universidad del país vasco — euskal herriko unibertsitatea

Facultad de Informática — **Informatika Fakultatea**

TITULACIÓN:

Ingeniería Informática

Simulación de incendios forestales y urbanos

Alumno: Ander Arbelaiz Aranzasti

Director: Alejandro García-Alonso Montoya

Proyecto Fin de Carrera, 17 de Junio de 2013

Agradecimientos

En primer lugar quiero agradecer a mis padres Jose Manuel Arbelaiz y Maria Asunción Aranzasti el apoyo que me han dado a lo largo de toda la carrera.

También quiero agradecer a Aitor Moreno la oportunidad de realizar este Proyecto Fin de Carrera en Vicomtech-IK4 y a Alex García-Alonso por su orientación, sus consejos y la dirección de está memoria.

Finalmente, no me quiero olvidar de aquellos familiares, compañeros y amigos que también me han apoyado con sus mensajes de ánimo.

RESUMEN

La única forma que tienen los profesionales de aprender los procedimientos de extinción de incendios y gestión de los recursos es a través de maquetas o simulacros controlados.

Este proyecto hace énfasis en la verificación de un conjunto de algoritmos de simulación y propagación de incendios facilitando la validación de los mismos por expertos.

Los algoritmos validados se integrarán en un simulador interactivo dirigido al entrenamiento e instrucción del uso de los recursos para la extinción de incendios.

ABSTRACT

The project is divided in two stages. The first one focuses on the verification of a set of fire spread algorithms, in order to be validated by experts.

The second one integrates the previous validated algorithms in a training tool for the resource management on the fire extinguishment process.

LABURPENA

Proiektu hau bi fasetan zatitzen da, batetik suaren propagazioa eta portaera simulatzen dituen algoritmoen egiaztapenean, arloan adituak direnak hauek baliozkotu ahal izateko. Bestetik, sua itzaltzeko baliabideen erabilpenean zuzendua dagoen simulatzaile baten garapenean, baliozkotu diren algoritmoak erabiliz.

ÍNDICE

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Este Proyecto Fin de Carrera se enmarca dentro de un Proyecto más amplio abordado por el Centro Tecnológico Vicomtech-IK4. Seguidamente, se describe someramente el proyecto global y se especifica el alcance de este proyecto, enmarcándolo en un contexto más amplio.

El proyecto global tiene dos fases claramente diferenciadas. En la primera fase se ha puesto especial énfasis en el desarrollo y verificación de un conjunto de algoritmos que permiten simular la propagación de incendios en entornos virtuales. Las características principales de estos algoritmos son:

- Integración de entornos forestales y urbanos en un modelo único y compacto.
- Integración de información GIS heterogénea.
- Simulan simultáneamente ambos entornos.
- Permiten la propagación a distancia de los incendios (*Spotting Fires*).
- Se considera la simulación de la evaporación de los agentes extintores por medio del calor.
- Los algoritmos son eficientes para permitir la simulación interactiva en escenarios de entrenamiento y de *Decision Making*.

En consecuencia, se dispone de una librería de software que permite desarrollar aplicaciones para la extinción de incendios de manera interactiva (usuario) en entornos forestales y urbanos.

Esta primera fase consta de tres tareas principales:

- Estudio del problema y diseño de algoritmos.
- Implementación de algoritmos.
- Validación de los algoritmos.

La segunda fase del proyecto implica la implementación de una "Aplicación de Entrenamiento e Instrucción". Esta será la primera aplicación práctica desarrollada con la librería de simulación desarrollada en la fase I. Esta fase consta de las siguientes tareas:

- Definición de requerimientos.
- Diseño del Interfaz de Usuario.
- Arquitectura e implementación.
- Validación.

Dentro de este marco general, las fechas de desarrollo del Proyecto Fin de Carrera, han determinado en qué tareas del proyecto general se debía ver implicado.

Este condicionamiento ha determinado que la primera parte del Proyecto Fin de Carrera se enmarque en la tarea de validación de los algoritmos. El resto del proyecto se enmarca en todas las tareas de la segunda fase, excepto en la de validación (verificación con usuarios y expertos).

Este proyecto no va a incluir una descripción detallada de las librerías y algoritmos de simulación, excepto lo necesario para hacer comprensible las tareas aquí realizadas. Una exposición más completa se puede leer en la Tesis presentada recientemente en este

mismo departamento (Moreno 2013a) y en el artículo de Journal en el que he participado como co-autor (Anexo I).

2 DOCUMENTO DE OBJETIVOS DEL PROYECTO

En las siguientes secciones se concretan los objetivos y tareas de este Proyecto Fin de Carrera, pero para que esas descripciones tengan sentido, debe tenerse presente lo que en la introducción se acaba de comentar.

2.1 Objetivos

El objetivo principal que se plantea para la realización de este proyecto, es la de construir una aplicación que ayude en el proceso de aprendizaje a los profesionales que actúan en la extinción de incendios urbanos y forestales.

Además se marcan los siguientes objetivos no menos importantes:

- Colaborar en el proceso de validación de los algoritmos de simulación y propagación de incendios.
- Integrar los algoritmos de simulación y propagación de incendios en la aplicación.

2.2 Tareas

Este proyecto, siguiendo las pautas expuestas al comienzo de este capítulo, aborda las siguientes tareas:

1. Validación de algoritmos (en la Fase I),
2. Definición de requerimientos (en la Fase II),
3. Diseño de la Interfaz de Usuario (en la Fase II),
4. Arquitectura e implementación (en la Fase II)

La validación de las librerías implementadas en Vicomtech-IK4 durante la Fase I, se desarrolló siguiendo dos pautas principales, tal como se expone en (Moreno 2013a y 2013b). Este proyecto verificó que la propagación y extinción de incendios simulada mediante las librerías, sigue unas pautas razonables según los comentarios de la bibliografía y las experiencias de expertos.

Tras colaborar en el cierre de la Fase I se abordaron las tareas indicadas, ya dentro de la Fase II.

2.3 Estructura de descomposición del trabajo

La Figura 1 muestra la estructura de descomposición de trabajo, que está dividida en dos Fases.

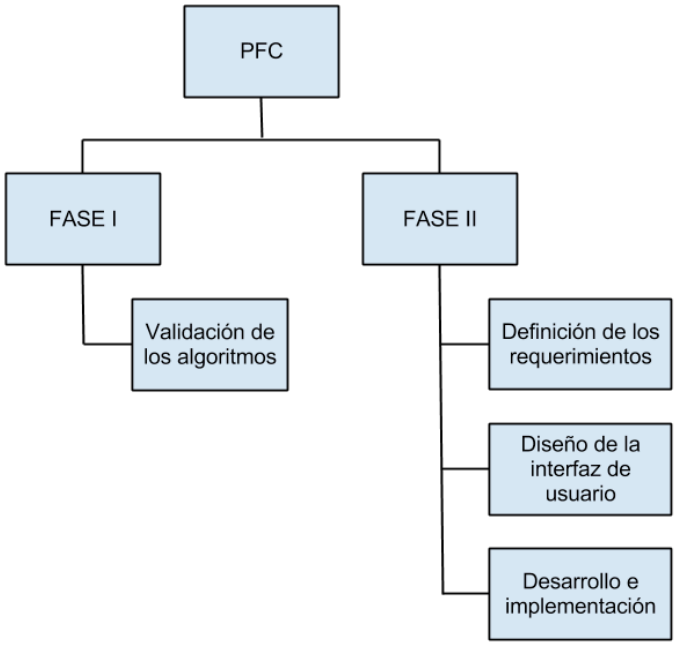


Figura 1 - EDT

2.4 Cronograma

En este apartado se presenta el cronograma del proyecto.

		Nombre	Duración	Inicio	Fin
1		Validación de los algoritmos de simulación y propagación de incendios	31.4d?	28/01/2013	22/02/2013
2		Definición de los requerimientos	7.4d?	22/02/2013	28/02/2013
3		Diseño de la interfaz	12d?	22/02/2013	05/03/2013
4		Desarrollo de la aplicación	95.4d?	27/02/2013	21/05/2013

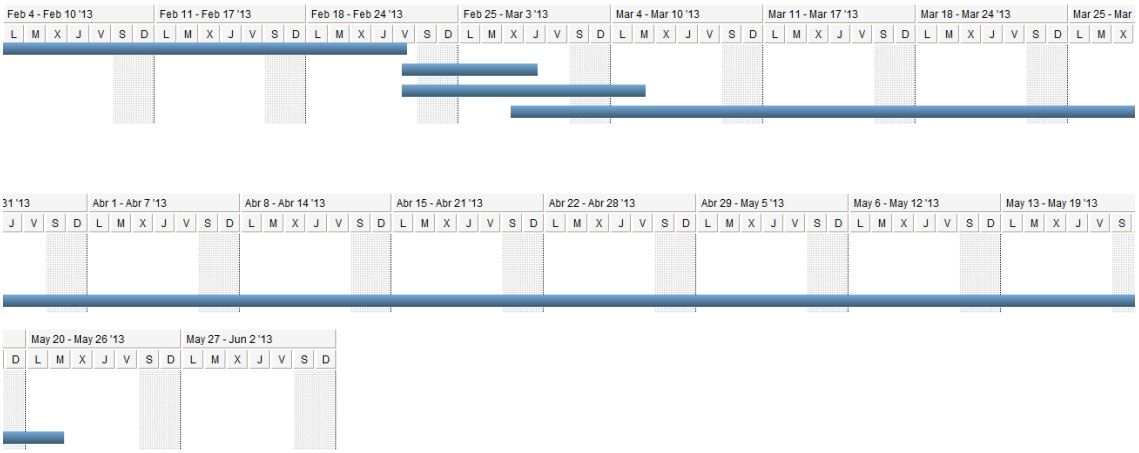


Figura 2 – Diagrama de Gantt

2.5 Contenido de la memoria

La memoria contiene un total de diez capítulos. En este apartado se describe brevemente el contenido en cada uno de ellos.

El primer capítulo introduce y pone en contexto el Proyecto Fin de Carrera.

El segundo capítulo presenta los objetivos y las tareas de este proyecto.

El tercer capítulo corresponde a la primera fase del proyecto, en donde se detalla las pruebas realizadas para la validación de los algoritmos de simulación y extinción de incendios.

El cuarto capítulo sintetiza los requerimientos de la aplicación de entrenamiento e instrucción en la extinción de incendios, siendo el primer capítulo perteneciente a la segunda fase.

El quinto capítulo presenta la interfaz de la aplicación, mostrando las diferentes ventanas que la componen con sus respectivas capturas de pantalla.

El sexto capítulo explica el proceso que se ha llevado a cabo en el desarrollo de la aplicación, así como el *workflow* empleado con el sistema de control de versiones.

El séptimo capítulo detalla el diseño de la aplicación dando a conocer las partes importantes del simulador y como se han plasmado en la implementación.

El octavo capítulo muestra la arquitectura de la aplicación con diagramas de clases y los casos de uso.

En el noveno capítulo se describen las conclusiones obtenidas tras la creación de la aplicación y las posibles líneas de futuro.

En el décimo capítulo se especifica la bibliografía y referencias más relevantes empleadas durante el proyecto.

Por último, en el Anexo I se adjunta el artículo enviado al Fire Safety Journal que se debe consultar para entender los algoritmos de simulación y propagación de incendios. En el Anexo II se describe la utilidad para la generación de escenarios creada para la validación de los algoritmos.

3 ANÁLISIS Y VALIDACIÓN DE LOS ALGORITMOS DE SIMULACIÓN Y PROPAGACIÓN DE INCENDIOS

En el documento de objetivos del proyecto se han expuesto las tareas. Allí se explicó como este Proyecto Fin de Carrera contribuye en la validación de la Fase I. Este capítulo expone las pruebas que se han realizado para validar esos algoritmos de propagación y extinción de incendios (tercera fase del proyecto global y primera tarea de este proyecto).

En este apartado, analizaremos visualmente el comportamiento de los algoritmos en función de los parámetros que alteran el comportamiento de la propagación del fuego, como en el caso de la pendiente de una colina. Se obtendrán datos visuales de las simulaciones con diferentes pendientes que permitirán a los expertos contrastar que el comportamiento del algoritmo está dentro de lo esperado. De forma similar, se analizarán otras características.

El análisis del rendimiento de los algoritmos es de obligado cumplimiento debido a la necesidad de comprobar la interactividad de los algoritmos en entornos de simulación de tiempo real o interactivos (Apartado 3.5). En el primer apartado del capítulo se introducen algunos de los términos y conceptos empleados.

Los resultados de este capítulo constituyen una parte importante del artículo enviado para su publicación al Fire Safety Journal. Por razones de extensión, aquí solo se describen las pruebas más relevantes, sin incluir todos los resultados de los extensos análisis realizados.

3.1 Modelo y algoritmos

Los algoritmos propuestos para la simulación de la propagación y extinción del fuego, han sido diseñados para funcionar en entornos urbanos y forestales.

En los algoritmos el terreno es un componente clave. El terreno está dividido en una malla regular compuesta por celdas y cada celda contiene las características que definen su comportamiento y un estado asociado a la misma. En cada iteración de los algoritmos, se mantiene en memoria la información de cada celda del terreno, actuando como fuente de datos de entrada y de salida en cada iteración.

Las características principales son las siguientes:

- La evolución del fuego está basada en la topología, los materiales y las condiciones ambientales.
- Los edificios son modelados por plantas.
- Se simulan medidas como las barreras o cortafuegos.
- Algoritmos con complejidad baja que permiten la simulación de incendios en tiempo real.
- Soporte para los procesos de extinción.

- Soporte para cambios dinámicos, como las condiciones ambientales.

En cada iteración del algoritmo se modifican los estados y parámetros de las celdas del terreno. El fuego siempre se propaga hacia las celdas vecinas. Cuando una celda no es afectada por el fuego, está en estado *safe*, mientras que cuando ha sido quemada pasa a un estado final, *burnt*. Cuando una celda está siendo consumida por el fuego podemos decir que está activa y por ende, en estado *Activated.* En algunas celdas, el fuego ha sido apagado pero no se han consumido del todo, es decir, que se encuentran en un estado *FireStopped.* Estas celdas pueden reavivarse volviendo a un estado *Activated* si se dan las condiciones idóneas. En el Anexo I se encuentra el diagrama con los estados de una celda. Este diagrama se explica con mayor detalle en el Anexo I. En ese mismo anexo se definen cada uno de los algoritmos.

Las gráficas que se muestran en este capítulo hacen uso del término *steps,* representando el número de pasos dados por los algoritmos para calcular el avance de la simulación. El *tiempo* en el apartado de rendimiento, hace referencia al tiempo de cálculo. En general, durante la memoria se utiliza los puntos cardinales y el sistema de referencia de coordenadas cartesianas que está formado por tres rectas perpendiculares entre sí (X,Y,Z). Utilizando el convenio que se muestra en la Figura 3.

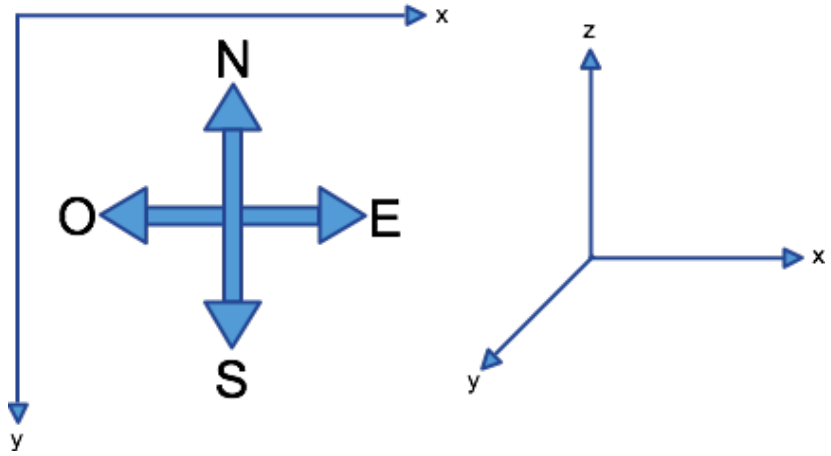


Figura 3 - Referencia de orientación en los escenarios

Para la validación, se han realizado una serie de simulaciones con escenarios predefinidos y sintéticos que se exponen en los siguientes apartados.

Los algoritmos contienen cierta aleatoriedad implantada para que no se den simulaciones idénticas. Si lo que pretendemos es una simulación lo más fiel posible a la realidad, repetiremos cada una de ellas un número determinado de veces hasta obtener la media y de este modo un resultado normalizado.

La validación prevista en este trabajo es la de contrastar visualmente el comportamiento del fuego simulado. Se generan imágenes que permitan validar que el avance del fuego es el esperado. Para ello, tras una cantidad de *steps* predefinida se captura un *snapshot* que represente el estado de la simulación. Además, se almacenan los datos de los parámetros más relevantes de una forma estructurada, de manera que posteriormente nos permita visualizar dichos datos en gráficas.

Para cada prueba se crea un escenario específico con el fin de validar un comportamiento concreto bajo unas condiciones aisladas.

Para el proceso de generación de los escenarios de forma cuasi automática se ha creado una utilidad escrita en *Python* y *Qt*. Al ser un programa pequeño y ajeno a los algoritmos, es fácilmente modificable para ayudar en la construcción de escenarios complejos. En el Anexo II, se puede encontrar más detalle sobre la utilidad y la composición de los terrenos utilizados en las pruebas.

3.2 Entorno forestal

Los incendios forestales son muy comunes en ciertos lugares del mundo donde, debido al clima y al tipo de vegetación, pueden provocar graves daños medioambientales. Puesto que hay varios factores que modifican el comportamiento del fuego en este entorno, a continuación generaremos las imágenes que se han usado para validar los puntos clave de los algoritmos.

3.2.1 Pendientes

La inclinación o pendiente de los terrenos es uno de los factores que más influye en la propagación del fuego, determinando su dirección y velocidad de propagación. Para validar el comportamiento de los algoritmos se han generado tres tipos de modelos y de los resultados se ha obtenido el comportamiento esperado.

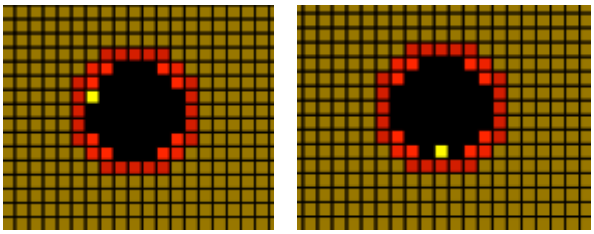


Figura 4 - Plano horizontal

El primer modelo consiste en un plano horizontal, sin inclinación alguna. Tras varias simulaciones se puede afirmar que las desviaciones producidas por la aleatoriedad de los algoritmos son pequeñas. Cuando no existen factores externos que alteren la trayectoria del fuego, éste se propaga de forma circular y uniforme (Véase Figura 4).

El segundo modelo emula las condiciones de una colina. El escenario consiste en un plano con diversas pendientes, véase Anexo II. El fuego tiende a subir hacia lo más alto de la colina. Para validar está tendencia se han generado distintas versiones de este modelo con diferentes grados de inclinación, tanto positivas como negativas, con el fin de representar pendientes ascendentes y descendentes.

Puesto que todas las simulaciones realizadas con este modelo han compartido el punto de origen del incendio en el mismo lugar, podemos comparar en el mismo paso de la simulación, el avance del incendio debido al cambio de pendiente con las fluctuaciones en la velocidad de propagación en el mismo.

Los resultados han constatado que aumentando la inclinación del terreno el fuego se propaga a mayor velocidad y, al contrario, cuando la inclinación es descendente el fuego se propaga a menor velocidad. Véase la Figura 5 donde se pueden diferenciar los dos escenarios. En ambos, el fuego empieza en una zona horizontal (sobre la línea) y luego se encuentra con una ladera. En un caso con pendiente positiva (el fuego sube con facilidad), en otro con pendiente negativa (el fuego desciende con mayor dificultad). Los dos escenarios corresponden al mismo instante de tiempo de simulación.

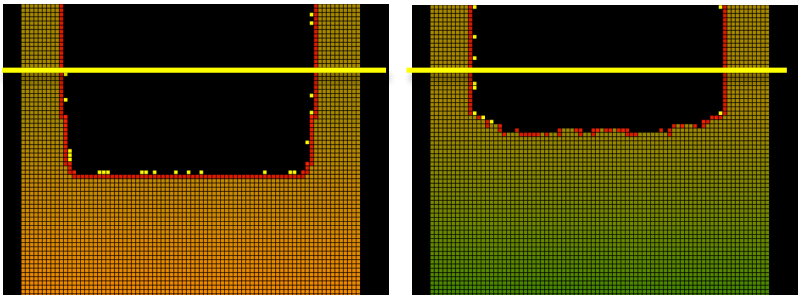


Figura 5 - Pendiente 60º (izquierda) y pendiente -60º (derecha)

El tercer modelo representa un escenario con una pendiente más suavizada, recreado con una campana de gauss en el centro del escenario. Las simulaciones han mostrado que el fuego se propaga hacia la punta, mientras que si invertimos la campana hacia dentro del terreno, el fuego rodea la campana antes de descender sobre ella. Véase Figura 6 y Figura 7.

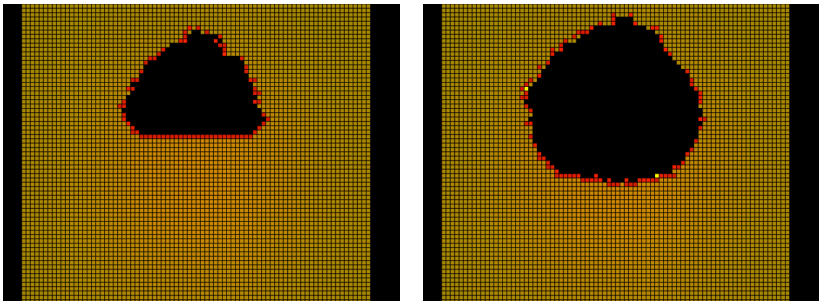


Figura 6 – Colina (campana de gauss)

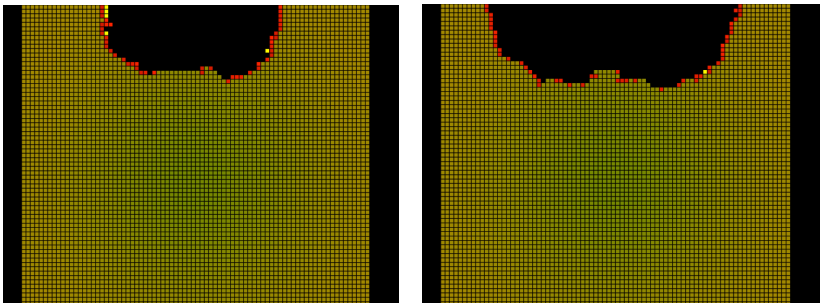


Figura 7 – Depresión (campana de gauss invertida)

3.2.2 Viento

Los incendios son alterados inesperadamente debido a las condiciones ambientales y el factor más agravante e importante de todos ellos es el viento. El viento influye en gran medida, imponiendo la dirección de la propagación.

Para la validación de este fenómeno natural, no se ha construido ningún modelo específico sino que se han utilizado los mencionados anteriormente. Validaremos el dinamismo del viento a lo largo de la simulación.

Si queremos validar algoritmos diseñados para simular en tiempo real, deberán ser capaces de tomar como entrada un viento variable, puesto que para una simulación que dura varias horas es normal que la dirección y la intensidad del viento sea cambiante.

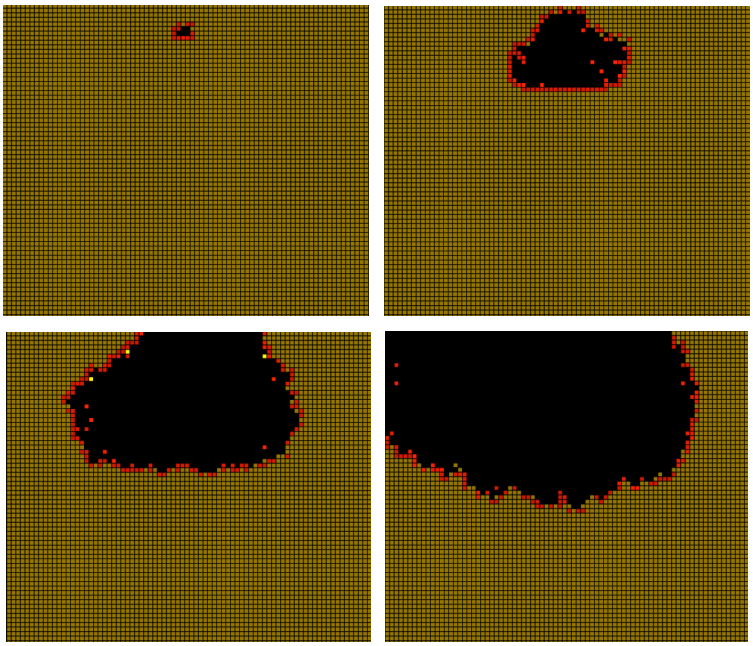


Figura 8 - Viento cambiante

En las simulaciones se ha hecho uso de un viento que cambia de dirección gradualmente. En la Figura 8 se puede ver cómo el fuego se propaga en dirección Sur en las primeras dos capturas superiores, mientras que en las inferiores, la dirección, del viento se altera hacia el Oeste. En la captura inferior derecha se puede apreciar como finalmente, la parte izquierda del terreno se ha consumido en mayor medida que la parte derecha.

3.2.3 Vegetación

Los algoritmos propuestos se adaptan al terreno y a su naturaleza. En los entornos forestales existe una gran diversidad de vegetación y cada tipo de vegetación tiene una combustión diferente.

Para las simulaciones se ha generado un modelo especial. El escenario consiste en un plano horizontal separado por tiras verticales compuestas de agua. Las dos barreras separan tres áreas compuestas por tres tipos de vegetación diferentes.

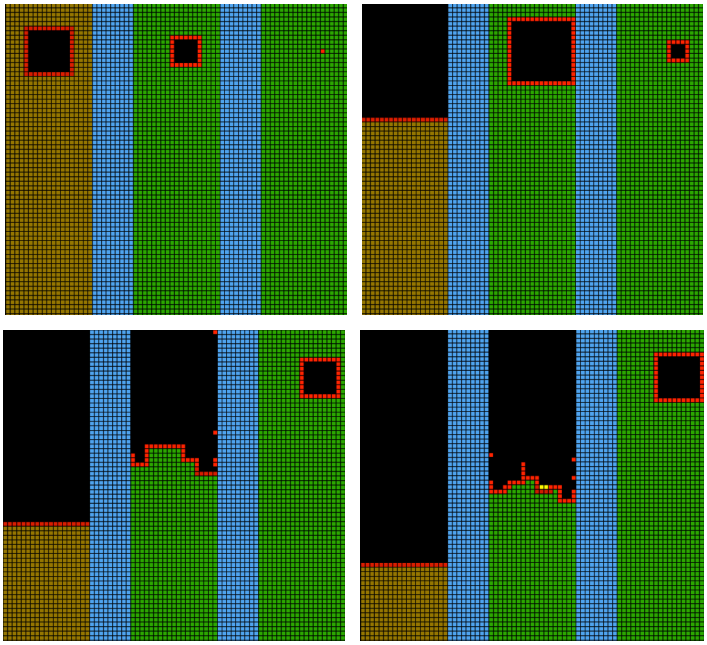


Figura 9 - Simulación en diferentes tipos de vegetación

En la Figura 9 se puede observar cómo originando un incendio en cada área al mismo tiempo y dependiendo de la vegetación, el fuego se propaga a una velocidad mayor o menor. El tipo de vegetación de la izquierda es de menor tamaño y prende más rápido debido a que tiene menos combustible para el fuego. Sin embargo, en la vegetación de la derecha que se compone de arboles de mayor altura y que contienen mayor cantidad de combustible para quemar, prende más despacio.

3.2.4 Barreras

Las barreras anti-incendios son un método de contención para parar y controlar el avance del fuego. Las barreras pueden ser naturales o artificiales. Entre las barreras naturales podemos destacar los ríos y los lagos y entre las barreras artificiales conocemos carreteras, pistas y zanjas. El método más común por el cual el fuego salta las barreras es el denominado *spotting fires*. Cuando esto ocurre, el material incendiario es transportado por el viento y la zona incendiada eleva la temperatura en la zona situada al otro lado de la barrera, posibilitando la propagación del incendio.

Para que el incendio sobrepase la barrera se tienen que dar las condiciones idóneas, tanto ambientales (viento) como geográficas. Los árboles y arbustos de gran tamaño en el lado activo de la barrera, ayudan a aumentar las posibilidades de que el fuego sobrepase la barrera, porque facilitan el transporte de material incendiario por el aire. Por otro lado, que en el otro lado de la barrera haya vegetación de ignición rápida, como arbustos y rastrojos, es un factor favorable.

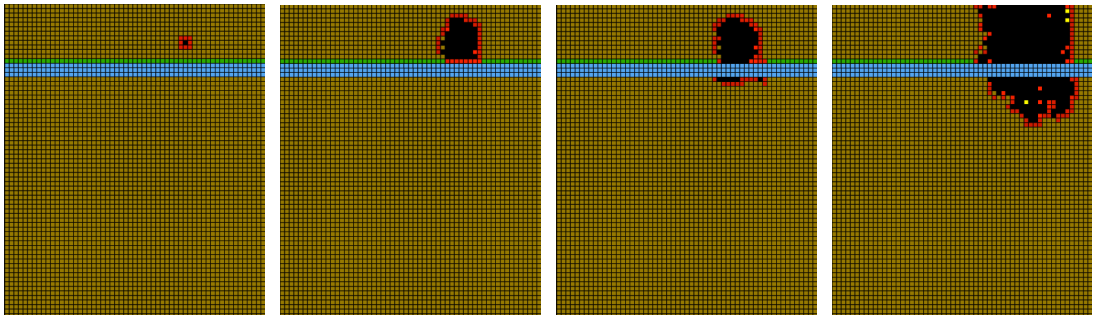


Figura 10 - Barrera de 9 metros

En esta serie de simulaciones se han generado unos escenario con una barrera horizontal. En la Figura 10, se puede apreciar cómo con el viento a favor, el fuego sobrepasa la barrera, mientras que con una barrera lo suficientemente ancha, mayor igual a 12 metros, se evita la propagación a pesar de las condiciones favorables para ello.

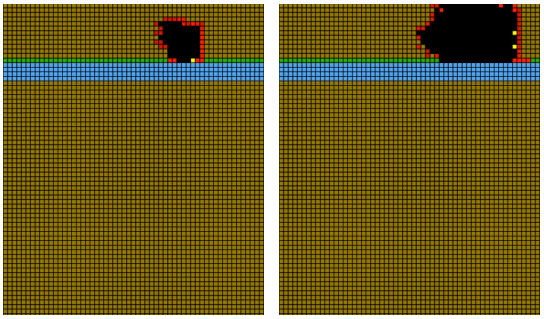


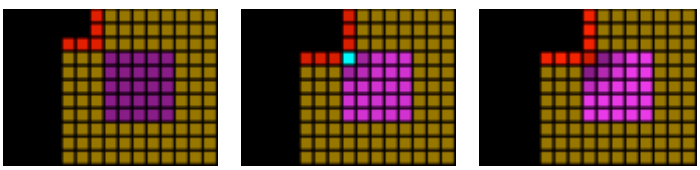
Figura 11 - Barrera de 12 metros

3.2.5 Uso de agentes extintores

El diseño de los algoritmos tiene como premisa soportar la extinción del incendio de manera interactiva. En el proceso de extinción se diferencian cinco casos derivados del tiempo en el que se interactúa con la simulación. En las simulaciones, se arroja agente extintor en una área cuadrada, pudiéndose apreciar en las imágenes por su color fucsia. El gradiente indica que cuando más oscura sea la celda, contendrá una menor cantidad de agente extintor. La cantidad de agua arrojada ha sido la misma en todas las simulaciones. El fuego se dirige hacia el sureste, propagándose en primer lugar a la celda superior izquierda del área que contiene el agente extintor.

3.2.5.1 Arrojar el agente extintor de forma preventiva

La acumulación de agente extintor en una celda evita el avance del fuego, pero debido a la intensidad del fuego en las celdas contiguas el agente extintor es finalmente evaporado y puede llegar a ser totalmente consumido por el fuego.



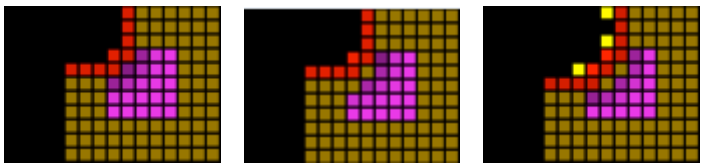


Figura 12 - Agente extintor acumulado

El la Figura 13 se puede observar la intensidad del fuego y el agua acumulada en la celda superior izquierda del área que contiene agente extintor. En el eje horizontal se representa el tiempo de simulación mediante *steps*. En primera instancia, se acumula una cantidad de agua entre los *steps* 720 y 820. En ese momento, el fuego alcanza la celda y empieza a evaporar el agua, no obstante, debido a que el agua sigue vertiéndose en la celda, el fuego en la celda se extingue y el agua vuelve a acumularse. En el *step* 1040 se deja de arrojar agua, y las celdas vecinas afectan a dicha celda evaporando el agua acumulada y reavivando el fuego en la misma hasta que finalmente se quema.

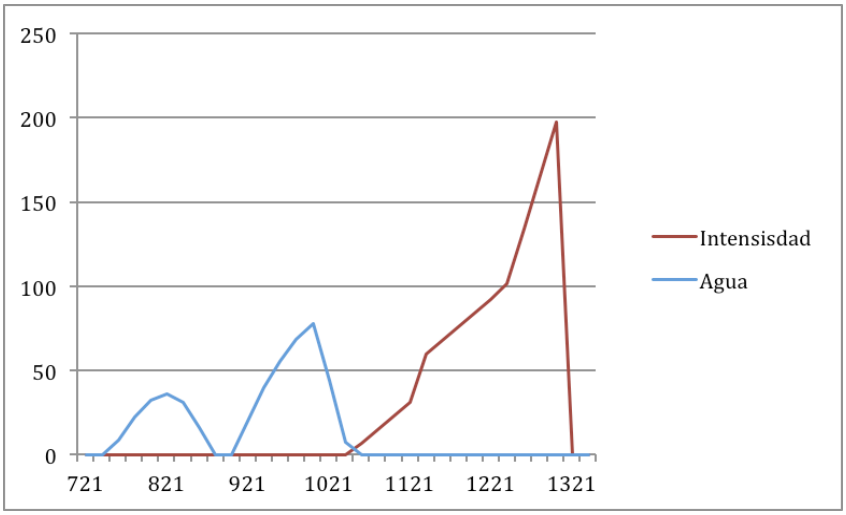


Figura 13 - Intensidad y agua acumulada en la celda

3.2.5.2 Arrojar el agente extintor nada más formarse el fuego en la celda.

Dado que al principio del ciclo de consumición de la celda, la intensidad del fuego no es muy elevada, el agente extintor consigue parar el avance del fuego de manera temporal, hasta que éste deja de ser arrojado.

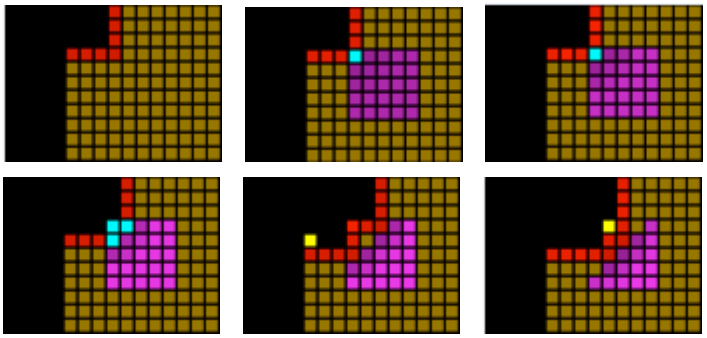


Figura 14 - Agente extintor en el instante inicial

En la Figura 15 se muestra, (como en el ejemplo anterior) una gráfica de la celda en la parte superior izquierda del área que tiene agente extintor acumulado. En primer lugar, la celda es afectada por el fuego. Nada más comenzar la ignición, se empieza a arrojar agua apagando el fuego completamente hasta el *step* 1120 donde, debido al calor que irradian las celdas vecinas, el agua es evaporada. Cuando finalmente se deja de arrojar agua, la celda termina consumiéndose.

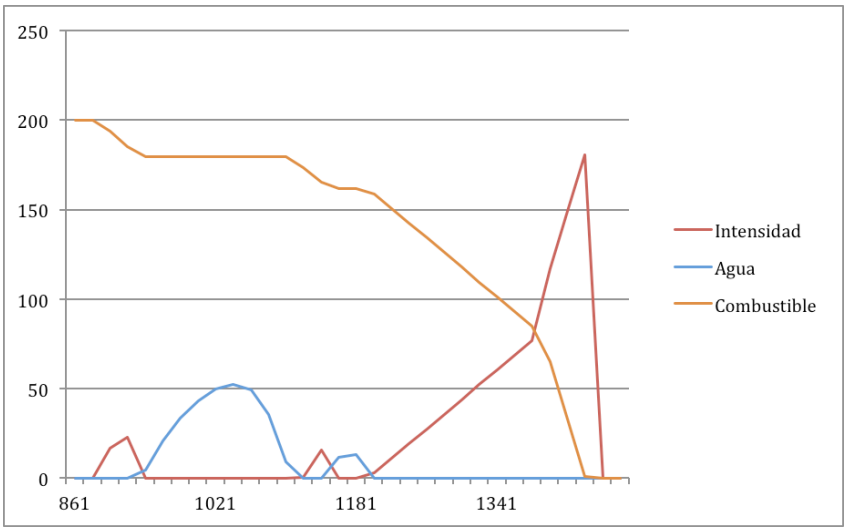


Figura 15 – Arrojar agua nada más incendiarse la celda

3.2.5.3 Arrojar el agente extintor cuando el consumo del combustible de la celda es aproximadamente la mitad.

La cantidad de agente extintor no consigue frenar el avance del fuego, sino que ralentiza su consumición mientras se arroja de manera continuada.

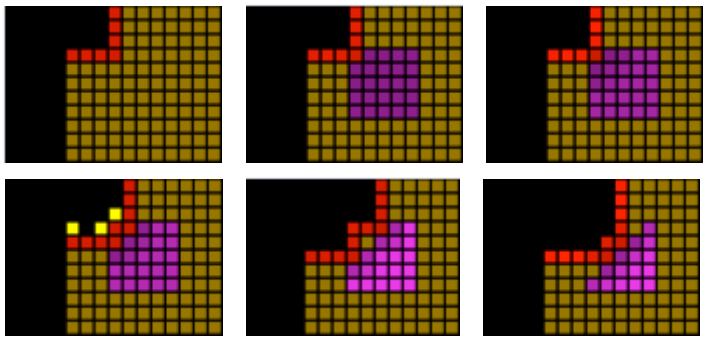


Figura 16 - Agente extintor en mitad del proceso de consumición de la celda

En la Figura 17 se muestra una gráfica con la misma celda que las anteriores. En éste caso, el combustible remanente de la celda es la que nos da una información clara de lo que sucede en ella. Cuando la celda es afectada por el fuego, la intensidad aumenta y la celda se consume. En el momento en el que empezamos a arrojar agua, la consumición de la celda se detiene temporalmente hasta que dejamos de hacerlo. Finalmente, la celda acaba consumiéndose, pero podemos decir que el agua ha detenido la consumición de la celda durante un tiempo.

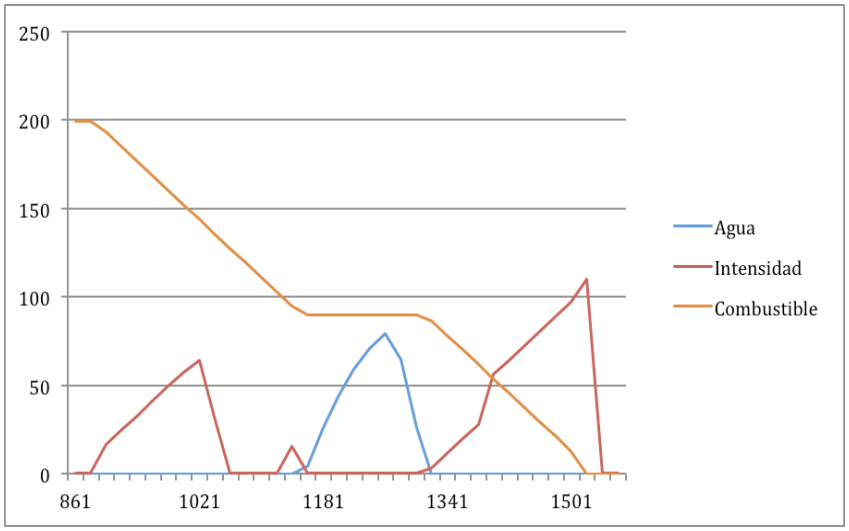


Figura 17 - Arrojar agua cuando el consumo de la celda es la mitad

3.2.5.4 Arrojar el agente extintor cuando el consumo del combustible de la celda es aproximadamente la totalidad.

La intensidad del fuego es muy grande como para parar la consumición de la celda, no obstante, ralentiza el proceso. Cuando se para de arrojar el agente extintor, éste se evapora y la celda se consume.

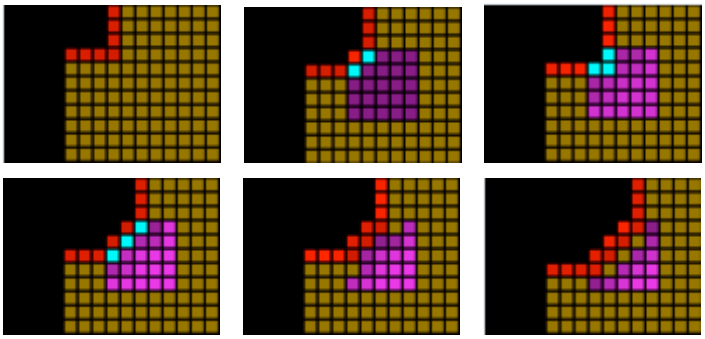


Figura 18 - Agente extintor en instante final de consumición de la celda

En la Figura 19 se muestra la gráfica de la celda superior izquierda. En ella se puede observar cómo el agua apenas disminuye la intensidad alcanzada por el fuego y se consume antes de poder paralizar la consumición de la celda.

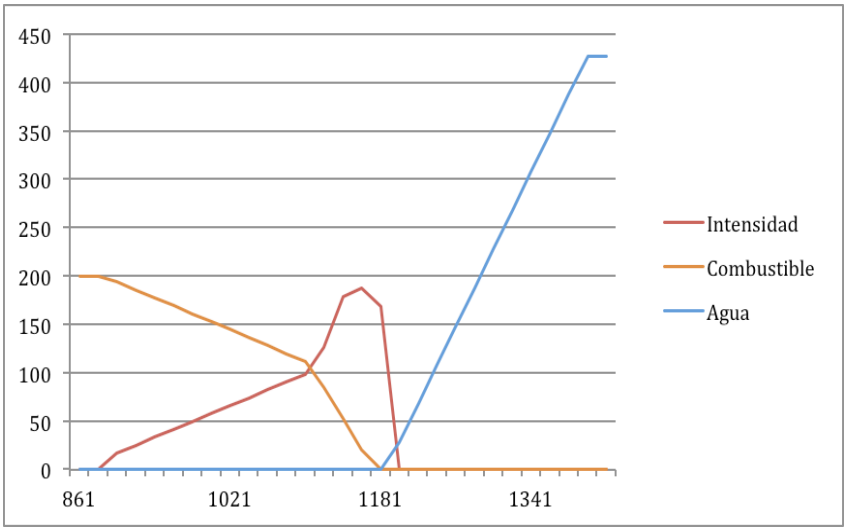


Figura 19 - Arrojar agua al final del proceso de consumición

3.2.5.5 Arrojar el agente extintor de forma continua o ilimitada.

La gran cantidad de agua imposibilita la consumición de la celda hasta el punto de parar la propagación.

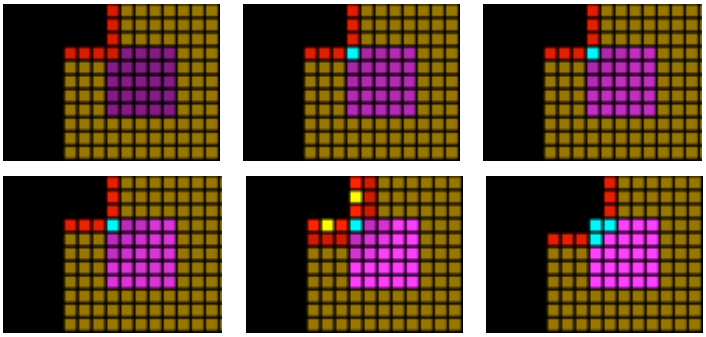


Figura 20 - Agente extintor arrojado de manera continua

En la Figura 21 se muestra una gráfica de la celda superior izquierda dentro del área que contiene agente extintor. En la gráfica se puede observar cómo arrojar agua de manera continua a un área impide la propagación del fuego, siempre y cuando la cantidad de agua arrojada sea lo suficiente como para contrarrestar el calor que emitan las celdas vecinas.

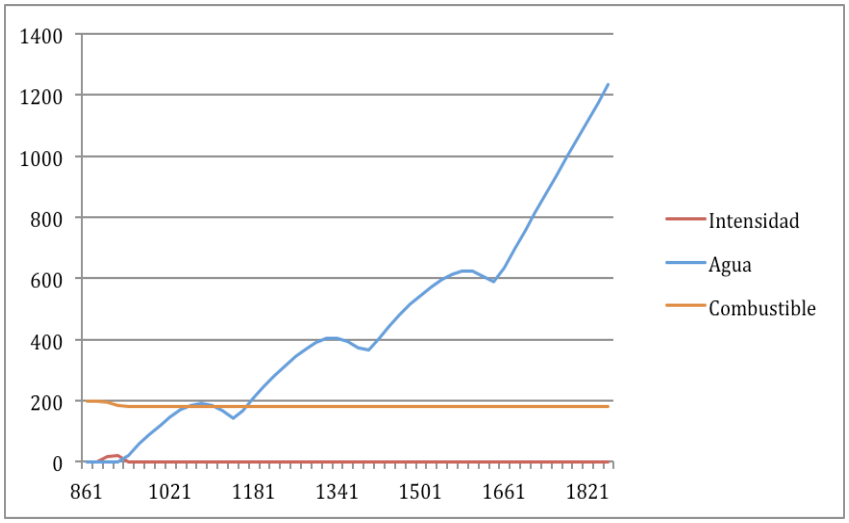


Figura 21 - Arrojar agua de forma continua

3.3 Entorno urbano

En el entorno urbano no se consideran los mismos factores y variables que en el entorno forestal. Para este entorno, se han realizado tres pruebas principales enfocadas en la propagación del fuego en los edificios y el proceso de extinción en los mismos.

3.3.1 Propagación vertical y horizontal

Se ha generado un escenario específico tanto para el caso de la propagación horizontal como para la propagación vertical.

Para la propagación horizontal se ha creado un edificio que está compuesto por un bloque de celdas con un solo nivel de altura, tres celdas a lo ancho y tres celdas de profundidad; es decir, (3,3,1) en los ejes (X, Y, Z).

En la Figura 22 se ha situado el origen del incendio en la celda central, en el *step* (800) de la simulación. Cuando la intensidad de la propagación de la celda llega a su apogeo, se produce la propagación entre plantas contiguas.

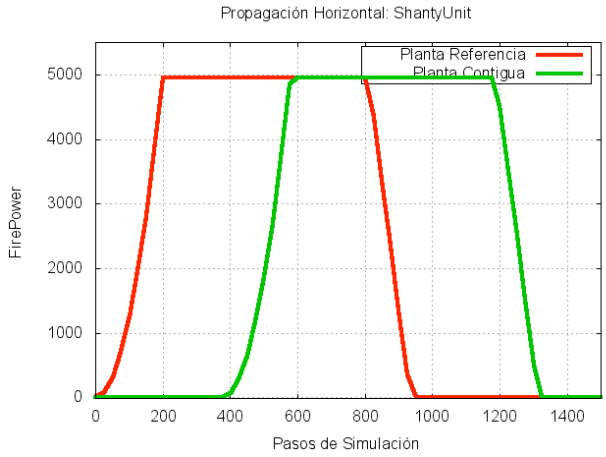


Figura 22 - Propagación horizontal entre plantas de un edificio

Para la propagación vertical se ha creado un edificio compuesto por un bloque de celdas apiladas, una encima de otra. El bloque de celdas conforma una configuración de (1,1,3) en el eje (X,Y,Z).

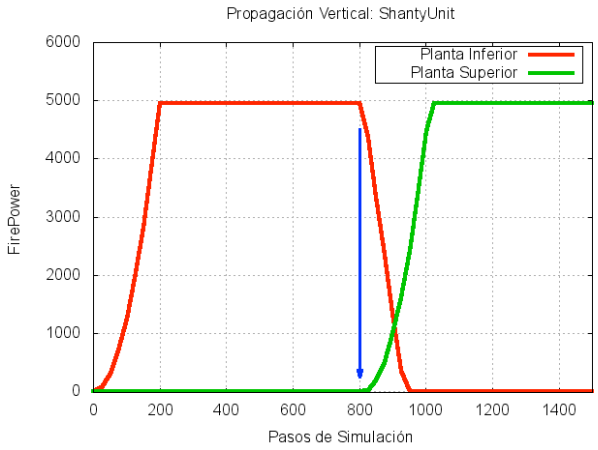
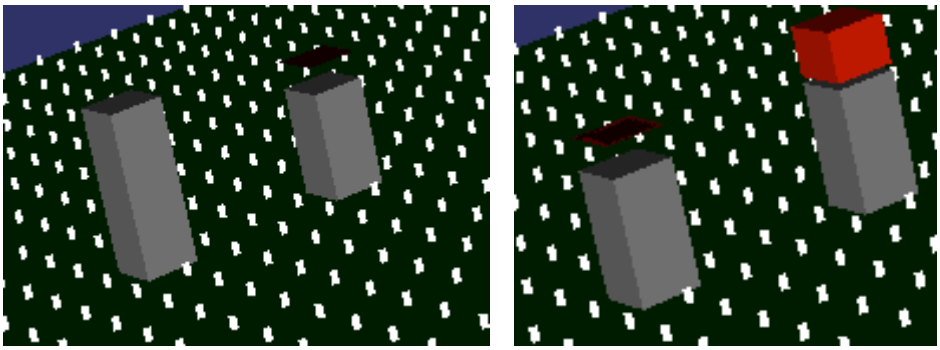


Figura 23 - Propagación vertical entre plantas de un edificio

Cuando la intensidad del fuego llega a su máximo y empieza a decrecer, es cuando el fuego se propaga a la planta superior. En la Figura 23 se puede ver la transición de nivel sobre el *step* 800.

3.3.2 Spotting fires

El efecto *spotting fires* también interviene en la propagación del fuego cuando las condiciones del viento son favorables. Los restos quemados y las brasas son transportados por el viento a toldos, ropa o material susceptible de ser incendiado situados en edificios adyacentes.



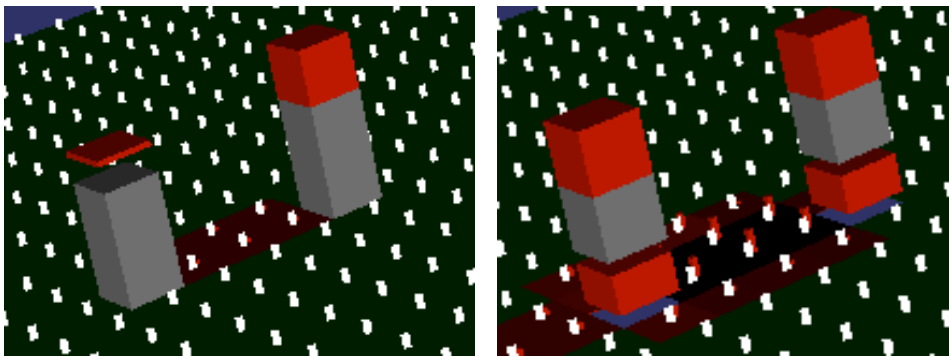


Figura 24 - *Spotting fires* entre edificios

Para las pruebas se ha creado una serie de escenarios con dos edificios colocados a diferentes distancias. Los dos edificios tienen la misma configuración de (1,1,3) y el viento es aplicado en la dirección favorable a la propagación, que para estos escenarios ha sido, dirección oeste.

Tras las simulaciones se concluye lo siguiente:

- La distancia máxima en la que se permite la propagación es de cuatro celdas, lo equivalente a doce metros y además, varía en función de los tipos de edificios.
- La propagación mediante el método *spotting fires*, sólo se produce desde la última planta del edificio.

3.3.3 Extinción

Se puede detener la propagación del fuego en los edificios arrojando agente extintor a las fachadas de los mismos. En la Figura 25 se muestran los resultados de la consumición de una celda de la fachada de un edificio, con y sin agente extintor.

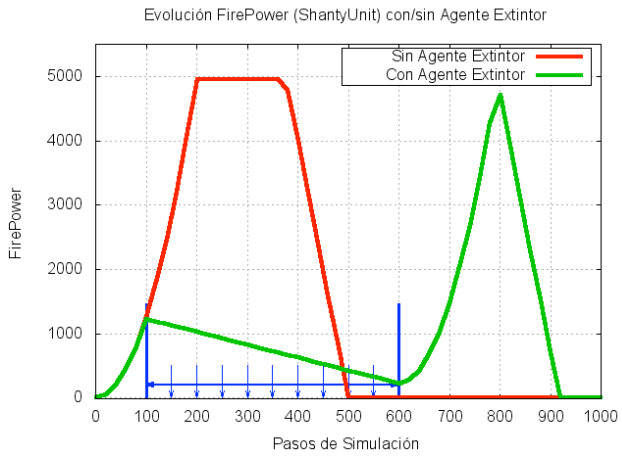


Figura 25 - Evolución del fuego con y sin agente extintor

Si queremos evitar la propagación del incendio a las plantas superiores o vecinas, se puede arrojar agente extintor a éstas como método preventivo, de igual forma que se ha señalado anteriormente en el entorno forestal.

3.4 Transición entre entornos

Los algoritmos están diseñados para simular tanto entornos urbanos como entornos forestales. De igual forma, son capaces de trabajar en un entorno mixto.

Para estas pruebas se ha generado un escenario sin pendiente en un edificio rodeado de vegetación alta.

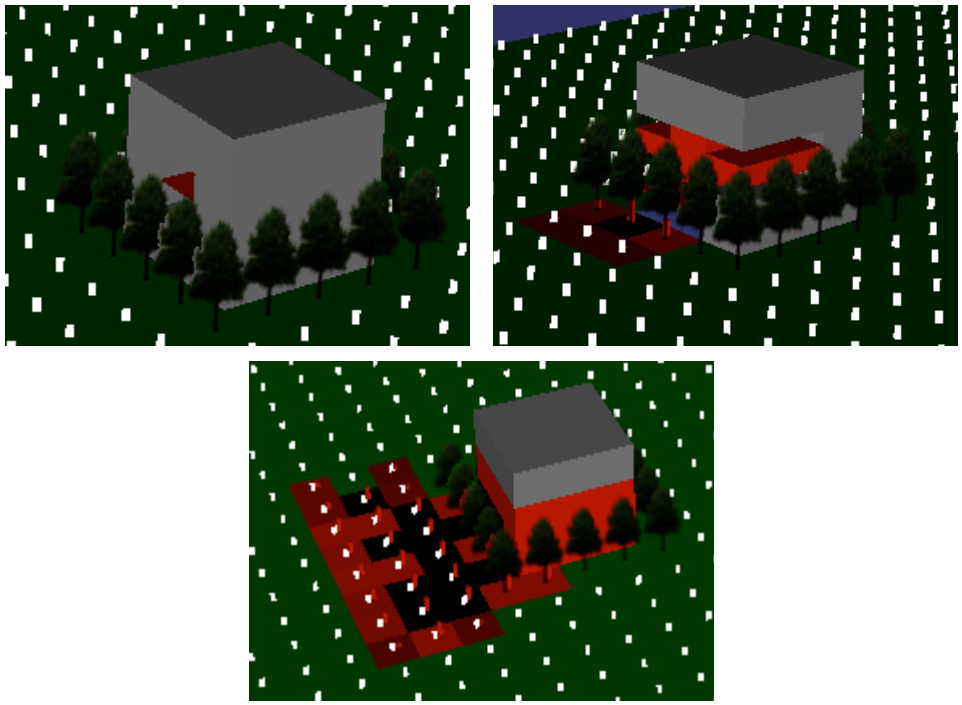
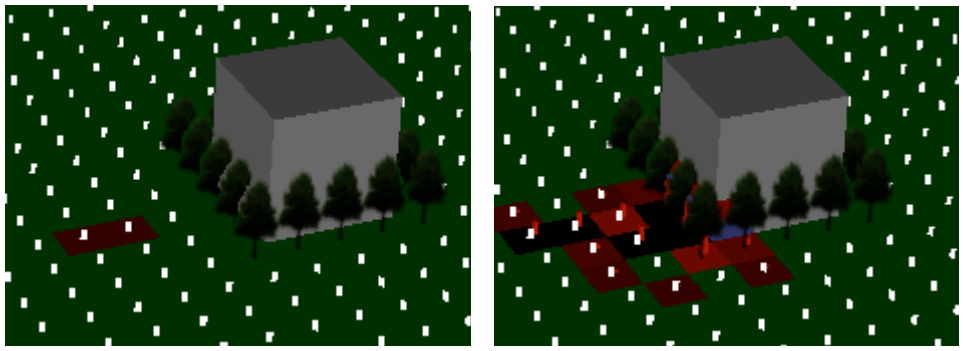


Figura 26 - Transición urbano-forestal

En la transición urbano-forestal, se aprecia que la propagación hacia la vegetación ocurre desde las primeras plantas, véase Figura 26. La probabilidad de que la propagación ocurra desde una segunda planta será siempre menor que desde la planta inferior. En el otro caso, para la transición forestal urbano, el fuego se propaga únicamente a través de la primera planta, tal y cómo se observa en la Figura 27.



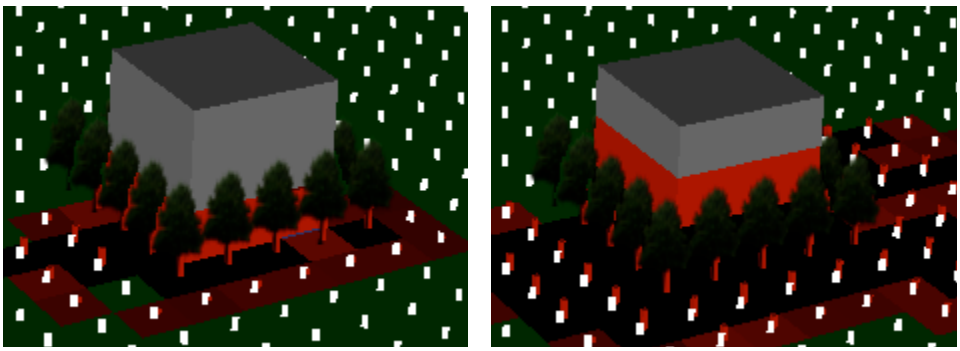


Figura 27 - Transición forestal-urbano

3.5 Rendimiento

Para comprobar el rendimiento de los algoritmos, se ha optado por medir el tiempo real de procesamiento necesario por cada paso de simulación. Las pruebas se han realizado con un ordenador de sobremesa estándar, sin utilizar un ordenador de gran capacidad de cómputo.

En cada paso de simulación se miden el número de celdas activas, que son las que alteran las condiciones actuales del incendio. En todas las simulaciones realizadas se ha medido el rendimiento.

En la Figura 28 podemos observar el número de celdas activas a lo largo del tiempo de simulación. El escenario corresponde a una de las pruebas con pendiente en forma de campana de gauss. Las condiciones del viento en este escenario son nulas.

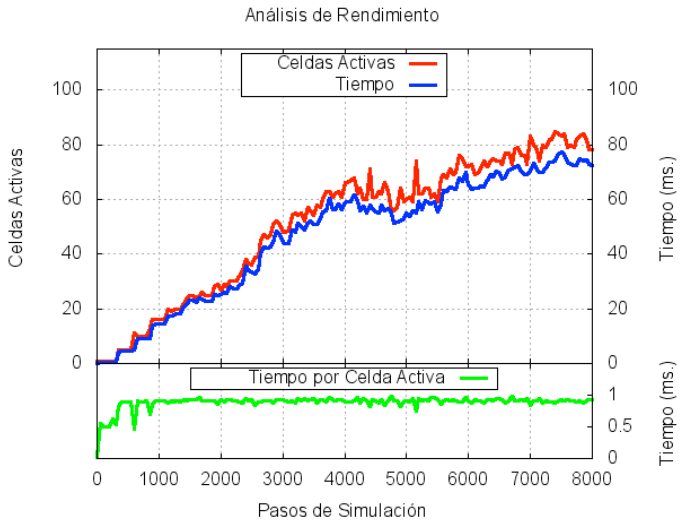


Figura 28 - Rendimiento de los algoritmos

El tiempo por cada paso de simulación ha sido siempre por debajo del umbral de 1 segundo, con una media de tiempo por celda activa menor a un milisegundo. Los tiempos en el rendimiento demuestran que los algoritmos propuestos son adecuados para simulaciones interactivas o que requieran tiempo real.

4 REQUERIMIENTOS

En este capítulo se presenta los requerimientos de la aplicación desarrollada en este PFC: "Aplicación de entrenamiento e instrucción para la gestión de recursos en la extinción de incendios". Nos referiremos a él cómo el simulador. que debe ayudar en el aprendizaje de la gestión de los recursos que intervienen en la extinción de los incendios.

Se distinguen los siguientes requerimientos, clasificados por categorías.

4.1 Usabilidad de la aplicación

La interfaz de la aplicación debe ser sencilla, proporcionando interacción tanto por la interfaz GUI, como del mundo virtual.

La interfaz debe ser dinámica, variando a la par con los sucesos de la simulación y siendo capaz de mostrar los datos relativos a la simulación en curso.

Dos actores o entidades con distintos roles definidos participarán en la aplicación. Por un lado, el alumno que ejecutará la simulación y por otro lado, el instructor que actuara como supervisor. En todo momento, la aplicación podrá ser utilizada sin la supervisión del instructor, siendo totalmente funcional.

4.2 Características de la simulación

La simulación tendrá dos tipos de cámara que facilitarán la navegación en el entorno simulado. La cámara estará situada en una perspectiva de tercera persona, distinguiéndose dos casos:

- *"God Mode"*: donde la cámara estará libre pudiéndose rotar y moverse a través del mundo simulado.
- *"Vista de águila"*: donde la cámara tendrá una perspectiva de vista aérea, pudiendo moverse por el escenario, sin rotación.

El simulador deberá integrar los algoritmos de simulación y propagación de incendios que se describen en el Anexo I. Estos algoritmos se integrarán en la aplicación mediante una librería externa denominada SuLib. Se requerirá identificar y detectar las incompatibilidades ayudando en el proceso de adaptación de la API de la librería, de modo que sea compatible con el simulador. Los escenarios se obtendrán del Framework de SuLib y habrá que adaptarlos para que sea compatible con la información extra de la que hará uso el simulador.

El tiempo es un factor crucial, tanto para la simulación en su totalidad, como en las acciones de los recursos. Será preciso llevar un correcto orden de las acciones en todo momento.

4.3 Recursos simulados

Los recursos serán las entidades que intervienen en la supresión del incendio y que serán controlados por el alumno. Los recursos deben ser, en la medida de lo posible, acordes con el mundo real. Por éste motivo, se tendrán en consideración los protocolos de actuación de los profesionales que se dedican a extinguir los incendios. Debido a la cantidad de recursos que puede haber, la primera versión del simulador se centrará en la representarán de dos tipos:

- Camiones de bomberos
- Bomberos

En cada escenario debe existir la posibilidad de utilizar una cantidad de recursos diferentes y se quiere poder definir las características de los mismos. El diseño ha de ser flexible posibilitando la integración de un mayor número de tipos de recursos en el futuro.

4.4 Utilización del simulador

Para poder iniciar la simulación, el usuario deberá escoger un escenario de los que están predefinidos y cargarlo. El escenario define el área y puntos de origen del incendio, así como los recursos que se pueden utilizar en el simulador y que el usuario debe gestionar correctamente.

En la parte izquierda de la interfaz del simulador estarán los paneles que controlan los recursos. Mediante estos paneles el usuario podrá utilizar los recursos, trasladándolos o echando agua al incendio.

Durante el ejercicio de simulación, el alumno actuará bajo la supervisión del instructor. alterando las condiciones del fuego y así dificultar las tareas de extinción al alumno si lo estima oportuno.

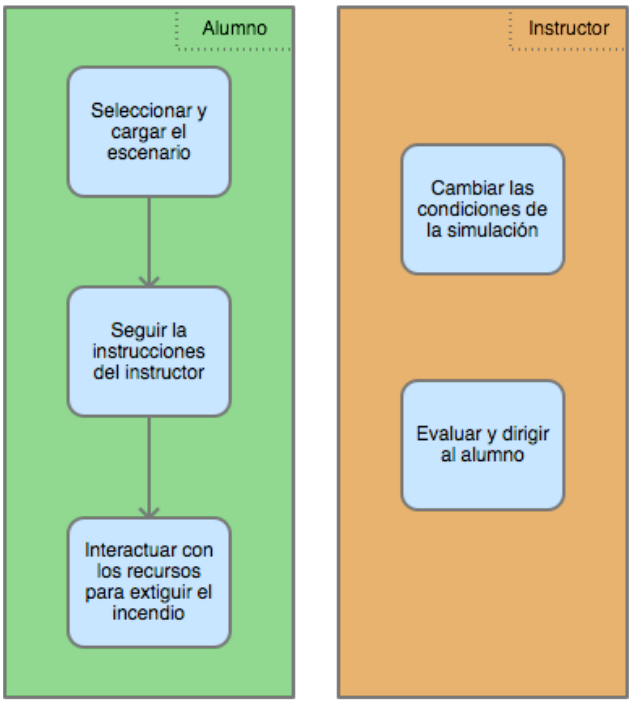


Figura 29 - Esquema de uso

4.5 Excepciones

No se crearán modelos 3D para el simulador. Realizar un modelado de los escenarios y recursos lo más realista posible no es el objetivo del proyecto; lo que se pretende es que los recursos sean representativos.

Algunos recursos como los agentes y vehículos se podrán desplazar por el mundo virtual. La movilidad de estos no estará limitada o guida por la aplicación. Un vehículo podrá desplazarse por todo el escenario sin restricciones. El usuario deberá ser consciente de la ruta que debe elegir para acceder a los diferentes lugares del escenario y moverá los recursos de manera lógica y adecuada por la ruta correcta.

La creación de una aplicación distribuida aunque pudiera ser apropiada para los roles alumno e instructor, no es un requerimiento de este proyecto. Para facilitar la implementación del simulador, los dos usuarios (alumno e instructor) actuarán bajo la misma instancia de la aplicación.

5 INTERFAZ DE USUARIO

En este capítulo se describe la interfaz del simulador, incluyendo los paneles y las herramientas de los que está compuesto. La construcción se ha llevado a cabo teniendo en cuenta las siguientes directrices como objetivo:

- Simplicidad.
- Facilidad de uso.
- Modificable por el usuario.
- Dinámico.

La arquitectura de los elementos aquí presentados se explicará más adelante en el capítulo 8.

5.1 Descripción

La interfaz parte de una configuración clásica utilizada en la mayoría de aplicaciones. Se compone de una **barra de menús** (menu bar), de **barras de herramientas** (toolbars) y de **paneles** (docks) que complementan al simulador.

El diseño se centra en mantener la ventana del mundo virtual como *widget* central, enfocando la atención del usuario en el simulador. A los lados de la misma se colocan los paneles que interactúan con la simulación en curso. Estos paneles o *docks* son movibles a ambos lados de la interfaz, pudiendo el usuario elegir la colocación de los mismos a su gusto.

Los paneles se pueden desactivar o activar desde la sección *Tools* situada en la barra de menú. La Figura 30 muestra una captura de la interfaz, cuando la simulación aún no ha sido iniciada.

El instructor accederá únicamente al panel *instructor* situado en la parte derecha de la aplicación, donde podrá cambiar la dirección del viento y otros parámetros de uso exclusivo para el *instructor*.

Ocasionalmente el instructor puede hacer avanzar la propagación del fuego o pedírselo al alumno, mientras evalúa sus acciones.

El resto de la aplicación está dirigido al alumno, quien deberá utilizar los paneles de la parte izquierda para gestionar los recursos y extinguir el fuego.

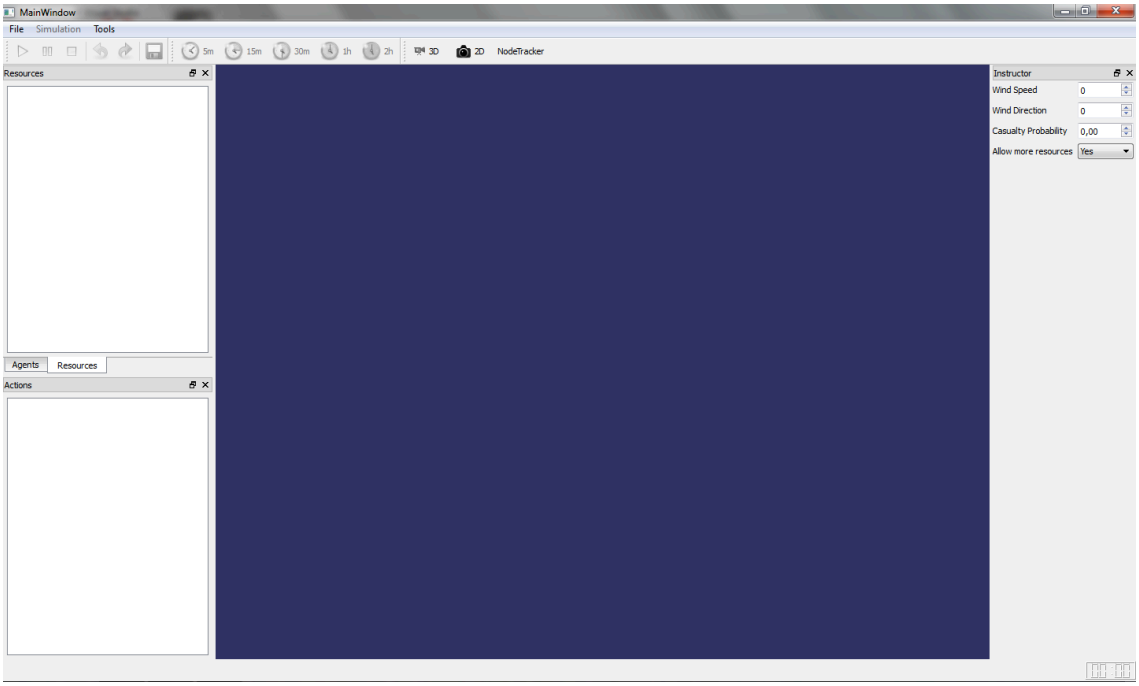


Figura 30 - Interfaz del simulador

El alumno procederá a cargar el escenario indicado por el instructor. Tras la carga del escenario a través de la barra de menú, se obtendrá un resultado similar al mostrado en la Figura 31 dependiendo de la localidad y los recursos que se hayan definido para dicho escenario.

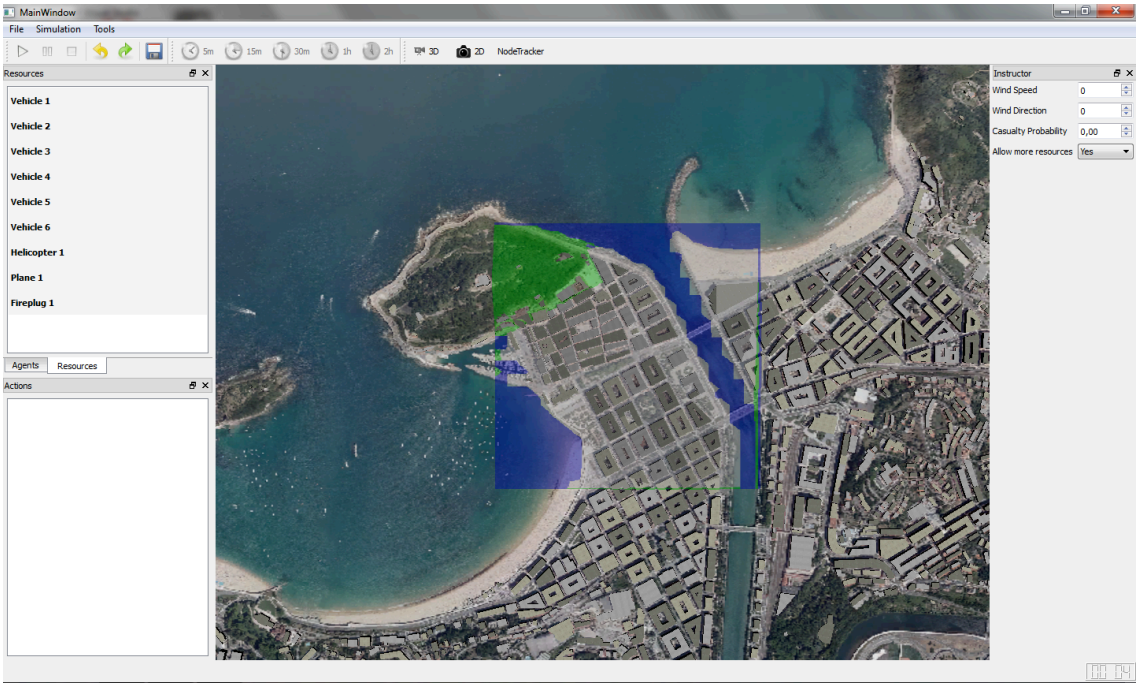


Figura 31 - Interfaz del simulador con el escenario cargado

5.2 Menu bar

La barra de menú se compone de tres secciones:

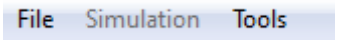


Figura 32 - Menu bar del simulador

- *File*. Nos permite cargar el escenario, seleccionando un archivo XML.
- *Simulation*. Contiene las mismas opciones que la barra de herramientas que controla el avance del simulador
- *Tools*. Permite activar o desactivar las diferentes barras de herramientas y paneles disponibles.

Desde la sección *File* de la barra de menú tenemos la opción *load simulation* que nos muestra una ventana emergente similar a la Figura 33. Esta ventana está configurada para que sólo se permita seleccionar ficheros del tipo *xml*. No obstante, cualquier fichero no es válido, sólo permitirá leer ficheros que contengan un escenario. Los escenarios se detallarán en el próximo capítulo.

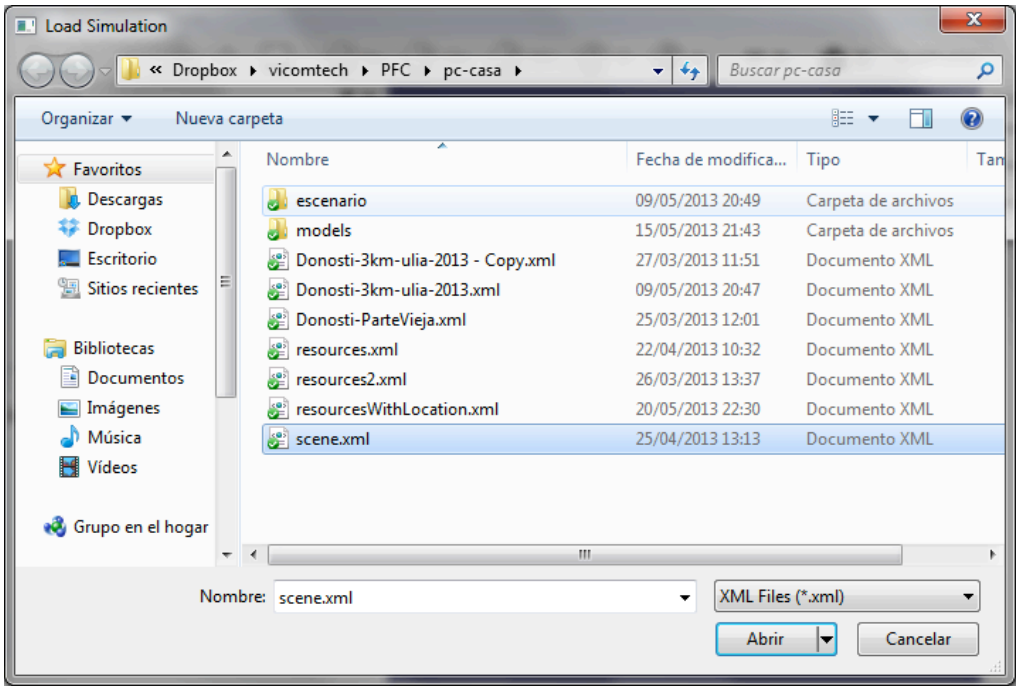


Figura 33 - Interfaz para la selección del escenario

5.3 Toolbars

Se han creado tres barras de herramientas. Por defecto, se han alineado en una misma barra horizontal, aunque si el usuario lo estima oportuno pueden separarse y alinearse para que resulte más cómoda su utilización. Cada opción de la barra de herramientas tiene un icono asignado y una descripción asociada para facilitar su comprensión.

Todos los *toolbars* están deshabilitadas por defecto, hasta que se carga el escenario que va a ser simulado.

5.3.1 Media toolbar

La barra de reproducción, permite, entre otras cosas, avanzar, pausar o parar la simulación. Se puede observar en la Figura 34, la representación de los iconos escogida, siguiendo con el esquema tradicional de los controles de reproducción de cualquier reproductor multimedia. Estos iconos facilitan la experiencia del usuario debido a que no necesitan instrucciones adicionales para entender su función.

Figura 34 - Barra de herramientas para el control de la simulación

Las opciones de reproducción de la simulación también se habilitan y deshabilitan automáticamente en función del estado de la simulación. Cuando la simulación está parada solo el botón *play* está habilitado, mientras que cuando la simulación está en curso, el botón *play* se deshabilita y los botones *pause* y *stop* se habilitan.

Cabe destacar que el botón *pause* y *stop*, no hacen la misma función, al igual que ocurre en el mundo multimedia. Cuando pulsamos *pause* la simulación de la propagación del incendio se detiene parando el tiempo en la simulación, mientras que cuando pulsamos el botón *stop*, se finaliza la simulación, dejándola lista para su reinicio. Al reiniciar la simulación, el estado de simulador vuelve al punto de inicio cuando el incendio todavía no ha empezado a propagarse.

5.3.2 Advance Time toolbar

La barra *advance time* permite controlar el avance de la propagación del fuego avanzando el tiempo de la simulación. En un momento dado se puede avanzar el tiempo desde cinco minutos hasta dos horas. Estas opciones requieren de tiempo para calcular el avance total de la propagación. Durante este periodo, las opciones de simulación quedan temporalmente deshabilitadas para el usuario. Además se mostrará una barra de progreso que indica al usuario el porcentaje calculado hasta el momento.

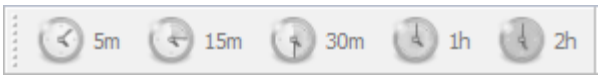


Figura 35 - Barra de herramientas para la gestión del tiempo

5.3.3 Camera toolbar

La barra de las cámaras nos permite seleccionar el tipo de cámara deseado en el simulador. Durante la simulación, se puede variar entre una cámara u otra.

Figura 36 - Barra de herramientas para la cámara

La primera cámara consiste en una cámara 3D, con la que podemos visualizar y navegar a través del mundo virtual con una perspectiva tridimensional. Después de seleccionar

esta cámara podemos interactuar con ella a través del ratón, haciendo clic sobre el *widget* central.

Con el botón izquierdo del ratón podemos rotar el escenario sobre la cámara. Los movimientos del ratón se mapean hacia una rotación orbital, donde la cámara está situado en el centro de la órbita. Cuando el terreno queda perpendicular respecto a la dirección a la que apunta la cámara, el movimiento se bloquea. De forma que la experiencia de usuario sea lo más cómoda posible y no se realicen giros completos que acaben mareando al usuario.

Con el botón derecho y los movimientos del ratón podemos trasladar la posición de la cámara a la dirección en la que señala.

Figura 37 - Acercamiento de la cámara 3D

La segunda cámara consiste en una visualización del escenario en 2D, utilizando siempre una vista alzada. La finalidad de esta cámara es permitir al usuario desplazarse por el escenario rápidamente, abarcando mayor distancia hasta poder llegar a la zona deseada, donde cambiará el tipo de cámara a 3D.

Los movimientos de la cámara se realizan del mismo modo que con el tipo de cámara 3D, a través de la interacción con el ratón en el *widget* central. Por defecto, al inicio de la simulación, la cámara está situada a una distancia en función del tamaño del escenario. Con el botón derecho del ratón y el desplazamiento hacia adelante y hacia atrás del mismo se puede acercar la cámara hacia el terreno o alejarla. El movimiento de la cámara en este caso se realiza modificando su posición perpendicularmente respecto al terreno.

Con el botón izquierdo y los movimientos del ratón podemos trasladar la posición de la cámara en paralelo con el terreno. Al trasladar la cámara mirando perpendicular al terreno, mantenemos la vista alzada del terreno o lo que es lo mismo, una vista bidimensional.

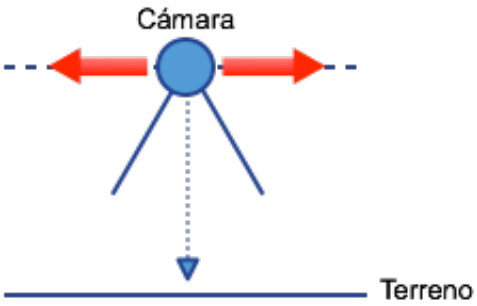


Figura 38 - Desplazamiento de la cámara 2D (perpendicular al terreno)

5.4 Docks

Los docks son paneles situados en los laterales de la aplicación, que son dinámicos o estáticos en función de la necesidad de la aplicación. Permiten interactuar con los recursos de la simulación, seleccionándolos o realizando acciones sobre ellos.

5.4.1 Instructor dock

El instructor dock es un panel enfocado al instructor. El objetivo de este panel es interactuar con el algoritmo de propagación y simulación de incendios. Influye en la simulación alterando variables que repercuten en los resultados, dificultando o facilitando las condiciones de la simulación al alumno.

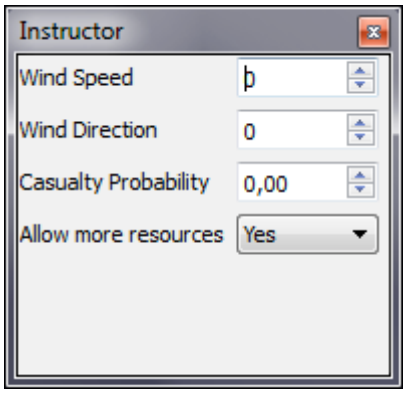


Figura 39 - Panel del instructor

Con la velocidad del viento regulamos la fuerza y con la dirección limitamos el sentido hacia el que se propaga el fuego. La dirección del viento se indica en grados, desde 0º hasta 359º. 0º grados indica una dirección hacia el Este, mientras que 90º grados señalan al Norte. Cambiar estos valores en ciertos momentos, puede dificultar enormemente la extinción del incendio con éxito.

La opción de probabilidad de víctimas no ha sido implementada en esta versión y es un concepto que se presentará como futura funcionalidad.

5.4.2 Resources dock

Resources dock muestra la lista de los recursos disponibles en la simulación, que se listan tras cargar el escenario.

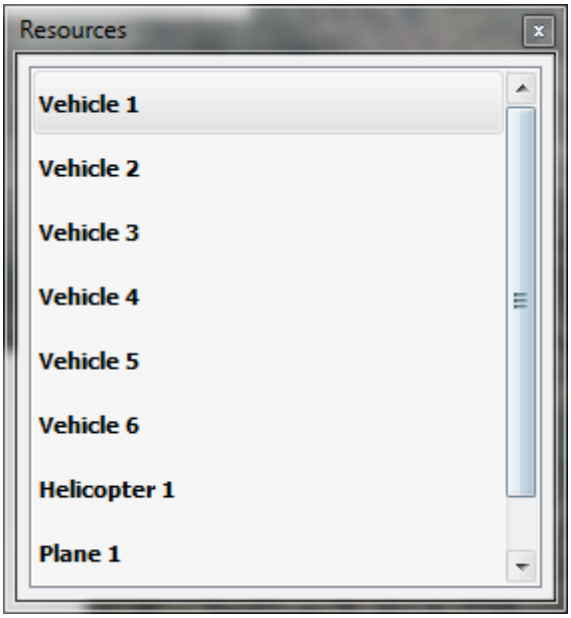


Figura 40 - Panel de los recursos

La Figura 40, muestra el ejemplo de los recursos disponibles en una simulación. El nombre de los recursos es configurable desde el archivo que define los recursos. Cuando se selecciona un recurso de la lista, éste es seleccionado en el mundo virtual y viceversa.

En el mundo virtual, cuando un recurso es seleccionado se dibuja una caja rectangular translúcida de color amarillo que engloba al objeto. De esta manera, se puede identificar fácilmente cuál es el objeto activo. Dicha caja corresponde al *Bounding Box* del objeto. En la Figura 41, se puede ver un ejemplo de un vehículo seleccionado.

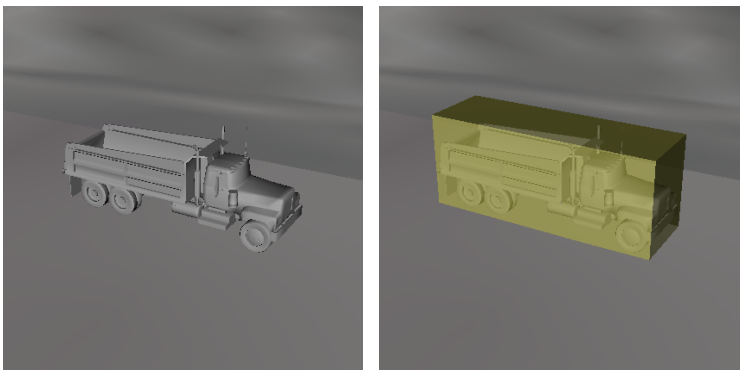


Figura 41 - Selección de un recurso

El alumno debe seleccionar primero un objeto para poder realizar una acción con él. Las acciones están asociadas al objeto y se mostraran en el panel de acciones.

5.4.3 Agents dock

El dock de los agentes, lista los agentes asociados a un recurso. En la Figura 42, se muestra un ejemplo de los agentes asociados a un vehículo.

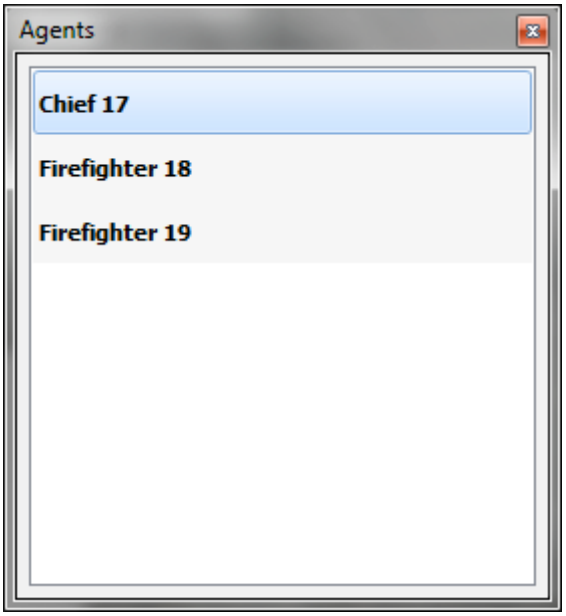


Figura 42 - Panel de los agentes

Al igual que en el anterior dock los nombres de los agentes se definen en el fichero que define los recursos. Los agentes asociados sólo se listan cuando estos pueden ser seleccionados, es decir, cuando previamente el usuario ha decidido desplegar los agentes de un vehículo.

Al seleccionar un agente en el mundo virtual, se autoseleccionan los correspondientes nombres, tanto en el panel de los agentes como en el de los recursos.

5.4.4 Output dock

El output dock está situado a lo ancho de la zona inferior de la aplicación. La finalidad de este panel es la de mostrar mensajes informativos, advertencias y mensajes que ayuden en el proceso de debugging de la aplicación.

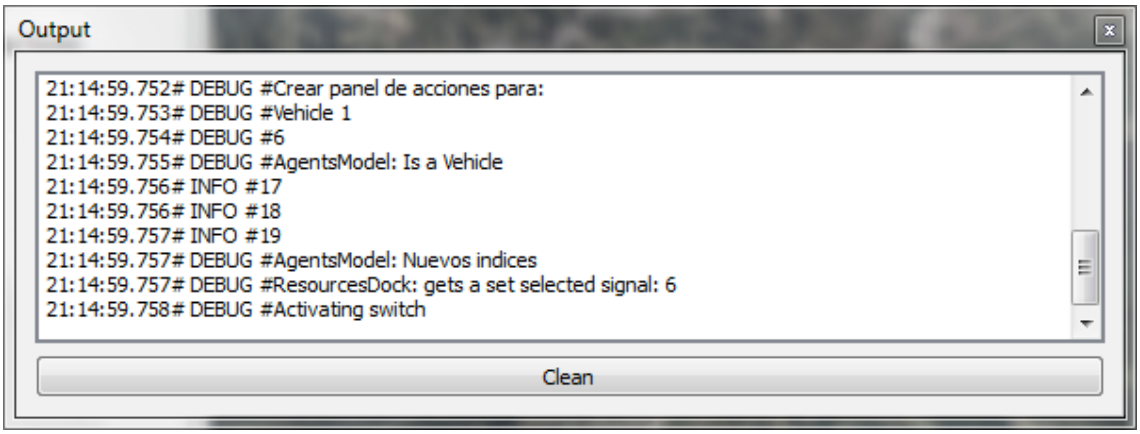


Figura 43 - Panel de salida para los mensajes

La Figura 43, muestra la interfaz del output dock que contiene un botón que permite borrar los mensajes mostrados hasta el momento.

5.4.5 Actions dock

El actions dock es un panel que contiene las acciones disponibles asociados al recurso seleccionado. Cada vez que se selecciona un recurso, el panel cambia en función del recurso y de su estado. Durante la simulación, dos recursos del mismo tipo pueden estar realizando acciones diferentes y además algunas acciones desencadenan otras. Por ello, este panel es actualizado cada vez que se selecciona un nuevo recurso y por lo tanto, es un panel totalmente dinámico.

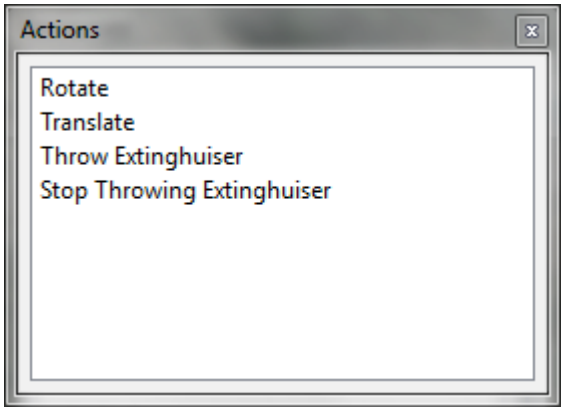


Figura 44 - Panel de las acciones

La Figura 44 muestra un ejemplo de las acciones disponibles de un agente.

La interfaz no sólo se limita a los botones y paneles, sino que también ofrece interacción en el mundo virtual. Al igual que se ha detallado previamente cómo el posicionamiento y el giro de la cámara se hace a través del mundo virtual, algunas de las acciones pueden desencadenar interfaces interactivas que aparecen en el mundo virtual. Estas interfaces están asociadas al objeto del que ha derivado la acción y permiten rotar y posicionar los recursos.

5.5 Status bar

La barra de estado, situado en la parte inferior de la aplicación contiene dos elementos. A la derecha de la barra se encuentra el reloj de la simulación indicando el tiempo transcurrido en el formato hh:mm (h: horas y m: minutos). A la izquierda se encuentra una barra de progreso, oculta por defecto. La barra se muestra sólo cuando el usuario decide avanzar el tiempo de la simulación, mostrando el progreso del cálculo realizado. En la Figura 45, se muestra la barra de estado, con la barra de progreso activa.

Figura 45 - Barra de estado

5.6 Interfaces en el mundo virtual

Al seleccionar ciertas acciones se activan interfaces dentro del mundo virtual. Estas interfaces están asociadas a los objetos seleccionados. En algunos casos, para poder

activar una interfaz se requerirá haber realizado una o una serie de acciones previamente.

En esta versión del simulador, las interfaces que se pueden activar en el mundo virtual son las rotación, la traslación y arrojar agente extintor.

5.6.1 Rotación de un recurso

Una vez seleccionado un vehículo o un agente, se ofrece la posibilidad de rotarlo en el panel de acciones. Cundo seleccionamos la acción *rotate* aparece un anillo interactivo alrededor del objeto en el mundo virtual.

El anillo nos permite rotar el recurso en su eje Z. Para ello, se debe clicar con el ratón sobre el mismo a la vez que desplazamos el ratón hacia la dirección en la que deseamos rotarlo. En la Figura 46, se muestra el anillo que permite realizar la rotación.

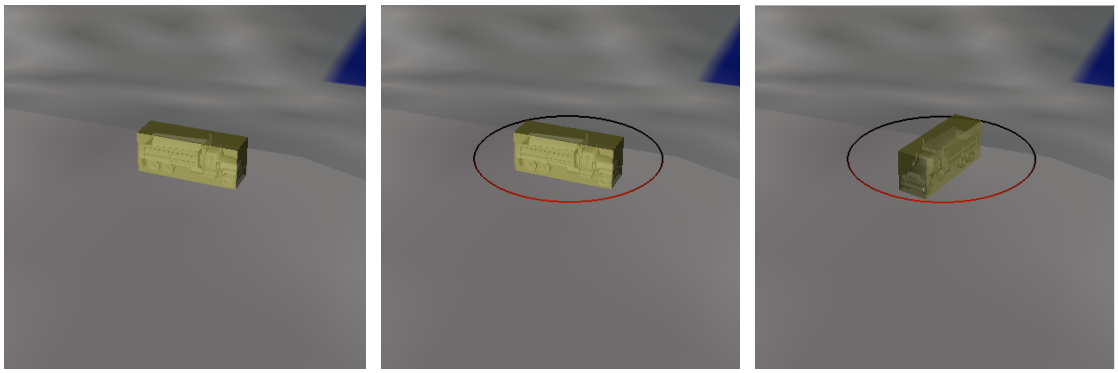


Figura 46 - Rotación de un vehículo

A la izquierda el vehículo en su estado inicial, en el centro el anillo y a la derecha el vehículo tras aplicar la rotación.

5.6.2 Traslación de un recurso

Los agentes se pueden mover por el escenario, para ello hay que seleccionar la opción *translate* en el panel de acciones de un agente. Cuando se activa la interfaz se dibujan dos flechas en forma de cruz. La cruz se activa en la planta del agente simbolizando que el agente puede ser desplazado.

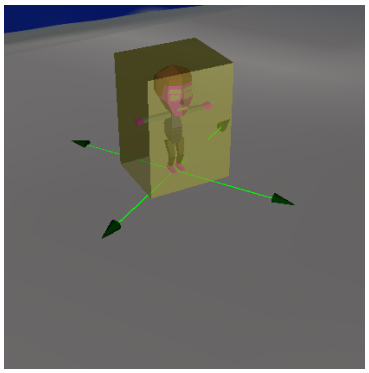


Figura 47 - Interfaz para el desplazamiento de un recurso

Para trasladar el agente sólo hace falta hacer clic con el botón derecho del ratón encima de la cruz y proceder a desplazar el ratón para desplazar el agente. En la Figura 47, se muestra un agente con las cruz activada.

5.6.3 Echar agente extintor

Una vez que las mangueras de un camión de bomberos han sido desplegadas se permite echar agua. La acción *throw water* es accesible para el agente que está situado en el extremo de la manguera.

Cuando las mangueras están desplegadas se dibuja un círculo con dos flechas en forma de cruz para identificar el lugar donde se echará el agua. El agua es suministrada de manera constante. Siendo necesario indicar qué punto queremos rociar con agua. Durante la acción es posible cambiar dicho punto sin parar de echar agua.

Los colores del círculo y la manguera señalan los estados de la misma. Se muestran tres colores:

1. Amarillo: No se está echando agua y hay agua en el depósito del camión.
2. Azul: Se esta echando agua y hay agua en el depósito del camión.
3. Rojo: No se está echando agua y no hay agua en el depósito del camión.

Cada manguera tiene su propio estado, y un camión puede tener más de una manguera en función de la dotación de bomberos que transporta.

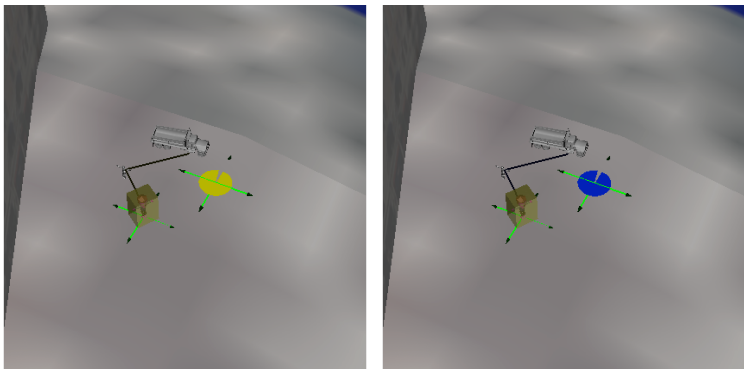


Figura 48 - Manguera desplegada. En espera (izquierda) y echando agua (derecha).

6 DESARROLLO DEL SOFTWARE

En este capítulo se introducen el modelo y sistema de control de versiones utilizados en el proceso de la implementación del simulador. Se divide en tres partes, una descripción del proceso de desarrollo utilizado, el modelo de *branching* empleado con el sistema de control de versiones y una descripción del sistema de incidencias.

6.1 Proceso del desarrollo del software

Cualquier proceso para el desarrollo de software establece una estructura que se aplica al propio desarrollo de un producto. Los procesos describen un modelo a seguir con una serie de actividades que se llevan a cabo en diferentes fases durante el ciclo de vida del proyecto. En la actualidad, hay definidos una gran variedad de procesos, donde cada cual tiene sus pros y sus contras. Ejemplo de ello son desde los clásicos modelos en cascada o espiral, hasta las metodologías ágiles.

Al inicio del proyecto se estableció que durante el desarrollo del mismo se realizarían *demos* periódicas con los avances realizados. Por este motivo, el proceso de desarrollo empleado ha sido iterativo e incremental, inspirado en los conceptos de las metodologías ágiles.

6.1.1 Proceso basado en el desarrollo iterativo e incremental

Del análisis de requerimientos y las especificaciones, se deducen una serie de tareas que se agrupan en funcionalidades, las cuales se añaden al producto en cada iteración. Las funcionalidades (denominadas *features*) se priorizan y se seleccionan al final y al inicio de cada iteración, tras haber enseñado los resultados al cliente.

El cliente es un componente clave del proceso, su participación en cada iteración ayuda a priorizar la lista de funcionalidades o en la alteración de las mismas si es necesario. Esta forma de trabajar ayuda en el desarrollo del producto, haciendo participe al cliente y detectando fallos en momentos que es posible rectificar o deshacer lo andado.

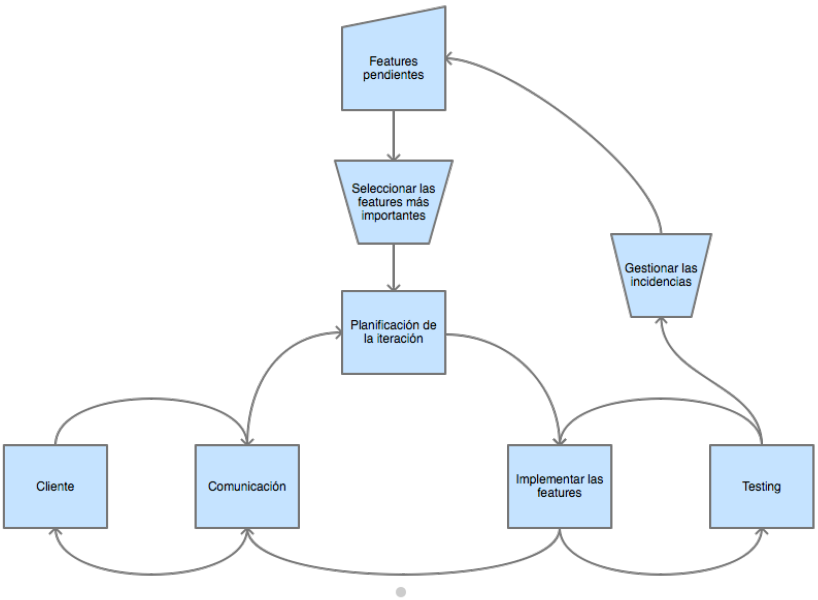


Figura 49 – Diagrama del proceso iterativo

Dado que los procesos software están diseñados para proyectos que se desarrollan en equipo, no se ha seguido ninguna de esas metodologías y se ha optado por definir un proceso propio. La Figura 49, muestra el esquema del proceso utilizado

6.2 Sistema de control de versiones

Un sistema de control de versiones nos permite llevar una gestión de los cambios que dan a lugar en los documentos. En este caso, ayuda a gestionar las diferentes versiones y cambios que se producen en el código implementado (SCM). Aunque mayoritariamente este tipo de sistemas se utilicen para facilitar el trabajo colaborativo, el uso de éstos para gestionar trabajos personales es igual de útil a la vez que recomendado.

6.2.1 Git

De entre las actuales herramientas de control de versiones, se ha optado por *git*, un software de control de versiones diseñado por Linus Torvalds. La principal razón por la elección de está herramienta ha sido la experiencia previa con este sistema.

Un sistema de control de versiones debe ayudar en el proceso de la gestión y versionado. Toda herramienta tiene una curva de aprendizaje previa, cualquier SCM supone un tiempo necesario de aprendizaje que puede dificultarnos las cosas hasta que dominamos dicha herramienta. Por ello, no se ha optado por cualquier otro sistema popular, como puede ser *Subversion o Mercurial*.

Git ofrece grandes ventajas, es un software rápido, eficaz y fiable. Utiliza una gestión distribuida donde cada usuario tiene el historial de desarrollo entero y los cambios en su repositorio local.

Es software open source, se puede encontrar más información acerca de *git* en http://git-scm.com y en https://github.com/git/git.

6.2.2 Bitbucket como repositorio

El repositorio es el lugar donde se almacenan los datos actualizados y el histórico de los cambios. *Git* es un sistema distribuido, pero a su vez podemos optar por utilizar un repositorio para sincronizar los distintos repositorios locales adoptando un esquema de arquitectura centralizado.

Bitbucket es un servicio de hosting para repositorios de sistemas de control de versiones distribuidos (DVCS), concretamente *git* y *mercurial*. El servicio también proporciona alojamiento de una wiki, un *issue tracker* (del cual hablaremos más adelante) e integración con servicios de terceros para ampliar sus funcionalidades.

Atlassian, compañía que proporciona *Bitbucket*, se dedica a proporcionar software como servicio para la gestión y desarrollo de proyectos. Además de *Bitbucket,* existen otras alternativas que alojan repositorios de *git* en la nube, entre otros, *Sourceforge*, *Google Code* o *Github*.

Hay dos motivos por los cuales se ha seleccionado *Bitbucket*; estos son la privacidad y el coste. A diferencia de los demás servicios mencionados *Bitbucket* permite disponer de manera gratuita un número ilimitado de repositorios privados con una cuota de un máximo de cinco personas por repositorio. Los otros servicios como *Github* premian la creación de repositorios Open Source, mientras que para la creación de repositorios privados es de pago.

6.2.3 Modelo de branching

Un modelo de branching no es más que una serie de procedimientos que todo miembro de un equipo de desarrollo tiene que seguir en su proceso de implementación para mantener coherencia y orden en el uso del sistema de control de versiones.

En este proyecto se ha hecho uso de un modelo presentado por Vicent Driessen conocido por el nombre *git-flow,* eso si, adaptado con ciertos matices para este proyecto.

En primer lugar hay que destacar que la configuración de los repositorios que funciona bien con este *workflow*, es la de mantener un repositorio central como punto de sincronización con los repositorios locales. El repositorio central estará alojado en *Bitbucket* y lo denominaremos *origin* por convenio. Debemos aclarar que a nivel técnico, en *git* no existe el concepto de repositorio central, simplemente lo consideramos repositorio central porque se utilizará con ese fin.

Cada vez que se quiera trasmitir los cambios de un repositorio local a los demás habría que subir los cambios al repositorio central para que los demás repositorios locales actualicen su repositorio a través de éste.

Gitflow define una serie de nombres para las ramas que se crean con *git*, diferencia entre las ramas principales y las ramas de soporte. En la Figura 50 obtenida del blog del autor

del modelo (Vicent Driessen, 2010) se puede apreciar el *roadmap* de las ramas que toman parte en *gitflow*.

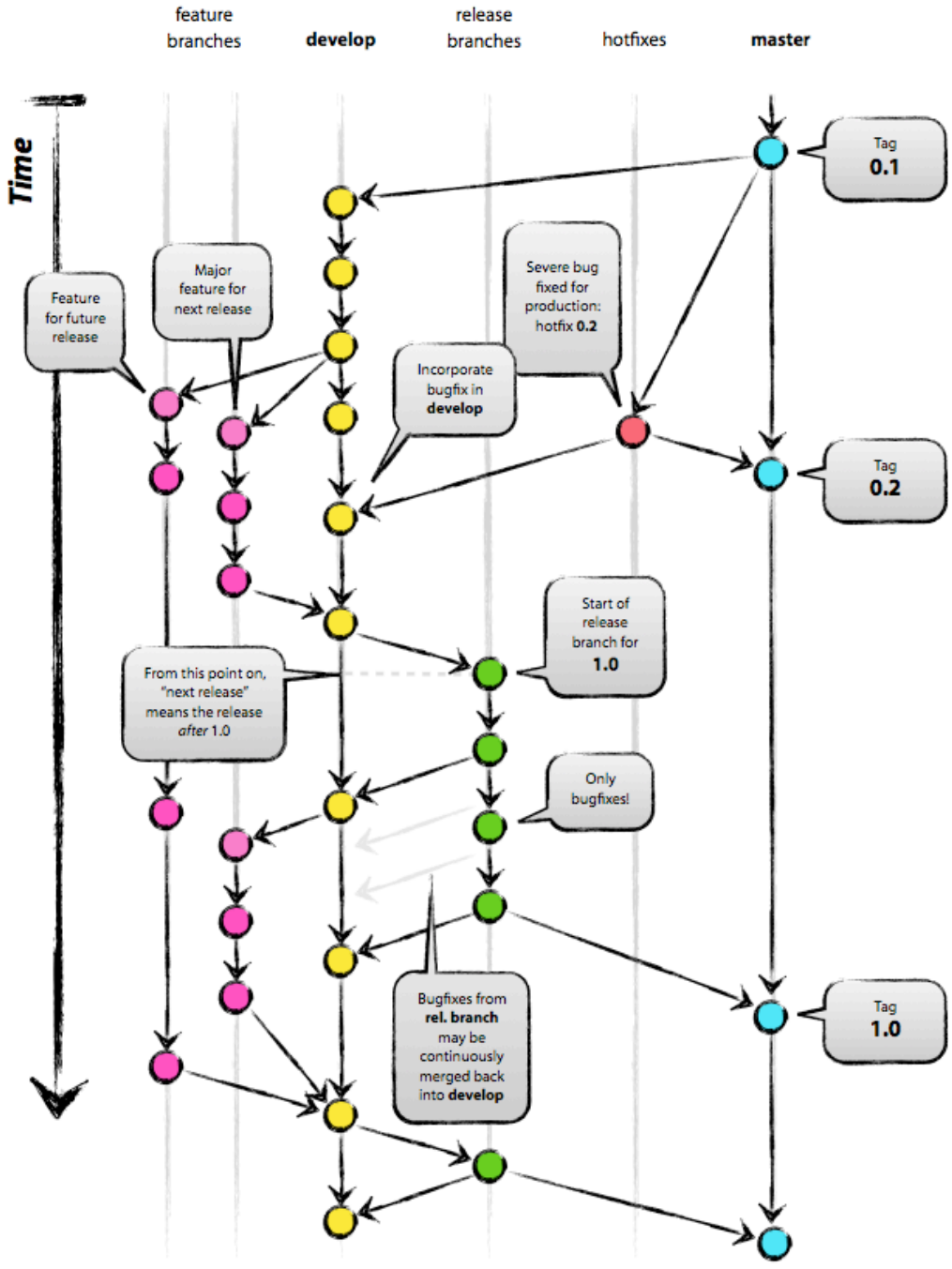


Figura 50 - Gitflow workflow

6.2.3.1 Ramas principales

Las ramas principales del modelo son dos: *master* y *develop*. La rama *master* contiene las diferentes versiones de los *releases* o el código preparado para distribuir al público y que define una versión de la aplicación. En resumen, es la rama que contiene el histórico con las versiones estables del código.

Por el contrario, la rama *develop* es la rama principal de trabajo que contiene el código en fase de desarrollo constante. Todas las demás ramas de soporte parten y se fusionan (*merge*) con esta rama.

Las dos ramas se mantienen durante todo el ciclo de vida del proyecto.

6.2.3.2 Ramas de soporte

Se distinguen tres tipos de ramas, como ramas de soporte:

- *Feature*
- *Hotfix*
- *Release*

Las ramas *features*, representan funcionalidades que se agregan a la rama *develop*, cada *feature* nace de la rama *develop* y una vez completada se fusiona de nuevo en la rama. De esta manera cada funcionalidad se implementa de forma aislada permitiendo desarrollar un conjunto de funcionalidades al mismo tiempo sin que la inestabilidad o problemas que surjan en una de las ramas afecten a otra.

Para cada ciclo del proceso iterativo se integran en *develop* un conjunto de ramas *feature*, las cuales representan el conjunto de funcionalidades seleccionadas al inicio del ciclo. Por tanto, la rama *develop* mantiene el conjunto de funcionalidades estables que no representan una versión completa de la aplicación, que a su vez es la utilizada para las demostraciones.

Una rama de *release* se abre desde *develop* cuando se han completado las funcionalidades que definen una versión completa de la aplicación y se cierra tras haber comprobado que no hay errores o tareas pendientes. Al cerrar la rama, se cierra tanto en *develop* como en *master*, siendo está última la que exclusivamente contiene *commits* que representan el histórico de versiones completas de la aplicación.

La rama *hotfix* se utiliza para corregir errores en alguna versión de la aplicación tras haber sido previamente cerrada e incorporada en la rama master. En este proyecto no se ha hecho uso de la misma, dado que sólo se ha creado una primera versión completa.

6.3 Sistema de seguimiento de incidentes

Un sistema de seguimiento de incidentes (*issue tracker*) nos permite mantener listas con las incidencias encontradas en el desarrollo de software. El servicio *Bitbucket* trae consigo un *issue tracker* sencillo que nos permite administrar estas listas.

No sólo nos permite registrar las incidencias sino que nos ayuda a catalogar cuatro tipos de listas ordenadas por prioridad.

- Bugs: fallos encontrados durante el desarrollo.
- Tareas: actividades pendientes por realizar, pueden no estar relacionadas con el desarrollo.
- Propuestas: una propuesta para trabajo futuro o anotación.

- Mejoras: mejoras que se pueden implantar en el desarrollo.

La Figura 51, muestra una captura de la interfaz del *issue tracker* proporcionado por *Bitbucket* durante el desarrollo del proyecto.

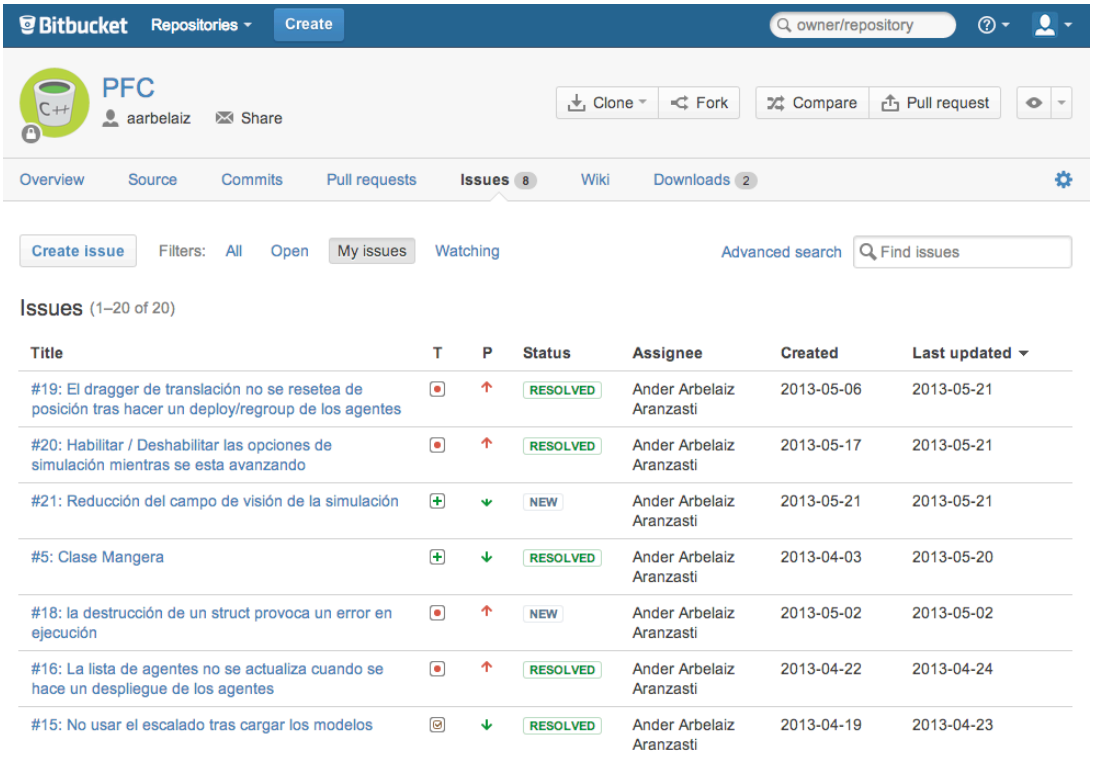


Figura 51 - Captura del *issue tracker* en *Bitbucket*

Una de las ventajas de este sistema de incidencias radica en la integración con el control de versiones, donde desde el mensaje adjunto a un *commit* se puede referenciar a una tarea o incidencia previamente definida, cerrándola, reabriéndola o comentándola.

7 DISEÑO

En este capítulo se introducen los componentes principales que conforman el simulador. Se muestra su diseño y cómo se ha plasmado cada concepto.

7.1 Recursos

Un recurso es una identidad que participa en el proceso de extinción de un incendio, puede ser de diversos tipos: un objeto, como una boca de riego, un vehículo, como un camión de bomberos, una persona, como un bombero.

La utilidad del simulador reside en el aprendizaje de la gestión de los recursos. Por ello, es importante analizar los recursos que toman parte en un entorno real, analizando los recursos clave y extrayendo las acciones que serán necesarias definir dentro de estos.

La bibliografía utilizada para este propósito ha sido el Manual de Extinción de Incendios para los bomberos de Navarra (José Javier et al, 2011), donde se detalla el protocolo de actuación contra incendios.

Hoy en día, existen una gran variedad de recursos y herramientas que se utilizan en el proceso de extinción. Puesto que sería abrumador implementar un simulador que recogiese todos los recursos está fuera del alcance de un Proyecto Fin de Carrera, se ha optado por fijar la atención en los camiones de bomberos, los agentes y las principales acciones de todos ellos. Se ha tenido siempre en cuenta la creación de una estructura que permitiese incorporar nuevos recursos y acciones de manera flexible y escalable desde el inicio de la implementación.

Tras el análisis de los protocolos de actuación, se ha obtenido una serie de conclusiones que se han tenido en cuenta a la hora de implementar el simulador:

- La información que recibe la centralita hace que se disponga de una mayor o menor dotación de salida. Esto es relevante, porque nos dice que la cantidad de recursos iniciales que se disponen en un escenario puede marcar el nivel de dificultad del mismo. Extinguir un incendio mayor de lo esperado con una cantidad de recursos limitada supone una mayor dificultad que teniendo una cantidad de recursos extra.
- Hay que identificar los diferentes focos del incendio ya que puede haber más de uno. Para ello, existe una etapa de evaluación sobre el terreno. Es necesario saber si hay que rescatar personas o animales ya que siempre se le da mayor prioridad que a la extinción del incendio. En el simulador se ha priorizado la implementación de la extinción del fuego, considerándose el rescate de victimas como trabajo futuro.
- La gestión de puntos de abastecimiento del agua es necesaria en incendios de gran envergadura. Los camiones no almacenan una cantidad de agua ilimitada, se tienen que reabastecer con bombas de agua en lagos y piscinas o conectándose a una boca de agua dentro de los entornos urbanos.

- El proceso de extinción se divide en dos fases: contención y extinción. La contención implica la acumulación de agua en zonas vecinas al fuego, algo que en la validación se demuestra que se puede realizar. En el simulador, cuando se echa agua a una zona que no está afectada por el fuego, ésta cambia a un color fucsia, indicando que contiene agua acumulada.
- El número mínimo para sujetar una manguera de incendios es de dos personas. El despliegue de mangueras se contempla en la simulación, situando a uno de los agentes en el extremo de la manguera.

Los recursos principales en los que se centra el simulador son los agentes y los vehículos. En un entorno más real, entran en juego una cantidad de recursos muy extensa con muchas variantes. Por tanto, el diseño y arquitectura de los recursos que se proponen, están basados en dos componentes: la flexibilidad en las características de un recurso y el aumento del factor de escalabilidad. Se quiere permitir que en la arquitectura se puedan añadir más recursos en el futuro, sin tener que reimplementar los recursos o minimizando el coste de implementar nuevos recursos.

Cada recurso del mismo tipo puede diferir uno de otro, es decir, podemos tener diferentes modelos de camión de bomberos en los que varían sus características, tales como la capacidad de agente extintor que puede transportar. Siguiendo el paradigma de programación orientado a objetos, consideraremos un recurso como un objeto. Cada recurso contendrá la información necesaria que lo describa y encapsulará cuatro componentes:

1. El grafo que representa el recurso.
2. El modelo, que define la geometría o el aspecto visual en el mundo virtual.
3. Los estados del objeto.
4. Las acciones asociadas que definen su comportamiento.

7.1.1 Modelo de un recurso

Durante el desarrollo del proyecto, no se ha creado ningún modelo 3D para los recursos. Se han utilizado los modelos de ejemplo que provee la librería gráfica OpenSceneGraph. Estos modelos son simplemente representativos. En un futuro se pueden sustituir por modelos manualmente generados o hechos por terceros que mejorarán el aspecto visual de la simulación.

Los modelos son asociados a un recurso en el momento de su inicialización, es decir, se definen externamente en un fichero y cuando se carga el escenario que contiene los recursos se asigna a cada recurso su modelo correspondiente.

Al abstraer los modelos de la implementación, podemos utilizar modelos externos y cambiarlos por otros sin la necesidad de alterar ninguna línea de código. En OpenSceneGraph el modelo con la información geométrica se establece en un nodo. Ese nodo se añadirá al grafo que compone el recurso.

Se ha utilizado esta estrategia para no condicionar el desarrollo del simulador con un modelo concreto y para facilitar el trabajo de desarrollo del simulador, separando el trabajo de modelado 3D de la implementación de los recursos.

El modelo se construye independientemente del simulador, aunque debe seguir ciertas directrices para ser compatible con la aplicación.

- **Proporción**: El modelo debe estar hecho a escala, manteniendo la proporción real del objeto. Las unidades de los modelos estarán en metros, donde 1 unidad equivale a 1 metro.
- **Sistema de coordenadas**: Por convenio se seguirá el utilizado en la figura ¿? del capítulo 2. La altura del objeto será el eje Z. El objeto estará orientado hacia el eje X, es decir, al (1,0,0) en (X,Y,Z).
- **Punto de referencia del modelo**: El centro del modelo estará situado en el centro de la planta del recurso.

Facilitando la adecuación de un modelo externo para el simulador, se ha construido una pequeña utilidad a partir de el programas *osgconv* de OpenSceneGraph.

7.1.2 Grafo de un recurso

OpenScenGraph (OSG) es una librería gráfica de código abierto para el desarrollo de aplicaciones gráficas interactivas y de alto rendimiento en 3D. Con OSG, una escena se representa mediante un grafo compuesto por nodos entrelazados. OSG se encarga de recorrer el grafo que compone la escena y renderizarla en pantalla usando OpenGL.

Cada objeto que representa un recurso contiene un grafo. Este grafo es un subgrafo de la escena total, es decir, todos los grafos que componen los recursos cuelgan de un nodo que a su vez forma parte del grafo de la escena.

Con los métodos definidos en cada objeto se puede acceder a los distintos nodos que componen el recurso y modificarlo. Al modificar los nodos también se cambia el recurso en el mundo virtual.

No todos los recursos mantienen el mismo grafo. En función de su comportamiento y acciones se enlazan de una manera distinta. Se han clasificado los recursos en tres categorías diferentes: objetos (*ítems*), vehículos (*vehicles*), aeronaves (*aircrafts*).

La Figura 52 muestra el grafo de un camión de bomberos. En el primer nivel se define un nodo del tipo *osg::Group*. Este nodo permite agrupar una serie de nodos y actúa como raíz del recurso. En él se añade el identificador del recurso que es utilizado para facilitar la selección del mismo desde el mundo virtual.

En el segundo nivel se coloca un nodo del tipo *osg::MatrixTransform*. Este nodo contiene la matriz de transformación que se utiliza para poder trasladar el recurso. Este último nodo tiene un hijo del tipo *osg::Switch*. Los nodos *Switch* facilitan activar o desactivar las ramas que cuelgan del mismo. Del primer *Switch* cuelga un segundo nodo del tipo *osg::MatrixTransform* con la matriz de rotación y un nodo del tipo *osg::Dragger*. En este recurso el nodo que contiene la matriz de rotación está siempre

activa, mientras que el *Dragger* es activado y desactivado en función de la acción de rotación.

Cuando el alumno selecciona la rotación del camión de bomberos, se activa la rama desde el *Switch* mostrando el *Dragger* en el mundo virtual. En el capítulo 5 bajo la sección de acciones, en la Figura 46, se puede ver el *Dragger* de rotación, con forma de un aro que rodea al recurso.

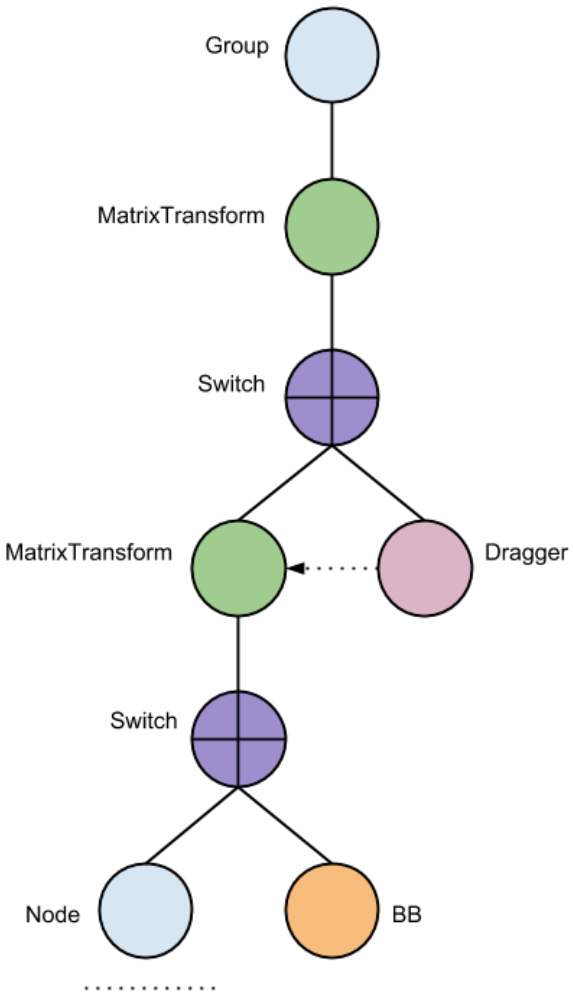


Figura 52 - Grafo del camión de bomberos

Un *Dragger* está compuesto por más de un solo nodo, por simplificación lo englobaremos en uno solo. Cabe destacar que, cuando el usuario rota el *Dragger* en el mundo virtual, éste actualiza el nodo que contiene la matriz de rotación del recurso.

En el último nivel del árbol hay un nodo del tipo *osg::Node* englobando todos los nodos que forman el modelo del recurso, y la caja de selección o *BoundingBox* del recurso. El segundo nodo *osg::Switch* se utiliza para desactivar o activar la caja de selección.

En la Figura 53, se encuentra el grafo que representa una boca de riego. Puesto se trata de un objeto estático su grafo es mucho más simple y reducido. Se compone de un nodo raíz, el modelo, la caja de selección y un *Switch* para activar o desactivar la selección.

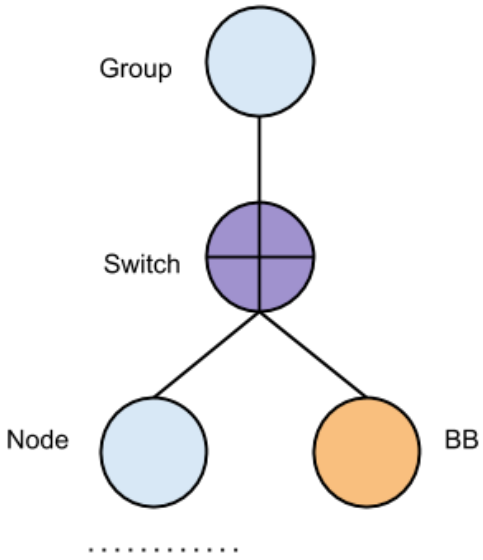


Figura 53 - Grafo de la boca de riego

La Figura 54, muestra el grafo correspondiente a los agentes. En el primer nivel se sitúa el nodo raíz del tipo *osg::Group*. A diferencia del camión de bomberos, el agente contiene tres nodos de *osg::MatrixTransform*, uno para la traslación y otro para la rotación. El nodo intermedio se utiliza en conjunción con la matriz de traslación siendo actualizado por el primero de los *Draggers*. Como un *Dragger* actualiza directamente la matriz de transformación se ha utilizado un nodo intermedio para evitar problemas. Si el *Dragger* actualiza directamente la matriz de traslación de la que cuelga, se produce una traslación incorrecta desplazando el recurso a mayor distancia de lo que debiera debido a su implementación.

El segundo de los *Draggers*, es el de rotación. Ambos, tanto el de traslación como de rotación cuelgan de sus respectivos nodos del tipo *osg::Switch* para poder ser activados y desactivados.

Finalmente, al igual que en los anteriores, en los últimos dos niveles tiene un nodo *osg::Switch* del que cuelgan el modelo del agente y la caja de selección.

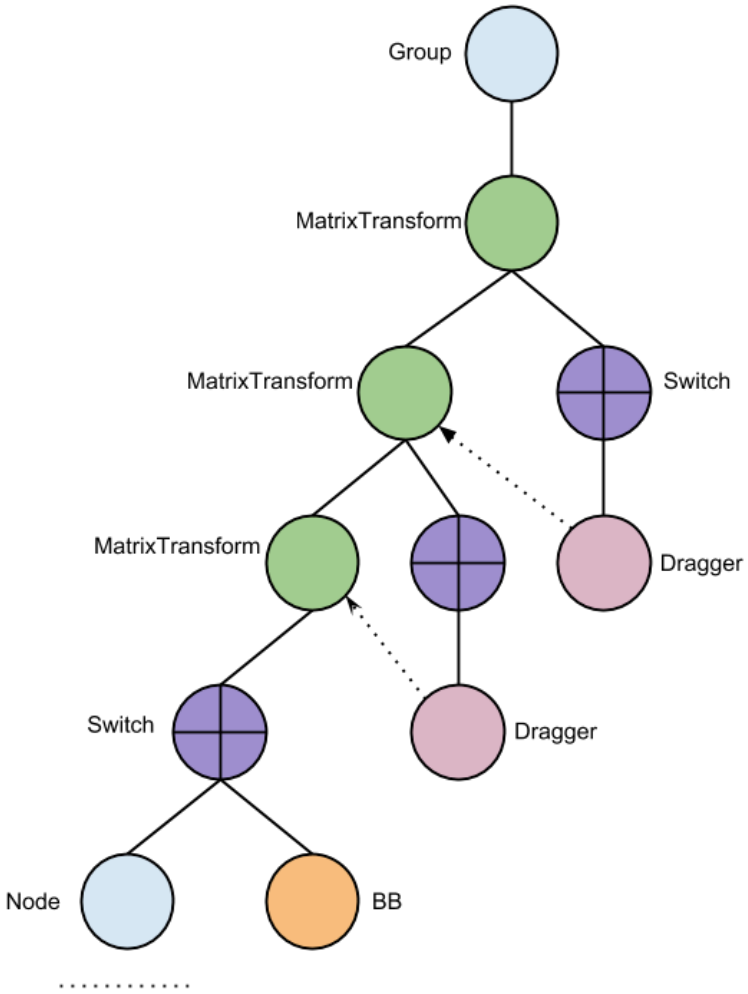


Figura 54 - Grafo de los agentes

La Figura 55 muestra la composición de la caja de selección que recubre cada recurso. Esta caja contiene una matriz de transformación para poder escalar y posicionar correctamente el *BoundingBox*. De su nodo raíz cuelgan a la izquierda un nodo del tipo *osg::Geode* y a la derecha un nodo del tipo *osg::StateSet*. La rama de la izquierda define la geometría del objeto con los vértices y la rama de la derecha el material de la geometría.

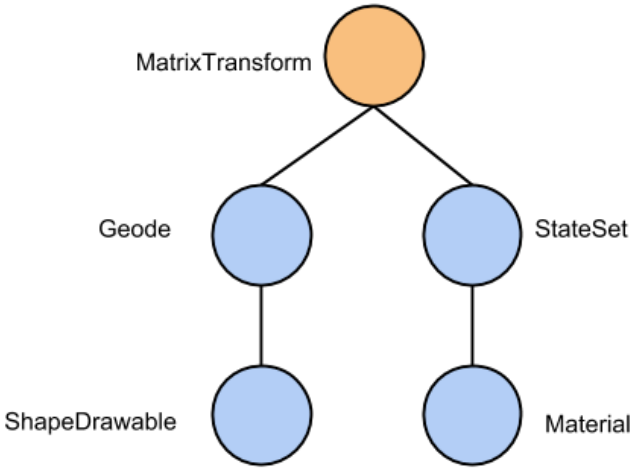


Figura 55 - Grafo de la caja de selección (*Bounding Box*)

7.1.3 Estados de los recursos

En el mundo virtual se simulan diferentes tipos de recursos. Además, puede haber recursos del mismo tipo con diferentes características y a esto hay que sumarle que cada cual tiene su propio estado.

La inicialización de los parámetros de una instancia de un recurso nos permite crear un recurso del mismo tipo con diferentes características. Aun así, los estados del recurso se mantendrán igual para cada tipo de recurso definido.

Cuando se están simulando dos vehículos, podremos desplegar sólo los agentes de uno de ellos mientras mantenemos los agentes del otro vehículo agrupados. Para ello, cada recurso debe mantener sus transiciones de estado independientemente de los demás.

El estado de un recurso está relacionado con las acciones que está realizando o va a poder realizar.

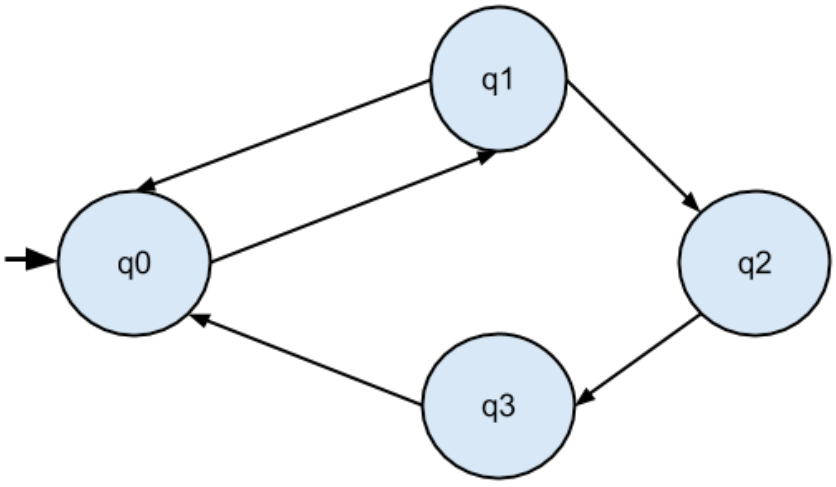


Figura 56 - Estados de los vehículos

La Figura 56, muestra los estados que tienen en común todos los vehículos. A medida que se quieran añadir más acciones a los recursos será necesario modificar el grafo

añadiendo nuevos estados y definir las transacciones entre ellos. Los estados son los siguientes:

- **q0** (*IDLE*). El vehículo está en estado de espera, no se ha realizado ninguna acción duradera sobre el mismo.
- **q1** (*AGENTS_DEPLOYED*). El vehículo tiene sus agentes desplegados. Mientras se encuentre en este estado no se podrá desplazar. Si el usuario decide volver a reagrupar a los agentes, pasa de nuevo al estado *IDLE*.
- **q2** (*HOSE_TO_BE_DEPLOYED*). Este es un estado intermedio. El alumno ha decidido desplegar las mangueras, pero debido al tiempo necesario y para sincronizar los *callbacks* de los acciones, debe pasar temporalmente por este estado.
- **q3** (*HOSE_DEPLOYED*). Estado que indica que las mangueras del vehículo se encuentran desplegadas.

Figura 57 - Estados de los agentes

En la Figura 57, se muestran los estados de los agentes. En la implementación actual solo se consideran dos estados.

- **q0** (*IDLE*). El agente está en estado de espera y su único cambio de estado posible es hacia *q1*.
- **q1** (*THROWING_WATER*). El agente está arrojando agua. Una vez que el alumno decide parar la acción, se vuelve al estado inicial.

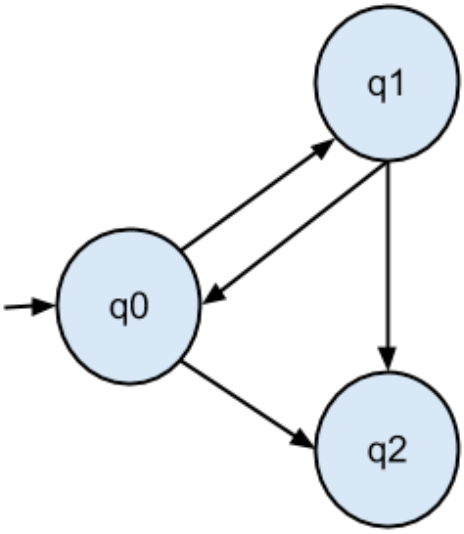


Figura 58 - Estados de la manguera

La Figura 58, muestra los estados de la manguera. Se definen los siguientes estados:

- **q0** (*IDLE*). En estado de espera, cuando las mangueras se acaban de desplegar del vehículo. En este estado, el lugar donde se echará agente extintor (*target*) se encuentra de color amarillo.
- **q1** (*THROWING_WATER*). Cuando la manguera está echando agente extintor, se encuentra en este estado donde el *target* se pondrá de color azul. El usuario puede decidir dejar de echar agente extintor, en ese caso se volvería al estado inicial. Si el vehículo se queda sin agente extintor, la manguera pasaría al estado *WITHOUT_WATER*.
- **q2** (*WITHOUT_WATER*). En este estado el target está de color rojo, indicando que el vehículo no contiene agente extintor.

Los estados de los recursos son necesarios para poder encadenar las acciones de más de un recurso. Teniendo en mente los estados anteriores tomaremos como ejemplo el proceso de extinción.

El alumno seleccionará el vehículo en primer lugar, bien por el panel de recursos o bien directamente en el mundo virtual. A continuación, desplegará los agentes para que el vehículo cambie al estado *AGENTS_DEPLOYED* permitiendo posteriormente desplegar las mangueras.

Una vez que los agentes están desplegados, el alumno podrá iniciar la acción de despliegue de mangueras. El vehículo pasará al estado *HOSE_TO_BE_DEPLOYED* indicativo de que el despliegue está en proceso. Cuando la acción se hace efectiva pasado el tiempo necesario para el despliegue, automáticamente el vehículo pasará al estado *HOSE_DEPLOYED*.

El estado actual impedirá que los agentes se puedan reagrupar en el vehículo si no han recogido las mangueras con anterioridad, es decir, si el vehículo no está en el estado *IDLE*. Las mangueras están desplegadas pero, a la espera de poder echar agente extintor (*IDLE*).

Conjuntamente con el agente situado en el extremo de la manguera, se puede iniciar el proceso de echar agente extintor. Entre las acciones del agente, el alumno podrá interactuar con la manguera a través de la acción *"throw extinguisher agent"* que cambiará tanto el estado de la manguera como el del agente a *THROWING_WATER*. Cuando la manguera detecte que no hay agente extintor, en el vehículo cambiará su estado a *WITHOUT_WATER* y el del agente a *IDLE*.

7.1.4 Las acciones de los recursos

Las acciones asociadas a un recurso están preestablecidas en el código. A diferencia del modelo y de sus características las acciones no se definen en ningún fichero externo o de configuración.

Se han implementado acciones para los siguientes recursos: el vehículo, el agente y la manguera.

7.1.4.1 Camión de bomberos

El camión de bomberos (*fire truck*), es un recurso que deriva de la categoría de los vehículos. Las acciones que se han implementado para el *fire truck* son las siguientes:

- Rotación.
- Desplegar y reagrupar mangueras.
- Desplegar y reagrupar agentes.

7.1.4.2 Bomberos

El bombero es un recurso que deriva del grupo de los agentes. Las acciones implementadas para este agente, son las siguientes:

- Rotación.
- Traslación.
- Echar y dejar de echar agente extintor.

7.1.5 Definición de los recursos para la simulación

Como se ha citado anteriormente, las características de los recursos se definen externamente en un fichero XML que se carga a la par con el terreno.

El número de recursos definidos en el fichero limita el número de recursos disponibles en el escenario para el alumno. Los modelos de cada recurso están definidos en formatos compatibles con OSG y siguiendo las normas de compatibilidad descritas anteriormente en este capítulo. El fichero enlaza cada recurso con la ruta que contiene su modelo y con las características definidas.

Para facilitar la escritura de la ruta en cada modelo, se ha creado una etiqueta "***<models_root>***" en la que se puede especificar una ruta raíz y de este modo en cada recurso se especificará una ruta relativa a aquél.

```
<data>
        <models_root>/Users/ander/models/</models_root>
</data>
```

Cada recurso que se define debe llevar un identificador único y las asociaciones entre los recursos se definen anidando uno dentro de otro. A continuación, se muestra un ejemplo de la definición de un camión de bomberos donde tres agentes están asociados al mismo.

```
<vehicle>
        <firetruck id="6">
            <agents>
                <agent>
                    <chieffireman id="17">
```

```
                <type>A</type>
                <equip>
                  <radio>true</radio>
                </equip>
                <r_name>Chief 17</r_name>
                <model>nathan2.osg</model>
            </chieffireman>
        </agent>
        <agent>
            <firefighter id="18">
                <type>B</type>
                <r_name>Firefighter 18</r_name>
                <model>robot2.osg</model>
            </firefighter>
        </agent>
        <agent>
            <firefighter id="19">
                <type>C</type>
                <r_name>Firefighter 19</r_name>
                <model>robot2.osg</model>
            </firefighter>
        </agent>
    </agents>
    <deposit>20000</deposit>
    <extinghuiseragent>
        <name>Water</name>
        <quantity>30</quantity>
        <factor>1</factor>
        <radio>3</radio>
        <volatibility>0.5</volatibility>
        <powerJet>20</powerJet>
    </extinghuiseragent>
```

```
            <location>
                <x>582542</x>
                <y>4797391</y>
            </location>
            <active>true</active>
            <model>dumptruck2.osg</model>
            <r_name>Vehicle 1</r_name>
        </firetruck>
</vehicle>
```

Como se puede apreciar las características del camión son configurables. Se pueden modificar las siguientes:

- Depósito. La cantidad de agente extintor máxima que puede almacenar el vehículo.
- Localización. Especificando el punto inicial del escenario en el que se cargará el recurso.
- Características del agente extintor y efectividad.
  - Nombre. Nombre del tipo de agente extintor.
  - Cantidad. Cantidad de agente extintor en el deposito.
  - Factor. Factor de efectividad del agente extintor.
  - Radio. Radio de alcance en el punto objetivo.
  - Volatilidad. Factor de evaporación del agente extintor.
  - PowerJet. Fuerza con la que se arroja el agente extintor en edificios.
- Activo. Indicando si el recurso esta oculto o disponible.
- Modelo. Ruta al fichero del modelo 3D.
- Nombre. Nombre del recurso.

Las etiquetas *activo*, *modelo*, *nombre* e *identificador* son generales para cualquier tipo de recurso.

7.2 Escenario

Para iniciar el simulador, es fundamental indicar el escenario que se quiere utilizar. El escenario contiene el modelo del terreno, información sobre el mismo y la definición de los recursos.

El modelo del terreno empleado en el desarrollo del simulador mantiene las coordenadas de sus vértices sobre el sistema de coordenadas UTM.

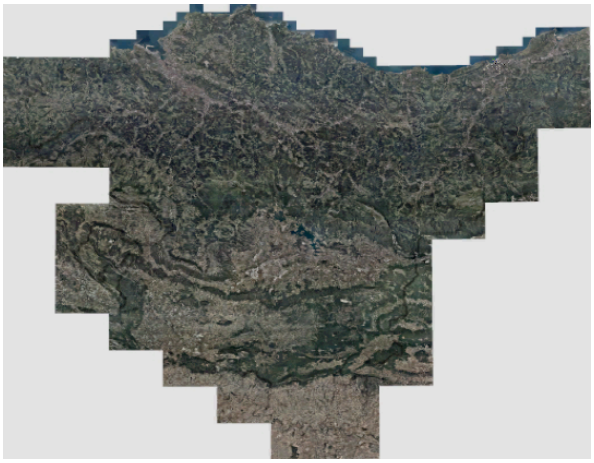


Figura 59 - Modelo obtenido del Framework de los algoritmos (Moreno 2013a)

Durante el desarrollo de la implementación se ha visto necesario obtener la altura del terreno en un punto dado. OSG contiene funciones que nos ayudan en este cálculo. Con la intersección del modelo en un punto del terreno conseguimos la altura y de este modo se sitúan los recursos sobre el terreno en el lugar correcto.

El escenario se compone de tres ficheros XML. El fichero raíz actúa de índice y contiene la localización de los otros dos ficheros. Uno contiene la información relativa al terreno y el otro, define los recursos.

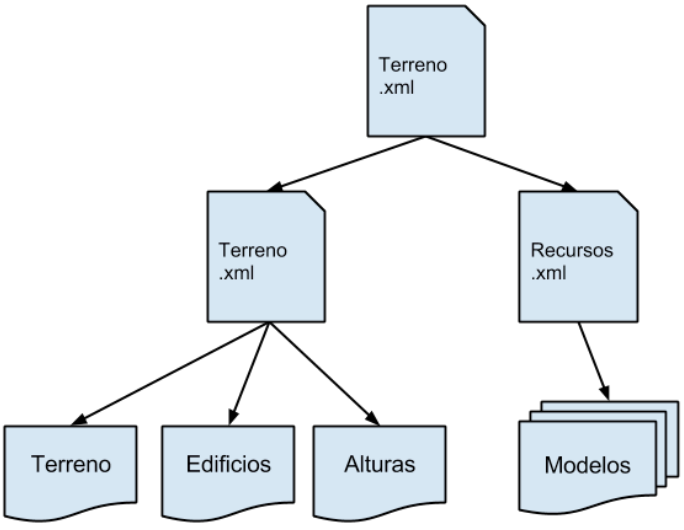


Figura 60 - Ficheros que componen el escenario

El fichero del terreno contiene:

- La localización de modelo del terreno
- La localización del modelo de los edificios.
- Información para la los algoritmos de simulación.

- Delimitación de las barreras.
- Delimitación de las zonas forestales.
- Delimitación de cada edificio.
- Alturas del terreno

El fichero de los recursos define los recursos disponibles y sus características (definido en el apartado anterior).

7.3 Integración de los algoritmos

La integración de los algoritmos de simulación y propagación de incendios en el simulador, es uno de los hitos y una de las razones de la existencia de este proyecto. Tras el análisis y la validación de los algoritmos, el siguiente paso era la integración de los mismos en un simulador. Este proyecto da pie a la adecuación de una librería que podrá ser incluida en otros simuladores.

La integración de la librería en el simulador requiere que el funcionamiento del simulador sea lo más independiente posible de la propagación del fuego. Cualquier interacción con el fuego se hace a través de la API de la librería. Algunas acciones de los recursos como echar agente extintor utilizarán llamadas al API para interactuar con la propagación del fuego.

Para conseguir una independencia entre la lógica del simulador y la librería SuLib, ésta controla una textura que se sobrepone al modelo del terreno en el mundo virtual del simulador.

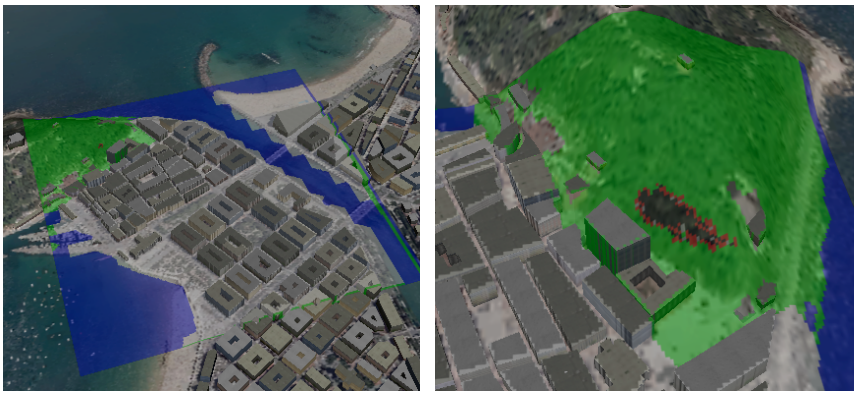


Figura 61 - Textura sobre la Parte Vieja de San Sebastián

Mediante *callbacks* procedentes de la librería se actualiza el estado y los colores de la textura para indicar visualmente el resultado de la simulación del avance del fuego.

El tiempo es gestionado por el simulador y se explicará más en detalle en el próximo apartado. El simulador será el encargado de llamar a la librería periódicamente para que está calcule el avance del fuego. Esto es posible ya que el tiempo del simulador y el paso de integración de los algoritmos del simulador están sincronizados.

7.4 Gestión del tiempo

Uno de los aspectos más importantes en la gestión de los recursos es administrar correctamente el tiempo. Cuando el incendio avanza, los movimientos de los bomberos deben estar bien coordinados, contemplando en todo momento que cada minuto que pase se complicará la extinción del incendio.

Una vez iniciada la simulación, el tiempo avanza en el simulador de manera constante a la vez que el fuego. Consideraremos el tiempo del simulador como una línea temporal que avanza desde el inicio hasta el momento actual de simulación.

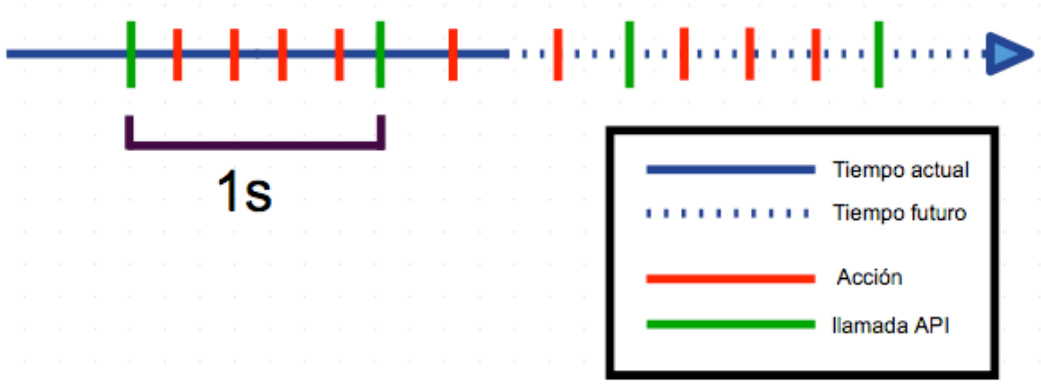


Figura 62 - Acciones y propagación del incendio en la línea de tiempo

El simulador utilizará llamadas a las rutinas del sistema operativo para obtener el tiempo del sistema y de esta forma sincronizar correctamente el reloj del simulador.

La gestión del tiempo implica organizar las acciones. Cada segundo se realizará una llamada para avanzar en la propagación del fuego. Entre estas llamadas periódicas se registrarán y se realizarán las acciones seleccionadas por el alumno.

Las acciones que el usuario realiza son registradas de manera ordenada en el tiempo. Se distinguen tres tipos de acciones:

- **Acción inmediata**. No son gestionadas dentro de la lógica del tiempo del simulador y porque se realizan al momento.
- **Acción duradera**. Estas acciones duran un tiempo determinado, como ejemplo, el despliegue y posicionamiento de las mangueras de un camión de bomberos. El resultado no se mostrará hasta pasado el tiempo predefinido para dicha acción. Estas acciones son siempre gestionadas por el manager del tiempo del simulador.
- **Acción contínua**. Estas acciones que se deben realizar de forma periódica.

La implementación de las acciones duraderas se ha realizado mediante el uso de *callbacks*. El lenguaje de programación empleado para la creación del simulador ha sido *C++*. Este lenguaje permite la creación de *callbacks* de distintas formas. Se ha optado por utilizar la manera más sencilla que es propia del lenguaje *C*, utilizando básicamente un puntero a una función.

Como se sigue el paradigma de programación orientado a objetos (OOP) además de un puntero a la función, se necesita un segundo puntero con la instancia del objeto a la que corresponde la función que será invocada.

Los *callbacks* quedan temporalmente almacenados en una tabla hash donde cada entrada especifica la marca de tiempo en segundos en la que se invocarán los *callbacks* asociados. Por otro lado, el valor asociado se compone de una lista donde se almacenan los *callbacks*.

Cuando el alumno selecciona una acción que requiere de un tiempo para realizarse (por ejemplo, el despliegue de los agentes de un vehículo), se añade a la tabla hash considerando el tiempo actual más el tiempo necesario para la realización del mismo. Para cada instante se comprueba que no haya acciones pendientes, de forma que cuando el tiempo actual alcance el tiempo que indica alguna de las entradas de la tabla hash, se invocarán las funciones asociadas realizando cada acción en el tiempo previsto.

Hay un tipo de *callback* que no se almacena en la tablas *hash*. Estos *callbacks* son continuos, es decir, tienen que ser invocados periódicamente cada segundo. Por facilidad, se almacenan en una lista ordenada por orden de inserción.

En un momento dado se puede querer avanzar la simulación tras un correcto posicionamiento de los recursos o simplemente acelerar la propagación. La Figura 63, muestra el proceso en el avance del tiempo.

Figura 63 - Líneas de tiempo del simulador al avanzar el tiempo

Cuando el alumno clica en el botón para avanzar en el tiempo, el reloj de la simulación se detiene y se crea un nuevo *thread* para proceder al cálculo del nuevo estado de la simulación. De esta manera, el alumno puede girar y apreciar el avance de la simulación sin que la aplicación quede bloqueada debido al tiempo de cálculo necesario.

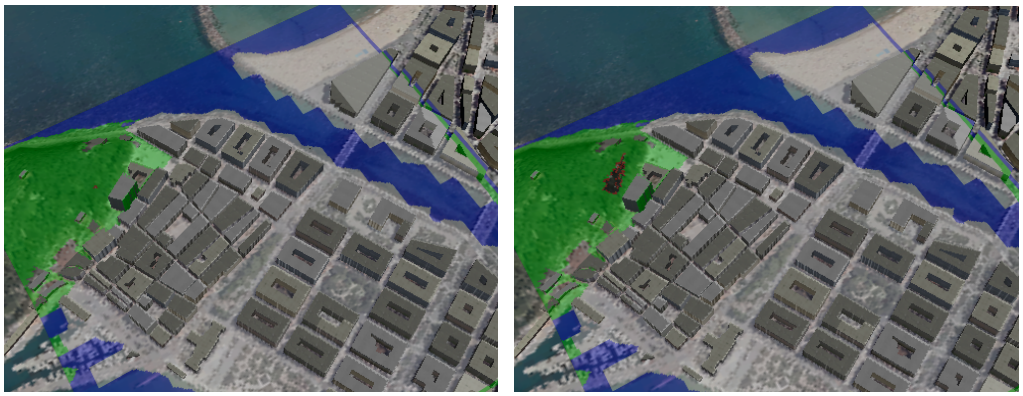


Figura 64 - Resultado de la simulación. Inicio (derecha) y transcurridos 5m (izquierda)

En la Figura 64, se puede ver un ejemplo del avance del fuego tras 5 minutos de simulación. El incendio tiene origen en el monte Urgull de Donostia-San Sebastián.

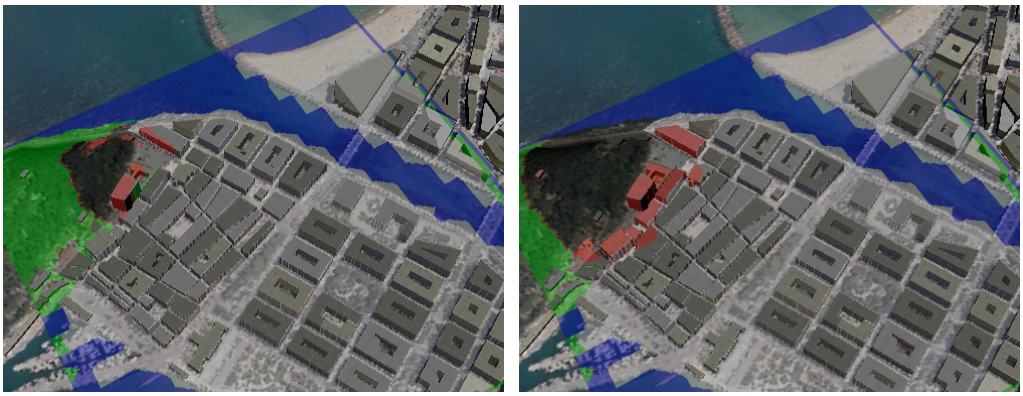


Figura 65 - Resultado de la simulación. 20m (izquierda) y 50m (derecha)

En la Figura 65, Se puede ver el avance del fuego tras 20 minutos a la izquierda y 50 minutos a la derecha.

8 ARQUITECTURA

En este capítulo se introduce la arquitectura del simulador. Se describen los módulos que lo componen, y se muestran tanto los diagramas de clases como los casos de uso que ayudan a comprender la composición del simulador.

8.1 Esquema del simulador

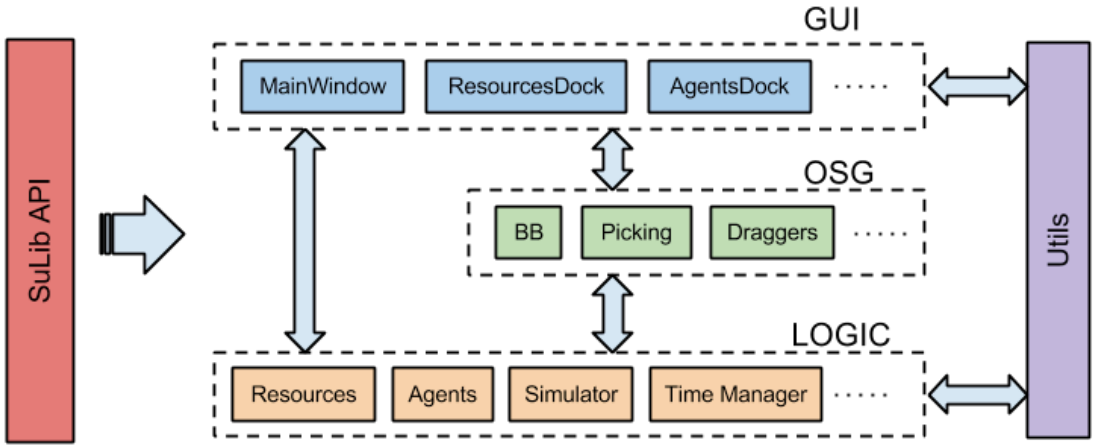


Figura 66 - Esquema del simulador

En la Figura 66, se muestra el esquema del simulador. En primer lugar tenemos la interfaz, el GUI, esta ha sido diseñada e implantada usando el framework *Qt*. En el cuarto capítulo se ha mostrado visualmente todos sus componentes.

La librería gráfica OpensSceneGraph es el segundo componente, siendo el encargado de renderizar la escena. Se ha visto necesario implementar algunas clases para interactuar con el mundo virtual utilizando las funcionalidades que ofrece OSG como el *picking*, los manipuladores de los recursos y la obtención del *Bounding Box* de un nodo.

Los *utils* contienen clases que cargan los modelos a través de ficheros externos paseándolos y transformándolos a objetos manejables por la lógica de la aplicación.

Todos los mensajes de *debug* e información pasan a través de la lógica de la aplicación y son mostrados en la interfaz gráfica.

La API de SuLib es utilizada mayormente a través de la lógica de la aplicación y OSG. En los siguientes sub-apartados se detallan en mayor medida los diferentes componentes.

8.2 Casos de uso

En los casos de uso de la aplicación se distinguen dos roles principales, en primer lugar, el alumno y en segundo lugar el instructor.

El alumno puede inferir en los siguientes casos de uso:

- Cargar el escenario. El alumno selecciona el escenario que simulará la aplicación.
- Controlar la simulación. El alumno puede controlar el estado de la simulación con los siguientes casos de uso:
    - Iniciar la simulación
    - Pausar la simulación
    - Reiniciar la simulación
    - Avanzar la simulación
- Cambiar el tipo de cámara. El alumno puede variar en cualquier momento el tipo de cámara pasando de una cámara 2D a 3D o viceversa.
- Seleccionar Recurso. El alumno pude seleccionar un recurso directamente desde el panel correspondiente o desde el mundo virtual.
- Seleccionar Agente: El alumno puede seleccionar un agente desde el mundo virtual o desde el panel correspondiente.
- Seleccionar Acción: El alumno puede ejecutar una acción sobre un recurso o agente desde el panel de acciones.

El instructor tiene los siguientes casos de uso:

- Avanzar el tiempo de simulación. Cuando el instructor vea conveniente para la prueba y evaluación del alumno avanzar el estado del incendio, avanzará el estado de la simulación.
- Modificar los parámetros de la simulación. Para dificultar o guiar la prueba simulada, el instructor puede modificar el incendio cambiando la dirección del viento.

La Figura 67, muestra los casos de uso del alumno y la Figura 68, las del instructor.

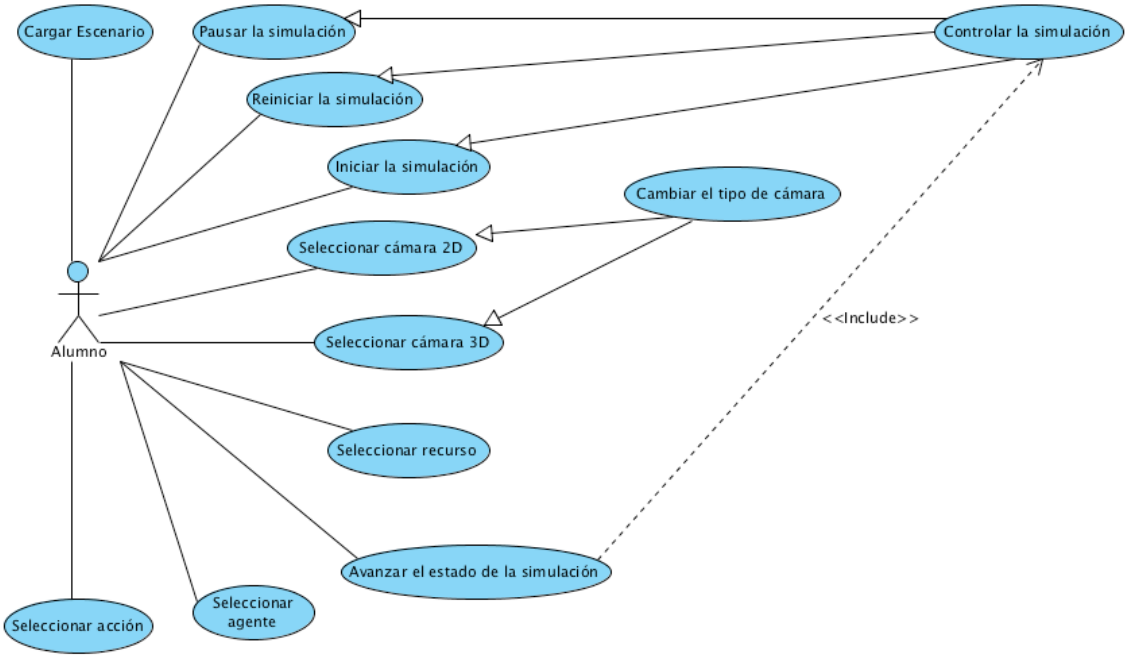


Figura 67 - Casos de uso del alumno

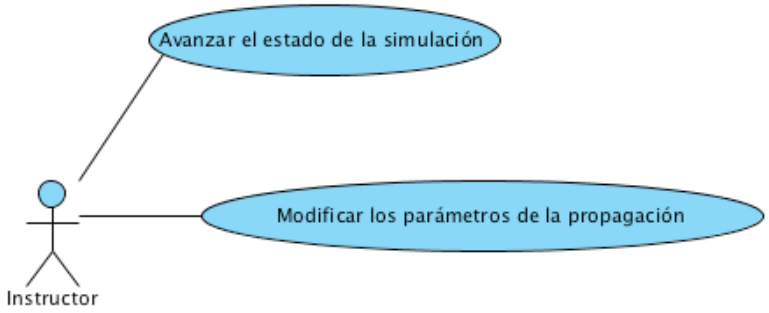


Figura 68 - Casos de uso del instructor

8.3 Diagrama de clases

En las siguientes subsecciones nos centraremos en algunas clases concretas para detallar y comprender mejor la aplicación.

8.3.1 Recursos

Para estructurar los recursos se hace uso del paradigma de programación orientado a objetos y sus ventajas. Se ha decidido separar los agentes de los recursos, aunque por definición se puedan clasificar dentro de la misma categoría. La separación se ha hecho por conveniencia durante la implementación, definiendo por un lado los recursos y por otro los agentes.

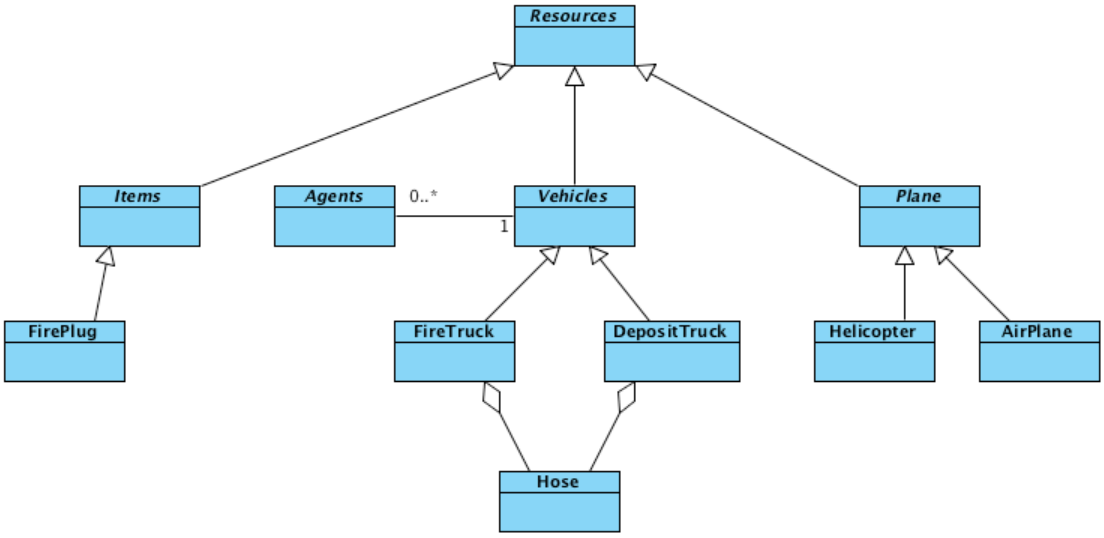


Figura 69 - Diagrama de clases de los recursos

Se han estructurado las clases haciendo uso de la herencia. En la Figura 69, se puede ver el diagrama de clases empleado para los recursos y en la Figura 70, para los agentes. Primero se ha definido una clase abstracta *Resource* de la cuál no se puede obtener una instancia y que se usará para definir los parámetros comunes de todos los recursos. Todos los recursos (exceptuando los agentes) heredarán de dicha clase. De esta forma podremos agrupar los diferentes tipos de recursos en una estructura de datos.

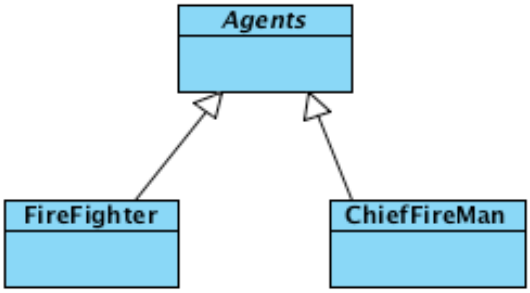


Figura 70 - Diagrama de clases de los agentes

En el segundo nivel, las clases que heredan de *Resource*, son las categorías que permiten diferenciar los tipos de recursos y agruparlos. Estas categorías o grupos son: *Items*, *Vehicles* y *Planes*. Estas clases también son abstractas porque agrupan recursos del mismo tipo aunque con características diferentes, definiendo métodos y datos comunes para estos. Un ejemplo de método común, es la creación de un *D*ragger (método protegido). Cada tipo de recurso contiene un *Dragger* que por norma general es idéntica para todos los de su misma categoría, de modo que no hace falta definirlo en cada uno de las subclases. Además, si esto no se cumpliera porque el *Dragger* para el recurso es especial con respecto a los demás, siempre se podría utilizar un *override* facilitando la implementación de la excepción.

En el tercer y último nivel de herencia, se concretan los objetos que representan un recurso real. Estos objetos no son abstractos, por tanto pueden instanciarse.

Al igual que con la estructura de los recursos, se define una clase abstracta *Agents* que contiene los parámetros y funciones globales para todos los diferentes tipos de agentes que hereden de ésta. Se han distinguido dos clases de agentes *FireMan* y *ChiefFireMan.* El primero representa al bombero mientras que el segundo es el jefe de brigada. Dado que los dos participan en el proceso de extinción del fuego no se ha hecho una distinción a nivel de implementación, aunque cabe destacar que los dos agentes tienen acciones diferenciadas en el mundo real.

8.3.2 Callbacks y estados

Para poder declarar las acciones en los recursos, se utilizan tres *callbacks* por recurso. En las clases abstractas *Resources* y *Agents,* se declaran tres métodos estáticos que se heredarán en todas las subclases de recursos y agentes. Estos métodos consisten en:

- **st_startOfCurrenctAction**. Esta función estática llama a la función interna de un objeto que inicia una acción.
- **st_interruptOfCurrentAction**. Esta función estática llama a una función interna de un recurso interrumpiendo la acción que en ese momento está realizando. Está pensada para parar o cambiar una acción continua.
- **st_endOfCurrenctAction**. Esta función estática llama a una función que detiene la acción en curso del objeto.

Cuando el método estático del objeto es invocado, éste llama al método correcto dentro del objeto. En este proceso se tiene en cuenta el estado del recurso para realizar la acción correcta que le corresponde.

Tras realizar la acción programada mediante la función adecuada del objeto al momento de recibir el *callback*, el recurso procederá a cambiar a un estado en el que el alumno podrá volver a interactuar con el recurso de nuevo. Por norma general, cuando un usuario pulsa una acción duradera sobre un recurso, se queda en espera de recibir el *callback*, denegando al usuario poder realizar una nueva acción sobre el mismo. Por tanto, las acciones de un recurso no pueden encolarse y hay que realizarlas una a una. Los estados de los recursos son los que controlan esta restricción.

En líneas generales, en el diagrama de la Figura 71, se muestra la relación de los *callbacks* de los recursos con el gestor del tiempo.

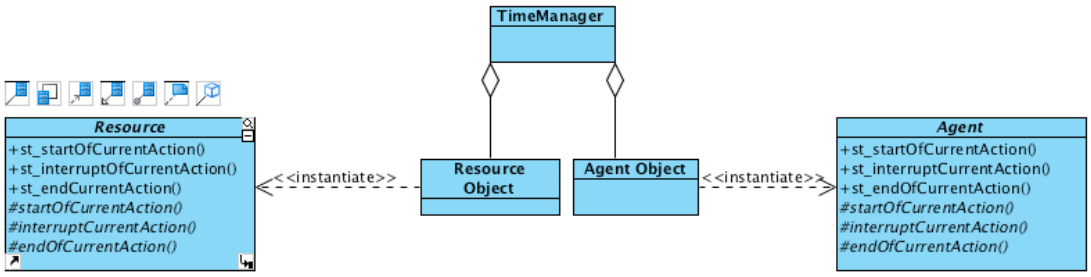


Figura 71 - Diagrama de clases para los *callbacks* de las acciones

El *ResourceObject* y *AgentObject* son clases que representan cualquier instancia de un recurso o agente, como pueden ser *FireTruck, Helicopter, FireMan*...

8.3.3 Utilidades

Las clases de las utilidades son las encargadas de parsear los ficheros externos de la aplicación. Concretamente se encargan de instanciar los recursos según se especifican en los ficheros. Para cada instancia de los recursos creada se inicializan sus parámetros y en caso de no estar definidos en los ficheros, se harán uso de valores por defecto. La Figura 72 muestra las clases que corresponde a las utilidades.

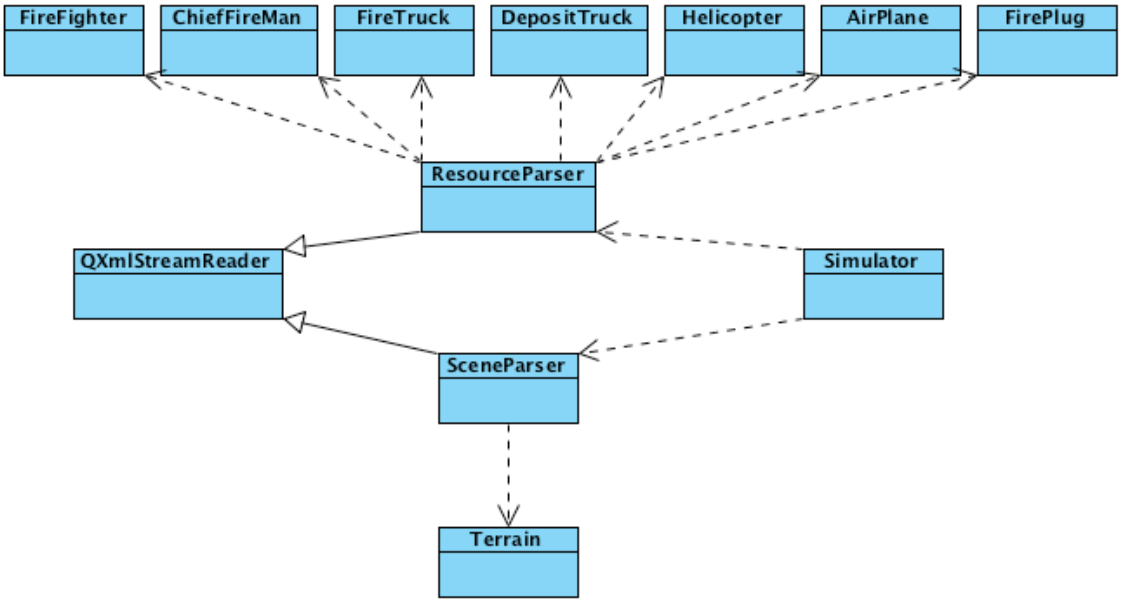


Figura 72 - Diagrama de clases de las utilidades

8.3.4 Gestión del tiempo

El control de la gestión del tiempo se lleva a cabo en una sola clase llamada *TimeManager*. Para esta clase se define un *thread* para el cálculo de la propagación cuando se quiere avanzar en el incendio. Las acciones quedan centralizadas en dos estructuras de datos una tabla hash y una lista que son atributos privados de la clase.

Las señales son eventos que el gestor de tiempo tiene que manejar. Como eventos de entrada (*slots*) se consideran:

- Pausar la simulación.
- Reiniciar la simulación.
- Avanzar la simulación.
- Cancelar todas las acciones pendientes.
- Cancelar una acción de un objeto.

El gestor también acciona señales (*signals*) que van dirigidos a otros componentes de la aplicación; mayormente hacia la barra de progreso y el reloj de la simulación. Los dos están situados en la barra de estado de la aplicación.

- Mostrar la barra de progreso.

- Reiniciar la barra de progreso.
- Fijar valor en la barra de progreso.
- Ajustar el reloj.

8.4 GUI

Las clases que corresponden a la interfaz del simulador se pueden ver en la Figura 74. La clase *MainWindow* contiene los elementos de la ventana principal del simulador. Los paneles o *docks* del simulador son *widgets* que se añaden a la ventana principal y son referenciados desde *MainWindow*.

Qt utiliza un sistema de señales propio para manejar los eventos. La interacción del usuario con la interfaz se gestiona a través de estas señales (*signals*) y ranuras (*slots*). Cuando se acciona un botón de la interfaz, el objeto que contiene ese botón enviará una señal que recibirá un *slot* previamente definido. Las conexiones entre *signals* y *slots* se realizan en tiempo de compilación.

8.4.1 ResourcesDock y AgentsDock

Tomamos como referencia el tradicional Model-View-Controller(MVC) descrito en Design Patterns (Gamma et al, 1995). Con *Qt* en los paneles de los agentes y los recursos se utiliza el patrón Model-View, donde el View combina tanto el View como el Controller del MVC. Este patrón sigue separando el modo en que los datos son almacenados a como son representados. La Figura 73, muestra el esquema del patrón obtenido de la documentación de *Qt*.

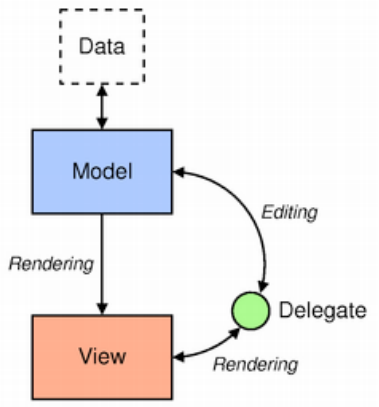


Figura 73 - Arquitectura Model/View en Qt

El modelo se comunica con los datos ofreciendo una interfaz a los demás componentes. La vista obtiene *Model Indexes* del modelo visualizándolos en una lista que permite al usuario seleccionar los datos. En caso de querer posibilitar la edición de datos desde la interfaz se hace uso de D*elegates* que se comunica también con el modelo a través de *Model Indexes*. Los *Delegates* no han sido necesarios en estos paneles, porque su funcionalidad se ha limitado a la selección.

Para los datos se ha utilizado una lista compuesta por los recursos, al igual que para los agentes.

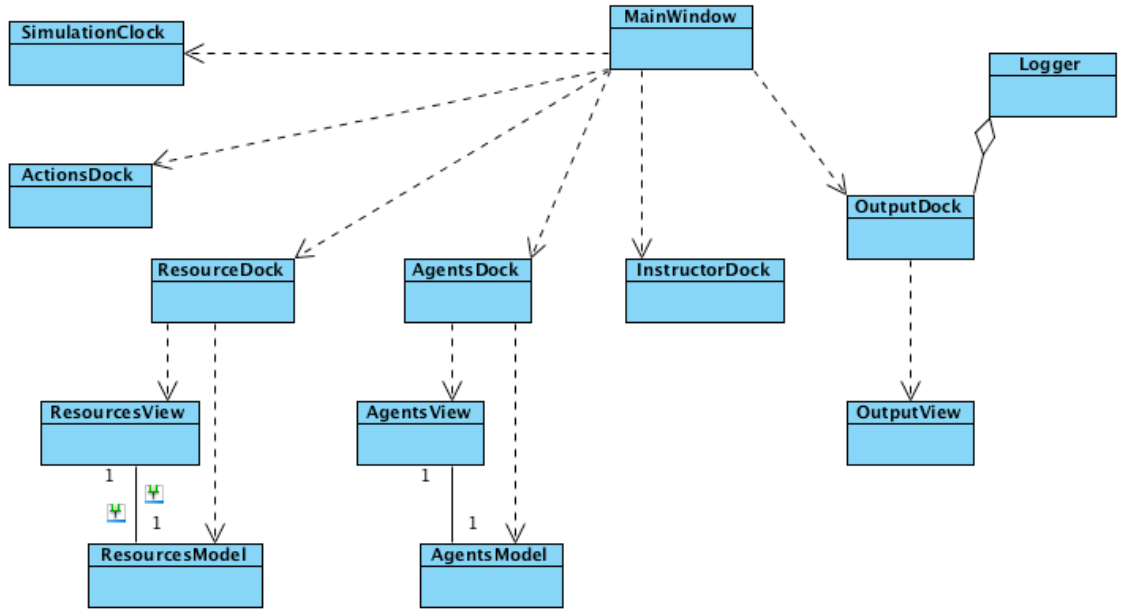


Figura 74 - Diagrama de clases de la interfaz

Como se puede observar, en los paneles tienen cabida tres clases. La clase *ResourcesDock* donde se define la ventana del panel y sus propiedades, *ResourcesModel* donde se define el modelo y se asocian los datos y por último *ResourcesView* donde se define la vista del contenido de la ventana.

Las señales de *Qt* se utilizan para la comunicación entre objetos. Por tanto, se han utilizado para cambiar la selección del recurso, la selección del agente asociado y para mostrar los datos del modelo en la vista.

8.4.2 OutputDock

Recordemos que para ayudar en el proceso de desarrollo y para informar al usuario de los que ocurre en la aplicación, se ha creado un panel que posibilita la exposición de los mensajes. Utilizando el patrón *singleton* (Gamma et al, 1995), de forma que solo hay una instancia del controlador del panel y que es accesible desde cualquier lugar de la aplicación. La clase *Logger* es la que envía las señales con los mensajes al *dock*, actuando de interfaz hacia la misma.

8.5 Escena

El grafo que representa la escena del mundo virtual está formado por una variedad de nodos procedentes de las diferentes clases de la librería gráfica OpenScenGraph. La Figura 75, muestra el grafo de la escena completa.

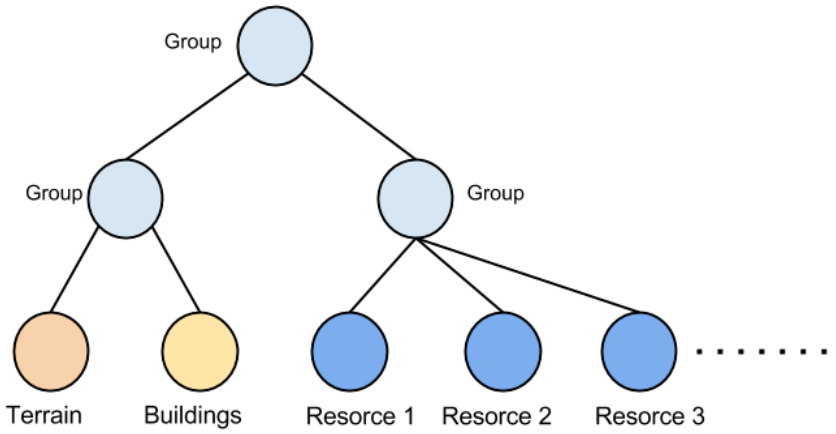


Figura 75 - Grafo de la escena

Se estructura de la siguiente forma; en el primer nivel tenemos el nodo raíz del que cuelga toda la escena. El segundo nivel se divide en dos ramas, el terreno y los recursos. El nodo del terreno engloba los modelos 3D del terreno y los edificios. Del nodo de los recursos, cuelgan los diferentes sub-grafos que compone un recurso o un agente siguiendo la definición especificada en el apartado 7.1.2.

8.6 OpenSceneGraph

En este proyecto se hace un gran uso de OpenScenGraph, no sin ciertos problemas que se ha visto necesario afrontar.

8.6.1 Picking o selección

La interacción con el mundo virtual es uno de los requisitos de la aplicación. Con el *picking* se resuelve el problema de identificar qué objeto ha sido seleccionado por el puntero del ratón en el escenario.

OSG contiene clases para facilitar el desarrollo de esta funcionalidad que consiste en implementar un *NodeCallback*. El reto es identificar qué objetos ha seleccionado el alumno al clicar en el mundo virtual. Para resolver este problema, a cada nodo raíz de los recursos y agentes se les asigna un identificador. Cuando se clica en el mundo virtual OSG devuelve una lista de nodos que están dentro de la intersección de la posición clicada. Recorriendo la lista se buscarán nodos que contengan un identificador y con él, se obtendrá la instancia del objeto.

Una vez obtenida la instancia del objeto, se usará el método que invoque la señal de selección a los paneles de los recursos y agentes, y que su vez active la caja de selección del mismo.

8.6.2 Bounding Box

El *Bounding Box* es el área que recubre un objeto en el mundo virtual. OSG trae consigo la funcionalidad de obtener el *Bounding Sphere* a partir de un nodo.

Aunque en primera instancia se optó por utilizar esta función integrada en el *toolkit*, finalmente, se descartó debido a que cuando se empezó a utilizar modelos 3D externos, el *Bounding Sphere* no se ajustaba al recurso debido al tamaño desproporcional del mismo. La esfera del recurso creaba un recuadro demasiado grande

En vez de usar la función de OSG se ha creado una clase propia utilizando el patrón *NodeVisitor*, por la que se visitan todos los nodos del modelo y se obtiene el *Bounding Box*.

Con la clase implementada, al recorrer el grafo, por cada nodo que contiene una matriz de transformación la almacena en una pila. Las matrices de transformación se almacenan con el fin de calcular la posición real de cada vértice mientras se recorre el grafo y actualizar los valores mínimos y máximos del objeto.

8.6.3 Manipuladores

En la búsqueda para el desarrollo de la manipulación de los recursos (rotación, traslación…) el *core* de OSG no contiene clases que faciliten dicha función. Se presupone que la manipulación de objetos puede ser implementada a partir de OSG *core.* Si bien esta opción puede proporcionar más flexibilidad, se decide descartarla y optar por un NodeKit. *osgManipulator* amplía las funciones de OSG soportando manipuladores 3D interactivos.

Los distintos *Draggers* implementados que ofrece como partida no son los requeridos por la aplicación y se han modificado adecuándolos a las necesidades de cada recurso.

A cada *Dragger* se le ha añadido la restricción de limitar la posición del recurso a la altura del terreno, para que cuando el alumno los desplazase, se encuentran siempre a la altura del terreno.

Durante el desarrollo se ha detectado un fallo, previamente reportado, en el *NodeKit osgManipulator* que no permitía añadir restricciones a una variante de *Dragger* concreta. Este error está resuelto en las últimas versiones de OSG, no obstante, se ha sorteado de forma que la aplicación funciona tanto en las antiguas como nuevas versiones.

9 CONCLUSIONES

En general, estoy satisfecho con el resultado del proyecto ya que he cumplido con los objetivos marcados. El proyecto ha supuesto una ampliación y puesta en práctica de los conocimientos adquiridos en diversas materias de la carrera, sobre todo, las orientadas a la programación y gráficos por computador. En definitiva, toda una experiencia que no se puede sintetizar en esta memoria y que me ha ofrecido la oportunidad de poder conocer más de cerca el entorno de investigación en un centro tecnológico.

Con la elaboración de este proyecto, se ha demostrado que los algoritmos de simulación son válidos para integrarlos en una herramienta virtual de entrenamiento. La validación y prueba del simulador no se ha llevado a cabo y se deja en manos de un trabajo futuro.

La aplicación tiene varias líneas de trabajo que permiten convertirlo en una herramienta útil en el proceso de aprendizaje de futuros profesionales:

- Añadir un mayor número de recursos y acciones.
- Modificar la arquitectura para que sea una aplicación distribuida, donde el instructor y alumno operen en distintos equipos.
- Crear o añadir modelos 3D para dar más realismo a la simulación.

10 BIBLIOGRAFÍA

Aitor Moreno (2013a), *Propagación y Extinción de Incendios Forestales y Urbanos para Entornos de Entrenamiento Virtuales*, Tesis presentada en la UPV-EHU, Departamento de Ciencias de la Computación e Inteligencia Artificial.

Aitor Moreno, Jorge Posada, Álvaro Segura, Ander Arbelaiz, Alex García-Alonso (submitted 2013b), *Interactive Fire Spread Simulations With Extinguishment Support for Virtual Reality Training Tools*, Fire Safety Journal.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides (1995), *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley.

José Javier Boulandier, Felix Esparza, Javier Garayoa, Carlos Orta, Pedro Anitua (2011), *Manual de Extinción de incendios*, Pamplona.

Paul Martz and Skew Matrix Software LLC (2007). *OpenSceneGraph Quick Start Guide*.

Rui Wang, Xuelei Qian (2010). *OpenSceneGraph 3.0: Beginner's Guide*, Packt publishing. ISBN 1849512825

Rui Wang, Xuelei Qian (2012). *OpenSceneGraph 3 Cookbook,* Packt publishing. ISBN 184951688X

Vicent Driessen (2010). *A successful Git branching model*. http://nvie.com/posts/a-successful-git-branching-model/

OpenSceneGraph Wiki Web site (2013), http://www.openscenegraph.org/

Qt 4.8 Documentation Web site (2013), http://qt-project.org/doc/qt-4.8/

ANEXO I

INTERACTIVE FIRE SPREAD SIMULATIONS WITH EXTINGUISHMENT SUPPORT FOR VIRTUAL REALITY TRAINING TOOLS

Interactive Fire Spread Simulations With Extinguishment Support for Virtual Reality Training Tools

Aitor Moreno[a,*], Jorge Posada[a], Álvaro Segura[a], Ander Arbelaiz[a], Alex García-Alonso[b]

[a] *Vicomtech-IK4, Mikeletegi Pasealekua 57, 20009 San Sebastián, Spain*
[b] *University of the Basque Country, Paseo Manuel de Lardizabal 1, 20018 San Sebastián, Spain*

Abstract

Virtual Reality training for fire fighters and managers has two main advantages. On one hand, it supports the simulation of complex scenarios like big cities, where a fire cannot be simulated in the real world. On the other hand, fire fighting VR simulators allow trainees to experience situations as similar as possible to real fire, reducing the probability of accidents when they are practising exercises with real fire.

The success of the Virtual Reality training tools also depends on how close to reality the simulation process is. This work provides fire spread algorithms for forest and urban environments, which can be used at interactive rates. Due to the interactive nature of the algorithms, the users are able to fight the fire by throwing extinguishing agents.

Although the algorithms assume many simplifications of the problem, their behaviour is satisfactory. This is due to the efficient management of the cell states in a 3 m.×3 m. cell grid. Also the variables that have more influence in fire propagation constitute the core of the algorithms. The overall system deals with user extinguishment actions, natural and artificial firebreaks, variable wind conditions (even at cell level) and non-contiguous fire propagation (embers, spotting fires). The unified forest/urban model leads to a object oriented architecture which supports the fire propagation algorithms. This also allows the system to compute efficiently mixed forest-urban environments.

Keywords: Fire Spread Simulation, Wildfires, Urban Fires, Virtual Reality, Extinguishment Support, Firebreaks, Spotting Fires

1. Introduction

Any fire has a direct impact in the society due to economic losses, environmental impact or human causalities. Even with preventive measures, such fire events will happen and fire fighters will try to limit the devastation of the fires.

Fire fighters and managers have to be fully trained for fighting fires. The training is normally composed of a combination of theoretical content with some supervised exercises with real fire. In these exercises, trainees get used to wear the equipment in warm environments for a given period of time. For safety issues, the practices are limited.

Virtual Reality simulators have been used broadly in other fields (driving simulators, machinery handling simulators, etc.). The use of VR based techniques to train fire fighters increases the possible training scenarios, conditions and tactics. As there is no real fire, the required safety measures are reduced significantly.

One of the main algorithmic elements in such simulation tools is to simulate how fire spreads as simulation time advances. If a very unrealistic or simplistic algorithm were used, the simulation would not be precise enough. This reduces the effectiveness of the simulation tool. In contrast, introducing the most complex mathematical models in the simulation increases the realism of the fire spread. However, due to the high computational cost, a real time implementation will be less feasible and the user interactivity required cannot be

*Corresponding author
Email address: amoreno@vicomtech.org (Aitor Moreno)

achieved. Thus, to meet the interactivity requirements, a simplification of the algorithms is needed.

The VR training tools we have tested with the algorithms here proposed deal with forest and urban fires. The existence of different types of terrain and buildings modifies how the fire spread, including the apparition of spotting fires (jumping over barriers) or the fire spread between urban and vegetation zones. The suppression of the fire is supported in two ways: self-extinguishment (fuel combustion) and by the action of the fire-fighters (throwing extinguishing agent).

In this work, we present fire spread algorithms that can be used in real time within interactive Virtual Simulations. The algorithms are intended to produce approximated but fast results that could be used in the training of fire fighters and managers.

In the next section, some of the related work for fire spread will be reviewed. Next, the proposed algorithms for the forest and urban environments will be described, followed by the numerical results and the validation. Finally, the conclusions and future work will be addressed.

2. Related Work

There are two major models of fire simulation: empirical models and physical models.

The empirical models follow to the experiences gathered with real fires. These models use statistical relationships found between the fire evolution and different parameters tested on the field [17]. Within this group, we can mention FARSITE [7], which uses Huygens principle of wave propagation.

The physical models use convection and heat transfer mechanisms and/or Computational Fluid Dynamics methods. The main mathematical tools they use are partial differential equations and reaction diffusion systems. Fire Dynamic Simulator (FDS, National Institute of Standards and Technology - NIST) or FIRETEC [10] follow this approach.

Seron et al. [18] and Ferragut et al. [5, 6] provided physical models with some empiric variables, which makes their solutions hybrid. The advantage of these models is their accuracy in the fire prediction. Morvan et al. [15] used them to study the interactions between fire fronts, but the computational effort is very high. The mathematical models are too complex and computers can only provide approximate solutions [4]. Another consequence is that increasing the spatial resolution causes too long computational times.

Unlike the two previous models, other research works have taken a different direction from the complex mathematical models. Their objective is to reduce computation time and to implement a real time simulation. Achtemeier [2] presented the *Rabbit Model*, a collection of basic rules of fire evolution, which are implemented as autonomous agents (the rabbits). The scope of the *Rabbit Model* is limited to the evolution of wildland fires.

Lee et al. [9] proposed a physical model for urban fires. The authors use equations to describe the heat transfer between buildings (radiation, convection), the temperature modification, and flame shape (direction, length) coming out through the windows.

Weise et al. [21] proposed a physics based model to simulate the fire spread in non-homogeneous cities with high resolution. Cheng et al. [3] modelled the fire spread in buildings taking into account the connectivity between rooms and stories. Stern-Gottfried et al. [19] introduced the *travelling fires* to support the fire dynamics in buildings.

Iwami et al. [8] introduced a very descriptive physical model for urban fires, providing different stages for each considered building type. Ohgai et al. [16] presented a physical model using cellular automata over grid of 9 m^2.

The algorithm proposed in this work presents an urban and forest fire spreading simulation, whose main characteristics are:

- The fire evolution in forest areas is based on the terrain topology, material and wind conditions. In urban areas, the different building characteristics are taken into account in order to obtain more accurate results.

- The fire may cross rivers, firebreaks or other barriers by throwing firebrands, producing spotting fires in the other side of the barrier. This mechanism is also used to spread the fire between buildings. In a similar way, urban fire can spread to forest areas and vice versa.

- Very low complexity, allowing real time simulation even with standard computing power.

- Fire suppression support. Throwing extinguishing agent affects how the fire spreads.

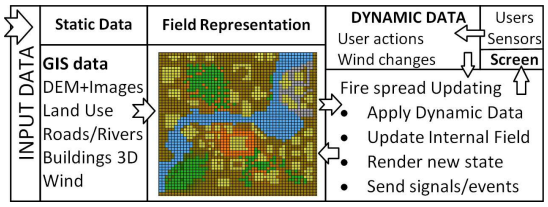


Figure 1: The simulation architecture construct the scenario from static information. During the simulation, the fire evolves according to the user actions, the fire spread algorithms and the wind changes.

Sections 3 and 4 describe the proposed algorithms. Following sections present results and performance analyses.

3. Fire Spread Algorithms for wildland areas

The fire fighting simulator has to get rapid results for the fire spread process in forest and urban areas. The system (see Figure 1) must provide a fast response to support the user interactions and dynamic changes of the wind conditions.

The main objective of this work is to provide novel algorithms to reduce the algorithmic complexity and the processing time. They follow and extend the works of Achtemeier [2], Iwami et al. [8] and some of our previous work [12]. The most relevant variables are taken into account: wind and slope [11, 21].

3.1. Field definition

The simulation algorithms utilise a regularly divided field (grid). Each cell of the field has its geometrical information (position and altitude) and its state. Figure 2 shows in a graphical way the terminology used to define the relationships of the cells in the field. For a given cell, we define its neighbours as the 8 closer cells. The surroundings of a cell include a set of cells enclosed in an elliptical or circular region.

Each cell has a type, which determines the nature of the cell (dry grass, tall trees, water bodies, roads, etc.) and its behaviour. Also, each cell has variables which are updated by the algorithms in each simulation step: *State*, *FirePower*, *MaxFirePower*, *Fuel*, *AmountAgent*, etc. These variables will be defined in the algorithms.

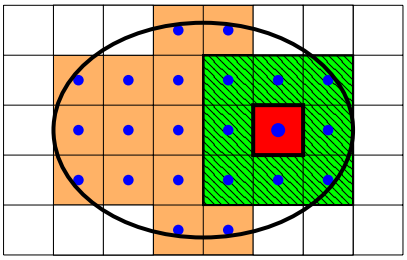


Figure 2: Terminology: The neighbours of the red cell (in striped green) and its sorrounding cells (cells inside an ellipse or a circle).

3.2. Cell States

All the cells in the field have an internal state which describes the state of the fire that exists in such cell. The different states are `Safe`, `Activated`, `Burnt`, `Survive`, `BarrierCrossing` and `FireStopped` (see Figure 3). An extended description of the states can be found in [12].

When a cell is in `Safe` state, there is no fire in it. `Activated` state indicates that there is an active fire in the cell.

`Burnt` is the final state of a cell. All its fuel is burned and it has cooled down. `Survive` state is a pseudo-final state, where all its fuel is burnt, but still has residual heat (*FirePower*). Even when its fuel is completely consumed by the fire, the cell can irradiate some heat to other cells. Eventually, the cell will pass to the `Burnt` state.

`BarrierCrossing` is a state that controls whether the fire spreads through cells that represent a river or firebreak (Spotting Fires).

`FireStopped` means that the fire in the cell has been stopped by an external extinguishing agent.

3.3. Simulation Step

Algorithm 1 shows the *SimulationStep* procedure, which is run in each simulation step and defines the main simulation loop. Firstly, the cells that were activated in the previous step (stored in the temporary *ActivatedCellList*) are moved into *ActiveCellList*.

Afterwards, the *UpdateState* method of each active cell is called. If the state of the cell is `Burnt`, the cell is removed from the list.

3.4. Update Step method

The *UpdateState* method is presented in Algorithm 2. This is a generic version of the method,

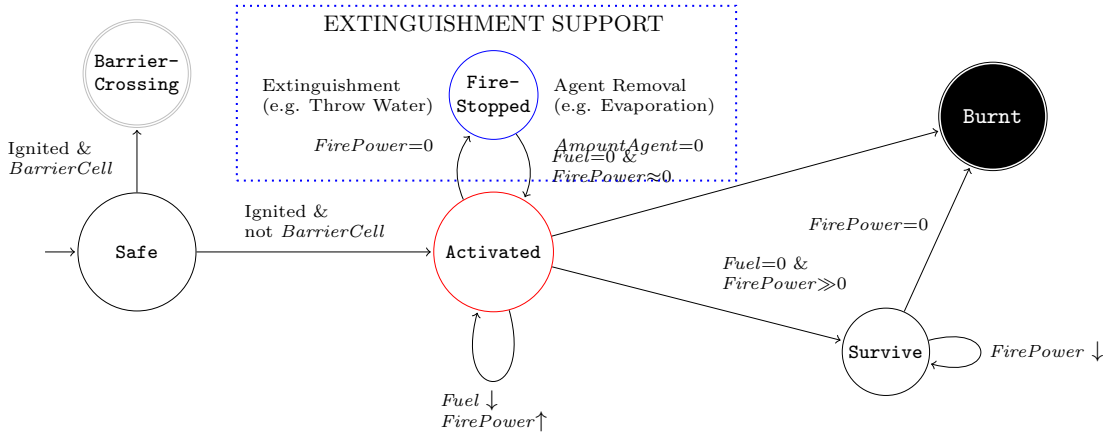


Figure 3: Cell states and transitions.

Algorithm 1 The *SimulationStep* procedure defines the main simulation loop.

```
1:  procedure SimulationStep()
2:      // Update list ActiveCellList
3:      for each cell cc in ActivatedCellList do
4:          ActiveCellList.Add(cc)
5:          ActivatedCellList.Remove(cc)
6:      end for
7:      // Compute new states
8:      for each cell cc in ActiveCellList do
9:          cc.UpdateState()
10:         if cc is Burnt then
11:             ActiveCellList.Remove (cc)
12:         end if
13:     end for
14: end procedure
```

since every class of *Cell* will provide its own implementation. The method contains three main actions: *i)* consume fuel, *ii)* evaporate water in the surroundings and *iii)* spread to other cells (ignite cells).

The *ConsumeFuel* function will reduce a certain amount of fuel, calculated as a function of the existing *FirePower*. As the fuel is consumed, the *FirePower* will be increased. As an example, a grass cell will consume a small amount of fuel at the beginning, but it will increase the *FirePower* very fast, increasing the amount of consumed fuel in each step.

The *FirePower* cannot grow infinitely, so a maximum *FirePower* (*MaxFirePower*) is defined for each cell type. When a cell is in `Survive` state, the *FirePower* decreases in each step. Having no fuel, the cell will pass to the `Burnt` state when the residual *FirePower* becomes zero.

The *Evaporation* method reduces a certain amount of extinguishing agent in the surrounding cells (the method is addressed in the Section 3.9).

The *SpreadSlopeWind* and *SpreadSpotting* methods evaluate if the cell triggers the ignition of new cells. The methods are described in their corresponding sections. The activation of the ignited cells is performed in the *IgniteFire* method.

The *UpdateState* method finishes checking if the remaining fuel is zero. In this case, the cell state is set to `Burnt` or `Survive`.

3.5. IgniteFire method

The *IgniteFire* method is presented in Algorithm 2 and it is run when a cell activates another cell. The state of the target cell is set to `Activated` and an initial *FirePower* is calculated. This value takes into account the *FirePower* in the cell which is igniting this cell, and the *FirePower* of the neighbour cells.

The newly ignited cell is added to *ActivatedCellList*.

3.6. Spread fire: Slope and Wind

SpreadSlopeWind method is presented in Algorithm 3. The `Safe` neighbours of a given cell are visited. For each one, the slope is calculated. A probability (p) is computed considering slope, wind

Algorithm 2 The methods *UpdateState* and *IgniteFire* are the basic actions when any active cell is updated or ignites another cell.

```
1:  method Cell::UpdateState()
2:      if state == FireStopped then return
3:      ConsumeFuel ()
4:      Evaporation ()
5:      // Spread to other Cells
6:      SpreadList.Add (SpreadSlopeWind())
7:      SpreadList.Add (SpreadSpotting())
8:      for each cell cc in SpreadList do
9:          IgniteFire(cc)
10:     end for
11:     if Fuel == 0 then
12:         SetState (Burnt or Survive)
13:     end if
14: end method
1:  method Cell::IgniteFire(Cell c)
2:      calculate intensity ( neighbour cells)
3:      c.SetState ( Activated )
4:      c.SetFirePower ( intensity )
5:      // In next step, this cell c will be active
6:      ActivatedCellList.Add(c)
7:  end method
```

Algorithm 3 The method *SpreadSlopeWind* checks neighbours for ignition.

```
1:  method Cell::SpreadSlopeWind()
2:      create new Cell list IgnitedList
3:      for each neighbour cell vc do
4:          calculate slope
5:          // p ∈ (0, 1) is a function of
6:          // slope, wind and FirePower
7:          p = CalculateProbability()
8:          if PassTest (p) then
9:              IgnitedList.Add (vc)
10:         end if
11:     end for
12:     return IgnitedList
13: end method
```

and *FirePower*. *PassTest* generates a random number; if p is greater, then a function returns *true*, and the cell is selected for ignition.

3.7. Barriers: Rivers, Firebreaks, Roads

Barrier cells model non-combustible barriers such as rivers or roads.*BarrierCell* class has its own implementation of *UpdateState* and *IgniteFire* methods. They are presented in Algorithm 4 and they have significant differences from the generic implementation, explained in sections 3.4 and 3.5.

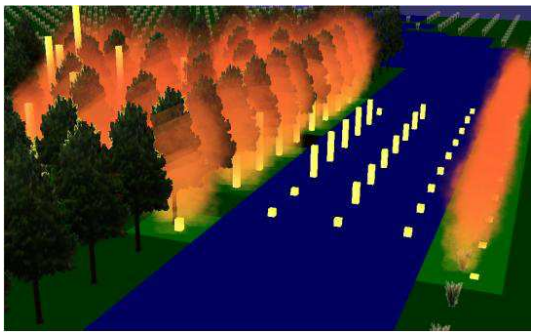


Figure 4: Visual representation of the *CrossingIntensity* value when the fire crosses a river.

Algorithm 4 The methods *UpdateState* and *IgniteFire* for barrier cells.

```
1:  method BarrierCell::UpdateState()
2:      SpreadList.Add (SpreadSpotting())
3:      for each cell cc in SpreadList do
4:          IgniteFire(cc)
5:      end for
6:  end method
1:  method BarrierCell::IgniteFire(Cell c)
2:      calculate crossIntensity (neighbours)
3:      if crossIntensity > 1 then
4:          c.SetState(BarrierCrossing)
5:          // In next steps, c will be active
6:          ActivatedCellList.Add(c)
7:      end if
8:  end method
```

Only the *SpreadSpotting* method is used in *BarrierCell::UpdateState* (see Section 3.8) and the ignition probabilities are heavily reduced (75%).

When a *BarrierCell* is activated, its state changes to `BarrierCrossing` instead of `Activated`. As there is no real fuel to burn, there is no *FirePower*. We define *CrossIntensity* as the intensity of the radiation of the fire through a *BarrierCell*. The initial value is the 20% of the sum of its neighbours' *FirePower*. If the cell is surrounded only by *BarrierCell*'s, the *CrossingIntensity* will be set as the 50% of the maximum *CrossingIntensity* value found in its neighbours.

3.8. Spread by Spotting Fires

Spotting fires provide a mechanism to spread fires further than neighbour cells. This mechanism also

Algorithm 5 The method *SpreadSpotting* checks the surrounding cells for ignition (*Spotting Fires*).

```
1: method Cell::SpreadSpotting()
2:     create new cell list IgnitedList
3:     create new cell list Candidates
4:     get ellipse parameters (wind)
5:     Candidates = SelectCells (ellipse)
6:     for each cell vc in Candidates do
7:         calculate slope
8:         // p ∈ (0, 1) is a function of
9:         // slope, wind and FirePower
10:        p = CalculateProbability()
11:        if PassTest (p) then
12:            IgnitedList.Add (vc)
13:        end if
14:    end for
15:    return IgnitedList
16: end method
```

allows fire to propagate over natural or artificial barriers (roads, rivers, etc.) and to the surrounding cells (see Figure 2).

SpreadSpotting method is presented in Algorithm 5. Its behaviour, although similar to *SpreadSlopeWind*, takes into account the non-combustible surrounding cells.

In this method, an active cell may ignite other cells within an elliptical shape. The ellipse is a function of the wind direction and velocity. For each surrounding cell, an ignition probability is calculated. If the *PassTest* function returns *true*, the cell is selected for ignition.

The vegetation type is very important in the ignition of spotting fires. Tall trees are more prone to propagate embers through the air (reaching distant areas), than grass or bushes.

Figure 4 shows in vertical bars the *CrossingIntensity* in the cells which represent the river. In the figure, the fire spreads to the other bank.

3.9. Extinguishment

The calls to *ThrowWater* are triggered by user actions. This is done out of the main simulation loop (*SimulationStep*). This method is presented in Algorithm 6. A given quantity of extinguishing agent reduces instantaneously some *FirePower* which depends on the type of the agent.

When a cell has no *FirePower*, the state changes to `FireStopped` and the remaining agent is accumulated in the cell. In this state, any additional

Algorithm 6 The method *ThrowWater* is triggered by the user actions and supports the suppression of the fire with extinguishing agents.

```
1: method FloorCell::ThrowWater(factor)
2:     calculate reduction ( factor, Type )
3:     if state == Activated then
4:         FirePower -= reduction
5:         if FirePower <= 0 then
6:             SetState (FireStopped)
7:             AmountAgent = -1 ×FirePower
8:             FirePower = 0
9:         end if
10:    else if state == FireStopped then
11:        AmountAgent += reduction
12:    end if
13: end method
```

thrown agent increases the cell *AmountAgent*. This accumulated agent can be evaporated by nearby active cells.

The *Evaporation* and *EvaporateWater* methods are presented in Algorithm 7. *Evaporation* is called within the *UpdateState* performed in each simulation step (see Algorithm 2). An active cell selects surrounding cells within a circular area, which radius is proportional to the *FirePower*. The, the *EvaporateWater* method of the selected cells is called with a reduction factor as parameter. This factor takes into account the distance between the cells and the *FirePower*.

The *EvaporateWater* method uses the calculated factor and the extinguishing agent type to calculate the amount of agent to reduce. If *AmountAgent* goes below zero, the cell's state changes to `Activated`. In the next simulation step, the fire will be effectively rekindled.

4. Fire Spread Algorithm for urban areas

To support the fire spread in urban areas, the *BuildingCell* and *FloorCell* classes are introduced. A *BuildingCell* is composed of a vertical stack of *FloorCell* cells. Urban cells require the definition of new types of materials. Iwami et al. [8] classify buildings according to their structure. We follow a similar classification that considers three types: *ShantyUnit*, *WoodenUnit* and *SecureUnit* [12].

The *UpdateState* method for *FloorCell*'s is similar to Algorithm 2. The consumption function depends on the building type (see Figure 5).

Algorithm 7 The methods *Evaporation* reduces the accumulated extinguishing agent in the surroundings cells by means of the supporting *EvaporateWater* method.

```
1:  method Cell::Evaporation()
2:      create new cell list Affected
3:      // this active cell evaporates cells
4:      // within a radius (range)
5:      Affected=SelectCells(FirePower,range)
6:      for each cell cc in Affected do
7:          if cc has extinguishing agent then
8:              set factor (FirePower, distance)
9:              cc.EvaporateWater ( factor )
10:         end if
11:     end for
12: end method
1:  method Cell::EvaporateWater(factor)
2:      calculate reduction ( factor, Type )
3:      AmountAgent -= reduction
4:      if AmountAgent <= 0 then
5:          AmountAgent = 0
6:          if state == FireStopped then
7:              SetState (Activated)
8:          end if
9:      end if
10: end method
```

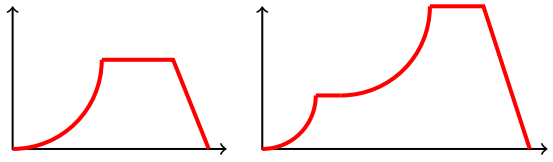


Figure 5: Evolution of *FirePower* for *SecureUnit* and *WoodenUnit* building types.

The *Evaporation* method reduces some of the accumulated extinguishing agent in the neighbour cells. In buildings, neighbour cells are contiguous *FloorCell*'s in the same floor, plus two other cells: the cell above and the cell below the *FloorCell* (if they exist).

After checking if the cell can still be active, the fire spreading methods are run. In the buildings, they are *SpreadHorizontal* and *SpreadVertical*. If the cell is part of the facade, the fire can spread to the nearby buildings or to the existing vegetation using *SpreadSpotting* and *SpreadVegetation* methods.

Algorithm 8 The methods *SpreadHorizontal* and *SpreadVertical* provide the essential mechanism for fire spread in buildings.

```
1:  method FloorCell::SpreadHorizontal()
2:      Neighbours = GetNeighboursCells ()
3:      Candidates (Neighbours, FirePower)
4:      for each cell vc in Candidates do
5:          // p ∈ (0, 1) is a function of
6:          // slope, wind and FirePower
7:          p = CalculateProbability()
8:          if PassTest (p) then
9:              IgniteFire (vc)
10:         end if
11:     end for
12: end method
1:  method FloorCell::SpreadVertical()
2:      state = FloorAbove.getState()
3:      // p ∈ (0, 1) is a function of
4:      // slope, wind and FirePower
5:      p = CalculateProbability()
6:      if PassTest (p) then
7:          IgniteFire (FloorAbove)
8:      end if
9:      // Repeat for FloorBelow
10:     // with smaller probability
11: end method
```

4.1. Fire Spread algorithms

The *SpreadHorizontal* and *SpreadVertical* methods are presented in Algorithm 8. Some of the neighbours in the same floor are selected as candidates for ignition. The selection takes into account the *FirePower* and the building type of the cells. For each candidate, a probability is calculated and the cell is ignited if a function *PassTest* returns *true*.

The fire can spread vertically to the cell above and, sometimes, to the cell below. In the method *SpreadVertical*, a probability test is performed for the cell above. Another test is run for the cell below, using a much lower ignition probability.

The *SpreadSpotting* method is presented in Algorithm 9. It is only run for cells in the building facades.

This method calculates an elliptical shape around a burning *FloorCell*, whose parameters depend on the wind direction and velocity. All facade *BuildingCell*'s inside this 2D ellipse are considered for ignition. The ignition probability for each *FloorCell* is computed as a function of the wind condition,

Algorithm 9 The method *SpreadSpotting* for *FloorCell*'s provides the *Spotting Fires* mechanism for buildings.

```
1: method FloorCell::SpreadSpotting()
2:     if isNotFacade() then return
3:     get ellipse parameters (wind)
4:     List = SelectBuildingCells ( ellipse )
5:     for each BuildingCell b in List do
6:         for each FloorCell fc do
7:             dist = distanceTo (fc)
8:             α = angleDeviation (fc)
9:             // p ∈ (0, 1) is a function of
10:            // wind, distance and α
11:            p = CalculateProbability()
12:            if PassTest (p) then
13:                IgniteFire (fc)
14:            end if
15:        end for
16:    end for
17: end method
```

Algorithm 10 The method *SpreadVegetation* provides the *Spotting Fires* mechanism between buildings and vegetation cells.

```
1: method FloorCell::SpreadVegetation()
2:     if isNotFacade() then return
3:     List = SelectNeighbourCells ()
4:     for each cell cc in List do
5:         // p ∈ (0, 1) is a function of
6:         // wind and FirePower
7:         p = CalculateProbability()
8:         if PassTest (p) then
9:             IgniteFire (cc)
10:        end if
11:    end for
12: end method
```

the distance and the 3D angular deviation from the candidate *FloorCell* to the already burning *FloorCell*.

The *SpreadVegetation* method is presented in the Algorithm 10. As the previous one, it only can be run for cells in the building facades and only the vegetation cells in the neighbourhood are taken into account. For each selected cell, an ignition probability is calculated as a function of the *FirePower*, the vegetation type and the wind direction and velocity.

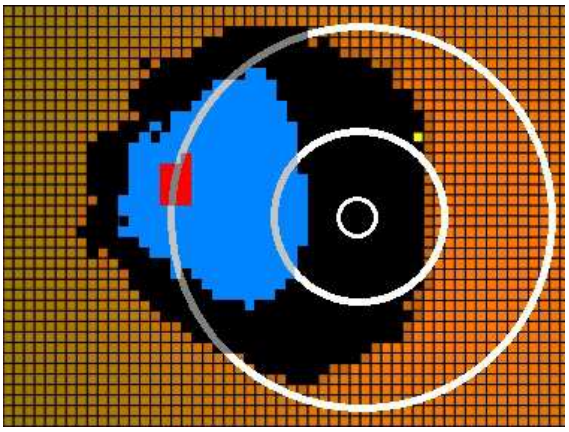


Figure 6: The fire spreads in a terrain with a hill (represented with white circles) and no wind. The fire started in the red zone and then it spread faster (blue zone) towards the top of the hill. Finally, the fire spread sideways and went down the other side of the hill.

4.2. Extinguishment Support

The mechanism to support the suppression of the fire is the same in the buildings and in vegetation cells. The method *ThrowWater* is presented in the Algorithm 6.

The *ThrowWater* method is not used directly in the buildings. We have introduced the *WaterJet* concept, which emulates the actions of the fire fighters with high pressure hoses in the facade of the buildings.

A *WaterJet* is conceptualized as a water jet thrown from the exterior of a building to a target *FloorCell*. It has to be in the facade of the building. The *WaterJet* is parameterized by its *WaterJetLevel*, which determines how many *FloorCell*'s are reachable in a straight line. If the value is 1, only the target cell receives the extinguishing agent. If the value is 2, the agent reaches to an extra cell in the water jet direction. Both of them receive half the amount of the agent.

5. Analysis of the Algorithms

This section measures and analyses the main parameters involved in the fire spread algorithms. Results show the coherent behaviour of the algorithms regarding their expected behaviour.

5.1. Forest Environments

Firstly, we analyze the behaviour of the algorithms in forest environments, which is composed

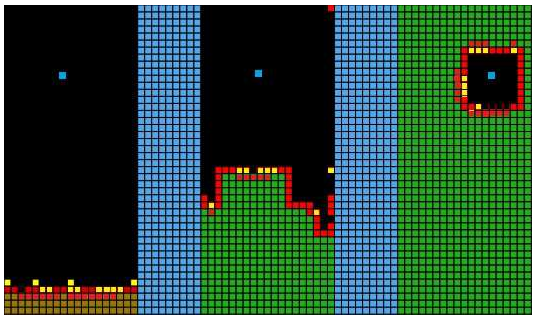


Figure 7: Scenario with different vegetation types, horizontal terrain and no wind. The fire spreads at different speed. Each zone (grass, bush, trees) is separated with water bodies (rivers). The ignition points are in similar positions (in blue).

of the following key factors: the terrain (slope and composition), wind conditions and spotting fires. Also, we present the behaviour of the extinguishment support.

5.1.1. Terrain slope and composition

The slope of the terrain has a direct influence on the fire spread direction and speed. In horizontal fields, the algorithm tends to create circular shapes. On slopes, the fire tends to go up, following the slope direction. The tests confirm that the speed of the fire propagation is the function of the slope (see Figure 6) .

Different cell types also modify the fire spread results. Changing the type or the quantity of the existing fuel in a cell impacts the simulation result. Figure 7 shows three vegetation cell types (grass, bush and tall trees) in an horizontal terrain with no wind. Each vegetation strip is separated from each other with water cells. The simulation shows the different spread speed of each cell type.

5.1.2. Wind direction

Wind is one of the most influential variables in fire spread. The wind direction and speed modify the fire spread direction and velocity. Figure 8 shows how the wind spreads the fire in the corresponding direction.

As the wind conditions are used in each simulation step, they can be changed dynamically. This feature of the algorithms can be used to provide more realistic scenarios with changing conditions.

5.1.3. Spotting Fires

The algorithms simulate how the fire spreads by throwing embers (*Spotting Fires*). When there is

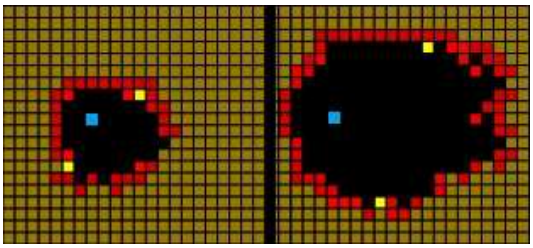


Figure 8: Two time steps at an horizontal scenario with uniform vegetation and constant wind (from the West). The fire spreads faster in the wind direction.

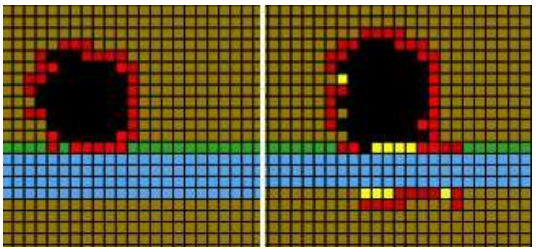


Figure 9: Two scenarios with a river, horizontal terrain and constant wind (coming from the North). The fire only spreads to the other bank in the scenario with the narrower river.

a barrier (river, road...), the spread probability depends on the width of the barrier and on the vegetation in both sides. Figure 9 shows two examples with the wind coming from the North. The only difference is the width of the barrier. The first barrier is wide enough to stop the fire. The second one does not stop the fire spread.

5.1.4. Extinguishment

The extinguishment support was tested with different experiments. Figure 10 shows the evolution of the *FirePower* in a given cell with extinguishing action from Step 1020 to Step 1260.

Initially, the fire evolves freely until the extinguishing action starts (Step 1020). The extinguishing agent manages to control the fire and starts to accumulate in the cell (Step 1150).

After stopping the extinguishing action in Step 1260, the sourrounding active cells evaporate gradually the accumulated extinguishing agent. Once the remaining extinguishing agent is evaporated (Step 1320), the cell rekindles and its *FirePower* starts to increase till the fuel is fully consumed (Step 1540).

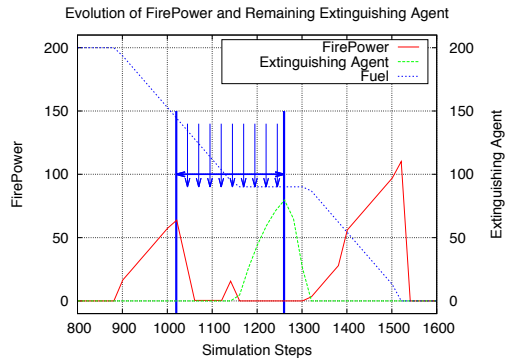


Figure 10: The evolution of the cell is explained in Section 5.1.4.

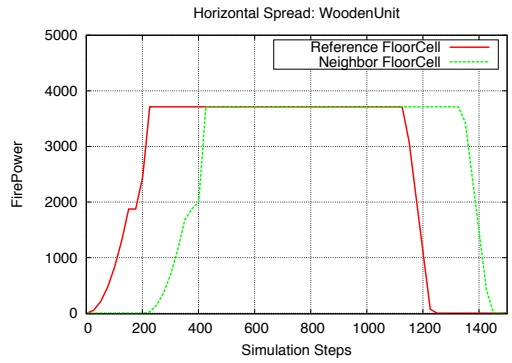


Figure 11: An urban fire spreads from a *WoodenUnit FloorCell* to another.

5.2. Analysis of Urban Environments

In the urban environments, the fire spreads inside the buildings or by spotting fires from one building to another. In this section some case studies are presented.

Figure 11 shows the evolution of the *FirePower* in a *WoodenUnit FloorCell*. As the fire evolves freely, eventually it spreads to a neighbor cell.

The main extinguishing action is to throw extinguishing agent to a facade of a *FloorCell*, trying to reduce its *FirePower*. Figure 12 presents two possible *FirePower* evolution in a *ShantyUnit FloorCell*. The thrown extinguishing agent thrown by the *WaterJet* is not enough to put out the fire and the *FirePower* rises again when the extinguishing action stops.

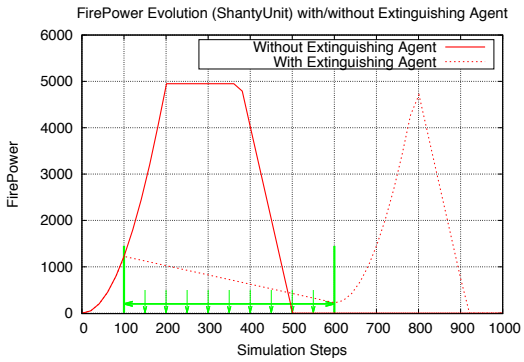


Figure 12: Free fire evolution in a *ShantyUnit FloorCell* (solid red line). Throwing extinguising agent reduces *FirePower* (dotted green line). In the simulation step 600, no more agent is thrown and the fire rekindles.

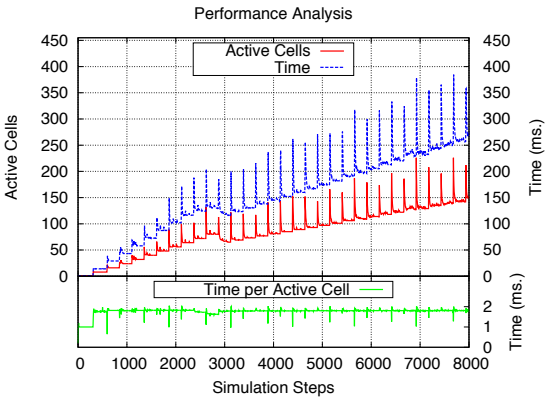


Figure 13: Performance in a forest scenario.

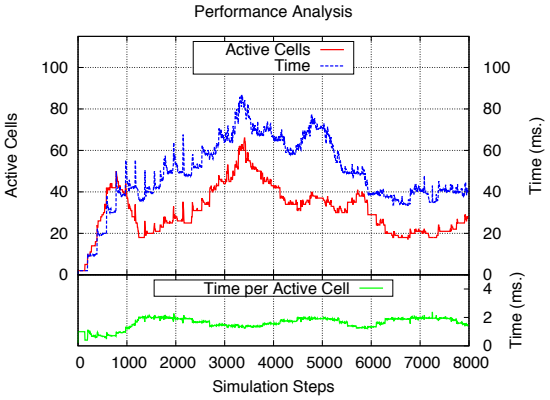


Figure 14: Performance in a mixed scenario (forest and urban).

6. Performance Analysis

The analysis of the computation performance of the algorithms has been made through multiples scenarios under different conditions. All the measures were obtained using an Intel Quad Core Q9400 processor, 4 GB of RAM and a GeForce GTX 285,Windows 7 64 Bit (Service Pack 1) with the latest stable graphics drivers. In all simulations, the number of active cells per step and the time needed by the algorithms are registered.

Figure 13 shows the behaviour in a scenario with a constant 45 degree slope, composed of one single vegetation type and no wind. The number of active cells increases gradually, as the fire spreads. The figure shows a linear evolution in the number of active cells. This is consistent with the algorithms because burnt cells are removed from the active cell list as the new ones are added. The expected quadratic behaviour is converted into linear by the algorithm: the active cells represent approximately the fire front.

Figure 14 shows performance in an urban area with constant wind. The number of active cells is more irregular, depending on the evolution of the fire across the buildings. Buildings have different number of storeys. In the statistics, a *BuildingCell* is counted as one active cell, even when it is composed of multiple floors.

In both examples, the simulation time per active cell is around 2 ms. We find 300 ms. as the maximum computation time for one simulation step. As the simulation time per step is below 1 s. (the simulation Step), the interactivity capabilities of the simulations are guaranteed.

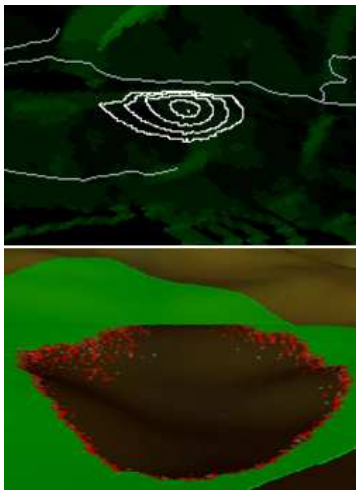


Figure 15: Visual comparison between the results obtained in FARSITE (top) and the simulation results using the proposed algorithms (bottom).

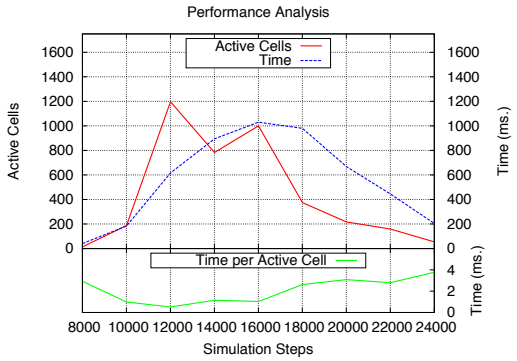


Figure 16: The time spent per active cell is around 4 ms. The shown scenario is an urban scenario with buildings with 3 stories. The number of active cells varies depending on the fire spread in the buildings and between them.

7. Validation

The validation of the presented algorithms has been performed in three steps. The first one addresses the utilisation of FARSITE software to validate the forest fires simulation. The second step compares with the simulation output obtained by Zhao [23] in a urban scenario. For the last step, we have contacted fire fighting experts to get their opinion about our simulation achievements.

7.1. Forest Areas

FARSITE [7] is a well-known fire simulation application oriented to forest environments and its simulation results are usually utilised in scientific comparisons [20, 22]. It is based on the Huygens principle of wave propagation to imitate the fire propagation in a highly realistic way by using complex calculations.

The field spatial resolution can be configured to match the field resolution used in this work, i.e., 3×3 m. cells. However, FARSITE can not be configured to set a simulation step of 1 s. The minimum selectable value is 1 min.

FARSITE calculates the fire spread as a collection of fire fronts, represented as contours. The humidity, the terrain aspect, dead fuel models and other variables are used in the FARSITE to calculate the simulation.

Figure 17: Simulation of a theoretical mixed fire in San Sebastian (Spain). The terrain, the land use and the building information have been constructed from open data sources.

We have used a section of the tutorial field included in the FARSITE installation for a qualitative comparison with the proposed algorithms. The size of this test area is 81 ha. and it is sampled by 300×300 cells.

The simulation results after one hour are similar on both cases (see Figure 15). The real difference is in the simulation time. Our algorithms are much faster than the FARSITE simulation, even disabling some features in FARSITE (varying wind and dead fuel models).

7.2. Urban Areas

Zhao [23] has researched the fires after earthquakes in urban environments. In his work, a real urban fire scenario is presented and we have used it to validate the behaviour of our algorithms. We have constructed a cell representation of one urban scenario analysed by Zhao. As the number of stories is not described, we have modelled the buildings with 3 stories. The roads and streets are considered non combustibles, so the fire spreads by jumping over streets (Spotting Fires).

Figure 18 shows the real fire (reported by Zhao), the Zhao's simulation [23] and our results. They correspond to 7 h. after fire starts (our step is 25.200). The figure shows that simulation results are similar (spatially and temporally).

From the performance point of view, Figure 16 shows that the simulation time per active cell is about 4 ms. The simulation time is below 1 s. threshold, except in a brief period of time. That precise moment correspond to an extreme situation, where almost 70% of the buildings are burning at the same time.

Figure 17 shows another use case study. The sampled area is part of San Sebastian (Spain). It is a coast zone with a small hill near the old-town of the city.

The scenario contains a urban zone, composed of wooden buildings, surrounded by vegetation (including a small forest):

- The Digital Elevation Model (DEM) was constructed from open data sources.
- The land use (water, forest, vegetation, roads and buildings) was loaded as Shapefiles (SHP). As building footprints came from different providers, an intelligent combination of the available data sources was performed to create an improved scenario.
- The number of storeys of the buildings was not available. Thus, we provided a method to calculate the height of the buildings by using the difference between the DEM and the Digital Surface Model (DSM).

With the composed scenario, different fire situations can be simulated. Figure 17 shows how a fire started in the vegetation zone and due to the wind conditions, it managed to spread to the urban area.

7.3. Validation with Experts

We have collaborated with the *Centro de Jovellanos* [1] in the validation phase of the algorithms. *Centro de Jovellanos* provides training courses for different professionals such as fire fighters, brigades or civil protection. A selection of experts have tested the algorithms through a simulation tool, specifically designed for that purpose (see Figure 19).

7.3.1. Validation of Forest Scenarios

The experts confirmed that the fire behaviour on the forest environment met their experience. They noticed how the wind and the slope modified the fire spread behaviour. They appreciated how the fire could jump over a river or road if the wind conditions were favourable.

In a second run of testing, the experts were provided with a virtual hose, so they could try to suppress a fire. Their experiences were satisfactory.

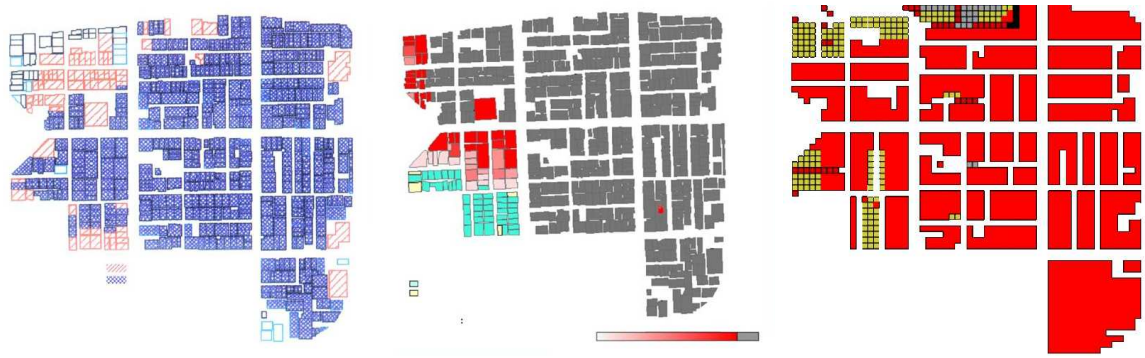


Figure 18: Validation of urban algorithms. From left to right and up to bottom: *a)* real status of the fire after 7 h.; *b)* Zhao's simulation after 7 h. and *c)* simulation output with our algorithms.

Figure 19: Fire training facilities at *Centro Jovellanos* [1], validation session and two screenshots of the simulation tool.

However, in their opinion, the behaviour of the extinguishing agent was not correctly balanced. However, the *factors* of the extinguishing agents can be easily modified to match the expected behaviour.

7.3.2. Validation of Urban Scenarios

The experts explained that it is quite difficult to determine the correct behaviour of a big scale urban fire because it does not happen very often and there are few records. One of their main concerns was related to the modelling of the interior of the buildings, as it is important to determine how the fire would spread.

The experts verified that the suppression techniques with virtual hoses (the *WaterJet* method described in the algorithms) fulfill a great variety of suppression tactics from the exterior of the buildings.

8. Conclusions and Future Work

This work presents fire spread and extinguishment algorithms for forest and urban environments which can be used at interactive rates. This is a requirement that must be met by virtual environment based training systems: fire fighting trainees have to throw extinguishing agents and get an immediate feedback of their actions. Although the algorithms have been simplified to match this requirement, they support variables and models that have a great influence in fire evolution. The analyses and validations show that the simplifications proposed keep proper fire spread behaviour. The paper shows numerical and graphical results. It also summarizes assessments from experts, who state the consistency of the system behaviour with their experiences.

The algorithms support different types of vegetation and buildings. Barriers like roads, rivers or firebreaks can be jumped by the fire creating spotting fires (given the proper wind and slope conditions). Wind can be set at local level (each 3×3 m. cell) without any performance decay. Wind and slope have a significant influence in the fire spread behaviour.

Urban fires are seamlessly integrated in the field definition. The specific fire behaviour for buildings includes horizontal and vertical spread mechanisms. Furthermore, the fire can spread from building to building or from building to vegetation areas (spotting fires). *WaterJet* conceptualizes the attack of urban fires with hoses from the exterior of the buildings.

The algorithms allow developers the design of elegant object oriented architectures which have as a central core a convenient definition of cell states and state transitions. These models, as previously published, allowed us the integration of geoinformation and semantic architectures [13].

The addition of other variables not covered in this work (e.g. weather, season, temperature and relative humidity) would provide better simulations, but the impact in the performance has to be evaluated. It is also important to use more detailed information, including the typology of buildings, land use and extinguishing agents to achieve more accurate simulations. Using GPU or GPGPU implementation techniques will bump up the performance as the algorithms are highly parallelizable.

The utilisation of the algorithms in the early stages of a real emergency can help in the decision making process. We have been working in some preliminary analyses [14].

9. Acknowledgements

This work was supported by COST Action TU0801 "*Semantic Enrichment of 3D City Models for Sustainable Urban Development*", and it was carried out in the context of project SIGEM, funded by the Spanish Industry Ministry through its *Avanza I + D Programme*. Dr. García-Alonso was supported by the Spanish MEC TIN2009-14380 and the Basque Government IT421-10.

Authors thanks Basque Country government for the *Open Data* initiative, which provided the geographic information needed to construct the scenarios.

References

[1] *Centro de Seguridad Marítima Integral Jovellanos: http://www.centrojovellanos.com*, 2012.

[2] G. L. Achtemeier, *Rabbit rules. an application of stephen wolfram new kind of science to fire spread modeling*, Technical Program of the Joint 2nd International Wildland Fire Ecology and Fire Management Congress and 5th Symposium on Fire and Forest Meteorology (2003).

[3] H. Cheng and G. V. Hadjisophocleous, *Dynamic modeling of fire spread in building*, Fire Safety Journal **46** (2011), no. 4, 211–224.

[4] Y. Dumond, *Forest fire growth modelling with geographical information fusion*, Information Fusion, 11th International Conference on (2008), 1–6.

[5] L. Ferragut, M. I. Asensio, and S. Monedero, *Modelling radiation and moisture content in fire spread*, Communications in Numerical Methods in Engineering **23** (2007), 819–833.

[6] L. Ferragut, S. Monedero, M. I. Asensio, and J. Ramrez, *Scientific advances in fire modelling and its integration in a forest fire decision system*, Modelling, Monitoring and Management of Forest Fire I. WIT Transactions on Ecology and the Environment, 2008, pp. 31–38.

[7] M. A. Finney, *Farsite: Fire area simulator-model development and evaluation*, Res. Pap. RMRS-RP-4. Ogden, UT: USDA Forest Service, Rocky Mountain Research Station **1** (1998), 47.

[8] T. Iwami, Y. Ohmiya, Y. Hayashi, K. Kagiya, W. Takahashi, and T. Naruse, *Simulation of city fire*, Fire Science and Technology **23** (2004), no. 2, 132–140.

[9] S. Lee, R. Davidson, N. Ohnishi, and C. Scawthorn, *Fire following earthquake-reviewing the state-of-the-art of modeling*, Earthquake spectra **24** (2008), no. 4, 933–967.

[10] R. Linn, J. Reisner, J. J. Colman, and J. Winterkamp, *Studying wildfire behavior using FIRETEC*, International Journal of Wildland Fire **11** (2002), 233–246.

[11] F. Morandini, X. Silvani, L. Rossi, P. A. Santoni, A. Simeoni, J. H. Balbi, J. L. Rossi, and T. Marcelli, *Fire spread experiment across mediterranean shrub: Influence of wind on flame front properties*, Fire Safety Journal **41** (2006), no. 3, 229–235.

[12] A. Moreno, A. Segura, A. Korchi, J. Posada, and O. Otaegui, *Interactive urban and forest fire simulation with extinguishment support*, Advances in 3D Geo-Information Sciences, Lecture Notes in Geoinformation and Cartography, 2011, pp. 131–148.

[13] A. Moreno, A. Segura, S. Zlatanova, J. Posada, and A. García-Alonso, *Benefit of the integration of semantic 3D models in a fire-fighting VR simulator*, Applied Geomatics **4** (2012), no. 3, 143–153.

[14] A. Moreno, A. Segura, S. Zlatanova, J. Posada, and A. García-alonso, *Introducing GIS-based simulation tools to support rapid response in wildland fire fighting*, Modelling, Monitoring and Management of Forest Fires III, 2012, pp. 163–174.

[15] D. Morvan, C. Hoffman, F. Rego, and W. Mell, *Numerical simulation of the interaction between two fire fronts in grassland and shrubland*, Fire Safety Journal **46** (2011), no. 8, 469–479.

[16] A. Ohgai, Y. Gohnai, S. Ikaruga, M. Murakami, and K. Watanabe, *Cellular automata modeling for fire spreading as a tool to aid community-based planning for disaster mitigation*, Recent advances in design and decision support systems in architecture and urban planning, 2005, pp. 193–209.

[17] R. C. Rothermel, *A mathematical model for predicting fire spread in wildland fuel*, U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station, 1972.

[18] F. J. Serón, D. Gutiérrez, J. Magallón, L. Ferragut, and M. A. Asensio, *The evolution of a wildland forest fire front*, Visual Computer **21** (2005), 1–18.

[19] J. Stern-Gottfried and G. Rein, *Travelling fires for structural design - Part II: Design methodology*, Fire Safety Journal **54** (2012), no. 0, 96–112.

[20] G. A. Trunfio, D. D'Ambrosio, R. Rongo, W. Spataro, and S. Di Gregorio, *A new algorithm for simulating wildfire spread through cellular automata*, ACM Transactions On Modeling And Computer Simulation **22** (2011), no. 1, 6:1–6:26.

[21] D. R. Weise and G. S. Biging, *Effects of wind velocity and slope on flame properties*, Can. J. For. Res **26** (1996), 1849–1858.

[22] X. Yan, F. Gu, X. Hu, and S. Guo, *A dynamic data driven application system for wildfire spread simulation*, Proceedings Of The 2009 Winter Simulation Conference (WSC 2009), Vol 1-4, 2009, pp. 2972–2979 (English). Winter Simulation Conference 2009, Austin, TX, DEC 13-16, 2009.

[23] S. Zhao, *GisFFE, an integrated software system for the dynamic simulation of fires following an earthquake based on gis*, Fire Safety Journal **45** (January 2010), 83–97.

ANEXO II

SCENE GENERATOR: UTILIDAD PARA LA GENERACIÓN DE ESCENARIOS

Durante el proceso de análisis y validación de los algoritmos de propagación de incendios se ha visto necesario la creación de una pequeña utilidad que permita generar algunos escenarios para tests de validación. Crearlos manualmente habría sido una gran pérdida de tiempo. Dado que un escenario, consiste en un archivo XML que contiene una descripción de cada celda del escenario, se ha optado por utilizar un lenguaje de programación que permitiera escribir el modelo de forma rápida y fuera fácilmente modificable. *Python* ha sido el lenguaje escogido debido a que es interpretado y permite crear un fichero de salida XML fácilmente.

En el script denominado *SceneGenerator* se ha hecho uso de la librería *PySide* como binding con *Qt*, de modo que se le ha añadido una simple interfaz.

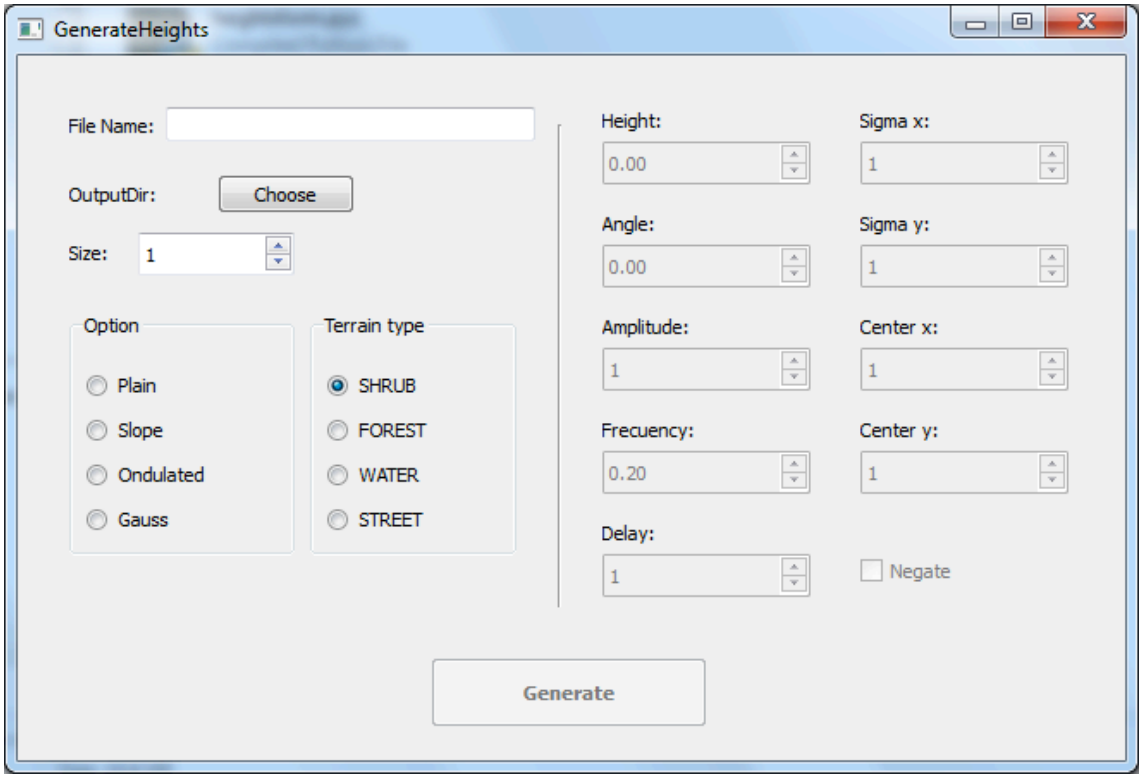


Figura 76 - GUI del SceneGenerator

Mediante la interfaz del generador de escenarios podemos generar:

- Escenarios horizontales a una altura y tamaño

- Escenarios con la pendiente deseada 
- Escenarios con oscilaciones 
- Escenarios con una pendiente gaussiana 

Formato de salida

El siguiente ejemplo muestra la representación de una celda dentro del entorno forestal, en formato XML:

```
<OUTSIDE>
    <GENERAL>
        <type>SHRUB</type>
        <height>0.0</height>
        <name>Outside Unit</name>
        <Xposition>44</Xposition>
        <Yposition>50</Yposition>
    </GENERAL>
    <SPECIFIQUEOUTSIDE>
        <FirstStory>false</FirstStory>
        <food>200</food>
    </SPECIFIQUEOUTSIDE>
</OUTSIDE>
```

Las etiquetas tienen el siguiente significado:

- **Type**: El tipo de vegetación.
- **Height**: La altura de la celda sobre el nivel del mar u otra referencia local.
- **Xposition** y**Yposition**: Indica la posición 2D en unidades de celda.
- **Name**: Etiqueta para la celda.
- **Food**: La cantidad de material combustible de la que está compuesta.
- **FirstStory**: Indicando si la celda es iniciada con fuego o no.

El siguiente ejemplo muestra la representación de una celda dentro del entorno urbano, en formato XML:

```
<BUILDING>
    <GENERAL>
        <type>BUILDING</type>
        <height>0</height>
        <name>A</name>
        <Xposition>16</Xposition>
        <Yposition>15</Yposition>
    </GENERAL>
    <SPECIFIQUEBUILDING>
        <numBuilding>0</numBuilding>
        <nFloor>2</nFloor>
    </SPECIFIQUEBUILDING>
    <FLOORS>
        <FLOOR>
            <numFloor>0</numFloor>
            <type>SecureUnit</type>
            <firstStory>false</firstStory>
            <windowSize>1</windowSize>
        </FLOOR>
        <FLOOR>
            <numFloor>1</numFloor>
            <type>SecureUnit</type>
            <firstStory>false</firstStory>
            <windowSize>1</windowSize>
        </FLOOR>
    </FLOORS>
</BUILDING>
```

Las etiquetas tienen el siguiente significado:

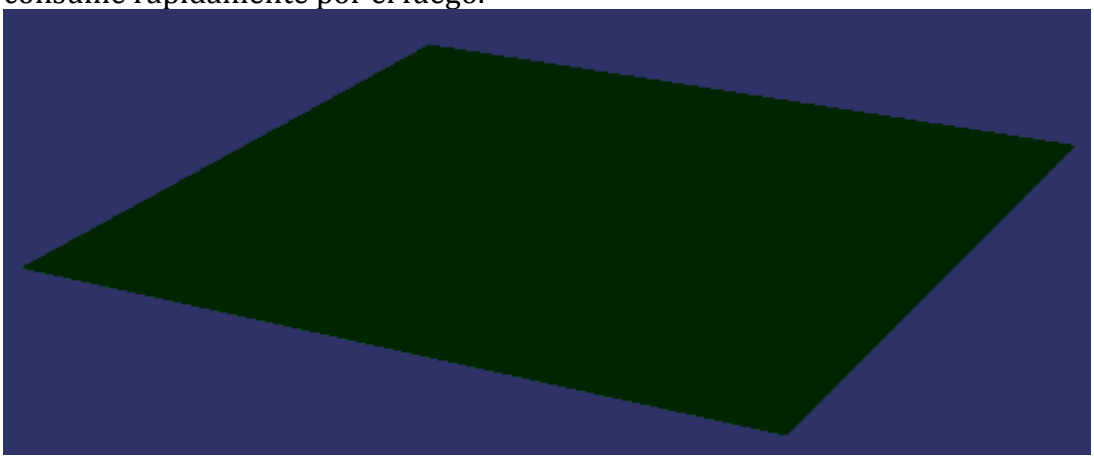
- Datos de carácter general.
    - **type**: Tipo de edificio.
    - **height**: Altura de la celda base.

    - **name**: Etiqueta del edificio.
    - **Xposition** y **Yposition**: Indicando la posición de la celda base.
- Datos que definen el edificio.
    - **numBuilding**: El id dentro del bloque que conforma un edificio.
    - **nFloor**: El número de plantas del edificio.
- Por cada planta del edificio encima de la celda base.
    - **numFloor**: El número de planta al que corresponde.
    - **type**: El tipo de edificio.
    - **firststory**: Indica si la planta tiene fuego inicialmente.
    - **windowSize**: Tamaño de la ventana

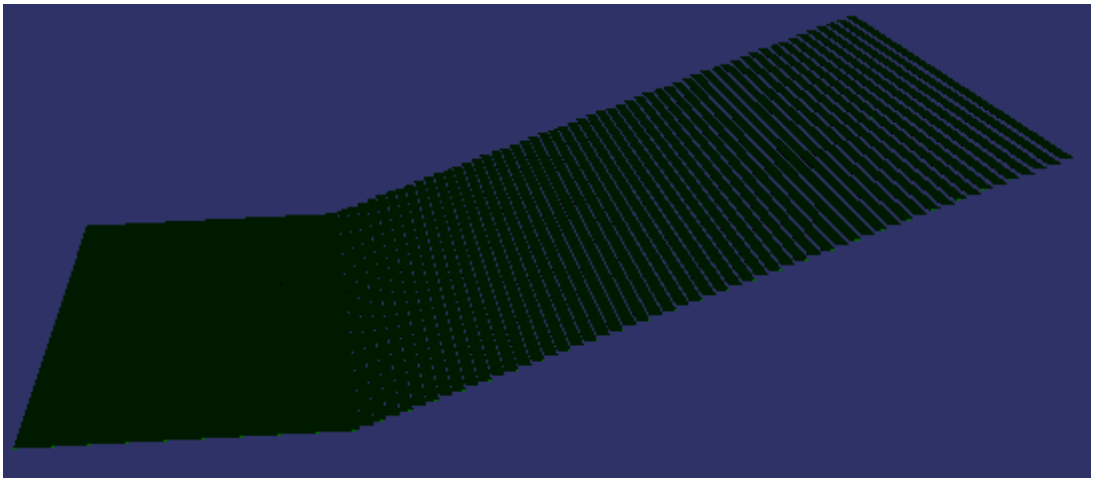
Ejemplos de escenarios

Con *SceneGenerator*, se han generado los siguientes escenarios:

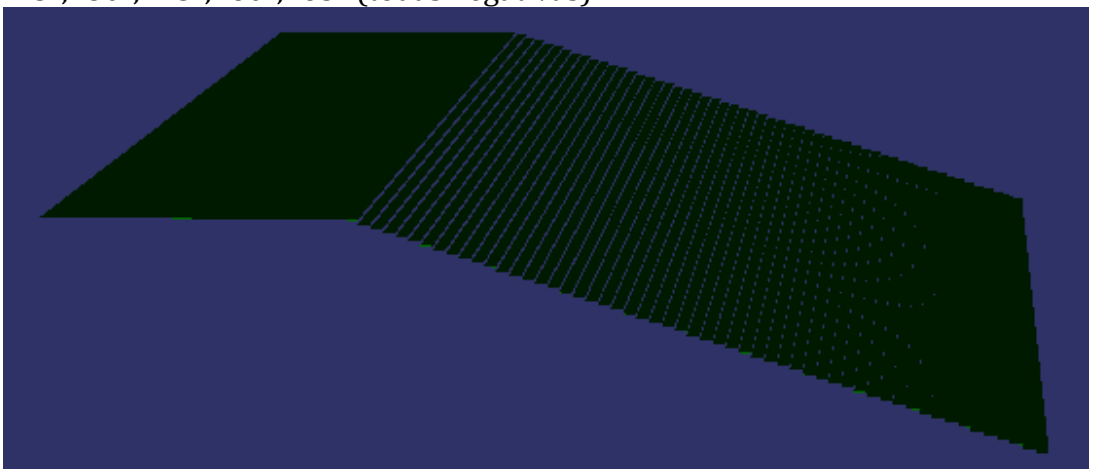
- **Horizontal**: El escenario consiste en un plano horizontal, compuesto de un solo tipo de vegetación tipo *SHRUB.* El tipo *SHRUB* es una vegetación que se consume rápidamente por el fuego.



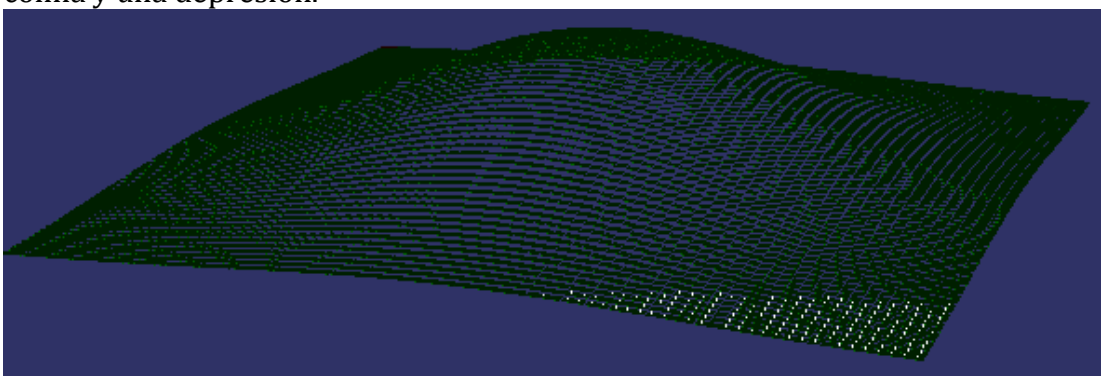
- **Pendiente+Y:** El escenario lo conforman dos planos, un plano horizontal y en su extremo el inicio de otro plano con un grado de inclinación predefinido. Las pendientes generadas para este segundo plano han sido de 15°, 30°, 45°, 60°, 65° (todas positivas).



- **Pendiente-Y:** El escenario lo conforman dos planos, un plano horizontal y en su extremo el inicio de otro plano con un grado de inclinación predefinido. Las pendientes generadas para este segundo plano han sido de -15°, -30°, -45°, -60°, -65° (todas negativas).

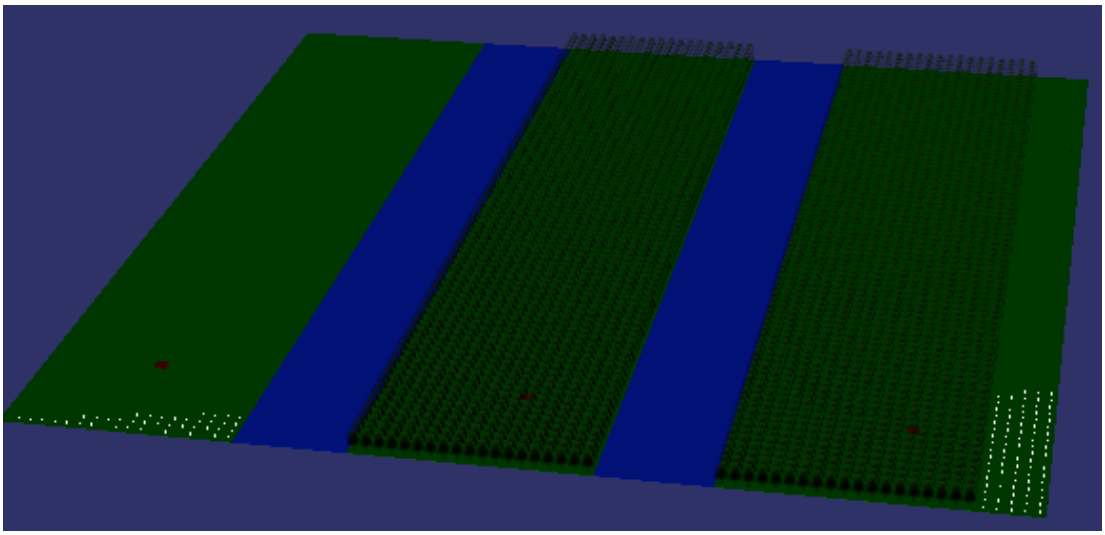


- **Gauss**: Un escenario con una pendiente que representa una curva Gaussiana bidimensional positiva y otra negativa, con el fin de recrear una colina y una depresión.

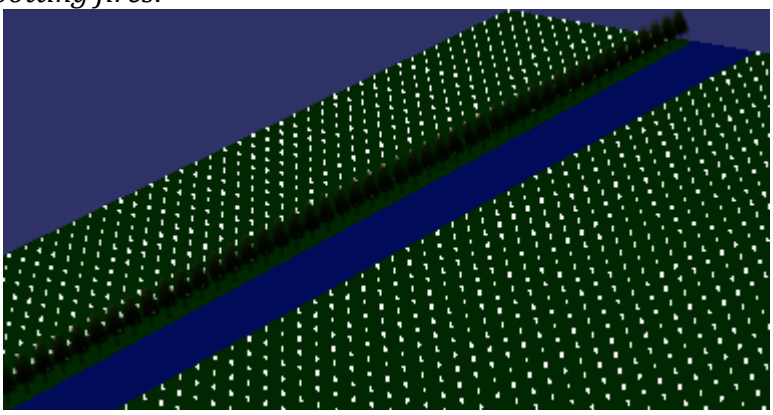


Los demás escenarios se han generado manualmente con las herramientas *BuildingFire* y *SceneGenerator*, han consistido en:

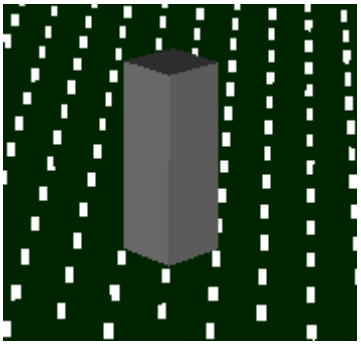
- **Tiras**: El escenario está compuesto por tres secciones separadas por dos barreras de agua. Cada sección está hecha con un tipo de vegetación diferente y está configurado de tal forma, que el incendio se produzca de manera simultánea en las tres zonas.



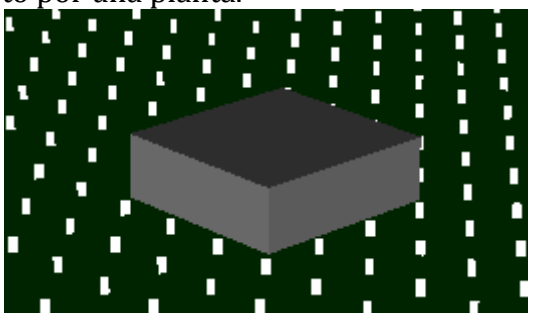
- **Barrera**: El escenario está dividido por un rio de diferentes grosores, el rio actúa de cortafuegos impidiendo la propagación. A un lado del rio se coloca vegetación con gran altura, para favorecer la propagación mediante el método *spotting fires*.



- **Urbano – 1x1x3:** Un escenario horizontal con un edificio de una unidad compuesto por tres plantas.



- **Urbano – 1x3x3**: Un escenario horizontal con un edificio de nueve unidades compuesto por una planta.



- **Urbano – 2edif:** Un escenario horizontal con dos edificios de una unidad compuesto por tres plantas separados a diferentes distancias. Las distancias utilizadas han sido desde tres a doce metros.