ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

SIMULADOR DE CONDUCCIÓN 3D

Luis Salvador Roa Rodríguez

2 de junio de 2010

ESCUELA SUPERIOR DE INGENIERÍA

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

SIMULADOR DE CONDUCCIÓN 3D

- Departamento: Lenguajes y Sistemas Informáticos
- Directores del proyecto: Manuel Palomo Duarte, Francisco Palomo Lozano
- Autor del proyecto: Luis Salvador Roa Rodríguez

Cádiz, 2 de junio de 2010

Fdo: Luis Salvador Roa Roddríguez

Agradezco a mis padres todo su apoyo,
me han hecho posible terminar estos estudios.

Licencia

Índice general

Indice de figuras

Capítulo 1

Introducción

Este proyecto desarrolla un simulador de conducción en tres dimensiones.

Un simulador es un sistema que recrea una situación concreta del mundo real. Existen muchos tipos de simuladores. Los más sencillos pueden ser una pequeña aplicación, y también hay grandes infraestructuras que combinan maquinaria y software.

Un simulador de conducción es un simulador cuyo objetivo es situar al usuario a los mandos de un vehículo. En este caso, los más conocidos son los simuladores del campo de los videojuegos y máquinas recreativas. Hay simuladores para coches, motos, aviones, y casi cualquier medio de locomoción que exista.

Es un simulador en tres dimensiones. Las aplicaciones gráficas se pueden dividir en dos grupos, dos dimensiones y tres dimensiones. En dos dimensiones significa que toda la acción se desarrolla en el mismo plano. En las tres dimensiones se añade un tercer eje que da volumen o profundidad.

1.1. Objetivo y alcance

Este simulador de conducción está ideado para recrear mapas de carreteras reales. Centrado en este aspecto, toma una fuente de datos veraz y fiable sobre las carreteras existentes en todo el mundo para recrearlas en una aplicación 3D. Luego se añade la posibilidad de conducir un coche y explorar el mapa.

El punto de partida es el proyecto OpenStreetMap. Es un proyecto que ofrece información sobre mapas de carreteras entre otros datos cartográficos. Utilizar OpenStreetMap permitirá que el escenario sea una representación realista.

Para el apartado gráfico y desarrollo de la aplicación se utiliza el motor gráfico Panda3D que soporta las tres dimensiones, es una herramienta que proporciona un conjunto de funciones, clases y estructuras de datos para el desarrollo de videojuegos y renderizado en tres dimensiones. Los esfuerzos están orientados a la representación del trazado del mapa de carreteras y la circulación de los vehículos. Los elementos visibles están menos trabajados, intentando que sean lo más representativos posibles sin pretender ser un reflejo fiel de la realidad.

En cuanto al movimiento por el mapa de carreteras se realiza a los mandos de un vehículo. Los aspectos físicos del movimiento no son tratados aquí. El objetivo es que se muevan siguiendo las carreteras

de manera ordenada y correcta.

1.2. Definiciones y abreviaturas

2D: dos dimensiones.

3D: tres dimensiones.

API: es el conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Cliente HTTP: software que permite hacer solicitar información a un servidor HTTP.

DOM: Document Object Model, interfaz de programación de aplicaciones que proporciona un conjunto estándar para documentos HTML y XML.

DTD: definición de tipo de documento (siglas en inglés de Document Type Definition) es una descripción de estructura y sintaxis de un documento XML o SGML.

FPS: Frames Per Second, cuadros por segundo, indica cuantas veces se dibuja la ventana de una aplicación en un segundo.

Fuerzas G: es una medida de aceleración, 1G es equivalente a la aceleración producida por la gravedad estándar.

Fuente tipográfica: Conjunto de de letras utilizadas en un escrito con formato y estilo común.

GIMP: GNU Image Manipulation Program. Programa de manipulación de imágenes.

HTTP: Hypertext Transfer Protocol.

IDE: entorno de desarrollo integrado.

Main loop: bucle de ejecución de instrucciones de una aplicaciones.

Motor gráfico: herramienta para el desarrollo de aplicaciones gráficas.

OSM: OpenStreetMap.

Panda3D: motor gráfico para renderizado 3D y desarrollo de videojuegos.

Paquete deb: extensión del formato de paquetes de software de Debian.

Plugin: complemento que añade alguna característica específica a una aplicación.

Repositorio: lugar centralizado donde se almacena y administra información digital.

SVN: subversion. Es un sistema de control de versiones que permite administrar un repositorio.

Capítulo 2

Antecedentes

2.1. Simuladores

Existen muchos tipos de simuladores y de propósitos muy diversos. Los más conocidos son los creados para entretener, videojuegos y maquinas recreativas.

Un tipo de simulador es, por ejemplo, el utilizado para que los pilotos de aviones militar entrenen su resistencia a las fuerzas G. También son simuladores las máquinas utilizadas para hacer los test de rodaduras de los neumáticos. Lo son también los túneles de viento utilizados para analizar la aerodinámica de un vehículo.

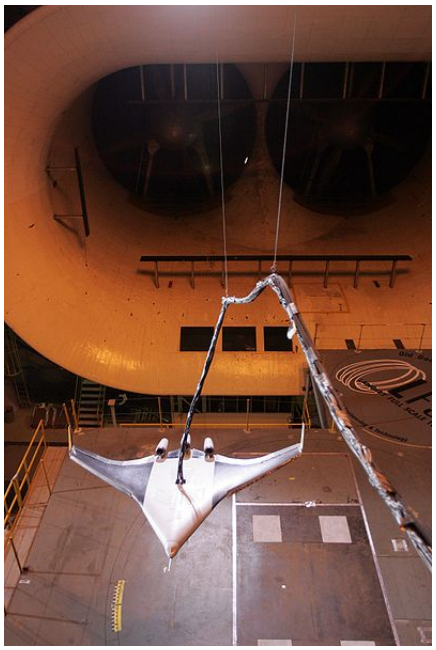

Figura 2.1: Túnel de viento, Wikipedia.

En el campo de los videojuegos hay todo tipo de simuladores: deportivos, conducción, baile, rol, etc. En la rama de los simuladores de conducción, la mayoría centran su realismo en el aspecto visual de los vehículos y del escenario, en detrimento del realismo aportado por la física y las matemáticas. En pocos años se ha pasado de aplicaciones sencillas y simples, a un grado de realismo muy alto, con aplicaciones

que se acercan mucho al comportamiento real del vehículo y calidad en el acabado visual.

Por ejemplo, los videojuegos sobre Formula 1 son los más realistas en cuanto a movimiento y funcionamiento de los vehículos. La serie Gran Turismo destaca por su calidad de aspecto visual. O el famoso Microsoft Flight Simulator, simulador de pilotaje de aviones que recrea con exactitud los controles de un número elevado de aeronaves.

2.2. OpenStreetMap, breve descripción

OpenStreetMap es un mapa libremente editable de todo el mundo. Permite ver, editar y usar información geográfica de manera colaborativa desde cualquier lugar del mundo. Es un proyecto dirigido expresamente a crear y ofrecer datos geográficos libres, tales como planos de calles, a cualquiera que los desee. El proyecto comenzó debido a que muchos mapas que se cree que son libres, tienen en realidad restricciones legales o técnicas para su uso, lo cual evita que la población los utilice de forma creativa, productiva o inesperada.

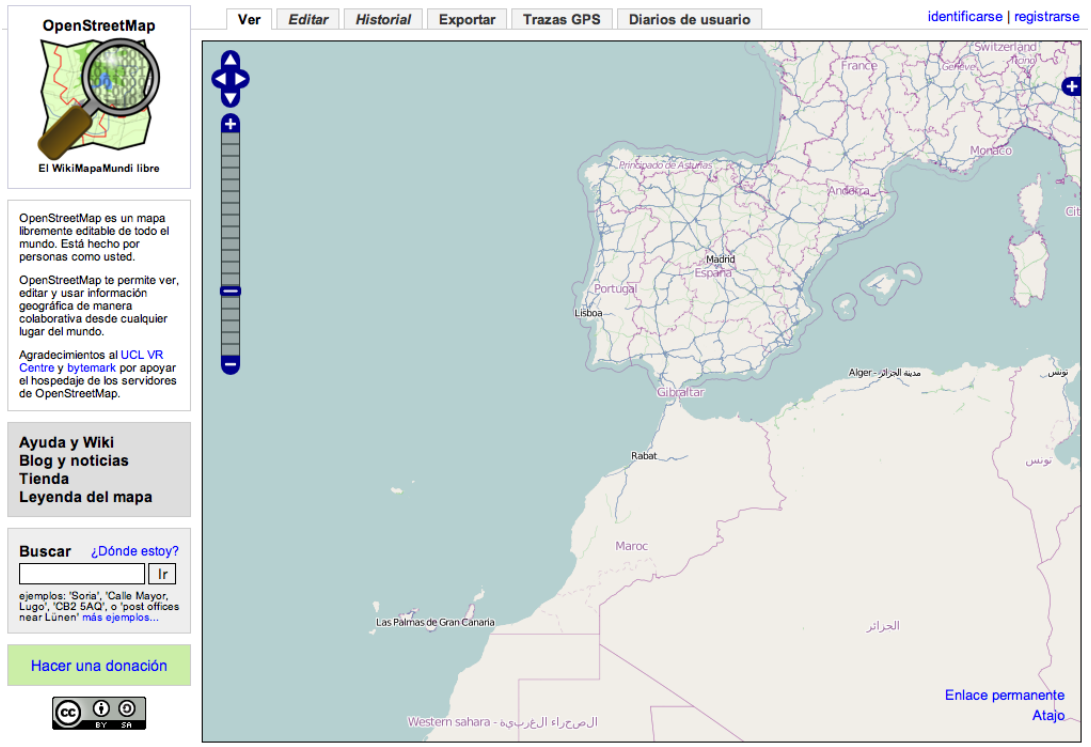

Figura 2.2: OSM, página principal.

El mapa se puede ver desde la web principal [7]. Para extraer parte de él y darle uso, hay varias maneras:

- Mediante la exportación a un documento con formato OpenStreetMap XML.
- Mediante imagen de Mapnik.
- Mediante imagen de OSMRender.

- HTML para pegar.
- Mediante cliente HTTP.

La información que contiene el mapa esta principalmente orientada a los planos de carreteras pero acepta cualquier tipo de información. Esto significa que cualquier elemento que se quiera representar en el mapa tiene cabida.

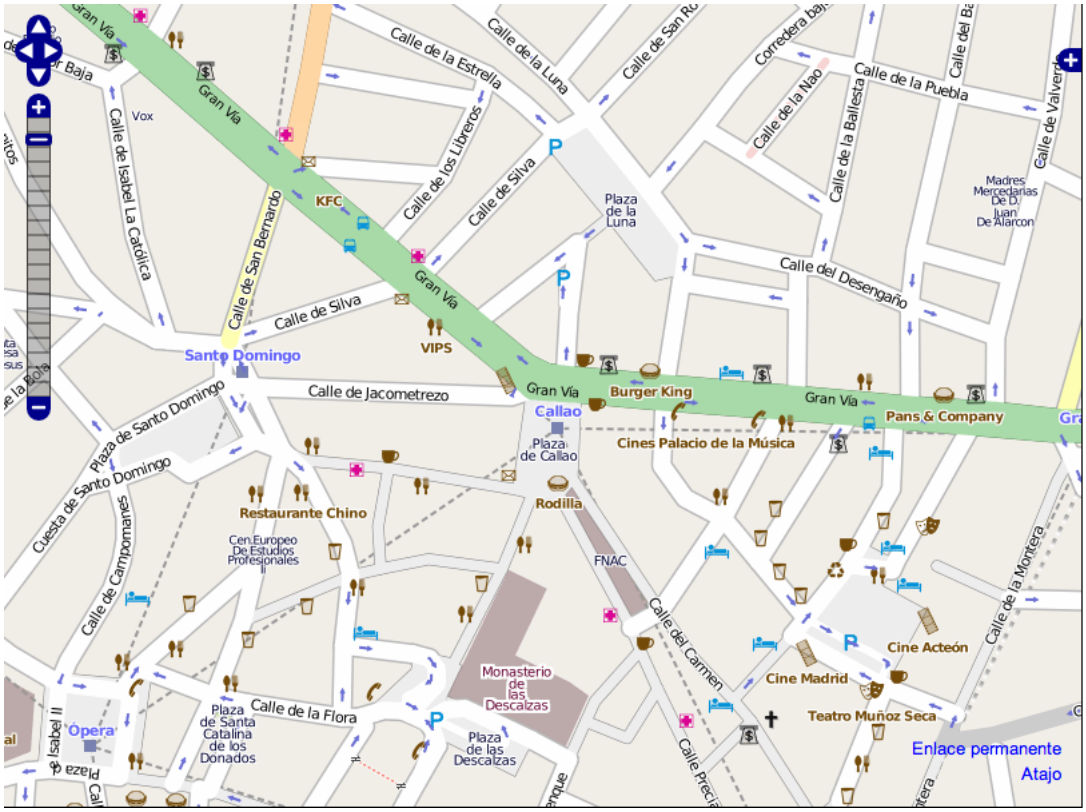

Figura 2.3: OSM, detalle del mapa.

Tiene disponible una wiki [21] donde se explican todas las características del proyecto. Hay un guía para comenzar a colaborar y subir trazados. También hay una sección de desarrollo dónde se detallan los aspectos técnicos del proyecto. Otra de las secciones está destinada a la cartografía.

Capítulo 3

Desarrollo del calendario

3.1. Delimitación del sistema y selección de tecnologías

1. Delimitación del sistema: 5 días.
2. Recopilación de información sobre las herramientas para el desarrollo de aplicaciones gráficas: 12 días.
3. Recopilación de información sobre las herramientas para el desarrollo de modelos 3D: 7 días.
4. Recopilación de información de información de viabilidad de uso del proyecto OpenStreetMap: 13 días.

Total: 37 días.

3.2. Aprendizaje

Antes de comenzar el proyecto ha sido necesario realizar una fase de aprendizaje de las herramientas utilizadas.

1. Estudio y aprendizaje del funcionamiento del motor gráfico Panda3D: 65 días.
2. Estudio y aprendizaje del uso del proyecto OpenStreetMap: 33 días.
3. Aprendizaje del manejo de la herramienta Deled3D: 4 días.
4. Aprendizaje del manejo de la herramienta GIMP: 4 días.
5. Aprendizaje del manejo de la herramienta Eclipse+Pydev+SVN: 4 días.

Total: 110 días.

3.3. Fase de análisis.

1. Elaboración de casos de uso: 2 días.
2. Elaboración del diagrama de clases conceptuales y modelo de comportamiento: 2 días.
3. Elaboración de la mecánica de la aplicación: 3 días.

Total: 7 días.

3.4. Fase de diseño e implementación

1. Trabajo de lectura del documento OSM: 20 días
2. Trabajo de implementación del cliente HTTP y uso de la APIv6 de OSM: 7 días.
3. Modelado 3D: 5 días.
4. Modelado 2D: 5 días
5. Implementación del menú: 14 días.
6. Implementación del escenario: 21 días.
7. Implementación de los vehículos: 17 días.
8. Implementación de la circulación: 34 días.

Total: 123 días.

3.5. Correcciones y mejoras

1. Corrección de errores: 23 días.
2. Mejoras de funcionamiento y eficiencia: 25 días.

Total: 48 días.

3.6. Elaboración de este documento

1. Elaboración y recopilación de gráficos: 1 días.
2. Elaboración del manual sobre Panda3D: 15 días.
3. Elaboración y recopilación de gráficos: 1 días.
4. Elaboración de este documento: 59 días.

Total: 76 días.

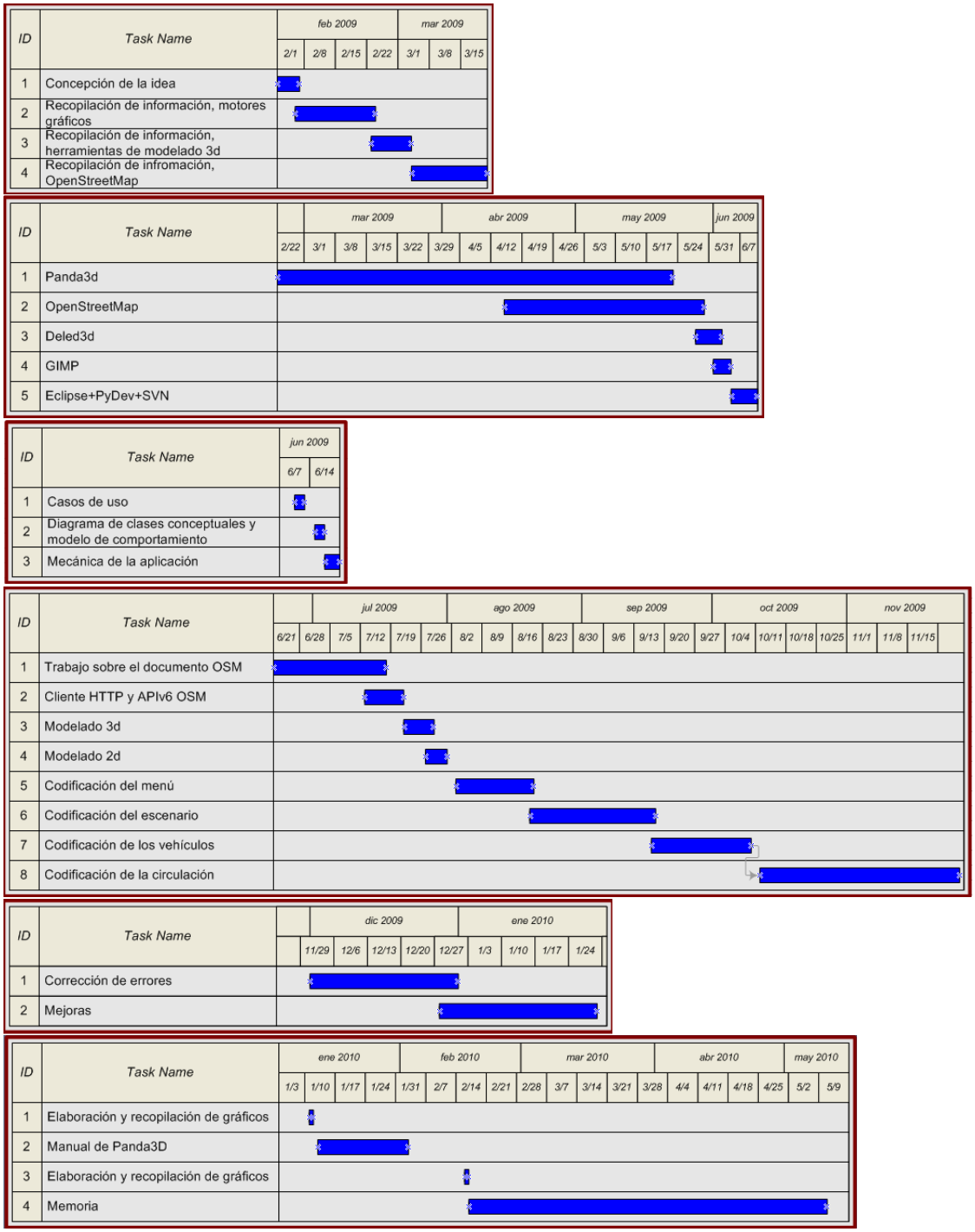

Figura 3.1: Diagrama de Gantt

Capítulo 4

Análisis

4.1. Modelos de casos de uso

A continuación se describen una serie de casos de usos que especifican el comportamiento deseado del sistema [16].

4.1.1. En el menú de la aplicación

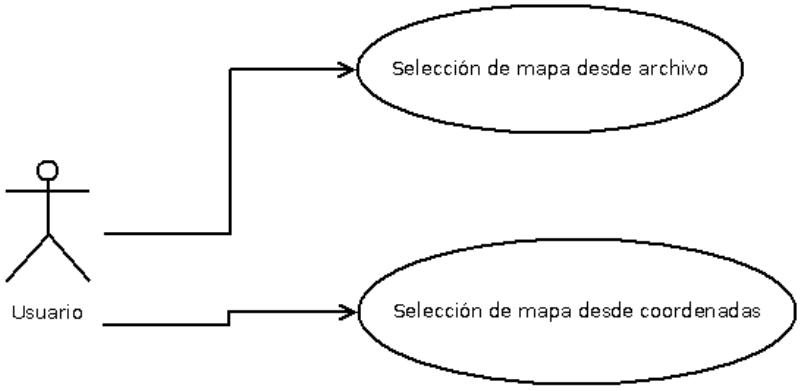

Figura 4.1: Diagrama de caso de uso 1.

Caso de uso: pantalla de selección mapas desde archivo.

- Actor principal: usuario.
- Precondiciones: existen mapas disponibles.
- Postcondiciones: el mapa seleccionado es cargado, son registradas las direcciones del mapa disponibles.
- Escenario principal:

  1. El usuario selecciona un mapa de la lista.
  2. Se genera un escenario a partir del mapa seleccionado.
  3. Se oculta el menú.

4. Se muestra el escenario.

- Escenario alternativos o de error:

  1a. No hay mapas disponibles. Se muestra un mensaje de advertencia.
  2a. No es posible crear el escenario. Se muestra un mensaje de error.
  * El usuario cambia a la pantalla de coordenadas.
  * El usuario cierra la aplicación.

Caso de uso: pantalla de introducción de coordenadas.

- Actor principal: usuario.
- Precondiciones: se dispone de conexión a internet.
- Postcondiciones: el mapa seleccionado es cargado, son registradas las direcciones del mapa disponibles.
- Escenario principal:

  1. El usuario introduce las coordenadas.
  2. Se genera un escenario a partir del mapa seleccionado.
  3. Se oculta el menú.
  4. Se muestra el escenario.

- Escenario alternativos o de error:

  1a. Las coordenadas son erróneas. Se muestra un mensaje de error.
  2a. No es posible crear el escenario. Se muestra un mensaje de error.
  * El usuario cambia a la pantalla de selección de mapa desde archivo.
  * El usuario cierra la aplicación.

4.1.2. En el escenario de la aplicación

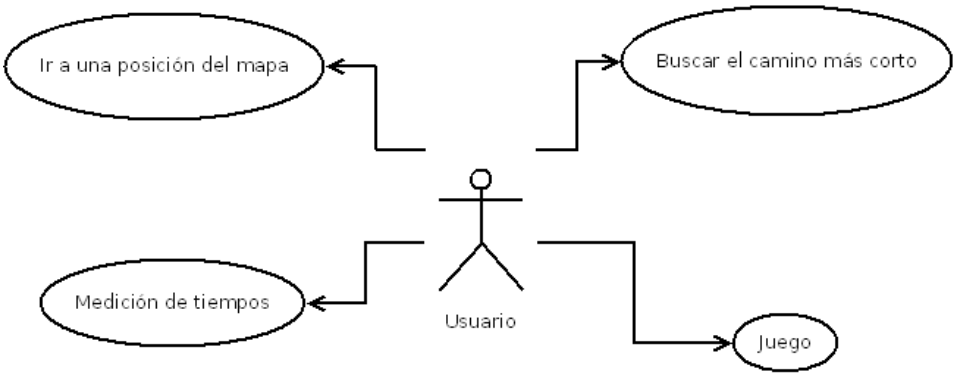

Figura 4.2: Diagrama de caso de uso 2.

Caso de uso: buscar el camino más corto.

- Actor principal: usuario.

- Precondiciones: existen direcciones registradas, existe un una ruta entre la posición actual y el destino elegido.
- Postcondiciones: se muestra camino más corto entre la posición actual y el destino elegido.
- Escenario principal:

    1. El usuario elige un destino.
    2. El sistema calcula la ruta más corta al destino.
    3. El sistema muestra al usuario la ruta.

- Escenario alternativos o de error:

    2a No hay conexión entre el origen y el destino. Se muestra un mensaje de error.

    * El usuario cancela la acción.
    * El usuario sale del escenario y vuelve al menú de la aplicación.
    * El usuario cierra la aplicación.

Caso de uso: Ir a una posición del mapa

- Actor principal: usuario
- Precondiciones: existen direcciones registradas.
- Postcondiciones: el usuario es transportado a la posición elegida.
- Escenario principal:

    1. El usuario selecciona una posición del mapa.
    2. El sistema transporta al usuario a la posición elegida.

- Escenario alternativos o de error:

    * El usuario cancela la acción.
    * El usuario sale del escenario y vuelve al menú de la aplicación.
    * El usuario cierra la aplicación.

Caso de uso: Medición de tiempos.

- Actor principal: usuario.
- Precondiciones: existen direcciones registradas.
- Postcondiciones: el tiempo queda registrado.
- Escenario principal:

    1. El usuario selecciona una dirección de destino.
    2. El sistema comienza a medir el tiempo transcurrido.
    3. Al llegar al destino, el sistema muestra el tiempo obtenido y lo registra.

- Escenario alternativos o de error:

  2a El sistema muestra registros anteriores si los hubiera.

  3a Si existe un mejor tiempo anterior no se registra el nuevo tiempo conseguido.

  * El usuario sale del escenario y vuelve al menú de la aplicación.

  * El usuario cancela la acción.

  * El usuario cierra la aplicación.

Caso de uso: Juego

- Actor principal: usuario.
- Precondiciones: existen direcciones registradas.
- Postcondiciones: ninguna.
- Escenario principal:

  1. El usuario activa el juego.
  2. El sistema muestra una dirección al azar.
  3. El usuario encuentra el destino.

- Escenario alternativos o de error:

  * El usuario cancela la acción.

  * El usuario sale del escenario y vuelve al menú de la aplicación.

  * El usuario cierra la aplicación.

4.1.3. Manejo del vehículo

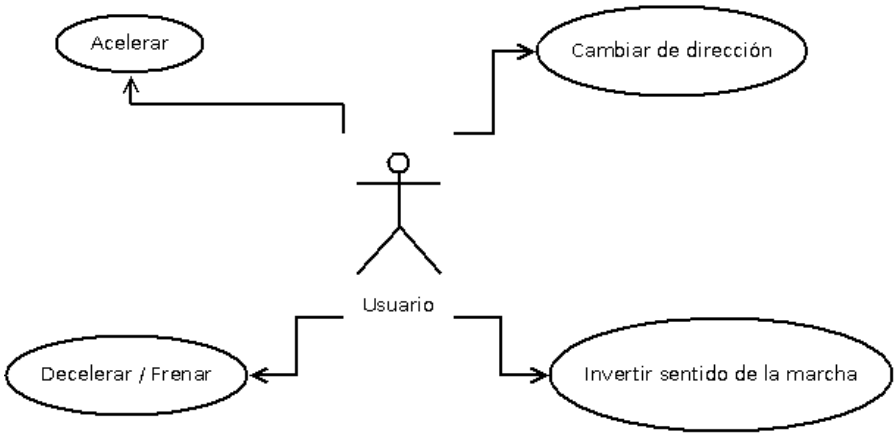

Figura 4.3: Diagrama de caso de uso 3.

Caso de uso: Acelerar

- Actor principal: usuario.

- Precondiciones: el vehículo no se desplaza a la velocidad máxima.
- Postcondiciones: el vehículo aumenta su velocidad.
- Escenario principal:

    1. El usuario solicita al sistema aumentar la velocidad del vehículo.
    2. El sistema comprueba que el vehículo no se desplaza a la velocidad máxima.
    3. El sistema aumenta la velocidad del vehículo.

- Escenario alternativos o de error:

    2a. El vehículo ya se desplaza a la velocidad máxima, mantiene su velocidad.
    * El usuario sale del escenario y vuelve al menú de la aplicación.
    * El usuario cierra la aplicación.

Caso de uso: Decelerar/Frenar

- Actor principal: usuario.
- Precondiciones: el vehículo no está parado.
- Postcondiciones: el vehículo disminuye su velocidad.
- Escenario principal:

    1. El usuario solicita al sistema disminuir la velocidad del vehículo.
    2. El sistema el sistema comprueba que el vehículo no está detenido.
    3. El sistema disminuye la velocidad del vehículo.

- Escenario alternativos o de error:

    2a El vehículo se desplaza ya a la velocidad mínima y el sistema lo detiene.
    * El usuario sale del escenario y vuelve al menú de la aplicación.
    * El usuario cierra la aplicación.

Caso de uso: Invertir sentido.

- Actor principal: usuario.
- Precondiciones: el vehículo se encuentra en una carretera de doble sentido.
- Postcondiciones: el vehículo cambia el sentido de la marcha.
- Escenario principal:

    1. El usuario solicita al sistema cambiar el sentido de la marcha.
    2. El sistema comprueba que el vehículo está en una carretera de doble sentido.
    3. El sistema cambia el sentido de la marcha del vehículo.

- Escenario alternativos o de error:

    2a El vehículo se encuentra en una carrera de sentido único.
    * El usuario sale del escenario y vuelve al menú de la aplicación.
    * El usuario cierra la aplicación.

Caso de uso: Cambiar dirección

- Actor principal: usuario.
- Precondiciones: existe más de una dirección posible.
- Postcondiciones: el vehículo cambia la próxima dirección a tomar.
- Escenario principal:

  1. El usuario solicita al sistema cambiar la próxima dirección a tomar.
  2. El sistema comprueba que hay más de una dirección posible.
  3. El sistema cambia la próxima dirección a tomar por el vehículo.

- Escenario alternativos o de error:

  2a Sólo hay una dirección posible.

  * El usuario sale del escenario y vuelve al menú de la aplicación.

  * El usuario cierra la aplicación.

4.2. Modelo conceptual de datos

A continuación se describen las estructuras de datos del sistema y las relaciones estáticas entre ellas [15].

El modelo conceptual de datos está reducido a las clases relevantes de la aplicación. Las clases pertenecientes al motor gráfico o creadas para simplificar otras clases son obviadas para que sea más comprensible el diagrama.

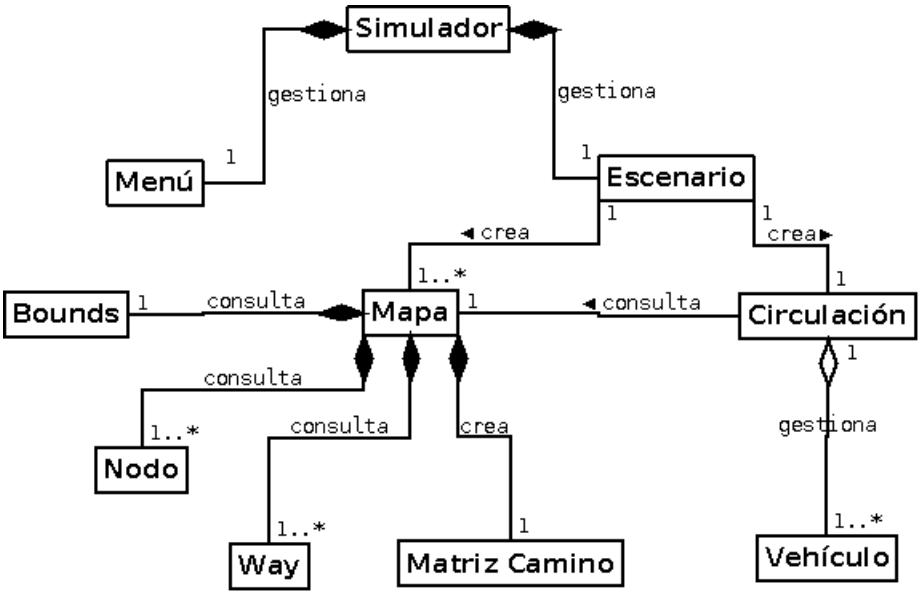

Figura 4.4: Diagrama de clases conceptuales.

Las clases que no aparecen en el diagrama son Area, Camino, Cielo, Figura, Flecha, FlechaMini, Highway, MapaMini, Nombre, Records, Suelo y Manejador.

La clase *Simulador* es la clase que desencadena el funcionamiento de la aplicación. Establece la configuración inicial del motor gráfico y crea las clases *Menu* y *Escenario*. Realiza las transiciones entre el menú y el escenario. También aporta a la clase *Escenario* los datos introducidos por el usuarios recogidos por la clase *Menu*.

Clase Simulador:

- __init__(): constructor de la clase, crea los objetos *Menu* y *Escenario*.
- MenuIrEscenario(): realiza la transición del menú al escenario.
- EscenarioIrMenu(): realiza la transición del escenario al menú.

La clase *Menu* se encarga de mostrar el menú de la aplicación y de recoger los datos que necesitará la clase *Escenario*.

Clase Menu:

- __init__(): constructor de la clase, crea y establece la configuración inicial de las estructuras de datos necesarias. Obtiene la lista de mapas disponibles desde archivo.
- Entrar(): muestra el menú.
- Salir(): oculta el menú.
- CambiarModoMapa(): realiza la transición entre la pantalla para seleccionar el mapa desde archivo y la pantalla para introducir las coordenadas.

La clase *Escenario* recibe la información introducida por el usuario en el menú como punto de partida para crear el escenario. El resto de información que necesita la toma de la clase *Mapa*. Con todo, pinta el escenario y crea la clase *Circulación*.

Clase Escenario:

- __init__(): constructor de la clase, crea y establece la configuración inicial de las estructuras de datos necesarias.
- Entrar(): crea y muestra el escenario.
- ComenzarCirculacionManual(): carga el escenario y la circulación a partir de una posición de inicio especificada por el usuario.
- ComenzarCirculaciónAutomático(): carga el escenario y la circulación a partir de una posición al azar.
- ActulizarEscenario: realiza el seguimiento al vehículo del usuario.
- LimiteBounds(): controla la salida del usuario de los límites del mapa.
- LimiteMiniBounds(): controla la salida del usuario de los límites visibles del mapa.
- ProcesarPorcionMapa(): procesa la información de la porción de mapa visible al usuario.

- MenuDireccion(): muestra los controles para elegir una dirección del mapa.
- MenuRuta(): muestra los controles para buscar una ruta a un destino seleccionado.
- CrearRuta(): muestra el camino a un destino seleccionado.
- MenuIrA(): muestra los controles para ir a un destino seleccionado.
- IrA(): desplaza al vehículo a una posición elegida.
- MenuRecord(): muestra los controles para comenzar una contrarreloj.
- MedirRecord(): realiza la medición de tiempos de la contrarreloj.
- MenuJuego(): muestra los controles para comenzar un juego.
- ComenzarJuego(): comienza la acción de juego.
- Salir(): oculta el escenario y elimina las estructuras de datos creadas.

La clase *Mapa* se encarga de obtener y procesar la información del mapa indicado en el menú. Haciendo uso de las clases *Bounds*, *Nodo* y *Way* para almacenar la información y la clase *Camino* para obtener la matriz de camino del mapa.

Clase Mapa:

- __init__(): constructor de la clase, obtiene y procesa toda la información referente al mapa. Genera la matriz de camino.
- CalcularPorcionMapa(): a partir de una posición y una distancia, determina que elementos y parte de la matriz de camino están a menor distancia de la posición que la especificada, este radio de acción es la parte visible del mapa.
- getPorcionMapa(): devuelve un conjunto de elementos perteneciente a la porción de mapa visible.
- getPorcionCamino(): devuelve la porción de la matriz de camino correspondiente a la parte visible del mapa.
- getRuta(): a partir de un origen y un destino determina el camino mínimo entre ambas posiciones del mapa.
- getBounds(): devuelve las posiciones límite del mapa.
- getNodoPosById(): devuelve la posición de un nodo a partir de su identificador.

La clase *Bounds* representa los límites del mapa.

Clase Bounds:

- __init__(): constructor de la clase.
- setMinLat(): establece la latitud mínima del mapa.
- getMinLat(): devuelve la latitud mínima del mapa.
- setMaxLat(): establece la latitud máxima del mapa.

- getMaxLat(): devuelve la latitud máxima del mapa.
- setMinLon(): establece la longitud mínima del mapa.
- getMinLon(): devuelve la longitud mínima del mapa.
- setMaxLon(): establece la longitud máxima del mapa.
- getMaxLon(): devuelve la longitud máxima del mapa.
- setMinPos(): establece la posición mínima del mapa.
- getMinPos(): devuelve la posición mínima del mapa.
- setMaxPos(): establece la posición máxima del mapa.
- getMaxPos(): devuelve la posición máxima del mapa.

La clase *Nodo* es utilizada para almacenar la información de las posiciones del mapa.

Clase Nodo:

- __init()__(): constructor de la clase.
- setDatos(): inicializa los datos del nodo.
- getPos(): devuelve la posición del nodo en el mapa.
- getTagValue(): devuelve el valor de una etiqueta especificada.
- getOneWay(): devuelve el valor oneway asociado al nodo.
- getWayName(): devuelve una lista de nombres de cada elemento del mapa que contiene este nodo.

La clase *Way* es utilizada para almacenar la información de los elementos que pueden aparecer en el mapa.

Clase Way:

- __init__(): constructor de la clase, registra el identificador en el mapa del elemento que representa.
- addref(): recibe un identificador de una posición del mapa y lo registra como asociado
- addTag(): registra una etiqueta (clave, valor),
- getListaRefs(): devuelve una lista de objetos de la clase *Nodo* asociado a elemento que representa.
- getTagValue(): devuelve el valor de una etiqueta especificada.
- setTipo(): establece el tiempo de elemento del mapa que representa.
- getTipo(): devuelve el tipo de elemento del mapa que representa.

La clase *Camino* se encarga de crear la matriz de camino.

Clase Camino:

- __init__(): constructor de la clase, recibe las posiciones y los elementos del mapa, y crea la matriz de camino.

- getListaNodos(): devuelve la lista de posiciones de la matriz de camino.

- getListaAdyacentes(): dada una posición del mapa, devuelve una lista con las posiciones del mapa accesibles desde ésta.

La clase *Circulación* se encarga de manejar la ubicación y movimiento de los vehículos.

Clase Circulación:

- __init__(): constructor de la clase, crea las estructuras de datos necesarias para manejar los vehículos.

- Crear(): a partir de un mapa y una posición crea los vehículos y su configuración inicial.

- Eliminar(): elimina los vehículos.

- ActualizarCamara(): realiza el seguimiento del vehículo del jugador por el mapa.

- Movimiento(): se encarga de controlar el movimiento de los vehículos.

- setRumbo(): establece las direcciones tomadas por los vehículos.

Clase Vehículo:

- __init__(): constructor de la clase, establece la configuración inicial del vehículo.

- Acelerar(): controla la velocidad del vehículo.

- Parar(): detiene el vehículo.

- Reanudar(): reanuda la marcha del vehículo.

- setPos(): sitúa al vehículo en la posición especificada.

4.3. Modelo de comportamiento

4.3.1. Selección de mapa desde archivo

Diagrama de comportamiento

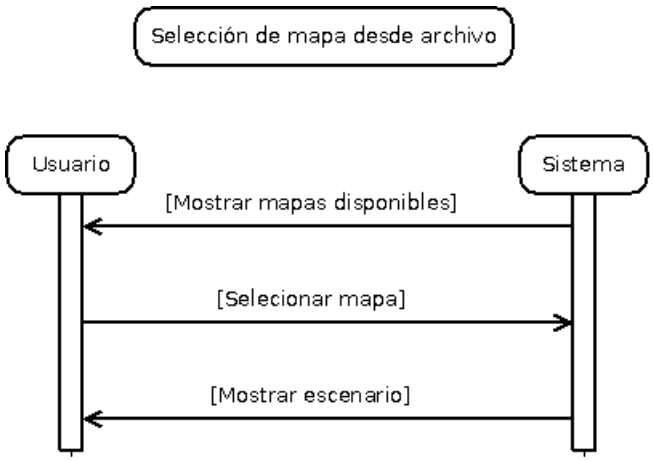

Figura 4.5: Modelo de comportamiento: Selección de mapa desde archivo.

Contrato de las operaciones

Operación: Mostrar mapas disponibles.

Responsabilidad: muestra una lista de mapas disponibles.

Precondiciones: hay mapas almacenados.

Postcondiciones: se elabora una lista de mapas disponibles.

Operación: Seleccionar mapa.

Responsabilidad: selecciona un mapa de la lista.

Precondiciones: hay mapas disponibles.

Postcondiciones: ninguna.

Operación: Mostrar escenario.

Responsabilidad: accede al escenario a partir del mapa seleccionado.

Precondiciones: el mapa elegido contiene información suficiente.

Postcondiciones: se crea un objeto *Escenario* a partir del mapa seleccionado.

4.3.2. Selección de mapa desde coordenadas.

Diagrama de comportamiento

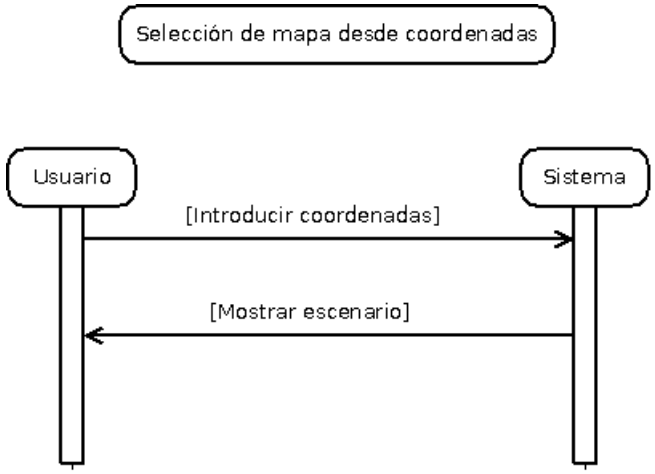

Figura 4.6: Modelo de comportamiento: Selección de mapa desde coordenadas.

Contrato de las operaciones

Operación: Introducir coordenadas.

Responsabilidad: el usuario selecciona un mapa a partir de las coordenadas.

Precondiciones: se dispone de conexión a internet, las coordenadas son correctas.

Postcondiciones: el mapa es descargado desde el servidor.

Operación: Mostrar escenario.

Responsabilidad: se accede al escenario a partir del mapa seleccionado.

Precondiciones: el mapa elegido contiene información suficiente.

Postcondiciones: se crea un objeto *Escenario* a partir del mapa seleccionado.

4.3.3. Buscar el camino más corto.

Diagrama de comportamiento

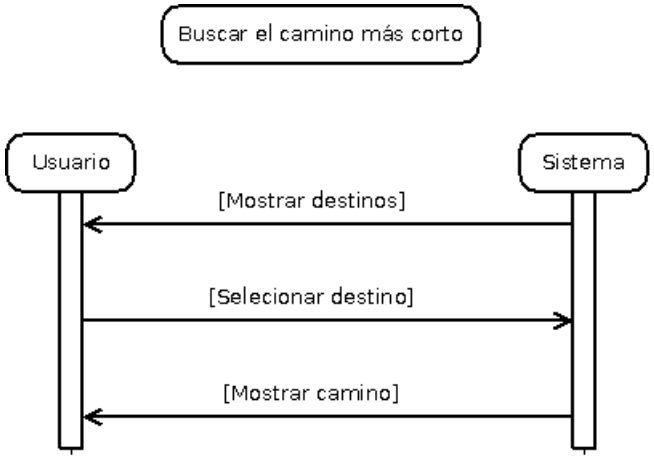

Figura 4.7: Modelo de comportamiento: Buscar el camino más corto.

Contrato de las operaciones

Operación: Mostrar destinos.

Responsabilidad: muestra los posibles destinos al usuario.

Precondiciones: el objeto *Mapa* contiene las direcciones del mapa.

Postcondiciones: se elabora una lista con los destinos disponibles.

Operación: Seleccionar destino.

Responsabilidad: selecciona un destino.

Precondiciones: existen destinos disponibles.

Postcondiciones: ninguna.

Operación: Mostrar camino.

Responsabilidad: muestra el camino hacia el destino seleccionado.

Precondiciones: existe, al menos, un camino entre el origen y el destino.

Postcondiciones: ninguna.

4.3.4. Ir a una posición del mapa

Diagrama de comportamiento

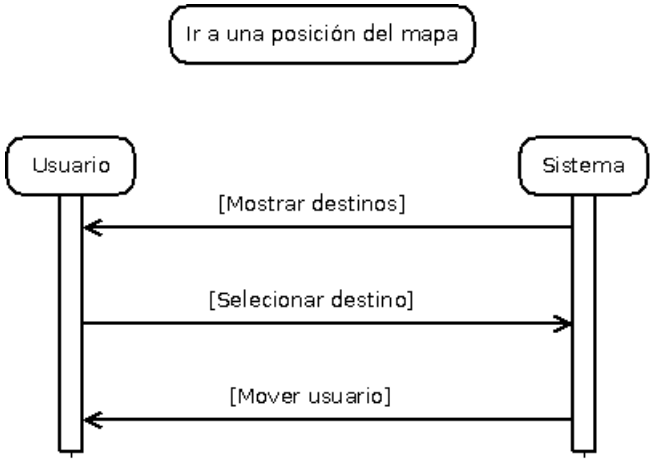

Figura 4.8: Modelo de comportamiento: Ir a una dirección del mapa.

Contrato de las operaciones

Operación: Mostrar destinos.

Responsabilidad: muestra los posibles destinos al usuario.

Precondiciones: el objeto *Mapa* contiene las direcciones del mapa.

Postcondiciones: se elabora una lista con los destinos disponibles.

Operación: Seleccionar destino.

Responsabilidad: selecciona un destino.

Precondiciones: existen destinos disponibles.

Postcondiciones: ninguna.

Operación: Mover usuario.

Responsabilidad: lleva al usuario a la posición del mapa de la dirección seleccionada.

Precondiciones: ninguna.

Postcondiciones: el vehículo del usuario es desplazado a la dirección seleccionada.

4.3.5. Medición de tiempos

Diagrama de comportamiento

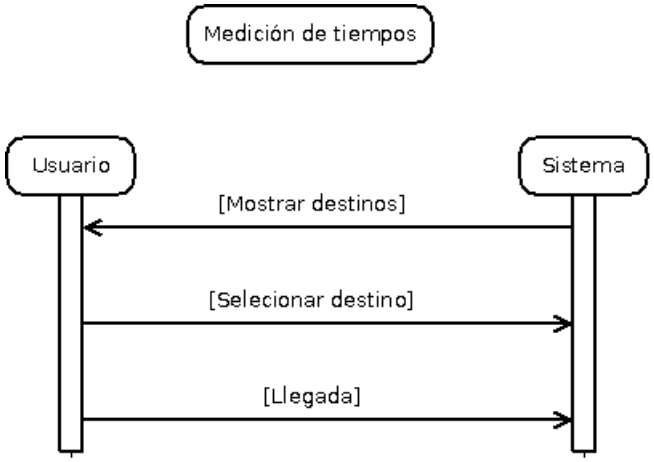

Figura 4.9: Modelo de comportamiento: Medición de tiempos.

Contrato de las operaciones

Operación: Mostrar destinos.

Responsabilidad: muestra los posibles destinos al usuario.

Precondiciones: el objeto *Mapa* contiene las direcciones del mapa.

Postcondiciones: ninguna.

Operación: Seleccionar destino.

Responsabilidad: selecciona un destino.

Precondiciones: existen destinos disponibles.

Postcondiciones: —

Operación: Llegada.

Responsabilidad: el usuario ha llegado al destino.

Precondiciones: existe un objeto *Escenario* creado.

Postcondiciones: —

4.3.6. Juego

Diagrama de comportamiento

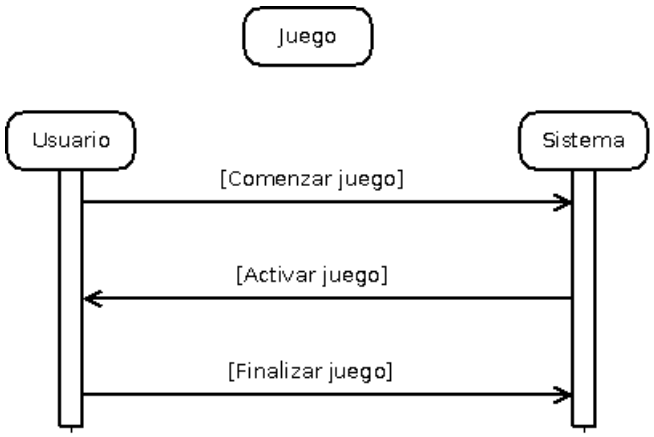

Figura 4.10: Modelo de comportamiento: Juego.

Contrato de las operaciones

Operación: Comenzar juego.

Responsabilidad: el usuario solicita comenzar el juego.

Precondiciones: existe un objeto *Escenario*, el objeto *Mapa* contiene las direcciones del mapa.

Postcondiciones: ninguna.

Operación: Activar juego.

Responsabilidad: el sistema arranca el juego.

Precondiciones: existe un objeto *Escenario*, el objeto *Mapa* contiene las direcciones del mapa.

Postcondiciones: ninguna

Operación: Finalizar juego.

Responsabilidad: el usuario finaliza el juego.

Precondiciones: existe un objeto *Escenario*, el objeto *Mapa* contiene las direcciones del mapa.

Postcondiciones: ninguna.

4.3.7. Acelerar

Diagrama de comportamiento

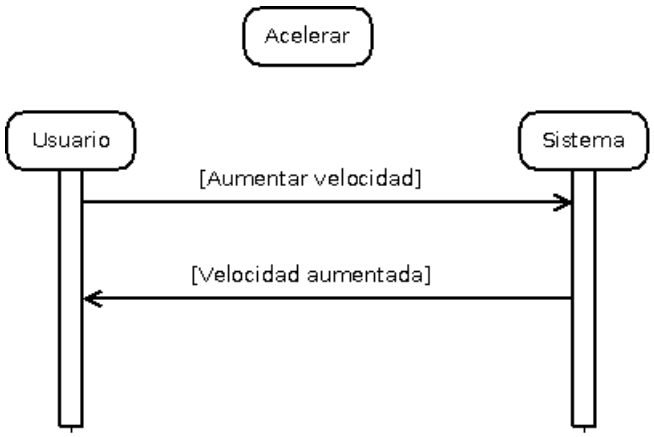

Figura 4.11: Modelo de comportamiento: Acelerar

Contrato de las operaciones

Operación: Aumentar velocidad.

Responsabilidad: el usuario solicita aumentar la velocidad del vehículo.

Precondiciones: existe un escenario y un vehículo del usuario.

Postcondiciones: ninguna.

Operación: Velocidad aumentada.

Responsabilidad: el sistema aumenta la velocidad del vehículo.

Precondiciones: existe un escenario y un vehículo del usuario.

Postcondiciones: la velocidad del vehículo del usuario es aumentada.

4.3.8. Decelerar

Diagrama de comportamiento

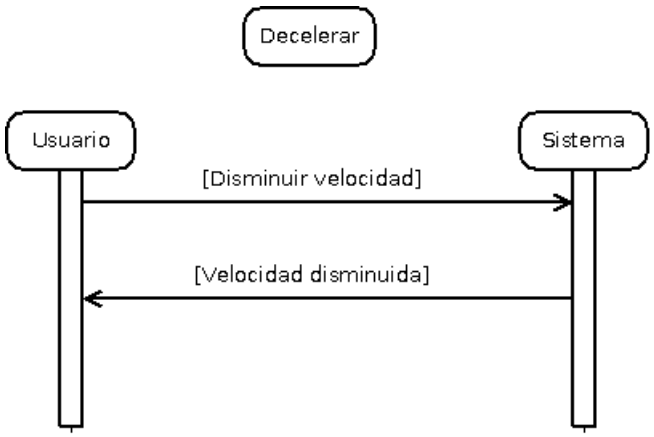

Figura 4.12: Modelo de comportamiento: Decelerar

Contrato de las operaciones

Operación: Disminuir velocidad.

Responsabilidad: el usuario solicita disminuir la velocidad del vehículo.

Precondiciones: existe un escenario y un vehículo del usuario.

Postcondiciones: ninguna.

Operación: Velocidad disminuida.

Responsabilidad: el sistema disminuye la velocidad del vehículo.

Precondiciones: existe un escenario y un vehículo del usuario.

Postcondiciones: la velocidad del vehículo del usuario es reducida.

4.3.9. Invertir sentido

Diagrama de comportamiento

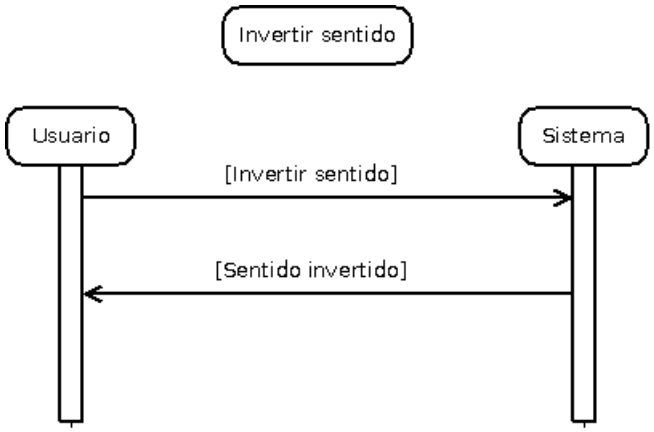

Figura 4.13: Modelo de comportamiento: Invertir sentido.

Contrato de las operaciones

Operación: Invertir sentido.

Responsabilidad: el usuario solicita modificar el sentido de la marcha.

Precondiciones: existe un escenario y un vehículo del usuario.

Postcondiciones: ninguna.

Operación: Sentido invertido.

Responsabilidad: el sistema modifica el sentido de la marcha.

Precondiciones: existe un escenario y un vehículo del usuario.

Postcondiciones: el vehículo del usuario cambia el sentido de la marcha.

4.3.10. Cambiar dirección

Diagrama de comportamiento

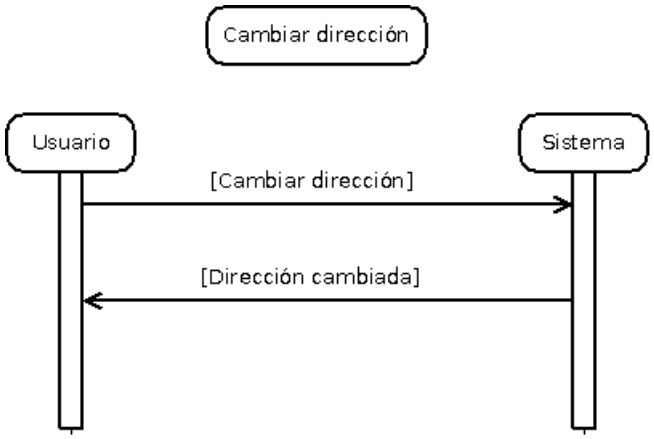

Figura 4.14: Modelo de comportamiento: Cambiar dirección.

Contrato de las operaciones

Operación: Cambiar dirección.

Responsabilidad: el usuario solicita modificar el sentido de la marcha.

Precondiciones: existe un escenario y un vehículo del usuario.

Postcondiciones: ninguna.

Operación: Dirección cambiada.

Responsabilidad: el sistema modifica el sentido de la marcha.

Precondiciones: existe un escenario y un vehículo del usuario.

Postcondiciones: el vehículo del usuario cambia el sentido de la marcha.

4.4. Mecánica de la aplicación

La ejecución comienza cargando y creando las estructuras de datos necesarias para acciones posteriores. Inicialmente se accede al menú. Desde el menú se accede al escenario. En cualquier momento se puede salir de la aplicación.

4.5. Funciones del producto

Esta aplicación permite acceder al mapa de carreteras proporcionado por OpenStreetMap en un formato distinto a la web, permite calcular la ruta más corta entre un origen y un destino, y da una perspectiva de exploración diferente, recreando el mapa a escala real en un estilo propio de los videojuegos.

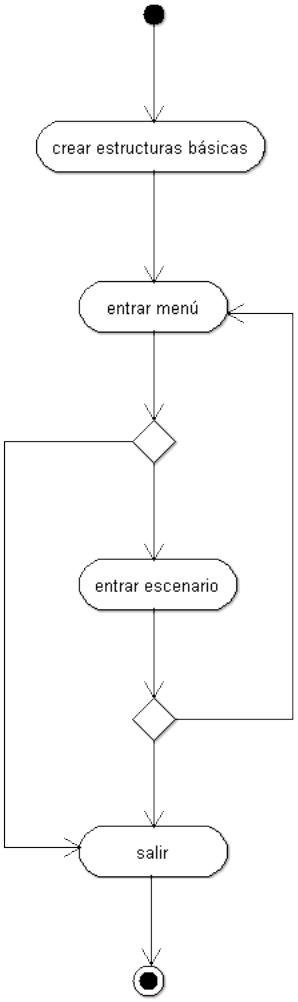

Figura 4.15: Diagrama de actividad de la aplicación.

Capítulo 5

Diseño

5.1. Arquitectura del sistema

El sistema se compone de dos partes, una formada por el motor gráfico y otra formada por la implementación de la funcionalidad del sistema y utiliza al motor gráfico.

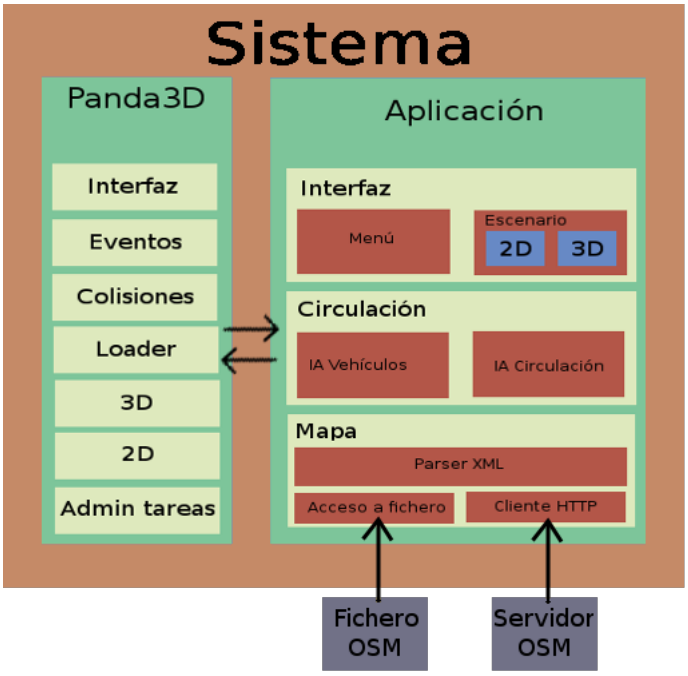

Figura 5.1: Arquitectura del sistema.

5.1.1. El motor gráfico

El motor gráfico tiene muchas características y algunas de ellas forman parte esencial de sistema. Del motor gráfico se hace uso de: el administrador de tareas, renderizado 2D, renderizado 3D, el módulo *loader* para cargar archivos de música o modelos 3D entre otros, el sistema de detección de colisiones, el administrador de eventos, y un conjunto de recursos para manipular la interfaz de la aplicación.

El administrador de tareas se encarga de la ejecución del bucle principal y permite añadir y manipular el código a ejecutar en él. El renderizado 2D y 3D se componen de un conjunto de estructuras para la creación y manipulación de todos los elementos gráficos. El sistema de detección de colisiones gestiona las colisiones producidas entre los elementos de la escena. El administrador de eventos se encarga de gestionar todos los eventos ocurridos en el sistema: colisiones, teclado y eventos propios. Panda3D permite modificar la interfaz manipulando desde la ventana de ejecución hasta la cámara en el escenario.

5.1.2. La aplicación

La aplicación se puede dividir en tres partes, con tres propósitos diferenciados. Una primera se encarga del interfaz, la parte visual de la aplicación. La segunda se encarga de los vehículos y la circulación. La tercera se encarga de procesar los mapas y procesar la información que contienen.

El interfaz se encarga de todo lo que es visto por el usuario, menú y escenario. El escenario contiene elementos 2D y 3D

La parte de circulación se encarga de gestionar cada vehículo y la circulación en general, y además, controla al vehículo del jugador.

La última parte, procesa los mapas y hace disponible al resto del sistema la información. Integra un parser XML para extraer la información. La fuente de datos del sistema puede ser un documento OSM almacenado en disco o la base de datos de OpenStreetMap a través de su API para consultas.

5.2. Diseño de arte

5.2.1. Banda sonora

Hay dos melodías de ambientación. Una sonará cuando se muestre el menú y la otra cuando se muestre el escenario. Ambas en un “loop” infinito.

5.2.2. Fuentes

Panda3D incorpora su propia fuente tipográfica por defecto. Pero se utiliza un conjunto de fuentes proporcionadas por el proyecto GNU FreeFont.

Figura 5.2: Tipografía.

5.2.3. Arte 2D

Son necesarias un conjunto de flechas utilizadas para realizar indicaciones, el fondo de pantalla del menú, un marcador de posición del vehículo y el círculo de llegada a destino. Todos los elementos 2D han sido creados con GIMP.

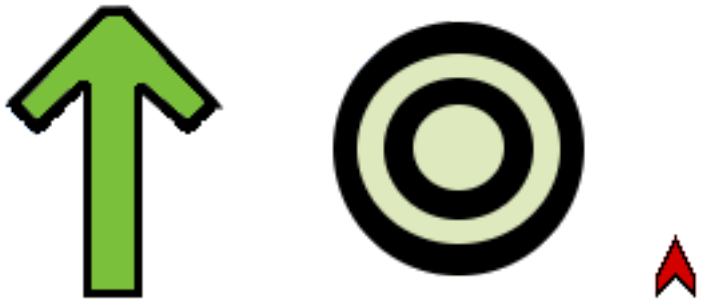

Figura 5.3: De izquierda a derecha: flecha, diana de llega al destino, marcador de posición del vehículo.

Ejemplo de creación con GIMP

Utilizando la herramienta de trazo, se puede crear el contorno de la flecha. Luego con la herramienta de relleno se puede colorear su interior. Finalmente, basta con exportar en formato PNG para tener la imagen disponible para usar.

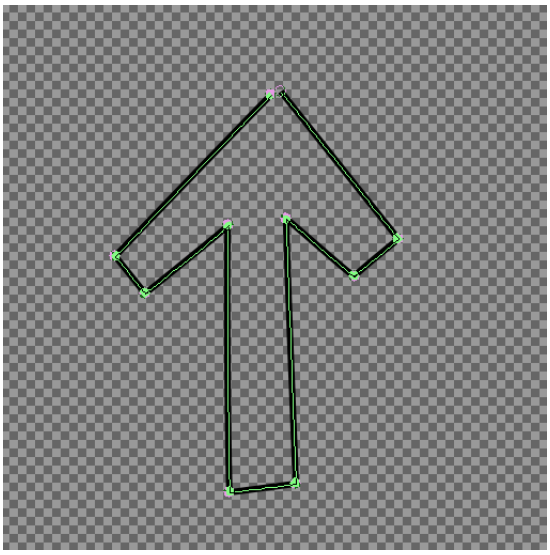

Figura 5.4: Herramienta de trazo, GIMP.

5.2.4. Modelos 3D

Panda3D pone a disposición de los desarrolladores un conjunto de modelos. En este proyecto se han utilizado tres de esos modelos. Uno para los vehículos, otro para la cúpula del cielo, y otro para un personaje infantil.

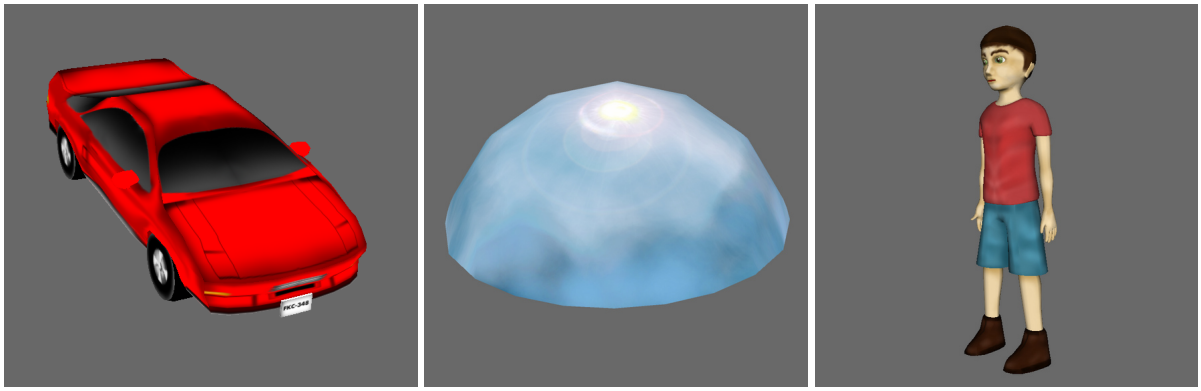

Figura 5.5: De izquierda a derecha: vehículo, cúpula del cielo, personaje infantil.

Son de creación propia los modelos para representar las carreteras y el nodo de unión entre los tramos de carreteras.

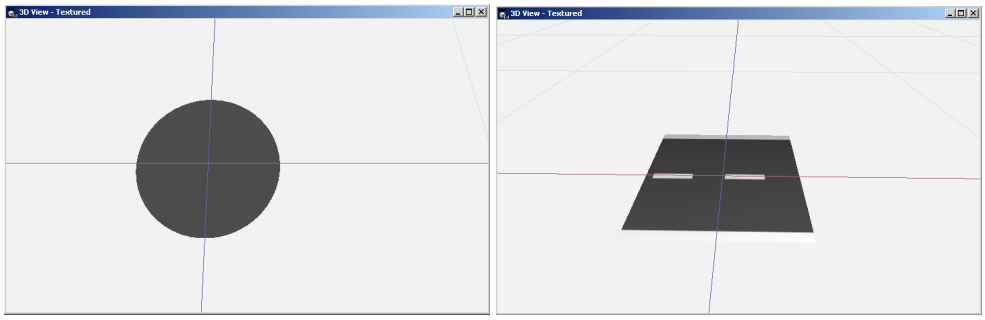

Figura 5.6: Nodo de unión entre tramos de carreteras y tramo de carretera.

En los lugares donde hay un elemento reseñable se coloca un hito con forma de cubo, pintado con una imagen que lo identifica. El cubo está en el aire y una línea une el cubo con el suelo. Junto al hito puede aparecer el nombre del elemento y en otros casos sólo aparecer el nombre.

Figura 5.7: Hito de representación.

Para representar plazas y jardines se dibuja sobre el suelo el área contenida con una textura identificativa.

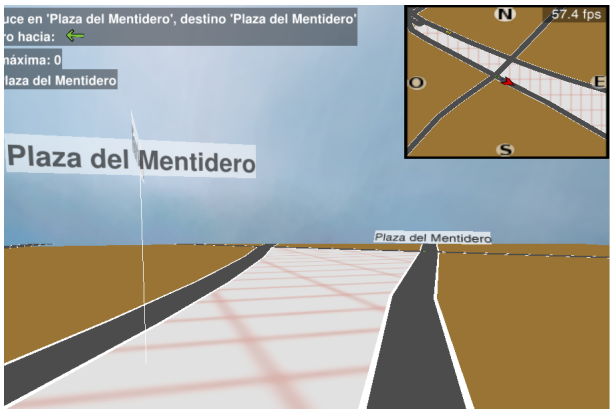

Figura 5.8: Plaza dibujada en la escena.

Ejemplo de creación con Deled3D

Para crear el tramo de carretera se selecciona la figura ’cube’ en el panel. Luego se reescala a las siguientes medidas absolutas: 32 píxeles de ancho, 32 píxeles profundidad y 0 píxeles de alto.

Luego se debe aplicar la textura adecuada. Seleccionando el editor de materiales, creando un nuevo material para crear la textura que luego será aplicada a la figura.

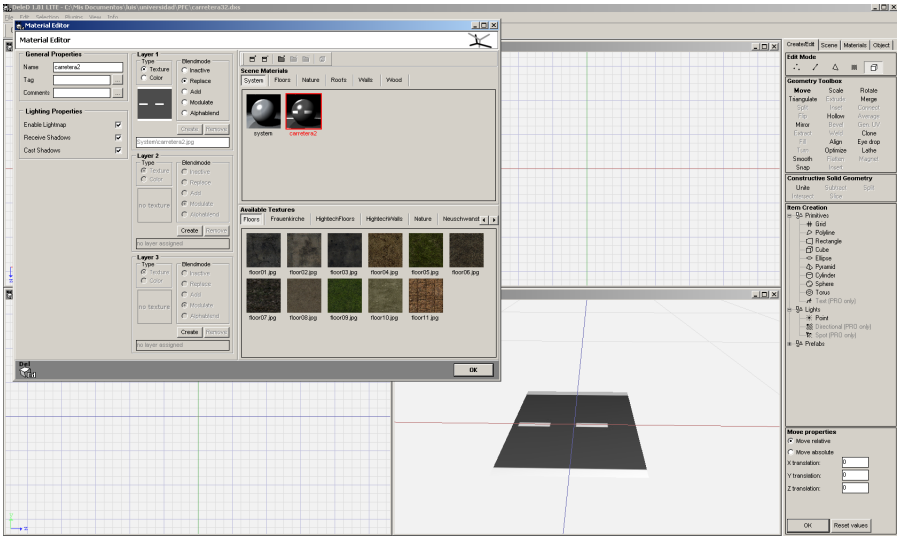

Figura 5.9: Creación del tramo de carretera.

5.3. Estrategia general para la creación de objetos

La estrategia seguida consiste en crear todos los objetos necesarios y establecer las configuraciones iniciales antes de que el motor gráfico arranque el bucle principal de ejecución. Luego, según la ejecución de la aplicación, se mostrarán o modificarán los objetos creados. La excepción sucede con la clase *Mapa*. Un objeto de la clase *Mapa* sólo puede ser creado cuando el usuario haya elegido un mapa, y no antes.

5.4. Menú de la aplicación

Al inicio de la aplicación se muestra el menú. El menú tiene como único objetivo permitir al usuario elegir a que mapa desea acceder. Está pensado para que sea lo más sencillo posible. Se compone de dos pantallas. En la primera pantalla se pueden seleccionar los mapas que están almacenados en archivos. En la segunda pantalla se pueden introducir las coordenadas para obtener el mapa desde el servidor de OSM.

Para poder seleccionar un mapa guardado previamente, la aplicación debe identificar cuáles son los documentos OSM disponibles. Se elabora una lista con los nombres de los documentos, y se da a elegir al usuario. En el caso de que el usuario quiera acceder al escenario a través de coordenadas deben ser numéricas, pueden tener signo negativo, y pueden tener decimales. El símbolo decimal es el punto.

En ambas pantallas se mostrarían los avisos si se produjera algún error. Los posibles problemas pueden ser: no hay mapas almacenados y error en el formato de las coordenadas. Seleccionado el mapa, si se produjera algún fallo o problema al leer el documento OSM no es responsabilidad del menú.

5.5. Acceso a la información

Para crear cada escenario la información es extraída del proyecto OpenStreetMap. Los datos se pueden obtener de dos maneras: de un archivo de texto con formato OSM, o utilizando un cliente HTTP.

5.5.1. El mapa desde un archivo de texto

Aunque es posible editar a mano el documento OSM, lo lógico es obtenerlo desde la web de OpenStreetMap. En la Web se puede exportar el mapa a un archivo de texto. Este archivo debe incluirse en la carpeta *mapas* de la instalación. Al comienzo de la ejecución se ofrecerá la posibilidad acceder al mapa que se corresponde con el archivo guardado. Accediendo al mapa a través del archivo de texto sólo se tendrá acceso a la región de mapa correspondiente a la información que este documento contenga.

5.5.2. El mapa desde el servidor

La segunda forma de obtener los datos del mapa es indicando a la aplicación unas coordenadas geográficas. La aplicación solicitará al servidor de OpenStreetMap los datos sobre una región del mapa centrada en dichas coordenadas.

La APIv0.6 para consultas a la base de datos de OpenStreetMap especifica el formato de la petición HTTP. Se utilizan cuatro parámetros: *left*, *bottom*, *right* y *top*; que delimitan la región del mapa. Especifican la longitud y latitud, mínima y máxima, del mapa. [22]

- left: valor de la longitud mínima del mapa.
- bottom: valor de la latitud mínima del mapa.
- right: valor de la longitud máxima del mapa.
- top: valor de la latitud máxima del mapa.

La región de mapa solicitada tiene las dimensiones de un cuadrado de X grados de lado. Si el vehículo del usuario llegase a los límites del mapa, se volvería a solicitar al servidor los datos de una nueva región del mapa centrado en la posición actual. Se elimina la limitación a la región de mapa seleccionada en primera instancia existente con los escenarios generados a partir de los archivos de texto.

5.5.3. Estructura del documento OSM

El documento OSM, tiene estructura de documento XML. Su correspondiente DTD se incluye en el anexo C al final de este documento.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="OpenStreetMap server">
  <bounds minlat="36.457984" minlon="-6.214474"
    maxlat="36.461603" maxlon="-6.208959"/>
  <node id="328975854" lat="36.4497096" lon="-6.2212205"
    version="1" changeset="748364" user="jmdg" uid="52498"
    visible="true" timestamp="2009-01-06T10:48:37Z"/>
  <node id="328975850" lat="36.452674" lon="-6.2185444"
    version="1" changeset="748364" user="jmdg" uid="52498"
    visible="true" timestamp="2009-01-06T10:48:36Z"/>
  <node id="328975844" lat="36.4586026" lon="-6.2130274"
    version="1" changeset="748364" user="jmdg" uid="52498"
    visible="true" timestamp="2009-01-06T10:48:36Z"/>
```

```
    <way id="29853941" visible="true" timestamp="2009-01-06T10:49:07Z"
    version="1" changeset="748364" user="jmdg" uid="52498">
      <nd ref="328975844"/>
      <nd ref="328975850"/>
      <nd ref="328975854"/>
      <tag k="place" v="city"/>
      <tag k="name" v="San Fernando"/>
      <tag k="landuse" v="residential"/>
    </way>
  </osm>
```

La raíz del documento es el nodo *osm*. Este nodo contiene, entre otros, tres nodos usados en este proyecto. El nodo *bounds*, el nodo *node* y el nodo *way*. [20]

El nodo *bounds* contiene la información sobre los límites del mapa. Significado de los atributos:

- minlat: latitud mínima del mapa.
- minlon: longitud mínima del mapa.
- maxlat: latitud máxima del mapa.
- maxlon: longitud máxima del mapa.

El nodo *node*, cada uno, representa una posición sobre el mapa. Existirán tantos elementos de este tipo como posiciones sean necesarias para describir a otros elementos. Contiene la información sobre una posición geográfica concreta del mapa. Si el nodo *node* representa un elemento del mapa formado por un solo nodo, entonces incluirá uno o más nodos *tag* para describirlo. Significado de los atributos:

- id: identificador único del nodo.
- lat: latitud del nodo.
- lon: longitud del nodo.

El nodo *way* representa cada elemento que pueda aparecer en el mapa [19]. Puede contener información sobre, una carretera, una línea de costa, un término municipal, un edificio, una plaza, un parque, etc. Se compone a su vez de dos clases de nodos: el nodo *nd*, utilizado para hacer referencia a la posiciones del mapa que definen a *way*; y el nodo *tag*, que describe a cada *way*.

El nodo *nd* contiene un atributo *ref*, que lo relaciona con el nodo *node* cuyo atributo *id* coincide con *ref*, así se posiciona cada *way* sobre el mapa.

El nodo *tag* contiene información muy diversa sobre el nodo *way* o *node* que lo contiene. Cada nodo *tag* tiene dos atributos *k* (key) y *v* (value) para almacenar la información.

5.5.4. Procesamiento del documento, OSM

Para procesar el documento OSM se utiliza SAX [13]. Es un analizador o parser, para documentos XML. Básicamente, recorre el documento linealmente proporcionando la información de cada nodo a un método designado para procesarlo.

La opción alternativa es DOM, que también es un parser, pero éste genera una estructura de árbol para trabajar con la información. Define varias interfaces con sus propios métodos y atributos. Permite modificar la estructura, navegar entre nodos y cambiar el estilo y contenido de los mismos. Este método necesita muchos más recursos en tiempo y espacio, que SAX, aparte de ofrecer muchas más posibilidades que no son necesarias. Cómo se emplea una estructura de datos a medida para la aplicación y los mapas son muy extensos, DOM no es una opción viable.

SAX funciona leyendo los nodos del documento XML linealmente, generando un evento al comienzo y final de cada nodo. Para dar respuesta a los eventos, el programador debe escribir su propia función, previamente debe conocer la estructura del documento. El parser ejecutará con cada evento la función que indique el programador, dándole la información sobre el nodo leído.

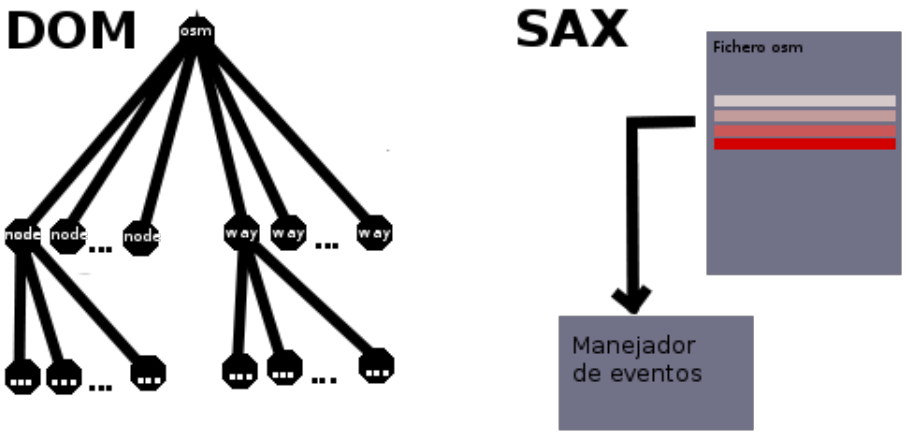

Figura 5.10: DOM vs. SAX.

5.5.5. Organización de la información del documento OSM

Es necesario que la información esté disponible durante la ejecución. En la lectura del documento OSM el parser almacena la información en una serie de estructuras de datos. Se puede decir que hay tres tipos de información: la referente a los límites del mapa, la referente a los nodos o posiciones del mapa, y la información sobre los elementos que aparecen en el mapa. Mediante colecciones de objetos la información se clasifica para conseguir que acceder a ella sea lo más eficiente posible.

Escalas y proporciones

El mapa definido en el documento OSM está medido en grados, latitud y longitud. Antes de guardar la información, para que sea útil, se debe establecer la conversión a una unidad utilizable por el motor gráfico. El tamaño y posición de cada elemento que aparece en la escena tiene que quedar proporcional con el resto de elementos. El método que utilizado es crear una tabla de equivalencias. Hay tres equivalencias básicas.

Se debe aclarar primero que el motor gráfico representa las posiciones en la escena mediante los ejes cartesianos x, y, z. Medido en unidades. Las equivalencias son:

- Grados, metros: OpenStreetMap proporciona las medidas en coordenadas geográficas, latitud y longitud. Como aproximación tomada, 1 grado equivale a 112.000 metros.
- Metros, unidades: Ésta es una correspondencia artificial, y he obtenido buenos resultados con valores entre 15 y 20 unidades para representar 1 metro en la escena.
- Km/h, m/s: Un kilómetro a la hora equivale a 0.278551532 metros por segundo.

Las dos primeras equivalencias sirven para pasar las mediciones en grados su correspondiente en unidades. La tercera es utilizada para calcular la equivalencia kilómetros a la hora/unidades por segundo, necesaria para el movimiento de los vehículos.

Conociendo la relación grado/metro/unidad, se pueden ubicar los elementos proporcionalmente y a escala, en la escena.

La relación metro/unidad permite calcular el tamaño de los elementos de la escena. Un elemento con factor de escala 1 conserva su tamaño original, con 0.5 será reducido de tamaño a la mitad y con valor 2 será el doble de grande. Si un elemento mide 32 unidades y debe medir 10 metros en la escena la cuenta es: (1 metro=20 unidades, por ejemplo) 20*10 / 32 = 6.25.

La información de los límites del mapa

La información sobre las dimensiones del mapa se guarda en la clase *Bounds*, almacena: latitud mínima y máxima, y longitud mínima y máxima. Estas coordenadas geográficas deben ser transformadas a una medida equivalente para el motor gráfico.

Los valores mínimos de latitud y longitud se hacen corresponder a la posición (0,0,0) de los ejes de la escena. La longitud se relaciona con el eje 'x' y la latitud con el eje 'y'. Ésta es la referencia para calcular la posición del resto de nodos. La conversión se realiza durante la ejecución del parser.

Las posiciones sobre el mapa, el nodo *node*

Un mapa OSM está compuesto por un conjunto de posiciones o nodos. Cada nodo señala que en ese lugar hay un elemento o parte de él. Los nodos contienen las coordenadas geográficas, además de un identificador único del nodo.

En ocasiones, un nodo puede constituir por sí mismo un elemento del mapa. Cuando esto ocurre, el nodo contiene además una serie de nodos *tag* dentro del propio *node*. Para los elementos del mapa formados por más de un nodo se describen a través del nodo *way*.

Los nodos *tag* sirven para describir al elemento. Cualquier característica o cualidad se describe a través de este tipo de nodos.

Los elementos del mapa, el nodo *way*

Los elementos del mapa se describen dentro del nodo *way*, y la información de este nodo se guarda en instancias de la clase *Way*. Los objetos toman como identificador el propio identificador único del nodo

way.

El nodo *way* contiene dos tipos de nodos, una serie de nodos *nd* y algunos nodos *tag*.

El nodo *nd* posee un atributo *ref* cuyo valor se corresponde con el identificador único de uno de los nodos *node*. Esta referencia posiciona sobre el mapa al elemento que se está describiendo. Para guardar estas referencias se utiliza una lista.

De nuevo, los nodos *tag* son utilizados para indicar las características de los elementos.

5.6. La matriz de camino

El mapa describe, entre otros elementos, las carreteras de la región seleccionada. Cada carretera, calle o autovía aparecen por separado cada una en un nodo *way*. A partir de esta información se crea la matriz de camino.

Una carretera se define por los nodos *nd*. Ubica la carretera y el sentido de la marcha. Cada uno de los nodos es tomado como un nodo de la matriz de camino. Se utilizan los identificadores únicos de los nodos *node*.

Un nodo está enlazado con el nodo inmediatamente siguiente en la lista de nodos de *way*. Si la vía es de doble sentido, el nodo es también enlazado con el nodo anterior en la lista. La conexión entre dos vías distintas está en que comparten nodos *node*.

La matriz de camino es un grafo disperso y débilmente conexo. La estructura de datos utilizada para la matriz es un conjunto listas de adyacencias. Las listas están contenidas dentro de cada objeto *Nodo* cuyo identificador de nodo coincide con el nodo referenciado en el nodo *way* leído.

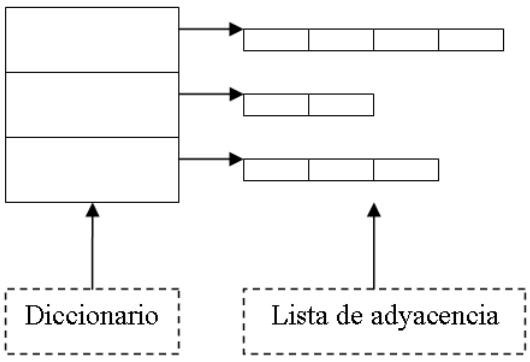

Figura 5.11: Representación de la matriz camino.

El trabajo de creación de la matriz la realiza el parser durante la lectura del documento OSM.

5.7. Creación del escenario

La elección del mapa desencadena el análisis del documento OSM y con la información organizada y optimizada se comienza a construir el escenario. No hay escenarios guardados previamente, no es posible.

5.7.1. Estrategia de creación del escenario

Un mapa puede llegar a ser demasiado extenso para que el motor gráfico lo maneje en su totalidad. Para evitar problemas de rendimiento se procesa sólo la parte del mapa que está en torno al jugador y se omiten las zonas lejanas.

La forma de saber cuáles son los elementos que se deben poner en la escena es comparar su posición en el mapa con la del vehículo del usuario. Si un elemento está lo suficientemente cerca de la posición de referencia, será creado. Este proceso se realiza a todos los elementos del mapa, incluidas las carreteras. Si el usuario se sale de la zona del mapa dibujada, se repiten los cálculos para la nueva región en torno a la nueva posición de referencia.

5.7.2. Listas de elementos para la optimización

Durante la lectura del documento OSM, se elaboran una serie de listas para la optimización de la construcción del escenario. Hay muchos elementos diferentes y muy numerosos, en cada mapa. La estrategia seguida para crear el escenario (sólo las regiones cercanas a posición del usuario) obliga a identificar a cada vez los elementos a representar. Una vez elegidos los elementos 'cercanos' hay que identificarlos para poder representarlos en la escena con el algoritmo correspondiente. El sentido de utilizar listas para agrupar los elementos procesados con el mismo algoritmo es evitar identificar los elementos del mapa a cada vez.

5.7.3. La red de las carreteras

La red de carreteras se compone de todas la carreteras definidas en el documento OSM. El mapa define cada carretera por separado, incluso puede que una misma carretera esté dividida en dos o más tramos y definida por partes. Cada definición en el documento OSM de una carretera o tramo, es procesada de manera independiente del resto. La forma en que las carreteras se representan crea el efecto de que forman un conjunto conexo. Para esta aplicación, las carreteras se han clasificado en dos tipos: de sentido único y de doble sentido; pero ambas son procesadas de manera similar.

El camino que sigue una carretera está determinado por un conjunto de nodos o posiciones sobre el mapa. Para formar las carreteras se coloca un tramo que conecta cada par de nodos de la carretera, y un tramo de unión sobre cada nodo.

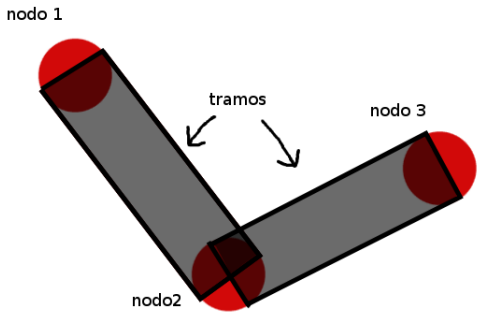

Figura 5.12: Formación de un tramo de carretera.

Para dibujar las líneas de los bordes de la carretera y las líneas de separación de carril se utiliza un sistema de capas superpuestas. Se emplean tres capas que luego se aplanan quedando unas sobre otras haciendo parecer que realmente las líneas han sido pintadas sobre el asfalto. La capa más baja es la más ancha y sirve para crear los bordes de la carretera. La capa intermedia es el asfalto. La capa superior es para las líneas de separación de carril. La líneas de separación de carril solo son incluidas en las carreteras de doble sentido.

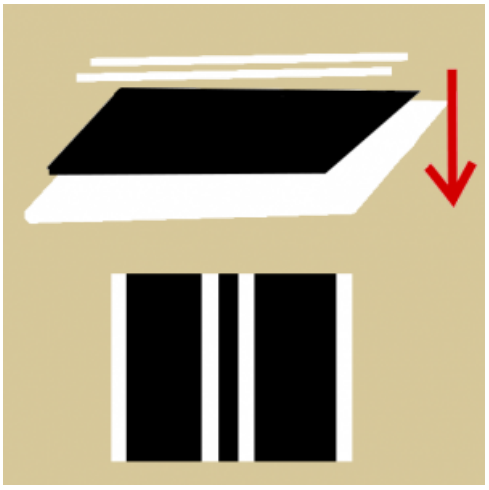

Figura 5.13: Formación de la carretera mediante capas.

5.7.4. La cúpula del cielo

La cúpula del cielo es un elemento que no está definido en el documento OSM. Consiste en una semiesfera que envuelve al vehículo del jugador. Sólo se ve su parte interior que está texturizada con una imagen de cielo, sol y nubes. Sigue el movimiento del vehículo por el escenario y se renderiza siempre en segundo plano respecto del resto de elementos.

5.7.5. El suelo del escenario

El suelo es un elemento que no está definido en el documento OSM pero que sí es importante para el aspecto general del mapa. Es la parte de escenario sobre la que se colocan el resto de elementos de la escena. Otros elementos pueden tener su propia área de suelo, pero estos espacios serán creados cuando se generen dichos elementos. Sirve para evitar que el escenario parezca estar 'en el aire'.

5.7.6. Elementos definidos como áreas

En el documento OSM está pensado para mapas de dos dimensiones, por lo tanto, muchos elementos están definidos mediante un polígono irregular o área. Los datos de origen informan de la ubicación de los vértices del polígono. Las áreas son pintadas sobre el escenario para hacerlas reconocibles.

5.7.7. Edificios

En cierta ocasiones las los elementos definidos como áreas son en realidad edificios. Para representarlos se utiliza un figura geométrica que abarca todo el espacio del área, con un cierto volumen. La texturización es genérica para todos los edificios.

5.7.8. Nombres sobre el escenario

Muchos de los elementos descritos en el documento OSM tienen un nombre. Este nombre no tiene por qué ser único, pero sí es de gran utilidad para el usuario al identificar los elementos del mapa.

La forma de indicar los nombres es mediante carteles sobre la posición del elemento. Son visibles desde todas las direcciones y están colocados en altura sobre una línea que los une al suelo.

5.7.9. Iconización de los lugares de interés

El documento OSM hace referencia a multitud elementos que pueden aparecer en un mapa: aparcamientos, iglesias, zonas de ocio, restaurantes, museos, plazas públicas, hospitales, farmacias, etc. Para señalar que en cierta posición del mapa hay elemento se utiliza un icono con una imagen identificativa, independientemente de la representación del propio elemento a través de otras formas visuales.

Un icono consiste en una figura geométrica con forma de cubo que en cada una de sus caras tiene una imagen características. Está situado a cierta altura y unido al suelo mediante una línea sobre la posición del elemento al que está asociado.

5.8. Circulación de vehículos

Sobre el escenario circulan vehículos de manera aleatoria y autónoma. Los vehículos se guían por las carreteras utilizando la matriz de camino. Su velocidad será la máxima permitida por la vía en que circulen, definida en el documento OSM.

Un número elevado de vehículos reduce drásticamente el rendimiento de la aplicación. Pero los mapas son tan extensos, que unos pocos vehículos repartidos por el escenario apenas aparecerían por delante

de la cámara. La solución es utilizar pocos vehículos que siempre se mueven alrededor del vehículo manejado del usuario. La idea es sencilla y crea el efecto de que son numerosos evitando problemas de rendimiento.

Cuando se busca una posición de inicio de un vehículo, se selecciona una posición aleatoria y cercana al vehículo del usuario. Los vehículos siguen una ruta aleatoria también. Si los vehículos se alejan del vehículo del usuario son recolocados, de nuevo, próximos al vehículo de referencia.

Los cálculos para mover los vehículos están dentro del bucle principal de ejecución, pero no la determinación de la ruta. Un vehículo se desplaza de tramo en tramo de carretera. Cuando llega al final del tramo se analizan las posibilidades y gira en una dirección nueva. La manera más eficiente de saber si un vehículo debe tomar una nueva dirección es mediante eventos. Al llegar un vehículo al final de un tramo un evento activa la selección de la nueva dirección.

Para evitar que los vehículos se atraviesen unos a otros se utiliza un sistema de detección y gestión de colisiones. Cada colisión produce un evento, y de nuevo se pone en marcha la solución para estas situaciones. Cuando un vehículo choca con otro se detiene inmediatamente, y se espera a que el otro vehículo se aleje para reanudar la marcha.

5.8.1. El vehículo del usuario

La forma que tiene el usuario para moverse por el escenario es un vehículo. Es un vehículo como el resto de los vehículos que componen el tráfico, pero el usuario controla su posición de partida, velocidad y dirección.

La ruta del vehículo se escoge en cada cruce de carreteras donde exista más de una opción posible. La opción elegida se observa mediante una flecha sobre el cruce y otra en lo alto de la pantalla. Además de la flecha se puede ver el nombre de la vía hacia la que se dirigirá el vehículo.

5.9. Búsqueda de la dirección seleccionada

Cuando el usuario está en el escenario puede solicitar a la aplicación que le muestre el camino hacia una dirección concreta. La ruta se muestra mediante un camino de flechas sobre el suelo y una diana en el destino. La ruta será siempre la más corta en caso de que sea posible llegar desde la posición de inicio al destino.

Figura 5.14: Camino más corto.

5.10. Traslado inmediato a una dirección

Siempre que existan direcciones disponibles, en caso de que el usuario quiera desplazarse a una dirección sin guiar al vehículo hasta ella, puede hacerlo. El vehículo manejado por el usuario será transportado a la dirección que se indique.

5.11. Medición de tiempos

El usuario tiene la posibilidad de cronometrar sus desplazamientos entre dos direcciones. Contando desde la posición actual, hasta llegar a un destino selecciona se mide el tiempo que tarde el usuario en llegar. Los tiempos mejores tiempos son almacenados, y son comparados con mediciones futuras. Se muestra una comparativa con un tiempo anterior y si mejora se almacena.

5.12. Un pequeño juego

La aplicación ofrece al usuario un juego sencillo. El juego consiste en que el sistema elegirá una dirección del escenario al azar, y con el cronómetro en marcha el usuario deberá encontrarla. La única ayuda será una flecha que actúe de brújula.

Figura 5.15: Juego.

Capítulo 6

Implementación

6.1. Cómo se obtiene el documento OSM y selección del mapa

Hay dos vías de acceso al documento OSM y selección del mapa. El documento puede estar en la carpeta `mapas` de la instalación. También se puede obtener, en tiempo de ejecución, utilizando un cliente HTTP.

6.1.1. Guardar el mapa en un fichero de texto

Aunque es posible editar a mano el fichero OSM, lo habitual es obtenerlo de la web [7]. En la parte superior de la página se encuentra la pestaña *Exportar*.

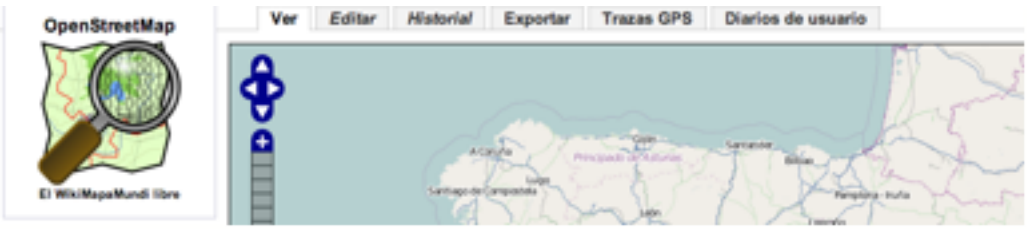

Figura 6.1: OSM, página principal.

En la pantalla de exportación, se ven varias opciones. Hay que seleccionar *Datos formato OpenStreetMap XML*. Al pulsar el botón *Exportar* se genera un documento OSM, con la información sobre la región de mapa seleccionada. Este archivo debe ser incluido en la carpeta `/mapas` de la instalación. La región de mapa a exportar se puede elegir desplazando el mapa visible con el ratón, o mediante coordenadas en el apartado *Área a exportar*. En caso de que se desee indicar las coordenadas, éstas se corresponden los límites del mapa en latitud mínima y máxima, y longitud mínima y máxima.

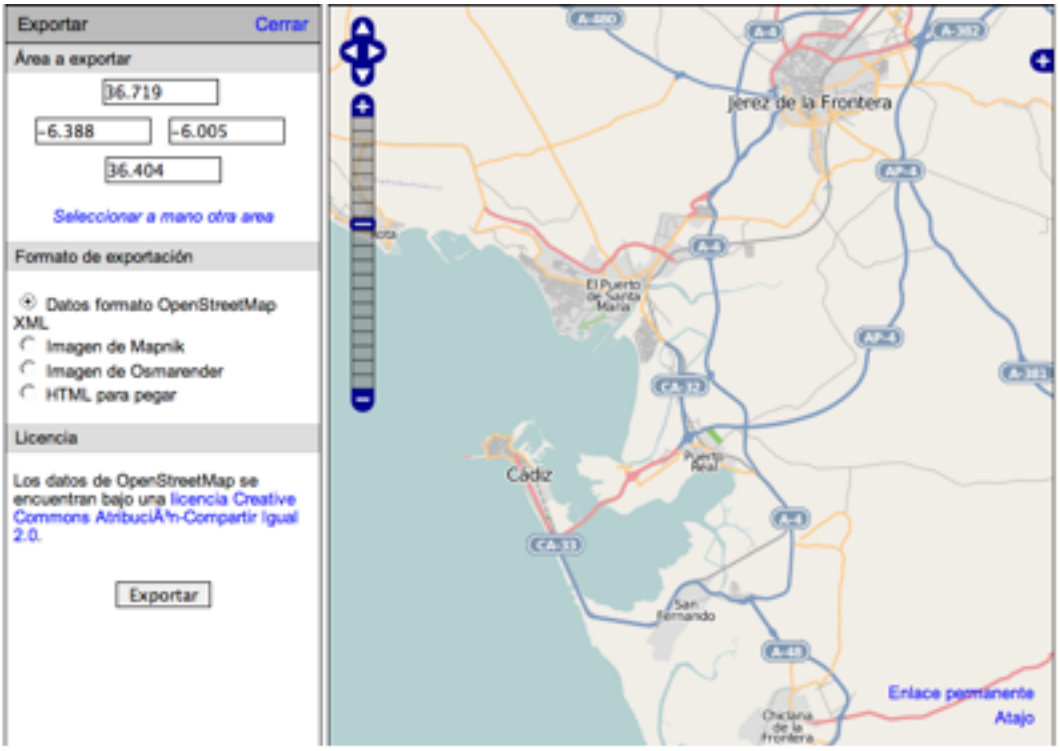

Figura 6.2: OSM, exportar mapa.

6.1.2. Selección y apertura del fichero OSM

Para saber que mapas están almacenados, la aplicación inspecciona la carpeta *mapas* y elabora una lista. Se identifican como mapas válidos aquellos archivos cuya extensión termine en 'xml' o 'osm'. La instrucción `os.listdir(dir)` devuelve una lista de nombres del contenido de la carpeta. El método *match* del módulo *re* de Python permite evaluar expresiones regulares. El patrón de búsqueda selecciona los nombres de lo archivos cuyos nombres comiencen por uno o más caracteres y terminen con ".osm" o ".xml".

```
1  import os, re
2
3  try:
4    for m in os.listdir('mapas'):
5      if re.match("^.+(\.(osm|xml))$", str(m)):
6        self.listaMapas.append(m)
7  except:
8    self.listaMapas.append("N/D")
9
10 if self.listaMapas.__len__() ==  0:
11   self.listaMapas.append("N/D")
```

6.1.3. Selección de un mapa mediante coordenadas

Si el usuario desea introducir las coordenadas del mapa lo puede hacer a través de dos cajas de entrada de texto. Las coordenadas deben tener un formato correcto, que se comprueba con las siguiente

expresión regular.

```
^(-?[0-9]+)(\.[0-9]+)?$
```

Las coordenadas deben ser numéricas, al menos un dígito. Si son de signo negativo, serán precedidas por el signo menos (-). En el caso de que contenga decimales, el símbolo decimal es el punto (.). Si las coordenadas no son correctas, se mostrará un mensaje de error al usuario. Si son correctas, se pone en funcionamiento el cliente HTTP.

6.1.4. Acceso al mapa a través de un cliente HTTP

OSM describe en su APIv0.6 cómo a través una petición HTTP, se puede obtener la información sobre una región concreta del mapa. En este caso se utiliza el cliente HTTP que proporciona Python, módulo *httplib* [12]. Primero se establece la conexión con el servidor, y luego se realiza la petición. Para seleccionar la región del mapa sobre la que queremos los datos se deben indicar los extremos del mapa en coordenadas geográficas. El código queda así:

```
import httplib

# establece la conexión
conn = httplib.HTTPConnection("www.openstreetmap.org")
# realiza la solicitud
conn.request("GET","/api/0.6/map?bbox=" + left + "," +
                                bottom + "," + right + "," + top)
# almacena la respuesta
r1 = conn.getresponse()
# variables de control
print r1.status, r1.reason  # si correcto, muestra: 200, OK
# almacena los datos solicitados
data1 = r1.read() # obtiene los datos solicitados
```

Las coordenadas introducidas por el usuario serán el centro del mapa que se solicite al servidor. El servidor recibe las variables *left*, *bottom*, *right* y *top*. Calculadas a partir de este centro aplicándole el incremento 0.02 para establecer las dimensiones del mapa.

6.2. Lectura del documento OSM

Los módulos necesarios son: *xml.sax.ContentHandler*, *xml.sax.make_parser* y *xml.sax.parseString*. Antes de utilizar el parser hemos de tener definido una clase *Manejador*, especialización de la clase *ContentHandler*. El parser recibe un objeto de esta clase, y le irá enviando la información a medida que avance leyendo el documento. Cada vez que el parser comience y termine de analizar un nodo invocará a los métodos *startElement* y *endElement* respectivamente. Ambos métodos reciben dos parámetros obligatoriamente, uno es el nombre del nodo y otro un diccionario que almacena los atributos del nodo. Es importante conocer la estructura del documento para poder analizarlo correctamente. [11]

```
# clase que gestiona los eventos producidos por SAX
```

```
class Manejador(ContentHandler):

  # constructor de la clase, puede recibir parámetros
  def __init__(self):
    print 'Manejador creado'
    # código...

  # se ejecuta la comienzo de cada elemento
  def startElement(self,nodo,atributos):
    if nodo == 'nombre_nodo':
      print 'Comienzo del nodo' + str(nodo)
      print 'Atributos: ', atributos.keys()
      print 'Atributo a1:', atributos.get('a1')

  def endElement(self,nodo,atributos):
    if nodo == 'nombre nodo':
      print 'Se ha procesado el nodo' + 'nombre_nodo'
```

Una vez definido la clase *Manejador* se puede utilizar el parser. En esta aplicación se utilizan dos analizadores diferentes, uno para los mapas almacenados en archivos y otro para el flujo de datos proporcionado por el cliente HTTP.

```
from xml.sax import ContentHandler
from xml.sax import make_parser

# crea un objeto manejador de eventos
manejador = Manejador()
# crea un objeto parser
parser = make_parser()
# asigna el manejador de eventos al parser
parser.setContentHandler(manejador)
# abre el fichero XML con los datos
fichero = open('./mapas/' + self.mapa, 'r')
# asigna al parser el fichero con la información
parser.parse(fichero)
```

Para el flujo de datos obtenido mediante el cliente HTTP es algo diferente.

```
from xml.sax import parseString

# stream_xml contiene el documento OSM
parseString(stream_xml, Manejador())
```

6.3. Almacenamiento de la información extraída del documento OSM

6.3.1. Guardando los límites del mapa

Para almacenar la información sobre los límites del mapa se utiliza la clase *Bounds*. El parser almacena las coordenadas geográficas en formato numérico decimal. También calcula la posición mínima y máxima del mapa. Tanto las coordenadas, como las posiciones, se almacenan en una tupla de tres componentes porque el motor gráfico trabaja sobre tres ejes de coordenadas.

```
# valores mínimos de las coordenadas
Bounds.setMinCoord(minlon, minlat, 0.0)

# posción mínima del mapa
Bounds.setMinPos(0.0, 0.0, 0.0)

# valores máximos de las coordenadas
Bounds.setMaxCoord(maxlon, maxlat, 0.0)

# posción máxima del mapa
Bounds.setMaxPos(Bounds.getMaxCoord() * factor_escala)
```

6.3.2. Guardando la información del nodo *node*

La información de cada nodo *node* del documento OSM se guarda en la clase *Nodo*. Cada objeto *Nodo* hace suyo el identificador único del nodo *node*. El parser realiza la conversión de las coordenadas de las que sólo se guardan las coordenadas cartesianas. Contiene un diccionario para guardar la información de los nodos *tag*.

```
# cálculo de la posición en el eje x
minlon = Bounds.getMinLon()
dif = longitud_nodo - minlon
x = dif * factor_escala

# cálculo de la posición en el eje y
minlat = Bounds.getMinLat()
dif = latitud_nodo - minlat
y = dif * factor_escala
```

Cada instancia de la clase nodo es incluida en uno de los diccionarios de la clase *Mapa*. La clave utilizada es el identificador único del nodo *node* del documento OSM. El valor es el propio objeto del tipo *Nodo*.

6.3.3. Guardando la información del nodo *way*

La información de cada nodo *way* se guarda en un objeto del tipo *Way*. Esta clase contiene un diccionario para almacenar el contenido de los nodos *tag* incluidos en *way*. La clase *Mapa* mantiene un diccionario donde para cada par clave/valor, se utiliza el identificador *way* y el objeto creado respectivamente.

6.3.4. Guardando la información del nodo *tag*

El nodo *tag* posee dos atributos: *k* y *v*. Tanto en la clase *Nodo* como en la clase *Way* se utiliza un diccionario donde se utiliza el atributo *k* como clave y el atributo *v* como valor.

6.3.5. Creación de la matriz de camino

Para establecer las conexiones entre los nodos, se necesita conocer qué elementos del mapa formarán parte del mapa de carreteras, cuáles son los nodos que contiene y las posiciones de estos. Durante el proceso de lectura del documento OSM se fue rellenando una lista que contenía los identificadores de cada elemento del mapa considerado una carretera.

El algoritmo para generar la matriz de camino recorre esta lista de identificadores de las carreteras para acceder al objeto de la clase *Way* correspondiente. Por cada objeto extrae la lista de los identificadores de los nodos que componen este elemento del mapa y enlaza cada nodo con el siguiente en la lista. Si la carretera es de doble sentido enlaza los nodos en sentido inverso también. Durante este repaso se realiza dos tareas adicionales: por cada nodo se indica si pertenece, al menos, a una carretera de doble sentido, y almacena el nombre del objeto *Way* visitado en cada uno los objetos *Nodo* que hace referencia. Estas dos operaciones son necesarias para hacer accesible esta información cuando se utilice la matriz de camino.

```
# crear el diccionario que contiene la matriz
self.MatrizCamino = {}

# recorre una lista con cada way considerada carretera
for way in carreteras:

  # lista de nodos que contienen way
  refs = w.getListaRefs()
  nrefs = refs.__len__()

  # determina si way es de doble sentido
  if w.getTagValue('oneway') == 'yes' or w.getTagValue('oneway') == '1':
    self.oneway = 'yes'
  else:
    self.oneway = 'no'

  # obtiene el nombre de way
  self.wayName = w.getTagValue('name')
  if self.wayName == None:
    if w.getTagValue('ref') != None:
      self.wayName = w.getTagValue('ref')

  # recorre la lista de nodos a los que hace referencia
  i=0
  while i < (nrefs - 1) :

    # registra el nodo como perteneciente a la matriz de camino
    if not self.MatrizCamino.has_key(refs[i]):
      # registra el nodo y crea su lista de adyacencia
      self.MatrizCamino[refs[i]] = []

    # enlaza el nodo i con i+1
    self.MatrizCamino[refs[i]].append(refs[i + 1])

    # si way es de doble sentido crea el enlace inverso
    if self.oneway == 'no':
      # registra el nodo como perteneciente a la matriz de camino
      if not self.MatrizCamino.has_key(refs[i + 1]):
```

```
          # registra el nodo y crea su lista de adyacencia
            self.MatrizCamino[refs[i + 1]] = []

        # enlaza el nodo i+1 con el nodo i
        self.MatrizCamino[refs[i + 1]].append(refs[i])

      # registra el nombre de way en el nodo
      if self.wayName != None:
        nodos[refs[i]].setWayName(self.wayName)

      # si no es un único sentido, modifica el atributo en el nodo
      # por defecto, todas las carreteras son de doble sentido
      if self.oneway == 'no':
        nodos[refs[i]].setOneWay(self.oneway)

      i = i + 1
    # end-while

    # comprueba si no se ha registrado el ultimo nodo de la lista
    if self.oneway == 'yes':
      if not self.MatrizCamino.has_key(refs[i]):
        self.MatrizCamino[refs[i]] = []
    # registra los dos atributos para el ultimo nodo de la lista
    if self.wayName != None:
      nodos[refs[i]].setWayName(self.wayName)
    if self.oneway == 'no':
      nodos[refs[i]].setOneWay(self.oneway)
  #end-for
```

6.4. Construcción del escenario

El escenario se crea cada vez que se accede a esta parte de la aplicación. No hay escenarios previamente guardados y son formados siempre en tiempo de ejecución. Se consigue, partiendo de las estructuras de datos que almacenan la información procesada del documento OSM.

6.4.1. Creación del cielo del escenario

El cielo del escenario está formado por un único modelo 3D en forma de cúpula. Está texturizada para que dé sensación de que se está viendo un cielo con sol y nubes.

Su tamaño está a escala para que la cúpula no quede fuera del alcance de la cámara, es renderizado de tal modo que siempre queda detrás de cualquier otro elemento de la escena. Realmente, la semiesfera no envuelve a todo el escenario, aunque así lo parezca porque el escenario es demasiado grande. Este efecto se consigue desplazando la cúpula persiguiendo a la cámara, ya que la cámara persigue al vehículo del usuario. En cada ejecución del bucle principal, se actualiza su posición.

6.4.2. Creación del suelo del escenario

El suelo se crea con un único modelo. Es un modelo de forma plana, rectangular y lo suficientemente grande, a escala, para que abarque toda el área visible por la cámara. En el bucle principal de ejecución

se actualiza su posición para que siempre esté situado, su centro, debajo de la cámara. La sensación que se consigue es que parezca que tiene un tamaño infinito.

6.4.3. Creación de la red de carreteras

Para representar las carreteras se utilizan dos modelos de formas básicas, un circulo y un cuadrado. Sólo se modifica la escala y la orientación (en el caso del cuadrado) para formar una sección de carretera.

La lista de nodos que define la carretera se recorre para colocar entre cada par de nodos un modelo en forma de cuadrado. La posición del cuadrado se calcula con el punto medio de las posiciones de los nodos. La orientación se calcula utilizando el algoritmo Pitágoras y tomando las posiciones de los nodos como extremos de la hipotenusa.

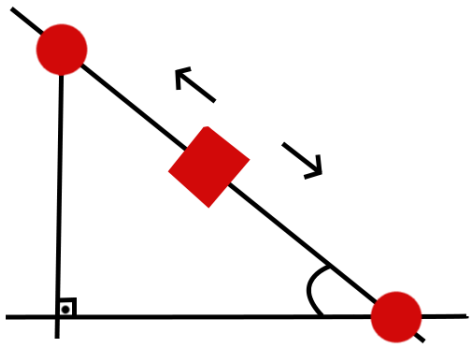

Figura 6.3: Formación del tramo de carretera

Por último, al recorrer la lista de nodos de la carretera, se coloca el elemento en forma de círculo en cada nodo para crear el efecto de unión y continuidad de la carretera.

```
def CrearCarreterea(self, way, parent, prioridad):

  # recorre la lista de nodos de la carretera
  # ref: todos menos el ultimo
  # ref+1: todos menos el primero
  for ref in range(way.getListaRefs()[:-1].__len__()):

    # obtiene las posiciones de los nodos
    pos1 = way.getListaRefs()[ref].getPos()
    pos2 = way.getListaRefs()[ref + 1].getPos()

    # carga el tramo de carretera
    modelo = loader.loadModel('modelos/carretera')

# se sitúa en el punto medio de las posiciones
    modelo.setPos((pos1 + pos2) / 2)

```

```
18      # orienta la carretera mediante el algoritmo de Pitágoras
19      self.Orientar(self.modelo, pos1, pos2)
20
21      # Establece el tamaño de la carretera, escala:(ancho, largo, alto)
22      escala = (pos1 - pos2).length() / 32 # el modelo original mide 32
            unidades
23      modelo.setScale(3 / 32, escala, 1)
24
25      # carga el nodo de unión entre tramos
26      union = loader.loadModel('modelos/circulo')
27      union.setPos(pos1)
28      union.setScale(Escalas * 3 / 32) # el modelo original mide 32 unidades
29
30    # end-for
31
32    # crea el nodo de unión para el último nodo de la lista
33    union = loader.loadModel("modelos/circulo")
34
35    # posición
36    union.setPos( pos2 )
37    union.setScale(Escalas * 3 / 32)
38
39  def Orientar(self, modelo, pos1, pos2)
40
41    # calcula la distancia entre nodos
42    vxy = pos2 - pos1
43
44    # si la distancia en el eje x es cero
45    # están alineados en la vertical
46    if(vxy.getX() == 0):
47      anguloDegrees = 90
48    # si la distancia en el eje y es cero
49    # están alineados en la horizontal
50    elif(vxy.getY() == 0):
51      anguloDegrees = 0
52  else: # cálculo del ángulo de orientación
53      angulo = abs( vxy.getY() / vxy.getX() )
54      anguloRad = atan(angulo)
55      anguloDegrees = degrees(anguloRad)
56
57    # traslado del ángulo al cuadrante correspondiente
58    if vxy.getX() <= 0 and vxy.getY() > 0: # segundo cuadrante
59      anguloDegrees = 180 - anguloDegrees
60    elif vxy.getX() <= 0 and vxy.getY() <= 0: # tercer cuadrante
61      anguloDegrees = anguloDegrees + 180
62    elif vxy.getX() > 0 and vxy.getY() <= 0: # cuarto cuadrante
63      anguloDegrees = anguloDegrees * (-1)
64
65    # establece la orientación del modelo
66    modelo.setH(anguloDegrees)
```

El esquema de algoritmo utilizado para crear las carreteras de uno y dos sentidos es el mismo. La diferencia radica en el tamaño de la carretera principalmente. La carreteras de doble sentido tiene los modelos a una escala mayor que los de un único sentido.

Para enlazar las carreteras de un único carril con las de dos carriles, los carriles de ambas carreteras deben coincidir. La carreteras de un único sentido deben entrar y salir de las de doble sentido por el carril derecho. En este caso el algoritmo de formación de la carretera lo detecta, y desplaza la posición de la carretera al lugar correspondiente.

```
# recibe dos tuplas con las posiciones de los nodos
def CalcularPosicionesCarril(self, pos1, pos2):

  # calcula la distancia entre nodos
  vxy = pos2 - pos1

  # si la distancia en el eje x es cero
  # están alineados en la vertical
  if(vxy.getX() == 0):
    anguloDegrees = 90
    #self.setHeading(90)
  # si la distancia en el eje y es cero
  # están alineados en la horizontal
  elif(vxy.getY() == 0):
    anguloDegrees = 0
    #self.setHeading(0)
  else: # cálculo del ángulo de orientación
    angulo = abs( vxy.getY() / vxy.getX() )
    anguloRad = atan(angulo)
    anguloDegrees = degrees(anguloRad)

  #traslado del ángulo al cuadrante correspondiente
  if vxy.getX() <= 0 and vxy.getY() > 0: # segundo cuadrante
    anguloDegrees = 180 - anguloDegrees
  elif vxy.getX() <= 0 and vxy.getY() <= 0: # tercer cuadrante
    anguloDegrees = anguloDegrees + 180
  elif vxy.getX() > 0 and vxy.getY() <= 0: # cuarto cuadrante
    anguloDegrees = anguloDegrees * (-1)

  # ángulos de orientación de la carretera
  alpha = anguloDegrees - 90
  alphaRadians = radians(alpha)
  # distancia del centro del carril
  # al centro de la carretera
  desplaz_carril = Escala * 1.2

  # desplazamiento en pos1
  pos1 = Point3(pos1[0] + cos(alphaRadians) * desplaz_carril,
                                pos1[1] + sin(alphaRadians) *
                                   desplaz_carril,
                                0)
  # desplazamiento en pos2
  pos2 = Point3(pos2[0] + cos(alphaRadians) * desplaz_carril,
                                pos2[1] + sin(alphaRadians) *
                                   desplaz_carril,
                                0)

  return (pos1, pos2)
```

6.4.4. Creación de hitos

Los hitos destacan lugares de interés señalados sobre el mapa. Se componen de dos elementos: un nombre y un icono; y no siempre aparecen los dos a la vez. Su posición puede estar definida explícitamente en el documento OSM o puede que necesite ser calculada. Para los casos en que la posición no es especificada se obtiene calculando el centroide del lugar en cuestión. El único caso en que la posición exacta de los elementos puede no aparecer es cuándo se traten de elementos definidos coma áreas.

```
def getCentroide(way):
  c = VBase3(0, 0, 0) # inicialización
  # recorre la lista de posiciones del elemento
  for nd in way.getListaRefs()[:-1]:
    c += nd.getPos()
    c = c / (way.getListaRefs().__len__() - 1)
  return c
```

El nombre del hito es el correspondiente a elemento que representa, especificado en el documento OSM. Se utilizan cuatro letreros solapados dos a dos en ángulo de noventa grados para que sea visible desde todas las direcciones. Para el icono se utiliza un modelo 3D genérico cúbico al que se le aplica la textura correspondiente en cada caso.

6.4.5. Creación de áreas

Muchos de los elementos que describe el mapa están representados en las dos dimensiones como un polígono. En muchos casos la única representación que se hace de este tipo de elementos es pintar esos polígonos sobre el suelo del escenario. Se utiliza un algoritmo de triangulación para representar los elementos. El algoritmo de triangulación parte de un conjunto de vértices y devuelve un conjunto de triángulos que en conjunto forman el polígono al que pertenecen los vértices.

6.4.6. Creación de edificios

Un edificio es una figura con volumen geométrico pero que en el mapa se sólo define el área o polígono que ocupa sobre el suelo. Mediante un proceso de triangulación se generan las caras del edificio por separado y se van situando una a una en su lugar. Se levanta una pared vertical entre cada par de nodos de describen el polígono de dos dimensiones. Al final, se genera el techo con la forma del polígono para cerrar la figura.

6.4.7. Gestión del escenario visible

La estrategia de creación del escenario permite mantener el rendimiento alto durante la ejecución. Pero también es importante que al llevar a cabo la idea el usuario no perciba este proceso.

El punto de referencia para saber que zona del mapa se va a crear es el la posición del vehículo del usuario. Si los elementos del mapa están lo suficientemente cerca se incluyen en una lista para procesarlos luego. Para que un elemento sea seleccionado basta que, al menos, uno de los nodos que lo define cumpla el requisito de cercanía. Los elementos se encuentran organizados por tipos en una serie de listas creadas durante la lectura del documento OSM, y se utiliza el mismo conjunto de listas pero con los

elementos seleccionados.

Determinada el área de mapa a representar en la escena el resto del mapa se obvia. Pero está porción de mapa es bastante más pequeña que el mapa completo. Cuando el usuario se sale del área representada se debe eliminar lo anterior y volver a representar el mapa. El control de la posición del usuario en la escena se realiza por eventos, mucho más eficiente que a través del 'main loop'. Aprovechando la detección de colisiones, una esfera de colisión. envuelve al escenario, y cuándo el usuario se salga de esta esfera significará que hay que volver a crear una nueva región del mapa. La esfera está centrada en la posición de referencia inicial y es algo más pequeña que la escena, para que desde la cámara no se vea el final.

6.5. Creación y gestión de vehículos

Todos los vehículos son instancias de la clase *Vehiculo* y son administrados por la clase *Circulacion*.Realiza las siguientes tareas: posicionamiento inicial, movimiento de los vehículos, gestión de las colisiones y reubicación de los vehículos.

Para representar los vehículos se utiliza el mismo modelo 3D al que se le aplican varias texturas. El vehículo del usuario es identificado por el color rojo.

6.5.1. Posiciones iniciales

La posición inicial está determinada por la elección del usuario antes de entrar al escenario y después de seleccionar el mapa. Tras seleccionar el mapa, la aplicación elabora una lista con todas las direcciones del mapa y la presenta al usuario para que elija. En el caso de que no hubiere direcciones disponibles, se seleccionará una dirección aleatoriamente.

El posicionamiento inicial de los vehículos tiene dos fases. La primera consiste en colocar el vehículo del jugador. La segunda fase consiste en colocar el resto de vehículos alrededor del vehículo del usuario.

El vehículo del usuario es fácil de ubicar. Si el usuario elige la posición de partida basta con colocarlo en su sitio. Si no hay posición de partida, se busca sobre la matriz de camino un nodo candidato. Un nodo es candidato si está conectado a, al menos, otro nodo a una distancia de tres saltos.

La colocación del resto de vehículos es más elaborada. Durante el proceso de selección de los elementos del mapa a colocar en la escena se realiza una segunda selección. Se seleccionan los nodos de la matriz de camino que están 'cerca' del usuario. Este conjunto de nodos estará siempre dentro de la región de mapa representada, y son las posiciones candidatas a posición de partida de los vehículos. Al igual que las listas de elementos, cuando el usuario se salga de la zona de mapa creada se volverá a elaborar esta lista de nodos. De inicio, se intenta colocar todos los vehículos (25 en total).

6.5.2. Reubicación de vehículos

En el escenario siempre habrá como máximo 25 vehículos controlados mediante IA. Para conseguir que siempre se muevan alrededor del vehículo del usuario se toma como referencia la posición de la cámara. Del conjunto de nodos de la matriz de camino que están dentro de la zona de mapa puesta en

la escena, se elige al azar un nodo que este a una distancia adecuada. Los nodos no estarán dentro del campo de visión de la cámara, pero a una distancia inferior a dos veces el horizonte visual de la cámara. Esta fórmula consigue que lo vehículos estén lo suficientemente cerca pero que no se vean aparecer como fantasmas en la escena.

Es posible que no se puedan colocar todos los vehículos en la escena porque no haya nodos disponibles suficientes. En tal caso, los vehículos quedan a la espera de ser reubicados más tarde. Se utiliza una lista para saber que vehículos se han quedado sin posición de partida o guardar los que se van quedando fuera posteriormente. El momento elegido para devolverlos al escenario se hace coincidir con la recolocación de los que ya estaban dentro. Por cada recolocación se intenta introducir un vehículo de la lista.

6.5.3. Guiado automático de vehículos

Todos los vehículos se controlan desde la clase *Circulacion*. Hay dos partes: determinación del rumbo, y actualización de la posición.

Los vehículos se mueven pasando por los nodos de la matriz de camino. Para controlar la ruta se emplean colisiones. Las colisiones generan eventos, y se evita tener que comprobar en un bucle si cada vehículo ha llegado o no a su destino. En la figura se puede observar un segmento situado en la posición de destino. Asociado al movimiento del vehículo hay una esfera que colisionará con el segmento en el momento que el vehículo llegue a su destino. Existe un par segmento-esfera, por cada vehículo en el escenario.

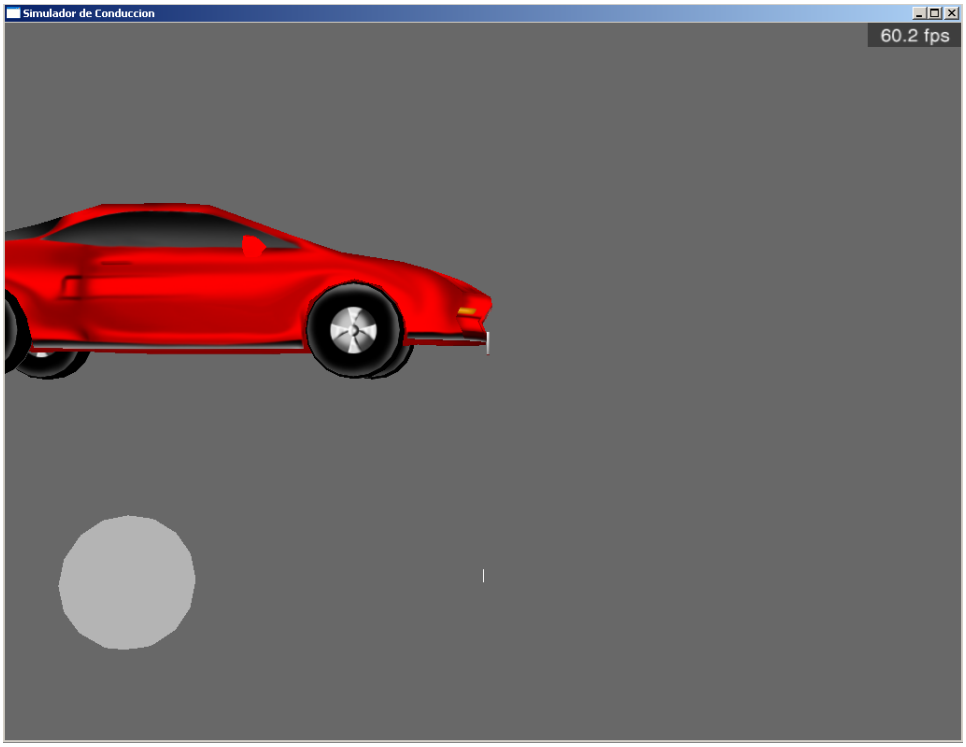

Figura 6.4: Vehículo, llegada a destino.

Cuando un vehículo llega a su destino la clase *Circulacion* recoge el evento y busca la siguiente posi-

ción de destino en la matriz de camino. Las posiciones de destino son un cálculo porque las posiciones de los nodos de la matriz de camino no coinciden con los carriles de las carreteras. Si la carretera es de doble carril la posición del nodo queda en el centro de la carretera. La posición correspondiente a cada carril se determina a partir del ángulo formado sobre el eje x la línea imaginaria que uniría a ambos nodos. A partir del ángulo se puede aplicar un desplazamiento a la posición para obtener el punto de partida real del vehículo.

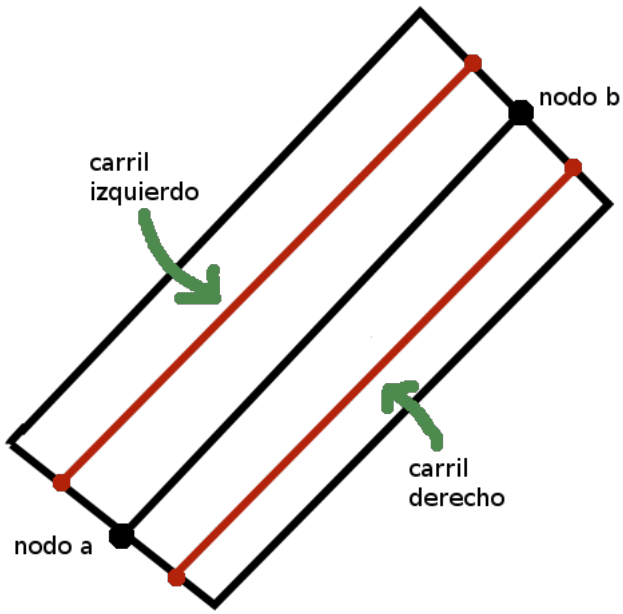

Figura 6.5: Cálculos de la posición del carril.

Cuando un vehículo llega al final del tramo de carretera entre dos nodos no se detiene al final del tramo. El punto de giro hacia el siguiente giro se realiza en un lugar determinado por el cruce de dos rectas. Las rectas son las que pasan por los puntos de inicio y final del tramo de carretera formado por los dos nodos atravesados y el tramo de carretera siguiente.

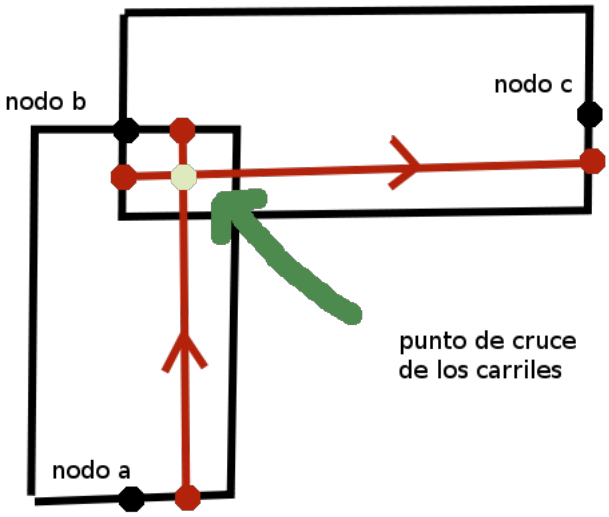

Figura 6.6: Cruce de carreteras.

6.5.4. Manejo del vehículo del usuario

Para que el usuario pueda controlar su vehículo se necesita una lógica más elaborada. El usuario puede controlar la velocidad y la ruta del vehículo.

La velocidad se modifica en valores discretos, cero, veinte, cincuenta, ochenta y ciento veinte kilómetros a la hora. Independientemente de la vía por la que circule.

Si el vehículo está es una vía de doble sentido se puede invertir el sentido de la marcha. El coche se desplazara al carril contrario automáticamente. Calculando por recorrer en el tramo de carretera, esta será la distancia recorrida en el carril contrario. Se utiliza el mismo sistema de colisiones empleado para saber cuándo el vehículo llega al final del tramo, pero el destino está ahora en el carril opuesto.

Al igual que el resto de vehículos, el vehículo del usuario se desplaza de tramo en tramo. En los casos en los que al pasar de un tramo sólo hay una opción el vehículo sigue su curso normal. Pero en los cruces donde hay más de una opción la clase *Flecha* se encarga de controlarlo. La clase flecha esta pensada para que el usuario indique cual es su elección y a la vez la muestre mediante un flecha la dirección a tomar. Partiendo primero de la posición inicial y luego desde cada cruce, la flecha explora la matriz de camino buscando los nodos que llevan a más de una opción. Si el nodo está enlazado con un sólo nodo se avanza al siguiente salto. La clase *Circulacion* mueve al vehículo por el escenario, pero si detecta que el vehículo ha llegado a un cruce con posibilidad de elección consulta al objeto de la clase *Flecha* para saber que eligió el usuario.

6.5.5. Detección de colisiones

Para evitar que los modelos de los vehículos se atraviesen entre si, se utiliza sistema de detección de colisiones que proporciona el motor gráfico. Cada vehículo está envuelto en una cápsula y delante de él se sitúa un segmento que hace de radar. Cuándo un segmento penetra en la burbuja, el motor gráfico lanza un aviso. La solución a la colisión consiste en detener al vehículo que colisiona, hasta que el vehículo colisionado se aleja.

6.5.6. Cuándo el usuario llega a 'el final' del mapa

La información que contiene el documento OSM es finita. El usuario puede llegar a 'el final' del mapa. Esto significa que no existan datos, en el documento OSM, para esa zona del mapa. En este punto se toman dos decisiones. Si el escenario se corresponde a un mapa almacenado en un fichero de texto se termina la partida. Si el mapa fue obtenido a través de Internet, se vuelve a solicitar al servidor de OpenStreetMap nuevos datos.

Para el caso del mapa seleccionado mediante coordenadas, el algoritmo que controla cuándo llega el usuario a un extremo del mapa, es independiente del algoritmo de gestión del escenario visible. El control se realiza a través de eventos. Aprovechando, de nuevo, el sistema de detección de colisiones, cuatro paredes confinan al mapa definido en el documento OSM. Cuándo el usuario atraviese una de estas paredes se solicitará al servidor el nuevo mapa. La paredes se colocar según la información del nodo *Bounds* del documento OSM. El proceso es similar al realizado cuando las coordenadas se introducen en el menú. La coordenadas de referencia serán las de la posición actual del vehículo del usuario. Realmente, las coordenadas se extraen del último nodo de la matriz de camino por el cual pasó el vehículo. La posición del vehículo, destino y otros datos son guardados. Los pasos son: guardar estado actual del vehículo, solicitar el nuevo mapa al servidor, eliminar escenario y vehículos, crear el nuevo escenario y vehículos, y por último, restaurar la configuración del vehículo. Para crear de nuevo el escenario se utiliza el mismo código utilizado para entrar a la escena desde el menú, pero pasándole los parámetros de configuración del vehículo.

6.6. Determinación del camino más corto

El usuario puede solicitar la ruta más corta desde su posición actual a un destino dado. La lista de direcciones disponibles es la misma lista ofrecida tras seleccionar el mapa y antes de acceder a la escena. El cálculo se realiza utilizando la matriz de camino y el algoritmo de Dijkstra [9]. El algoritmo de Dijkstra necesita una matriz de costes que se calcula a partir de la matriz de adyacencias. La matriz de costes se implementa también mediante listas, es un grafo disperso con muy bajo porcentaje de aristas respecto del número de vértices.

Para conseguir el camino más corto en un tiempo aceptable se utiliza la implementación del algoritmo de Dijkstra mediante colas de prioridad. Las colas de prioridad se implementan mediante montículos. El orden del algoritmo es de O(m+n Log n), para n vértices y m aristas. Es un algoritmo optimizado para el leguaje Python, y que se detiene cuando ya tiene la solución evitando calcular la ruta al resto de nodos pendientes.

6.7. Ir a un lugar seleccionado

Cuándo el usuario solicita ir a una dirección seleccionada el proceso es similar a si saliera del escenario y volviese a entrar a la nueva dirección. La lista de direcciones disponible es la misma lista ofrecida tras seleccionar el mapa y antes de acceder a la escena. Al solicitar una dirección, sin pasar por el menú ni reprocesar el documento OSM, se destruye el mapa en la escena y se crea uno nuevo, y por último, el vehículo es colocado en el lugar elegido.

6.8. Medición de tiempos

La clase *Records* de recibe, almacena y proporciona las mediciones de tiempo. La clase *Escenario* se encarga de presentar al usuario el menú para elegir un destino, realiza la medición del tiempo y envía/-solicita la información a la clase *Record*.

La lista de direcciones disponible es la misma lista ofrecida tras seleccionar el mapa y antes de acceder a la escena. Elegida la dirección de destino arranca el tiempo. Para saber que el usuario a llegado a su destino se utiliza el sistema de detección de colisiones. En la dirección de destino se coloca un elemento que al chocar con el vehículo produce un evento que detiene el cronómetro. Al terminar el cronometraje se solicita al objeto *Records* si hubiera alguna marca anterior para el mismo trayecto. Si hay un registro anterior pero mejor que el actual, se muestra al usuario pero no se guarda el registro nuevo. Si el registro anterior es peor, la nueva marca se guarda. Si no hay una marca anterior, se guarda la actual.

Los tiempos se guardan en un diccionario y éste a su vez es guardado en un archivo. Mediante la serialización, se guardan los datos de manera persistente. El módulo de Python *cPickle* lo hace posible.

```
try:
  import cPickle as pickle
except ImportError:
  import pickle

# Para cargar los datos

# abre el fichero
fileRecords = file('records.dat')
# carga los datos del fichero en la variable
records = pickle.load(self.fileRecords)
# cierra el fichero
fileRecords.close()

# Para guardar los datos

# abre el fichero en modo escritura
fileRecords = file('records.dat', 'w')
# guarda los datos, (variable, fichero, modo)
pickle.dump(records, self.fileRecords, 2)
# cierra el fichero
fileRecords.close()
```

6.9. Un pequeño juego

Por último se implementa un pequeño juego que consiste en encontrar un lugar aleatorio del mapa. En el lugar de destino se coloca un objeto ficticio con el que el vehículo colisiona y se detecta que ha encontrado la dirección. Sólo es posible acceder al juego si el mapa contiene los nombres de las calles y se cronometra el tiempo que el usuario emplea en encontrar la dirección. Como ayuda, bajo el vehículo se coloca una flecha que hace de brújula porque está orientada hacia el destino.

Capítulo 7

Pruebas

Las pruebas realizadas se pueden dividir en tres grupos. Hay tres aspectos básicos a probar: la información de los mapas se procesa correctamente, el escenario es dibujado acorde a la información leída y la gestión de los vehículos es correcta. Estos tres grupos dividen a la aplicación en tres partes diferenciadas que se prueban de manera diferente unas de otras.

Las pruebas se realizaron guiadas por las recomendaciones G. J. Myers [17]

7.1. Procesamiento de la información

La aplicación lee un documento XML y almacena la información en un conjunto objetos adaptados al uso posterior de esta información. Es importante que la información esté disponible, sea eficiente acceder a ella y sea correcta.

La información disponible es muy diversa y sólo se utiliza una parte de ella. Pero toda la información se almacena, con la idea de no tener que rediseñar en el futuro. En cualquier caso la información dada por OpenStreetMap es creciente y el sistema debe ser flexible a estos cambios.

Para saber si la información correcta, se han utilizado documentos XML relativamente cortos. Seleccionando nodos del documento XML que podían parecer problemáticos y mostrando por pantalla los datos almacenados. Es un trabajo de cotejación laborioso. Probando otras partes del software también se pueden descubrir este tipo de errores.

Para comprobar que la información se procesaba de manera eficiente se han utilizado documentos XML extensos. La comprobación consiste en observar la cantidad de memoria principal utilizada y midiendo si la espera es un tiempo aceptable para un posible usuario. Un tiempo aceptable pueden ser unos pocos segundos, es decir, no más de diez segundos en el peor de los casos. Cuanto menos tiempo esté el usuario esperando mejor impresión crea.

Cuando una operación es larga de realizar, el motor gráfico detecta que el reloj de la aplicación pierde sincronía respecto al reloj del bucle principal. Entonces muestra una advertencia del valor de reajuste del desfase en segundos.

7.2. Creación del escenario

Para comprobar que el escenario es creado y dibujado correctamente no hay más pruebas que las visuales. Es un trabajo de observación. La prueba consiste en crear el escenario y explorarlo visualmente. Cada elemento del mapa posee un identificador único en el documento XML, y es utilizado para hacer corresponder el elemento representado con su definición en el documento.

7.3. Gestión de los vehículos

Ahora las pruebas consisten en: probar el movimiento de los vehículos, probar que los vehículos circulan por donde se espera que lo hagan.

De nuevo las pruebas son principalmente visuales. Los vehículos deben seguir una matriz de camino. La matriz de camino es generada a partir de la información leída anteriormente. Se puede comprobar que la matriz de camino es correcta comprobando manualmente el documento XML con la matriz. Una prueba más exhaustiva es llenar el escenario de vehículos, sobrecargarlo, y observar movimientos extraños. La segunda prueba da mejores resultados porque la comprobación manual es muy costosa, y de esta manera los fallos saltan a la vista.

Capítulo 8

Conclusiones

8.1. Sobre el trabajo realizado

Este proyecto me ha permitido poner en práctica muchos de los conocimientos adquiridos durante estos años de estudio. Además de aprender cosas nuevas.

He podido trabajar con un motor gráfico. Hacia ya tiempo que tenía interés en desarrollar una aplicación de este tipo. Un tiempo atrás hice un juego sencillo con un motor gráfico 2D y desde entonces tenía la idea de pasar a las tres dimensiones y trabajar con una herramienta de mayor entidad.

El resultado respecto al esfuerzo no hace justicia. He tenido que trabajar duro para conseguir soluciones con cierto nivel de exigencia. Es fácil sobrecargar el motor gráfico, pero bastante difícil mantener el nivel de fps alto cuando la aplicación llega a cierto tamaño.

Me ha gustado programar utilizando Python. Tiene muy buena documentación online y la sintaxis es fácil de aprender.

Toda la documentación de Panda3D está en inglés. Así que he podido practicar un poco este idioma.

Respecto a las herramientas utilizadas, casi ninguna las había utilizado antes. No había utilizado Eclipse, no había utilizado SVN, no había utilizado Deled3D, no había utilizado SAX (es un módulo de Python), no había utilizado LaTeX y con GIMP había hecho lo justo.

8.2. Desarrollos futuros

Las características de este proyecto hace que esté totalmente abierto a nuevos desarrollo o modificaciones. Algunas de las nuevas características podrían ser:

- Representar el resto de elementos catalogados dentro de OSM. La lista de elementos del mapa es realmente larga, tiene un icono para casi todo lo que se pueda posicionar sobre un mapa: playas, parques, edificios, tiendas, señales de trafico, rutas ciclistas, etc.
- Incorporar la información topográfica del proyecto Shuttle Radar Topography Mission para representar las elevaciones del terreno. Hay otros proyectos asociados a OSM con resultados realmente impresionantes. Utilizar la información del proyecto SRTM casi constituye un proyecto por si mismo.

- Modificar el movimiento de los vehículos para darle más libertad. Los vehículos se mueven siguiendo la matriz de camino. Pero al vehículo del usuario se le podría dotar de más libertad de movimiento utilizando PyODE.
- Utilizar hilos de ejecución para ciertas operaciones y mejorar su eficiencia. Este modo de ejecución podría permitir acceder al servidor de OSM sin interrumpir la ejecución del bucle principal. O dar prioridad a un proceso encargado de manejar el vehículo del usuario.

Anexo A

Manual de usuario

A.1. Requisitos mínimos

Esta aplicación ha sido desarrollada en un equipo con las siguientes características:

- Procesador: Intel Core2Duo 1.8 MHz.
- Memoria principal: 2x DDR 667 1GB.
- Tarjeta Gráfica: Ati 1650x 128bits 256MB.
- Disco Duro: mínimo 200MB de espacio disponible.

Es imprescindible disponer de aceleración gráfica. No obstante, una manera de saber si la aplicación funcionará adecuadamente es utilizando un equipo que mantenga la tasa de FPS por encima de treinta.

La instalación incluye algunos mapas por defecto. Para obtener nuevos mapas o acceder al los mapas a partir de una posición geográfica es necesario disponer de acceso a Internet.

A.1.1. Limitaciones

Si el usuario descarga nuevos mapas desde la web de OpenStreetMap, debe tener en cuenta que el rendimiento puede verse afectado si el mapa alcanza cierto tamaño. En relación al tamaño del documento OSM su tamaño no debería superar los 5 MB.

Esta limitación es provocado por el aumento de la información que recibe la aplicación y el coste asociado a manejar esta información.

A.2. Instalación

Se pone a disposición en la web del proyecto [23] el instalador y código fuente del proyecto.

El instalador para MS Windows no tiene ninguna dependencia externa. Por defecto, se selecciona la unidad “C:” y se añade una nueva entrada al menú de inicio.

En el caso de utilizar el paquete deb, la instalación se realiza sobre la version 2.6 de Python. Si el intérprete no está en el sistema será instalado. Para arrancar la aplicación basta con escribir en la consola

“sim-conduccion".

En caso de que se desee ejecutar el código, no necesita compilación previa. Pero es necesario instalar, previamente, el software SDK de Panda3D [10] siguiendo las instrucciones.

A.3. Menú de entrada

La aplicación comienza mostrando un menú formado por dos pantallas. Desde ambas pantallas se tiene acceso al escenario. La transición entre estas pantallas se consigue pulsando la tecla *space*.

La primera pantalla muestra la lista de mapas ubicados en al carpeta */mapas* de la instalación. Con las teclas *arrow_up*/*arrow_down* se puede cambiar de una mapa a otro. En la esquina inferior derecha se muestra la numeración de los mapas. Para acceder solo hay que pulsar la tecla *enter*.

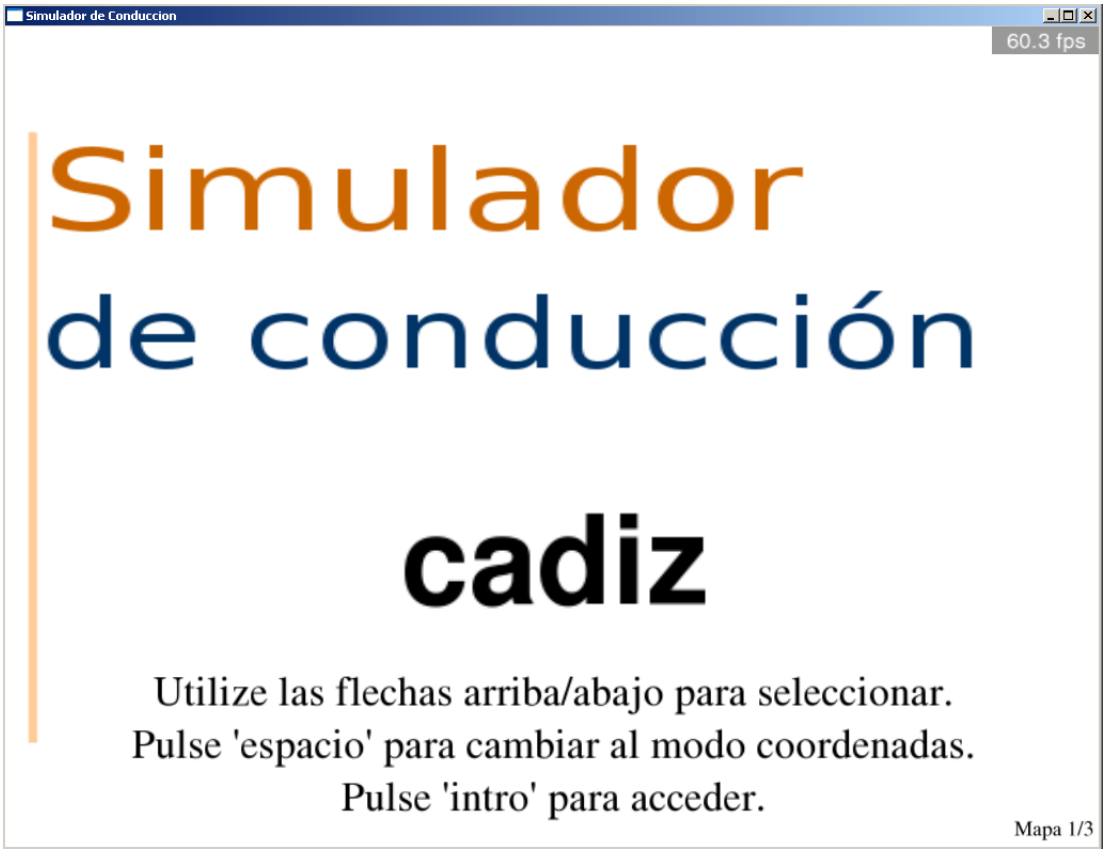

Figura A.1: Menú, selección del mapa.

En la segunda pantalla es dónde se puede introducir las coordenadas para descargar un mapa centrado en esa posición. Hay que escribir dos datos, latitud y longitud. El formato de entrada debe ser numérico, positivo o negativo, y el símbolo decimal es el punto. Si las coordenadas no tienen el formato correcto aparecerá un mensaje en la esquina inferior derecha de la pantalla. El desplazamiento del cursor entre las cajas de texto se hace con la tecla *tab*. Para acceder al escenario, igual que antes, hay que pulsar la tecla *enter*.

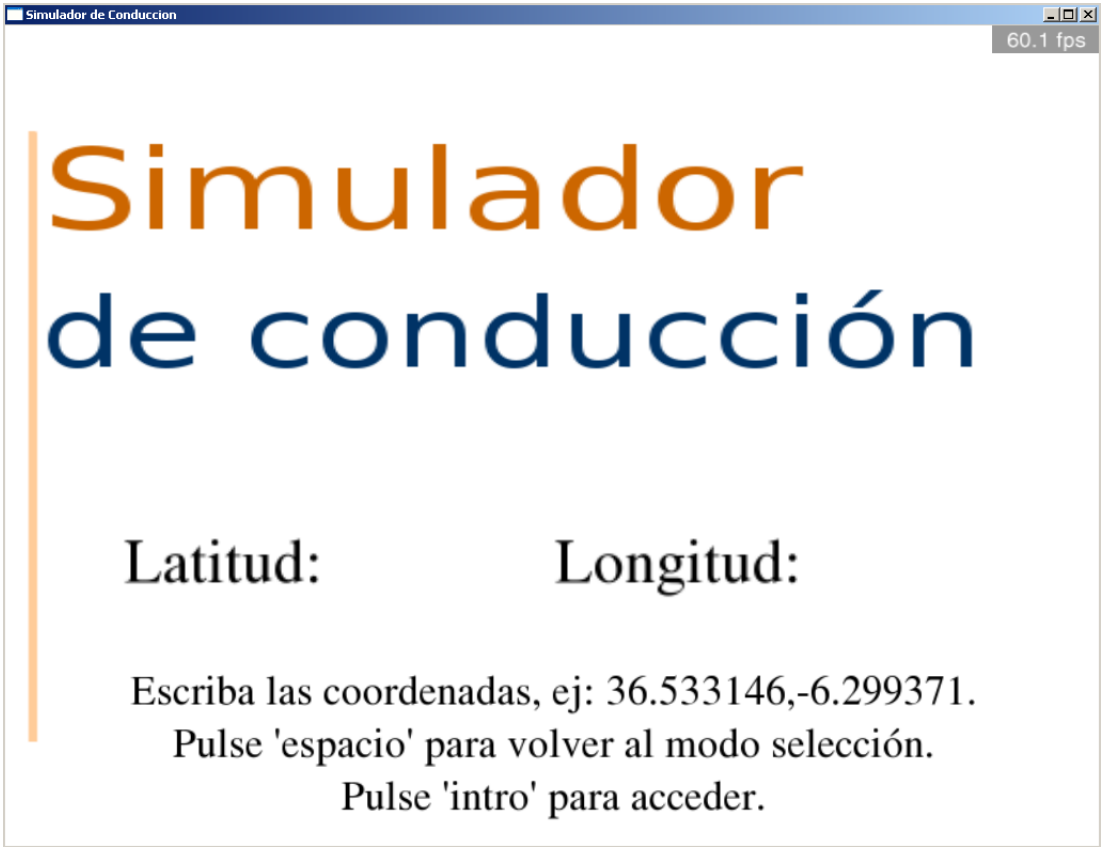

Figura A.2: Menú, introducción de coordenadas.

A.4. Escenario

Antes de entrar en el escenario hay que seleccionar la dirección de partida. Con las teclas *arrow_up*/*arrow_down* se puede seleccionar la dirección. Para realizar un filtro basta con escribir en la caja de texto y pulsar *shift-enter* para filtrar.

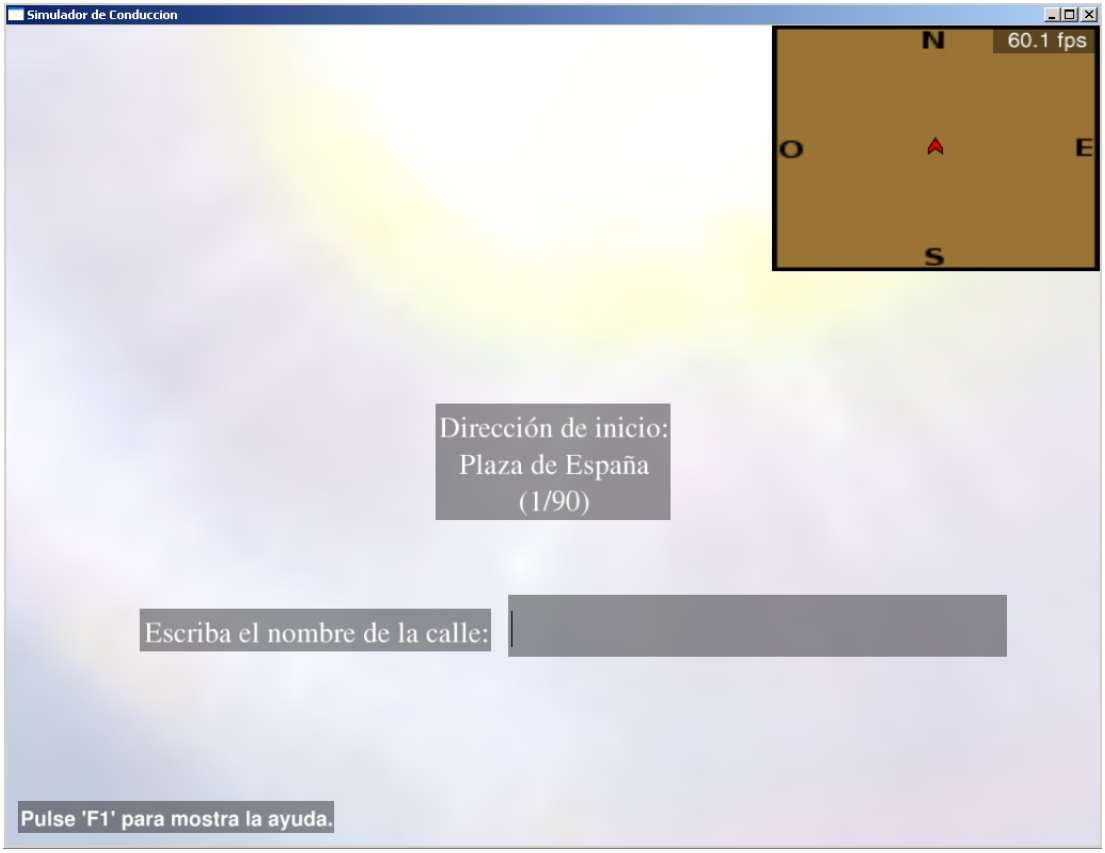

Figura A.3: Menú, selección de la posición de partida.

Los controles para manejar el vehículo son:

- Acelerar: *arrow_up*
- Decelerar/detener: *arrow_down*
- Invertir sentido de la marcha: *r*
- Cambiar dirección: *arrow_left/arrow_right*

Dentro del escenario hay cuatro posibles acciones. Buscar el camino más corto desde la posición actual hasta una dirección dada. Transportar el vehículo a una dirección dada. Realizar una contrarreloj desde la posición actual a una dirección dada y buscar una dirección al azar.

Para obtener el camino más corto a un lugar de destino, hay que pulsar la combinación de teclas *control-f*. Hay que seleccionar la dirección de destino y pulsar la tecla *enter*. Aparecerán las flechas sobre el suelo indicando el camino. La misma combinación de teclas para salir del menú de selección del destino.

Para ir directamente a un lugar, se consigue pulsando *control-g*. Basta con seleccionar la dirección. La misma combinación de tecla para salir del menú de selección del destino.

Para activar la contrarreloj se hace con las teclas *control-h*. Se mide el tiempo desde la posición actual hasta una dirección elegida por el usuario. Los mejores tiempos son guardados en un archivo. Pulsando de nuevo *control-h*, se puede cancelar.

Por último, con la combinación de teclas *control-j* se accede a un pequeño juego. Consiste en encontrar una dirección al azar del mapa. La misma combinación de teclas para cancelar.

Para volver al menú hay que pulsar *control-q*. En todo momento, para salir de la aplicación se hace pulsando *esc*.

A.5. Añadir nuevos mapas

Desde la web *www.openstreetmap.org*, en la parte superior de la página se encuentra la pestaña *Exportar*.

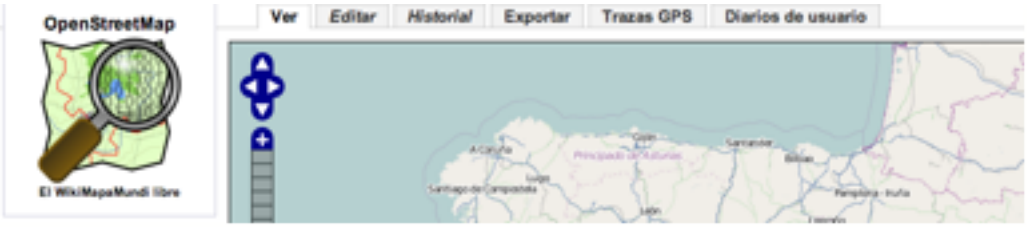

Figura A.4: OSM, página principal.

En la pantalla de exportación, se ven varias opciones. Hay que seleccionar *Datos formato OpenStreetMap XML*. Al pulsar el botón *Exportar* se genera un documento OSM, con la información sobre la región de mapa seleccionada. Este archivo debe ser incluido en la carpeta `/mapas` de la instalación. La región de mapa a exportar se puede elegir desplazando el mapa visible con el ratón, o mediante coordenadas en el apartado *Área a exportar*. En caso de que se desee indicar las coordenadas, éstas se corresponden los límites del mapa en latitud mínima y máxima, longitud mínima o máxima.

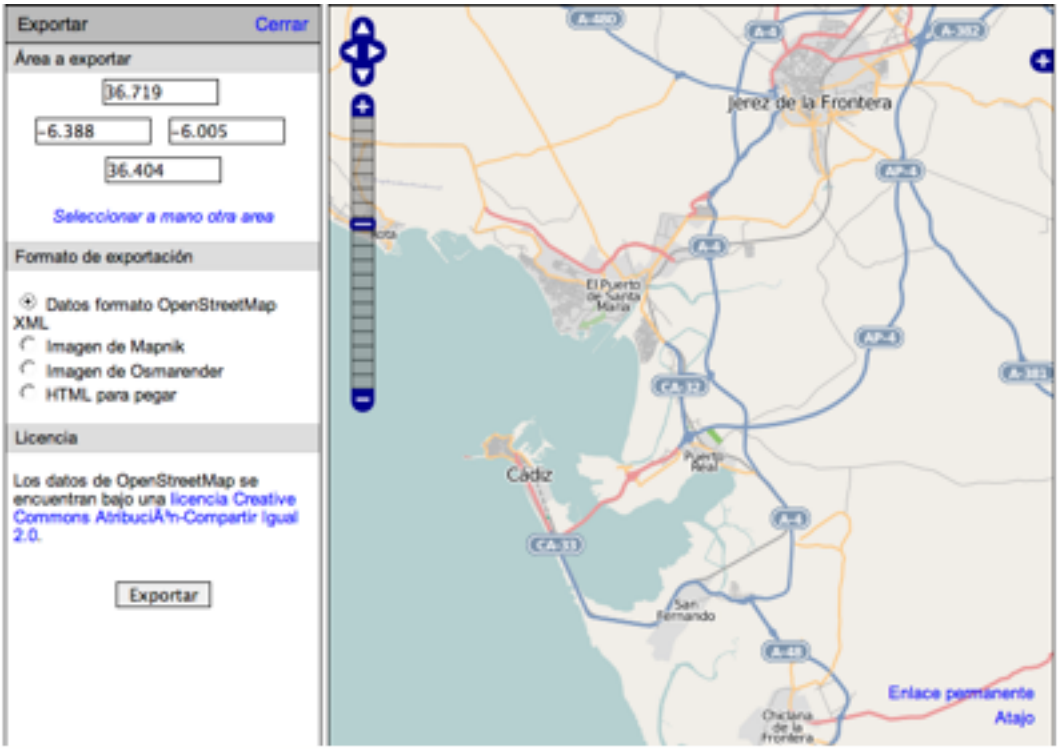

Figura A.5: OSM, exportar mapa.

Anexo B

Manual de Panda3D

B.1. Introducción

Éste es el manual del motor gráfico Panda3D. Este manual pretende ser una guía rápida que repase los aspectos más importantes del motor gráfico.

Para entender este manual, el lector debe conocer la programación orientada a objetos y la sintaxis de Python. En cualquier caso, no se pretende sustituir a la documentación disponible en la web oficial de Panda3D [10].

B.2. Qué es Panda3D

Panda 3D es un motor gráfico y como tal, proporciona un conjunto de funciones, clases y estructuras de datos para el desarrollo de videojuegos y renderizado en tres dimensiones.

Fue creado por la compañía Disney para el desarrollo del videojuego multijugador masivo online ToonTown. Se liberó bajo licencia Modified BSD en el año 2002. Actualmente está en desarrollo conjunto por Disney y Carnegie Mellon Entertaiment Technology Center. La licencia completa se incluye al final de este documento.

Algunas de sus características más destacables son: generador de shaders, incluye herramientas de monitorización, integración completa con Python, empaquetador para Windows, Linux y Mac; el único requisito es disponer de aceleración gráfica, incorpora las últimas características de OpenGL y DirectX, herramientas de debug, integra la biblioteca PyODE, etc.

B.3. Licencia

La licencia de Panda3D es 'Modified BSD License' [18], incluida al final de este documento y que debe ser leída con atención. Básicamente dice que cualquier distribución, código fuente o binarios, debe incluir la propia licencia acompañada de la cláusula de exención de responsabilidad de los autores, y ni el nombre de los autores u otros colaboradores, puede utilizarse con fines publicitarios sin el permiso previo por escrito (NO ES UNA ADVERTENCIA LEGAL).

B.3.1. A tener en cuenta

A continuación se describen una serie de recomendaciones, NO ES UNA ADVERTENCIA LEGAL, desde el punto de vista de la creación de un producto comercial utilizando Panda3D.

La licencia escogida debe cumplir estos términos:

- No puede prohibir la ingeniería inversa (LGPL).
- No se puede utilizar ninguna biblioteca con fines publicitarios (BSD/MIT).
- Se debe proporcionar el código fuente siempre que lo requieran los componentes GPL (LGPL).
- Debe incluir la licencia completa.

Su programa NO debe incluir:

- MP3
- MPEG
- Archivos 3D Studio Max, Maya, SoftImage, Milkshape 3D y GMax, al menos que se pague por el correspondiente SDK para convertirlo al formato egg.
- SSL si el software va a ser distribuido en países con leyes sobre criptografía (http://rechten.uvt.nl/koops/cryptolaw).

B.3.2. Desglose de licencias

Panda3D:

- Licencia BSD.
- A utilizar como el usuario guste.
- La licencia debe ser adjuntada al software, http://www.panda3D.org/license.php.

Python:

- Licencia PSF, http://www.python.org/download/releases/2.6.2/license/.
- Se puede utilizar libremente.

OpenAL:

- Licencia LGPL, debe enlazarse dinámicamente, http://www.gnu.org/licenses/lgpl-3.0.txt.

ZLib:

- Licencia ZLIP, http://www.zlib.net/zlib_license.html.
- Se puede utilizar libremente.

lippng:

- Licencia ZLIP, http://www.libpng.org/pub/png/src/libpng-LICENSE.txt.
- Se puede utilizar libremente.

Netpbm (todo lo relacionado con las imágenes en Panda3D)

- Licencias: Artistic License, GNU General Public License, MIT License; http://sourceforge.net/projects/netpbm/.
- Se puede utilizar libremente.

libjpeg:

- Licencia BSD-like, http://www.ijg.org/files/jpegsrc.v6b.tar.gz README.
- Debe incluir un archivo README.

libtiff:

- Licencia MIT-like, http://www.epsiia.com/licenses/libtiff.html.
- Se puede utilizar libremente, debe incluir una copia de la licencia.

FreeType (renderizado de texto):

- Licencias BSD y GPL, http://freetype.sourceforge.net/FTL.TXT.
- Debe incluir una copia de la licencia BSD.
- No se debe utilizar la licencia GPL.

Nvidia Cg Toolki:

- Licencia propietaria, http://developer.download.nvidia.com/cg/Cg_2.2/license.pdf.
- Se puede utilizar libremente.

FFMpeg:

- Licencia LGPL, debe ser enlazada dinámicamente, http://www.ffmpeg.org/legal.html, http://www.gnu.org/licenses/lgpl-3.0.txt.
- Se debe eliminar libpostproc (GPL) y libswscale (GPL).
- No se debe utilizar MPEG-4 (mp3, mpeg4, ...; por motivos de patentes).

Direct X:

- Licencia propietaria, http://www.microsoft.com/downloads/details.aspx?familyid=8b5cd64e-b4be-4135-95f8-ecfcf9182431&displaylang=en.
- Se puede utilizar libremente.

MFC (windows libs):

- Licencia propietaria.
- Se puede utilizar libremente.

ODE:

- Licencia BSD o GPL, http://www.ode.org/ode-license.html.
- No debe utilizarse GPL.

- Debe incluirse LICENSE-BSD.TXT.

squish:

- Licencia MIT-like.
- Se puede utilizar libremente.

openssl:

- Licencia MIT-like, http://www.openssl.org/source/license.html.
- No se debe incluir SSL para distribuciones en lugares con leyes sobre criptografía.
- Debe incluirse el siguiente agradecimiento: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)".

fcollada:

- Licencia MIT, http://www.opensource.org/licenses/mit-license.php.
- Se puede utilizar libremente.

libvrpn (Virtual Reality Peripheral Network):

- Es de dominio público, http://www.cs.unc.edu/Research/vrpn/obtaining_vrpn.html.
- Se puede utilizar libremente.

OpenCV (Web Cams capture):

- Licencia BSD, http://www.debian.org/misc/bsd.license.
- Debe incluirse una copia de la licencia.

B.3.3. Debe comprarse o no incluirse

FMOD:

- Licencia propietaria, http://www.fmod.org/index.php/sales.
- Se debe eliminar o comprar para distribuciones comerciales.

MP3:

- Patentado en EEUU, http://mp3licensing.com/royalty/software.html.

MPEG:

- Patentado en EEUU, http://www.mpegla.com/index1.cfm.

ARToolKit:

- Licencia GPL, http://www.hitl.washington.edu/artoolkit/license.html.
- Debe ser eliminado en distribuciones comerciales, a no ser que se obtenga bajo una licencia diferente.

FFTW (transformada discreta de Fourier):

- Licencia GPL, http://www.fftw.org/fftw3_doc/License-and-Copyright.html.
- Debe ser eliminado en distribuciones comerciales, a no ser que se obtenga bajo una licencia diferente.

B.4. Versiones disponibles, instalación, y otras descargas

Es un producto multiplataforma. Existen instaladores para las plataformas Windows, Linux y Mac; y también esta disponible el código fuente para la compilación manual. Junto con el código fuente se incluyen las instrucciones para la compilación y comprobación del éxito del proceso.

El motor gráfico está acompañado por un conjunto de aplicaciones third-party, necesarias para compilar Panda3D. También se incluyen una serie de ejemplos. Los ejemplos son los mismos referenciados en el manual. Por último, añaden una galería de modelos, creados por estudiantes, por si fueran de utilidad.

B.4.1. Instalación en Windows

El instalador para Windows es muy sencillo. Al ejecutarlo instala en el sistema todo lo necesario, estará disponible: el motor gráfico, el interprete de Python, las aplicaciones third-party y los ejemplos de los que se habla en el manual oficial. Para comprobar que la instalación se realizó correctamente es fácil ejecutando algún de los ejemplos. El interprete de Python que se debe utilizar es: `ppython`, situado de la carpeta `/python` de la instalación.

B.5. Documentación

Desde la web, Panda3D ofrece mucha documentación. Está disponible la API de referencia para clases, métodos y funciones; y está disponible un manual/tutorial. La API, aunque completa, no está totalmente documentada. La documentación aún está en proceso de elaboración, pero si está suficientemente completada para los elementos principales y más importantes. Respecto a el manual, más bien es un tutorial. Comienza desde cero y recorre con ejemplos las principales características. Pero no explica al cien por cien cómo utilizar el motor gráfico, y también está en proceso de elaboración. Se puede decir que la documentación, aunque no es completa, sí es suficiente. La API, el manual y los ejemplos están disponibles en la sección de descargas para consultarlos offline.

B.6. Editor y control de desarrollo

Panda3D se compone de código C++ y un conjunto de Python bindings. Para desarrollar una aplicación sólo es necesario escribir código C++ o Python utilizando la biblioteca. Se puede emplear cualquier editor de texto, y no hace falta un entorno de desarrollo específico. Pero siempre es más recomendable utilizar un editor especializado o que ofrezca algunas facilidades al desarrollador. Aquí se exponen algunas alternativas.

B.6.1. PyPE

PyPE [2] es un editor, bastante simple y ligero, multiplataforma y gratuito. Está publicado bajo la licencia GNU General Public License [14]. Es sencillo de utilizar e incluye características como el resaltado de sintaxis, se puede configurar el intérprete de Python y explorador de archivos entre otras.

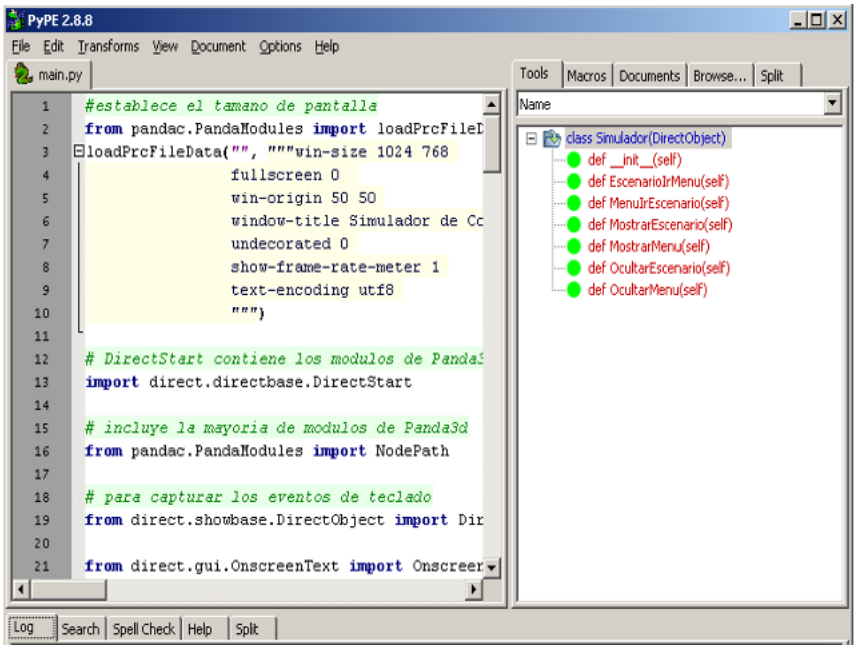

Figura B.1: Editor PyPe

B.6.2. Eclipse

Eclipse [6] es un IDE software libre multiplataforma para el desarrollo de aplicaciones. Está publicado bajo la licencia Eclipse Public License [1] y es gratuito. Se le puede incluir una serie de plugins que facilitarán el trabajo según las necesidades. Existen plugins para trabajar con diagramas UML o con repositorios SVN. En combinación con el plugin PyDev constituye un entorno de trabajo para aplicaciones escritas en Python.

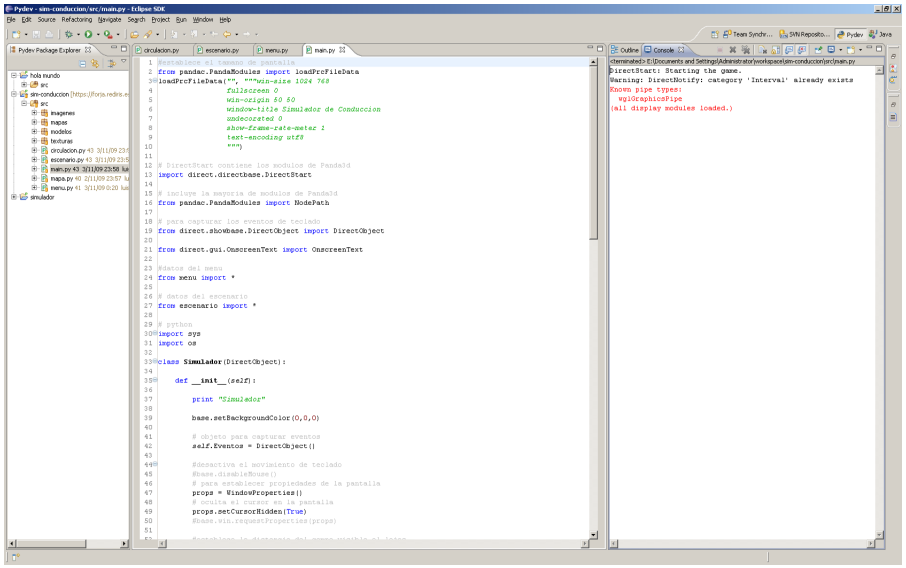

Figura B.2: Editor Eclipse.

PyDev [8] es un plugin que posibilita desarrollar aplicaciones utilizando Python, Jython y Iron Pyt-

hon; en Eclipse. Está publicado bajo licencia Eclipse Public License-v1.0 [1] y es gratuito. Incluye características como completado de código, resaltado de sintaxis, análisis de sintaxis, análisis de código, debug, consola interactiva, etc.

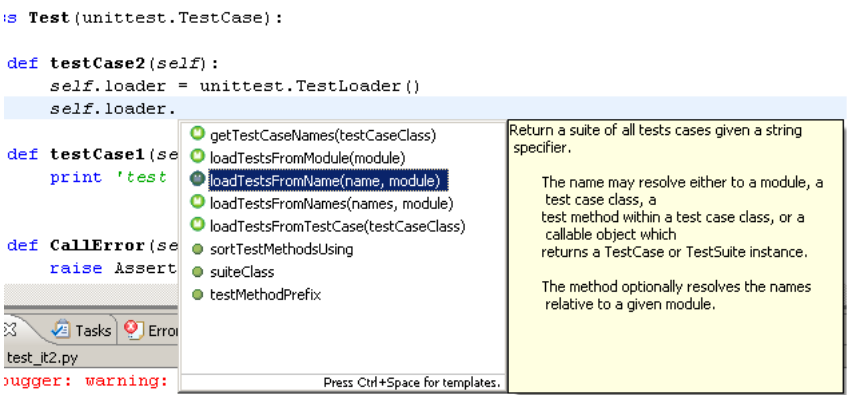

Figura B.3: Pydev, completado de código.

B.7. Editor 3D

Siempre que se crea un videojuego es imprescindible crea modelos que aparezcan en él. El formato nativo de Panda3D es el formato egg, legible y entendible tal cual. Trabajar con archivos egg puede ser lento, ya que está optimizado para la depuración, así que Panda3D tiene un segundo formato nativo, bam. El formato bam si está optimizado para la eficiencia. Para transformar modelos entre estos dos formatos se incluyen las herramientas `egg2bam` y `bam2egg` junto con la instalación. También existe la posibilidad de trabajar con el formato DirectX, y otra herramienta para transforma modelos con este formato al formato egg es `x2egg`.

B.7.1. Blender

Uno de los mejores editores es Blender [4]. Es un editor multiplataforma con licencia GPL [14] y gratuito. Constituye una herramienta muy potente cuyo único inconveniente quizá sea su interfaz algo complicada y que no se aprende a manejar en dos tardes. No obstante, existe mucha documentación disponible en la red y multitud de publicaciones. Tiene su propio plugin para exportar al formato egg o al formato DirectX.

B.7.2. Deled3D

Un editor sencillo y práctico es Deled3D [5]. Está disponible para la plataforma Windows, con una versión de pago, Deled Pro, y otra gratuita, Deled Lite. Ambas versiones se publican con la misma licencia, “license free", esto significa que todo lo que se haga, con propósito comercial o no, es propiedad de su creador sin restricciones legales ni condiciones. Desde su Web están disponibles varios plugins gratuitos, entres los cuales uno es para importar modelos en formato DirectX y otro para exportarlos. La versión de pago es bastante completa, pero con la gratuita es suficiente para crear modelos básicos, un usuario inexperto lo único que echará en falta es la herramienta para crear animaciones, sólo disponible en la versión Pro.

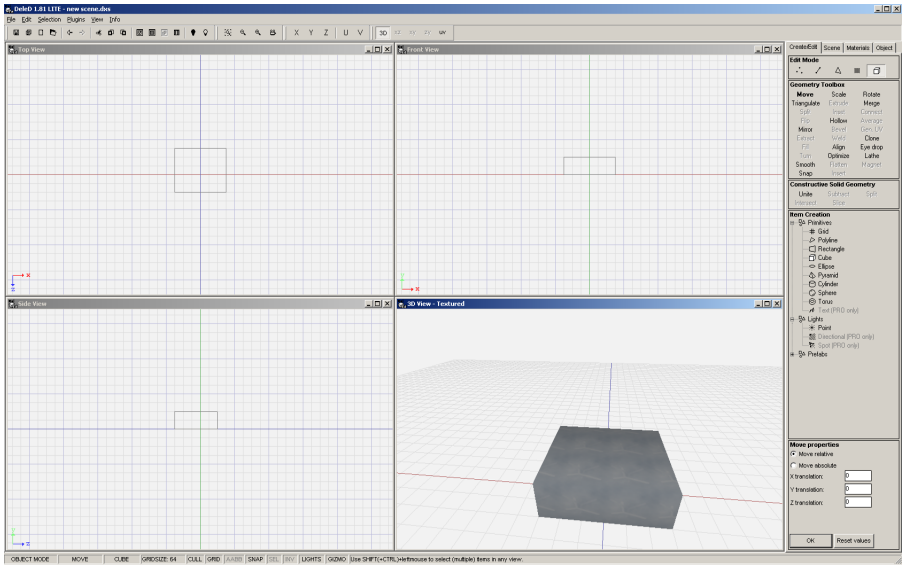

Figura B.4: Deled3D.

B.8. Editor de imágenes

Tan importante es el editor de modelos como el editor de imágenes. Las imágenes tienen mucha importancia en los videojuegos de dos dimensiones pero no dejan de ser igual de importante en los de tres dimensiones. Se utilizan para crear texturas de los modelos 3D, o para la parte de la aplicación que no necesariamente debe crearse en las tres dimensiones. Panda3D soporta varios formatos de imagen pero los recomendados son: PNG, TIFF y JPG.

B.8.1. GIMP

Una de la mejores elecciones para el editor de imágenes es GIMP [3]. Se publica con licencia GPL [14], es gratuito y multiplataforma. Posee varias herramientas de selección, herramientas de pintado, modificación de escala y dimensiones, herramientas de modificación de texto, filtros y manipulación de colores, manipulación de aspecto de imágenes, tijeras inteligentes, efectos de imagen, etc. Permite automatizar procesos mediante macros mediante el lenguaje Scheme entre otros y una de las características más importantes es su extenso catálogo de plugins creados por usuarios.

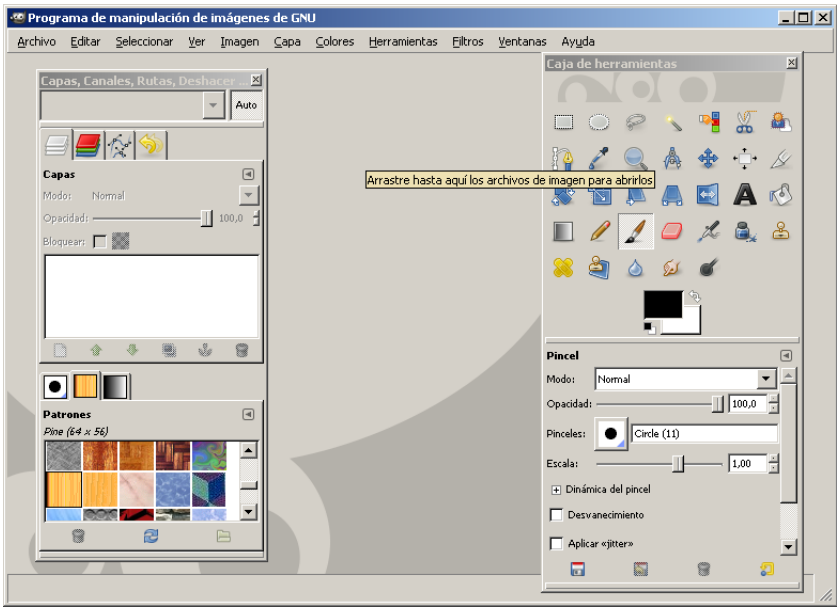

Figura B.5: GIMP.

B.9. Primer programa

Comenzar a utilizar Panda3D es muy sencillo. Con dos únicas instrucciones se obtiene la ventana de ejecución.

```
1 import direct.directbase.DirectStart
2
3 # código...
4
5 run()
```

La primera instrucción carga los principales módulos. La segunda es la que pone en marcha la aplicación, crea la ventana de la aplicación, comienza el bucle principal de ejecución, inicia el renderizado y administra todas la tareas de segundo plano. Son dos instrucciones imprescindibles y la primera siempre debe ser la primera línea del código, y la segunda la última. El resultado debe ser:

Figura B.6: Ventana de ejecución, Panda3D.

De aquí en adelante, las primeras instrucciones siempre serán para cargar el módulo necesario en cada ocasión. Para configurar la ventana de ejecución se utiliza el siguiente código:

```
from pandac.PandaModules import loadPrcFileData

loadPrcFileData("", """win-size 800 600
                fullscreen 0
                win-origin 50 50
                window-title Simulador de Conducción
                undecorated 0
                show-frame-rate-meter 1
                text-encoding utf8
                """)
```

Se establecen las siguientes propiedades las siguientes propiedades:

- win-size: establece el tamaño de la ventana, ancho alto.
- fullscreen: 1 para pantalla completa, 0 para modo ventana.
- win-origin: posición en pantalla de la ventana de ejecución, medida en píxeles, alto ancho.
- window-title: título de la ventana de ejecución.
- undecorated: establece el aspecto de la ventana de ejecución, valores: 1 ó 0.
- show-frame-rate: 1 muestra el contador fps, 0 lo oculta.
- text-encoding: establece la codificación utilizada.

B.10. Algunos conceptos importantes

Antes de continuar es necesario conocer algunos conceptos sobre el funcionamiento del motor gráfico. Para elaborar el gráfico de la escena, se utiliza una estructura de árbol que contiene los elementos a renderizar. Este árbol se compone de objetos de la clase *PandaNode*. La clase *PandaNode* es una superclase de una serie de subclases que conforman las funcionalidades del motor gráfico. El desarrollador controla la entrada y salida de los nodos del árbol, así como lo ancho o profundo que pueda llegar a

ser. Cada nodo guarda una relación con su nodo padre, existen una serie de atributos que cada nodo propaga a sus descendientes, a no ser que se indique explícitamente otro valor. Por ejemplo: si un nodo tiene aplicada una escala de 0.1, es aplicada esa misma escala a sus nodos hijos, o la posición de los descendientes se calcula a partir de la posición del nodo del que penden.

Panda3D crea, en realidad, dos árboles para pintar la escena. Uno es para la escena 3D y otro para la escena 2D. Las raíces de estos árboles son dos nodos especiales llamados *render* y *render2D* respectivamente. Cada estructura tiene su evidente propósito específico, elementos como botones, textos y similares van directamente al árbol destinado a las dos dimensiones; elementos como modelos, texto o imágenes que deben aparecer en la escena 3D, en definitiva, cualquier elemento que se desee ubicar desde una perspectiva de tres dimensiones, se debe colocar en el árbol cuya raíz es *render*.

En la escena 3D, las posiciones de sus elementos se define a partir de tres ejes cartesianos. El eje 'x' para la horizontalidad, el eje 'y' para la profundidad, y el eje 'z' para la altura. Una unidad sobre estos ejes se corresponde a la medida de un píxel. En caso de la escena 2D hay diferencias. El eje 'y' desaparece y el centro se sitúa en el propio centro de la ventana. La posición de los elementos se calcula de manera diferente. En un rango de valores [-1,1], por ejemplo el valor, [-1,-1] se corresponde a la esquina inferior izquierda de la pantalla. Esto es así porque su posición es relativa al tamaño de la ventana. Si las posiciones son relativas al tamaño de la ventana y la ventana no es cuadrada, se producen deformaciones. Para corregir este defecto existe un nodo especial, descendiente del nodo *render2D*, llamado *aspect2D*. Realmente, casi siempre se colocarán todos los elementos de la escena 2D como descendientes del nodo *aspect2D*.

La clase *PandaNode*, es una superclase para el resto de clases importantes en la estructura y funcionamiento del motor gráfico. Pero realmente el desarrollador va a trabajar con la clase *NodePath*. Esta clase es la unidad fundamental de interacción de alto nivel con el gráfico de la escena. Contiene una serie de métodos para manipular los "nodos" y la información que contienen. Un objeto *NodePath* es un puntero a un objeto *PandaNode* y cierta información adicional. Es un manejador de un puntero a un nodo. Algunas funciones trabajarán directamente con el objeto *NodePath* y otras directamente con el puntero. En cualquier caso el desarrollador debe trabajar y almacenar objetos *NodePath*, y obtener el valor del puntero mediante el método *node()*. Desde la clase *NodePath* es posible invocar a los métodos comunes de las subclases que heredan la clase *PandaNode*, y para los métodos especializados hay que trabajar con el nodo proporcionado por el método que se acaba de reseñar.

Para situar un nodo como descendiente de otro se hace de la siguiente manera:

```
miNodePath.reparentTo(render)
```

También es posible de esta otra forma:

```
# crea un nuevo NodePath
nuevoNodePath = render.attachNewNode('DummyNode')
# crea un nuevo NodePath a partir de un nodo de texto
nuevoNodePath = render.attachNewNode(TextNode)
```

Para descolgar un nodo del árbol existen dos métodos. El método *detachNode()* descuelga al nodo de su nodo padre. El método *removeNode()* desconecta al nodo del árbol y libera la memoria que tenía asignada eliminándolo por completo.

```
#desconecta el nodo del árbol
```

```
myNodePath.detachNode()
# elimina el nodo por completo
myNodePath.removeNode()
```

Si se desea recorrer el árbol a partir de un nodo dado, hay un método para acceder al nodo padre, y otro para obtener la lista de nodos hijos de un nodo.

```
parent = myNodePath.getParent()
children = myNodePath.getChildren()
```

Hay una serie de métodos para manipular los nodos a través de la escena que son los más comúnmente utilizados. Para establecer la posición del nodo sobre los ejes:

```
myNodePath.setPos(x,y,z)
myNodePath.setX(x)
myNodePath.setY(y)
MyNodePath.setZ(z)
```

Para establecer la orientación del nodo. En tres factores: header el giro a izquierda o derecha, pitch la inclinación hacia arriba o abajo, y roll para el giro sobre si mismo.

```
myNodePath.setHpr(header,pitch,roll)
myNodePath.setP(header)
myNodePath.setP(pitch)
MyNodePath.setR(roll)
```

En cambio, si lo que se quiere es orientar un nodo hacia otro concreto es fácil con el siguiente método. Es muy útil para que la cámara apunte hacia un nodo que se desea visualizar.

```
myNodePath.lookAt(node)
```

Existen otros dos métodos para mostrar u ocultar los nodos ante la cámara. Por defecto un nodo es visible.

```
# hace al nodo invisible
myNodePath.hide()
# hace al nodo visible
myNodePath.show()
```

B.11. Rutas y sintaxis

Antes de continuar es necesario comentar cómo es la sintaxis definida para Panda3D para las rutas de archivos. Para facilitar la portabilidad se emplea es estilo Unix, incluso en las plataformas que no son Unix. Con un ejemplo quedará claro.

```
# forma errónea
loader.loadModel("c:\\modelos\mimodelo.egg")
# forma correcta
loader.loadModel("/c/modelos/mimodelo.egg")
```

Trabajando en la plataforma Windows, es posible que sea necesario realizar conversiones entre los dos estilos de sintaxis. Algunas clases esperan como parámetro de alguno de sus métodos un objeto *Filename*. La clase *Filename* posibilita realizar conversiones entre el estilo de sintaxis utilizado por el motor gráfico y el estilo específico del sistema operativo en el que se esté trabajando.

```
from pandac.PandaModules import Filename

# partiendo de la sintaxis de Windows, por ejemplo
winfile = "c:\\micarpeta\\modelo.egg"
# se transforma en la sintásix de Panda3D
pandafile = Filename.fromOsSpecific(winfile)

# proceso inverso
# de la sintásix de Pand3D a la específica del OS
otherfle = pandafile.toOsSpecific()
```

B.12. Modelos y actores

Dentro de los elementos que se pueden añadir a la escena, los modelos y los actores son de los más importantes. Son objetos de tres dimensiones utilizados para representar escenarios y personajes. La diferencia entre un modelo y un actor está en su geometría o forma. Un modelo es una figura geométricamente estática, es decir, su forma es invariable. Un actor es un modelo cuya geometría puede variar. Según se desee cargar un modelo egg animado o inanimado, existe una clase específica. Una animación dirige el movimiento de un actor, puede estar dentro del archivo egg (puede causar confusiones) o en un archivo aparte.

B.12.1. La clase *Loader*, cargando modelos

La clase *loader*, entre otros elementos, permite cargar modelos desde archivo. Su método *loadmodel* recibe como parámetro la ruta del archivo y devuelve un objeto *NodePath*.

```
from direct.showbase.Loader import Loader
modelo = loader.loadModel('modelo.egg')
```

B.12.2. La clase *Actor*

Un actor se compone de un modelo no estático y un conjunto de animaciones. Se define para este propósito la clase Actor. Al cargar un actor se obtiene, de nuevo, un objeto *NodePath*.

```
from direct.actor.Actor import Actor

# carga un actor y sus animaciones
actor = Actor ('actor.egg',{'Nombre animación 1':'ruta animación 1',
                                        'Nombre animación 2':'ruta animaci
                                           ón 2'})
# elimina una animación cargada previamente
actor.unloadAnims({'Nombre animación 2':'ruta animación 2'})
```

B.13. Control de la cámara

Existe un nodo especial para poder visualizar los dos gráficos de la escena, el de tres y dos dimensiones. Este nodo es un objeto *NodePath* definido en *base.camera*. Por defecto, Panda3D implementa el control de la cámara a través del ratón. Se maneja de la siguiente manera:

- Botón izquierdo: desplaza a izquierda y derecha.
- Botón derecho: desplaza hacia delante y atrás.
- Botón central: rota el escenario alrededor del centro de los ejes de coordenadas
- Botón derecho y central: rota el punto de vista alrededor de los ejes vistos.

Casi con toda seguridad el control de la cámara incorporado por Panda3D entre en conflicto con el código escrito para el control de la cámara por el desarrollador. Para solucionarlo es posible deshabilitar el control por defecto de la cámara.

```
1 import direct.directbase.DirectStart
2 # desactiva el control por defecto de la cámara
3 base.disableMouse()
```

La cámara alberga un objeto *Lens* que establece las propiedades de visualización. Normalmente la configuración de la lente por defecto no necesite cambios, salvo un útil atributo que establece hasta donde alcanza es escenario visible.

```
1 import direct.directbase.DirectStart
2 # desactiva el control por defecto de la cámara
3 base.disableMouse()
4
5 # establece como limite de visibilidad respecto de
6 # la posición de la cámara, 1000 píxeles
7 base.camLens.setFar(1000)
```

B.14. Textos

B.14.1. Codificación de caracteres

Por defecto, Panda3D utiliza la codificación de caracteres *iso8859*, también llamada *latin-1* en Linux. Si fuera necesario cambiar a juego de caracteres es posible indicándolo en el archivos *Config.prc*. Por ejemplo, para cambiar a *utf-8*:

```
1 from pandac.PandaModules import loadPrcFileData
2
3 loadPrcFileData("", """text-encoding utf8""")
```

Elegida la codificación de los caracteres, todos las cadenas de texto deben emplear esta misma codificación y el código fuente también debe ser guardado de la misma manera.

B.14.2. La clase *TextNode*

La clase *TextNode* es la clase que permite crear y manipular textos. Utilizarla en muy sencillo.

```
from pandac.PandaModules import TextNode

# crea el objeto TextNode
texto = TextoNode('nombre')
# asigna el texto
texto.setText('aquí el texto a mostrar')
```

Ahora hay dos opciones, se puede colocar en las dos o las tres dimensiones.

```
# crea un Nodepath hijo de aspect2D
texto2D = aspec2D.attachNewNode(texto)

# crea un NodePath hijo de render
texto3D = render.attachNewNode(texto)
```

El nuevo objeto *NodePath* puede ser tratado como cualquier otro objeto de su clase, utilizándose métodos para definir su escala o posición.

Si Panda3D ha sido compilado con soporte para la biblioteca *FreeType*, se podrán carga archivos TTF. Mediante la clase *Loader* se puede acceder a esta posibilidad.

```
from direct.showbase.Loader import Loader

fuente = Loader.loadFont('arial.ttf')
# establece la calidad de la fuente
fuente.setPixelPerUnit(60)

# establece el tipo de fuente del texto
texto.setFont(fuente)
```

Algunos de los atributos propios de la clase *TextNode* son:

- Small caps: Especialmente útil para fuente que no soportan las minúsculas, las letras minúsculas se representan como letras mayúsculas pero de menor tamaño
- Slant: establece el grado de inclinación de los caracteres.
- Color: establece el color de la fuente.
- Shadow: estable la sombra de los caracteres.
- Wordwrap: por defecto, el texto es formateado en una sola línea, para acortar su longitud y utilizar varias líneas se emplea este atributo.
- Alignement: establece la justificación del texto, por defecto, a la derecha.

```
# small caps
texto.setsmallCaps(True)
texto.setSmallCapsScale(0.4)

```

```
# slant
texto.setSlant(0.3)

# color
texto.setColor(r,g,b,a)

# shadow
texto.setShadow(0.05,0.05)
texto.setShadowColor(r,g,b,a)

# wordwrap, medida en píxeles
texto.setWordWrap(15)

#justificación, [TextNode.ALeft,TextNode.ARight,TextNode.ACenter]
texto.setAlignement(TexNodeAcenter)
```

B.14.3. La clase *OnscreenText*

La clase *OnscreenText* es la forma más rápida para colocar textos en pantalla.

```
from direct.gui.OnscreenText import OnscreenText

texto = OnscreenText(text = 'texto a mostrar', pos = (x,z), scale = 0.3)
```

El constructor de la clase puede recibir estos parámetros:

- text: es el texto a mostrar, modificable mediante el método *setText()*.
- style: permite establecer un estilo configurando un conjunto de parámetros.
- pos: es la posición x, z del texto, es la escena 2D el valor del eje y es siempre cero.
- scale: establece el tamaño del texto, es un valor tipo float, se puede especificar mediante una tupla para la escala en cada eje.
- fg: color de primer plano del texto (r,g,b,a).
- bg: color de fondo del texto (r,g,b,a).
- shadow: color de sombreado del texto (r,g,b,a).
- frame: color de alrededor del texto (r,g,b,a).
- align: establece la justificación del texto [TextNode.ALeft,TextNode.ARight,TextNode.ACenter].
- wordwrap: establece la longitud máxima por línea del texto.
- font: establece la fuente del texto.
- parent: establece el nodo padre del nuevo *NodePath*, por defecto es *aspec2D*.
- mayChange: si es verdadero indica que el valor del texto puede variar en tiempo de ejecución, a falso crea un objeto estático optimizado para el uso de memoria.

La clase *OnscreenText* es heredera de la clase *NodePath*, por lo que todas las operaciones de esta clase están disponibles también.

Para cambiar el texto mostrado hay un método específico.

```
# establece el nuevo texto a mostrar
texto.setText('texto_nuevo')
```

Para eliminar el texto se debe hacer de la siguiente manera.

```
# texto es un objeto del tipo OnscreenText.
texto.destroy()
```

B.15. Imágenes

La forma más rápida de cargar una imagen es mediante el objeto *OnscreenImage*.

```
from direct.gui.OnscreeImage import OnscreenImage

imagen = OnscreeImage(image = 'imagen.png', pos = (x,0,z))
```

El constructor de la clase puede recibir estos parámetros:

- image: es la ruta de la imagen a mostrar.
- pos: es una tupla con la posición (x,y,z) de la imagen, en el caso de la escena 2D el valor de y debe ser 0.
- scale: es una tupla (x,y,z) indicando el tamaño de la imagen, en el caso de la escena 2D el valor de y debe ser 1.
- hpr: establece la orientación o giro, es una tupla (header,pitch,roll).
- color: establece el color de la geometría de la imagen, es una tupla (r,g,b,a).
- parent: establece el nodo padre del nuevo *NodePath*, por defecto es *aspect2D*.

La clase *OnscreenImage* es heredera de la clase *NodePath*, por lo que todas las operaciones de esta clase están disponibles también.

Para cambiar la imagen hay un método específico.

```
# establece la nueva imagen
imagen.setImage('nueva_imagen.png')
```

En el caso de que las imágenes contengan transparencias, es necesario especificarlo. Si no se indica nada y la imagen contiene transparencias, estas serán coloreadas en negro.

```
from pandac.PandaModules import TransparencyAttrib

imagen.setTransparency(TransparencyAttrib.MAlpha)
```

Para eliminar la imagen se debe hacer de la siguiente manera.

```
# imagen es un objeto del tipo OnscreenImage.
imagen.destroy()
```

B.16. Texturas

Aplicar una textura consiste en pintar un objeto con una imagen. Cuando se crea un modelo este puede incluir su propia información sobre las texturas que le son aplicadas, y Panda3D buscará estas texturas en un directorio relativo al archivo egg, o aplicadas en tiempo de ejecución.

B.16.1. Vértices uv

La forma en que las texturas son aplicadas está definida por la tupla uv. Cada vértice (u,v) relaciona un punto concreto de la geometría con un punto particular de la textura. Lo normal suele ser que estos vértices uv sean definidos al crear los modelos, hacer este trabajo manualmente puede ser bastante complicado. Pero en el caso de que sea necesario, en tiempo de ejecución, Panda3D posibilita crear los vértices uv siguiendo unos criterios.

B.16.2. Tamaños de la textura

Por motivos de eficiencia, las texturas deberían tener estos tamaños: 1, 2, 4, 8, 16, 32, 64, 128, 256, 1024, 2048. Texturas mayores pueden ser muy costosas de manipular. Las texturas no deben ser necesariamente cuadradas, pueden ser por ejemplo 64x32.

Si las texturas que se utilicen no tienen tamaños potencias de dos, es posible redimensionarlas. Realizando esta operación trabajar con las texturas será mucho más eficiente.

A la potencia de dos inmediatamente superior.

```
from pandac.PandaModules import loadPrcFileData

loadPrcFileData("", """textures-power-2 up
                """)
```

A la potencia de dos inmediatamente inferior.

```
from pandac.PandaModules import loadPrcFileData

loadPrcFileData("", """textures-power-2 down
                """)
```

Es posible que con ciertas configuraciones hardware esta configuración cause problemas. Se puede desactivar con la opción `texture-power-2 none`.

B.16.3. Modos de aplicación de texturas

Mediante los vértices uv se aplica la textura a una zona concreta de la geometría. Pero es posible que el resto del modelo también quede, o no, afectado por la textura. Se aplica en los dos vértices por separado.

```
from pandac.PandaModules import Texture

textura.setWrapU(modo)
textura.setWrapV(modo)
```

Modos:

- Texture.WMRepeat: la imagen es repetida hasta el infinito.
- Texture.WMClamp: los píxeles de los vértices son estirados hasta el infinito.
- Texture.WMBorderColor: el espacio se rellena con un color, *setBorderColor()*.
- Texture.WMMirror: la imagen se repite en espejo hasta el infinito.
- Texture.WMMirrorOnce: la imagen se repite en espejo una sola vez.

B.16.4. Aplicar texturas

Una textura puede ser aplicada en tiempo de ejecución. Tenga o no tenga ya el modelo una textura previa. Si ya tenía una textura previa se pueden aprovechar los vértices uv, o en cualquier caso redefinir los vértices.

```
from direct.showbase.Loader import Loader
from pandac.PandaModules import Texture

textura = loader.loadTexture('textura.png')
# el segundo parámetro le da prioridad a la textura
modelo.setTexture(textura,1)
```

Para redefinir los vértices uv existen varios modos predefinidos:

- TexGenAttrib.MWorldPosition: copia la posición (x,y,z) de los vértices a los vértices (u,v,w) de la textura.
- TexGenAttrib.MEyePosition: copia la posición (x,y,z) de la cámara a los vértices (u,v,w) de la textura.
- TexGenAttrib.MWorldNormal
- TexGenAttrib.MEyeNormal
- TexGenAttrib.MEyeSphereMap
- TexGenAttrib.MEyeCubeMap
- TexGenAttrib.MWorldCubeMap
- TexGenAttrib.MPointSprite
- TexGenAttrib.MLightVector

```
from direct.showbase.Loader import Loader
from pandac.PandaModules import Texture
from pandac.PandaModules import TextureStage

# carga una textura
textura = loader.loadTexture('texturas/'+ selección)
# establece el modo de aplicar la textura
textura.setWrapU(Texture.WMMirror)
textura.setWrapV(Texture.WMMirror)

# crea las propiedades de la textura
stage = TextureStage('ts')
# establece la coordinación de los vértices u,v
stage.setTexcoordName("coord")
# establece la textura(propiedades, textura, 1 indica prioridad sobre la
   textura anterior)
myNodePath.setTexture(stage, textura, 1)

# realiza la coordinación de vértices
myNodePath.setTexGen(stage, TexGenAttrib.MWorldPosition)
#establece la escala de la textura
myNodePath.setTexScale(stage,0.5,0.5)
```

B.17. Tareas (tasks)

Una tarea es un subrutina creada por el desarrollador que se ejecuta a cada *frame* de la aplicación. Panda3D genera automáticamente algunas tareas, y si se desea se pueden añadir otras más.

Todas las funciones o métodos que se quieran incluir en el bucle de ejecución se incluyen a través del administrador de tareas *taskMgr*. El método *add*, utilizado para añadir nuevas tareas, recibe tres parámetros: el primero es el nombre del método o función, el segundo es el nombre identificativo de la tarea, y el tercero es una lista de parámetros que puede recibir la función o método añadida.

La función que se añade al administrador de tareas debe recibir un primer parámetro *task* único y a continuación el resto de parámetros, con *appendTask=True* asegura que los parámetros son enviados.

```
from direct.showbase.DirectObject import DirectObject

# incluye una nueva tarea al administrador de tareas
taskMgr.add(Tarea,'nombre_tarea',[' de prueba'],appendTask=True)

def Tarea(task,texto):
        print 'Tarea' + texto
        return Task.done
```

Es posible comprobar cuanto tiempo ha estado en ejecución la tarea mediante el atributo *task.time*, o saber cuantas veces se ha ejecutado mediante *task.frame*. Para conocer el nombre de la tarea se hace a través del atributo *task.name*, y el identificador *task.id*.

Es importante el valor de retorno de la tarea. Cumple una función específica:

- Task.done: especifica que la tarea ha terminado y es eliminada del administrador de tareas.
- Task.cont: la tarea se repetirá en la próxima iteración.
- Task.again: la tarea se repetirá en la próxima iteración, con el retraso indicado inicialmente.
- Si no hay valor de retorno, al finalizar la tarea se elimina.

La tareas se pueden ejecutar con cierto retardo. Es especialmente útil para ejecutar código solo cada cierto tiempo. El valor de retardo es accesible mediante el atributo *task.delayTime*.

```
taskMgr.doMethodLather(delayTime, función, 'nombre')
```

Para eliminar una tarea del administrado hay dos opciones, eliminarlas todas, o indicando su nombre.

```
#elimina todas las tareas
taskMgr.removeAllTasks()

#elimina una tarea concreta
taskMgr.removeTask('nombre')
```

B.18. El manejador de eventos.

Un manejador de eventos es una subrutina que se ejecuta cuando ocurren cierto tipo de eventos en la aplicación. Un manejador de eventos de se define mediante un objetos de la clase *DirectObject*.

Mediante el método *accept* se puede definir la respuesta a cada evento. Recibe tres parámetros: el nombre del evento, la función o método de respuesta y la lista de parámetros enviada a la función a ejecutar. Si se quiere que solo se responda una única vez al evento se debe utilizar *acceptOnce*.

```
from direct.showbase import DirectObject

# define un objeto manejador de eventos
eventos = DirectObject()

# define una acción para este evento
eventos.accept("evento",funcion1,[texto])

def funcion1(texto)
        print texto
```

Para dejar de dar respuesta a un evento hay dos posibilidades, seleccionarlos uno a uno, o todos a la vez.

```
# elimina el evento pulsar la tecla 'a'
eventos.ignore("evento")

# elimina todos los eventos a la vez
eventos.ignoreAll()
```

Para crear y enviar eventos propios se hace con el siguiente código.

```
messenger.send('nombre_evento')
```

B.18.1. Soporte hardware

Por defecto, cada vez que una tecla de ratón o teclado es pulsada, Panda3D genera una serie de eventos asociados. El nombre de cada ejemplo está asociado al carácter que identifica la tecla, por ejemplo, tecla 'a' evento "a". Las teclas estás identificadas por las minúsculas o el símbolo sin la tecla shift.

Detectando los eventos se puede afinar más. Añadiendo *-repeat* para detectar la auto repetición, *-up* para detectar cuando se deja de pulsar, y sin añadidos para cuando la tecla es presionada.

En el caso del ratón el funcionamiento es el mismo. Los nombres de los eventos son: "mouse1", "mouse2" y "mouse3".

Estos son los nombres de otras teclas especiales: "escape", "f"+"1-12" (ej. "f", "f2", ..."f12"), "print_screen", "scroll_lock", "backspace", "insert", "home", "page_up", "num_lock", "tab", "delete", "end", "page_down", "caps_lock", "enter", "arrow_left", "arrow_up", "arrow_dow", "arrow_right", "shift", "lshift", "rshift", "control", "alt", "lcontrol", "lalt", "space", "ralt", "rcontrol".

B.19. Otras características

Panda3D es un motor gráfico pensado y hecho para que todo lo que se pueda necesitar este integrado en él. Así es que es mucho más completo de lo que aquí se expone. Si la documentación no aporta demasiado o no resulta clara, en el foro hay mucha información adicional.

Estas son algunas de las características que quién utilice Panda3D debe conocer y no han sido previamente descritas:

- Luces, partículas, niebla y otros efectos.
- Colisiones.
- Triangulación.
- Sonido.
- Shaders.
- Botones, cajas de texto y similares.

- Máquina de estado finito.
- PyODE integrado.
- Clases especificas para operaciones matemáticas (*LVector3F*, *LPoint3F*, *LMatrix3F*, etc).

Anexo C

Definición de tipo correspondiente al documento OSM

Éste es el documento DTD que describe la estructura y sintaxis del documento OSM para la APIv0.6:

```
<!ELEMENT osm (user|preferences|gpx_file|api|changeset|(node|way|relation)+)>
<!ATTLIST osm version            CDATA #Fixed "0.6">
<!ATTLIST osm generator          CDATA #IMPLIED>

<!--response to request message api/0.6/user/details -->
<!ELEMENT user (home?)>
<!ATTLIST user display_name      CDATA #REQUIRED>
<!ATTLIST user account_created   CDATA #REQUIRED>

<!ELEMENT home EMPTY>
<!ATTLIST home lat               CDATA #REQUIRED>
<!ATTLIST home lon               CDATA #REQUIRED>
<!ATTLIST home zoom              CDATA #REQUIRED>

<!--response to request message api/0.6/user/preferences -->
<!ELEMENT preferences (tag*)>

<!--response to request message api/0.6/user/gpx -->
<!ELEMENT gpx_file EMPTY>
<!ATTLIST gpx_file id            CDATA #REQUIRED>
<!ATTLIST gpx_file name          CDATA #REQUIRED>
<!ATTLIST gpx_file lat           CDATA #REQUIRED>
<!ATTLIST gpx_file lon           CDATA #REQUIRED>
<!ATTLIST gpx_file user          CDATA #REQUIRED>
<!ATTLIST gpx_file public        (true|false) "false">
<!ATTLIST gpx_file pending       (true|false) "false">
<!ATTLIST gpx_file timestamp     CDATA #REQUIRED>

<!--response to request message api/capabilities -->
<!ELEMENT api (version, area, tracepoints, waynodes)>

<!ELEMENT version EMPTY>
```

```
<!ATTLIST version minimum        CDATA #REQUIRED>
<!ATTLIST version maximum        CDATA #REQUIRED>

<!ELEMENT area EMPTY>
<!ATTLIST area maximum           CDATA #REQUIRED>

<!ELEMENT tracepoints EMPTY>
<!ATTLIST tracepoints per_page   CDATA #REQUIRED>

<!ELEMENT waynodes EMPTY>
<!ATTLIST waynodes maximum       CDATA #REQUIRED>

<!--response to request message api/0.6/changeset/*tbd* -->
<!ELEMENT changeset (tag*)>

<!--response to various request messages api/0.6/(create|delete|update)/ *tbd*
<!ELEMENT node (tag*)>
<!ATTLIST node id                CDATA #REQUIRED>
<!ATTLIST node lat               CDATA #REQUIRED>
<!ATTLIST node lon               CDATA #REQUIRED>
<!ATTLIST node changeset         CDATA #IMPLIED>
<!ATTLIST node visible           (true|false) #REQUIRED>
<!ATTLIST node user              CDATA #IMPLIED>
<!ATTLIST node timestamp         CDATA #IMPLIED>

<!ELEMENT way (tag*,nd,tag*,nd,(tag|nd)*)>
<!ATTLIST way id                 CDATA #REQUIRED>
<!ATTLIST way changeset          CDATA #IMPLIED>
<!ATTLIST way visible            (true|false) #REQUIRED>
<!ATTLIST way user               CDATA #IMPLIED>
<!ATTLIST way timestamp          CDATA #IMPLIED>

<!ELEMENT nd EMPTY>
<!ATTLIST nd ref                 CDATA #REQUIRED>

<!ELEMENT relation ((tag|member)*)>
<!ATTLIST relation id            CDATA #REQUIRED>
<!ATTLIST relation changeset     CDATA #IMPLIED>
<!ATTLIST relation visible       CDATA #IMPLIED>
<!ATTLIST relation user          CDATA #IMPLIED>
<!ATTLIST relation timestamp     CDATA #IMPLIED>

<!ELEMENT member EMPTY>
<!ATTLIST member type            (way|node|relation) #REQUIRED>
<!ATTLIST member ref             CDATA  #REQUIRED>
<!ATTLIST member role            CDATA  #IMPLIED>

<!ELEMENT tag EMPTY>
<!ATTLIST tag  k                 CDATA #REQUIRED>
```

```
<!ATTLIST tag  v                CDATA #REQUIRED>
```

Bibliografía

[1] Eclipse Public License-v1.0. http://www.eclipse.org/legal/epl-v10.html.

[2] Editor PyPe. http://pype.sourceforge.net/index.shtml.

[3] GIMP, GNU Image Manipulation Program. http://www.gimp.org.

[4] Página oficial de Blender. http://www.blender.org.

[5] Página oficial de Delgine. http://www.delgine.com.

[6] Página oficial de Eclipse. http://www.eclipse.org.

[7] Página oficial de OpenStreetMap. http://www.openstreetmap.org.

[8] Plugin PyDev Aptana. http://pydev.org/index.html.

[9] Jens Vygen Bernhard H. Korte. *Combinatorial optimization: theroy and algorithms, cuarta edición*. ISBN 3540718435, 9783540718437. 2008.

[10] Carnegie Mellon. Página oficial de Panda3D. http://www.panda3d.org.

[11] Daniel Gayo Avello. Lenguajes de consulta para XML, programación XML.

[12] Documentación Python v2.6.4. Módulo httplib. http://docs.python.org/library/httplib.html.

[13] Documentación Python v2.6.4. Módulo xml.sax. http://docs.python.org/library/xml.sax.html.

[14] Inc. Free Software Fundation. Gnu general public license. http://www.gnu.org/copyleft/gpl.html.

[15] Ivar Jacobson Grady Booch, James Rumbaugh. *El lenguaje unificado de modelado: guía del usuario*. ISBN 8478290761, 9788478290765. 2006.

[16] James Rumbaugh Grady Booch, Ivar Jacobson. *El lenguaje unificado del modelado: guía del usuario*. ISBN 8478290370, 9788478290376. 2000.

[17] Glenford J. Myers. *The art of software testing*. ISBN: 0-471-46912-2. Hoboken: John Wiley & Sons, segunda edición, 2004.

[18] Open Source Initiative. Modified BSD License. http://www.opensource.org/licenses/bsd-license.php.

[19] OpenStreetMap. Características del mapa osm. http://wiki.openstreetmap.org/wiki/Map_Features.

[20] OpenStreetMap. Elementos del documento osm. http://wiki.openstreetmap.org/wiki/Elements.

[21] OpenStreetMap. Wiki y manual. http://wiki.openstreetmap.org/wiki/ES:Main_Page.

[22] OpenStreetMap. APIv0.6. http://wiki.openstreetmap.org/wiki/API_v0.6, Abril 2009.

[23] Luis Salvador Roa Rodríguez. Web del proyecto sim-conduccion, 2010.

Modified BSD License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of Carnegie Mellon University nor the names of other contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.
This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.
We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.
A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.
A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.
The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.
A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".
Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.
The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.
The "**publisher**" means any person or entity that distributes copies of the Document to the public.
A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.
The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.
You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.
If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.
If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.
It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.
You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.
You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.
The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.
In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.
You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.
If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.
If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.
However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.
Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.
Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.
"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.
"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.
An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.
The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.
If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.