UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Carrera de Ingeniero Informático

SIMULACIÓN DE LA PROPAGACIÓN DE INCENDIOS FORESTALES MEDIANTE AUTÓMATAS CELULARES

NOMBRE DEL ESTUDIANTE : Jesús David Latorre García

Dirigido por: Alfonso Urquía Moraleda

Curso: 2014/15

SIMULACIÓN DE LA PROPAGACIÓN DE INCENDIOS FORESTALES MEDIANTE AUTÓMATAS CELULARES

Proyecto de Fin de Carrera de modalidad *oferta específica, modalidad B*

Realizado por: JESÚS DAVID LATORRE GARCÍA (firma)

Dirigido por: ALFONSO URQUÍA MORALEDA(firma)

Tribunal calificador:

Presidente: D./Da. .....
(firma)

Secretario: D./Da. .....
(firma)

Vocal: D./Da. .....
(firma)

Fecha de lectura y defensa: .....

Calificación: .....

RESUMEN

Realizamos una implementación en el lenguaje Java del autómata celular para la predicción de la extensión de incendios forestales desarrollado en el trabajo de Alexandridis et Alter, "A cellular automata for forest fire spread prediction: The case of the wildfire that swept through Spetses Island in 1990".

Se comienza con un breve repaso de la teoría de los autómatas celulares. Se continúa con una aproximación a diferentes modelos predictivos de la extensión de incendios forestales. Se indican diferentes paradigmas, en especial el de la simulación de los modelos por medio de autómatas celulares. Se realiza, igualmente, una aproximación al modelo "Cell-DEVS".

La implementación del autómata celular se realiza sobre el marco geográfico de la isla de Spetses, en Grecia. A partir de una imagen tomada con Landsat se ejecuta un algoritmo de clasificación de la imagen, denominado de Distancia Mínima, mediante el cual se obtiene la tipología del terreno. Sobre los datos del tipo y la densidad de la vegetación y de la elevación del mismo, y con los parámetros dados por el usuario de la velocidad y dirección del viento y de las coordenadas del foco inicial del incendio, se realiza la simulación en Java de la forma de la extensión del mismo. Se ofrecen igualmente varias simulaciones de prueba.

En el propio programa se expone toda la formulación del autómata de Alexandridis, junto con una explicación del algoritmo de la "Distanca Mínima" empleado para la clasificación de la imagen. El programa se ejecuta en un panel de pestañas, estando cada pestaña asociada a una parte del mismo. Se parte de una pestaña inicial, donde se explica la formulación empleada en el autómata. A continuación se pasa a la pestaña "vegetación" donde se expone el algoritmo de Distancia Mínima, los pasos intermedios del mismo y el resultado de su ejecución. Una vez obtenidos los datos de la densidad de la vegetación, a partir de los mismos se realiza una clasificación del tipo de combustible vegetal, lo cual se expone en la tercera pestaña, "combustibles". La siguiente pestaña, "viento", explica la influencia del viento en la formulación del modelo. Después, en la pestaña "elevación" se exponen los datos de la elevación del terreno y, finalmente, en la última pestaña, "autómata", se solicita al usuario la introducción de los parámetros restantes a fin de proceder a la ejecución del autómata. A su vez, se xponen varias ejecuciones de prueba.

Se realizan las pruebas de validación para comprobar la adecuación de la implementación realizada, y se presenta un manual de uso de la misma.

Se realiza, igualmente, una aproximación al mismo modelo de Alexandridis et Alter realizada en NetLogo, para comprobar la adecuación de NetLogo para la

simulación de autómatas celulares en particular y más específicamente para la predicción de la extensión de incendios forestales.

LISTA DE PALABRAS CLAVE

Java

Autómata celular

Incendio forestal

Algoritmo de clasificación de imágenes de Distancia Mínima

NetLogo

Modelado de eventos discretos

Simulación por ordenador

ABSTRACT

We present in the Java programming language an implementation of the cellular automaton for the prediction of the extension of wildfires given by the work of Alexandridis et Alter, "A cellular automata for forest fire spread prediction: The case of the wildfire that swept through Spetses Island in 1990".

We begin with a brief review of the cellular automata theory. Then, with an approximation to the different predictive models of wildfire extension. Various paradigms are named, in particular the simulation of models through cellular automata. The Cells-DEVS model is also viewed.

The implementation of the cellular automata is made using the geographical frame of the island of Spetses, Greece. Starting with a Landsat image of the island, an image classification algorithm, namely the "minimal distance" one, is executed in order to obtain the typology of the terrain. From the data obtained, type and density of the vegetation and elevation of the terrain, and with the parameters given by the user, wind velocity and direction and initial ignition point, a Java simulation of the shape of the wildfire extension is produced. Various examples with different initial parameters are given automatically by the program.

All the formulation used in the Alexandridis automaton is indicated in the program, next to an explanation of the "minimal distance" algorithm used in the image classification. The program is displayed as a "tab panel", where every tab is associated to a different part of the model. An initial tab explains all the principles and formulas used in the automaton. The following tab, "vegetation", explains the "minimal distance" algorithm, its intermediate steps and the results of its execution. Once the data of the density of the terrain is obtained this data is used to classify the type of the vegetal "fuel", which is described in the third tab. The next tab, "wind", shows the influence of the wind in the model formulation. The following tab, "elevation", displays the terrain elevation data and, finally, in the last tab, "automaton", the user is asked to give the rest of the parameters necessary for the execution of the Alexandridis model. With all the data the user may execute the simulation. A few example executions are given.

Some tests are executed to verify the validity of the implementation, and a user´s guide is produced.

Finally, we attach an approximation to the same Alexandridis et Alter model made with NetLogo, so that we test the adequacy of the use of NetLogo for the simulation of wildfire extension prediction through cellular automata models

KEYWORDS

Java

Cellular automaton

Wildfire

Minimal Distance algorithm for image classification

NetLogo

Discrete-event modelling

Computer simulation

ÍNDICE

LISTA DE FIGURAS

LISTA DE CÓDIGO

1 INTRODUCCIÓN, OBJETIVOS Y ESTRUCTURA

1.1 INTRODUCCIÓN

En el campo de la simulación, los modelos predictivos de incendios forestales consisten, en general, en un conjunto de ecuaciones cuya solución es uno o varios valores numéricos que sirven para modelar la evolución espacio-temporal del incendio.

Conforme a la formulación matemática mencionada, los modelos se pueden clasificar en tres tipos principales [Ljijlana, 2006] [Pastor, 2003]:

1. Modelos teóricos: se basan en las leyes de la termodinámica, tomando en consideración las leyes que gobiernan la mecánica de fluidos, la combustión y la transferencia de calor.

2. Modelos empíricos: las ecuaciones se derivan de correlaciones estadísticas extraídas de fuegos reales y/o simulados sobre un entorno particular.

3. Modelos semiempíricos: partiendo de ecuaciones teóricas, se realizan simulaciones sobre el entorno. A partir de los datos obtenidos, las ecuaciones teóricas se refinan y adaptan para el caso particular simulado.

En relación a los modelos semiempíricos, cualquier modelo predictivo de la extensión de un incendio forestal toma como entradas valores tanto meteorológicos como aquellos relacionados con las características propias del terreno forestal en consideración. Así, los factores principales que afectan al grado de la extensión del incendio son el combustible vegetal (tipo y densidad de la vegetación), la velocidad y dirección del viento y la inclinación del terreno.

Una vez planteado el modelo, se emplean técnicas computacionales para la implementación del mismo, construyendo de esta manera un simulador predictivo de la extensión del incendio forestal.

La simulación se realiza sobre una representación espacial del área geográfica donde se va a simular el incendio. Para modelar la propagación del fuego a lo largo del paisaje forestal se utilizarán, en general, una de las dos técnicas siguientes [Pastor, 2003]:

1. Representación del paisaje en redes ("grids", parrillas en inglés), por medio de las técnicas de percolación o de autómatas celulares.

2. Representación del paisaje en plano continuo (propagación por ondas, "wave propagation").

1.2 OBJETIVOS

Este trabajo implementa el modelo de autómata celular desarrollado por Alexandridis et Alter, "A cellular automata for forest fire spread prediction: The case of the wildfire that swept through Spetses Island in 1990" [Alexandridis, 2008].

Dicho autómata celular se codifica en el lenguaje de programación Java, utilizando el editor Eclipse. También se realizará una implementación del mismo en el lenguaje NetLogo.

La aplicación toma como datos de entrada una imagen Landsat, de la cual extrae datos de tipo y densidad de la vegetación del área representada en dicha imagen, y una imagen "png" representando la elevación del terreno correspondiente a ese mismo área.

Se emplea el algoritmo de clasificación de imágenes de Distancia Mínima para la obtención de los datos de la densidad de la vegetación [Santos].

Los datos de la elevación del terreno se extraen del procesado de los colores RGB de la imagen de la elevación del terreno.

Se solicita igualmente del usuario los siguientes datos:

1. Velocidad del viento.
2. Dirección del viento.
3. Coordenada x del foco inicial del incendio.
4. Coordenada y del foco inicial del incendio.
5. Velocidad del simulador (número de milisegundos por paso).

A partir de dichos datos de entrada (las imágenes y los introducidos por el usuario) el programa debe realizar una simulación de la extensión de un incendio forestal conforme a la formulación del modelo de Alexandridis.

La aplicación, a su vez, debe ser "auto explicativa", de forma que el usuario pueda ejecutarla sin necesidad de prácticamente ningún aprendizaje previo, simplemente leyendo la información que aparece en la misma.

Para conseguir estos objetivos se plantea realizar las tareas siguientes:

1. Búsqueda bibliográfica de diferentes modelos de implementación de autómatas celulares para la simulación de la propagación de la extensión de los incendios forestales. Para ello, será necesario adentrarse previamente en los fundamentos teóricos de los autómatas

celulares y en los diferentes conceptos relativos a la simulación de la propagación de los incendios forestales.

2. Elección de un modelo de simulación de la propagación de incendios forestales para su posterior implementación. Se escogerá un modelo, se presentará su fundamentación teórica y su formulación matemática y se expondrán los resultados obtenidos por el mismo.

3. Implementación en el lenguaje Java del modelo elegido. Se deberá realizar la aplicación y explicar el diseño y la codificación de la misma.

4. Realización de las pruebas que permitan deducir un correcto funcionamiento de la aplicación y su adecuación al modelo de simulación elegido.

5. Realización de un manual de uso de la aplicación implementada.

6. Establecimiento de las conclusiones obtenidas de la realización de la aplicación en cuanto a comprensión del dominio, factibilidad del desarrollo y posibles mejoras.

7. Finalmente, realización de la misma simulación en el entorno del lenguaje de sistemas "multiagente" NetLogo, a fin de estudiar las posibilidades de aplicación de dicho lenguaje para la simulación mediante autómatas celulares.

1.3 ESTRUCTURA

Esta memoria se estructura en varios capítulos, cuyos contenidos describimos a continuación:

1. Capítulo 1: Introducción, objetivos y estructura. Es el capítulo en curso.

2. Capítulo 2: Metodología de modelado. Se introducen los conceptos principales relacionados con los modelos predictivos de incendios forestales. Se exponen diferentes aproximaciones. Se describe el modelado mediante autómatas celulares y su aplicación a la predicción de los incendios forestales. Se realiza, igualmente, una introducción al modelo de Cell-DEVS.

3. Capítulo 3: Descripción del modelo implementado. Se describe con detalle el modelo de propagación de incendios seleccionado (el modelo de Alexandridis et alter).

4. Capítulo 4: Diseño e implementación de la aplicación. Se describe la aplicación desarrollada en Java que implementa el modelo descrito en el Capítulo 3.

5. Capítulo 5: Pruebas realizadas al simulador. Se describen las pruebas realizadas al simulador, a fin de verificar y validar el mismo.

6. Capítulo 6: Modo de empleo del simulador. Se describe, paso a paso, el modo de empleo del simulador.

7. Capítulo 7: Representación en NetLogo. Se realiza la implementación del modelo en NetLogo.

8. Capítulo 8: Planificación y presupuesto. Se describen las fases de desarrollo del proyecto, su secuencia temporal y un cálculo de los costes asociados.

9. Capítulo 9: Conclusiones y trabajos futuros. Se realiza una conclusión final del trabajo realizado y se plantean posibles mejoras futuras.

Finalmente, se proporciona un listado de la bibliografía empleada.

El código fuente, tanto de la implementación en Java como de la implementación en NetLogo, se encuentra en el CD adjunto. La estructura de dicho CD es la siguiente:

1. Carpeta "AlexandridisEtAlter". Contiene los archivos necesarios para ejecutar la aplicación en Java . En la carpeta, se añade un archivo de instrucciones sobre el uso de la aplicación.
2. Carpeta "ProyectoNetLogo". Contiene los archivos necesarios para ejecutar la aplicación en NetLogo. En la carpeta, se añade un archivo de instrucciones sobre el uso de la aplicación.
3. Archivo "instrucciones generales": archivo de texto que indica lo expuesto en esta misma numeración.
4. Archivo "PFC_DavidLatorre.docx": memoria del proyecto en formato word.
5. Archivo "PFC_DavidLatorre.pdf": memoria del proyecto en formato pdf.

2 MODELADO DE INCENDIOS FORESTALES

2.1 INTRODUCCIÓN

Existen diferentes modelos para la predicción de los incendios forestales.

En este capítulo realizaremos, en primer lugar, una clasificación de los modelos predictivos de incendios forestales, los cuales se pueden dividir en teóricos, empíricos o semiempíricos [Ljijlana, 2006] [Pastor, 2003].

Entre las diferentes formas de representar el paisaje a simular presentaremos los Autómatas Celulares, para lo cual realizaremos una revisión teórica del concepto de Autómata Celular para, a continuación, estudiar la aplicación de los mismos al caso de la predicción de los incendios forestales.

Presentaremos, seguidamente, un breve repaso de varios modelos predictivos de extensión de incendios forestales basados en Autómatas Celulares, realizando una discusión crítica sobre los mismos y unas conclusiones finales al respecto.

Existe una extensión del concepto de los Autómatas Celulares, la cual es el paradigma Cell-DEVS. Revisaremos dicho paradigma y su aplicación a la simulación de la predicción de los incendios forestales.

Finalmente, veremos las diferentes herramientas de modelado existentes para las simulaciones por ordenador y enumeraremos las utilizadas en la realización de este proyecto.

2.2 CLASIFICACIÓN DE LOS MODELOS PREDICTIVOS DE INCENDIOS FORESTALES

Los modelos predictivos de extensión de incendios forestales se basan, en general, en un conjunto de ecuaciones cuya solución define la evolución espacio temporal de una serie de variables, tales como velocidad de la propagación del perímetro del fuego, altura de la llama, riesgo de ignición y consumo de combustibles, entre otras [Ljijlana, 2008], [Pastor, 2003].

Conforme a las anteriores ecuaciones, los modelos se pueden clasificar en [Ljijlana, 2006] [Pastor, 2003] :

1. Teóricos: se usan las leyes físicas relativas a la mecánica de fluidos, combustión y transferencia de calor, de los cuales se deducirán las ecuaciones a aplicar.

2. Empíricos: a partir de los datos extraídos de fuegos en laboratorios o de fuegos reales, se formulan, mediante métodos de correlación estadística, las ecuaciones.

3. Semiempíricos: son una combinación de los dos métodos anteriores; a partir de un modelo teórico, las ecuaciones se refinan mediante experimentación y datos de fuegos reales.

Una vez construido el modelo, se pueden emplear técnicas computacionales para la implementación del mismo, construyendo, así, un simulador de extensión de incendios forestales.

Los simuladores de incendios forestales requieren una representación espacial del paisaje donde se va a simular el incendio. Entre las diferentes técnicas para representar el paisaje se utilizan, entre otras, los Sistemas de Información Geográfica o SIG (en inglés, GIS, Geographical Information Systems [GIS]), los cuales proporcionan representaciones digitales de las coordenadas espaciales. Otra posibilidad es la realización del tratamiento digital de las imágenes dentro de la misma aplicación, para los casos en que no se requiera un gran desplazamiento sobre el paisaje a simular. En ambos casos la propagación del fuego a lo largo del paisaje se realiza utilizando, por lo general, alguna de las técnicas siguientes [Pastor, 2003]:

1. Representación del paisaje en redes regulares (percolación, "bond percolation", y autómatas celulares).

2. Representación del paisaje en plano continuo (propagación por ondas, "wave propagation").

2.3 AUTOMATAS CELULARES

Supongamos una superficie de terreno vista desde arriba. Dividamos esta superficie en un número m de filas y en un número n de columnas, es decir, crearemos una red o malla de m*n elementos. Si a cada uno de estos elementos le llamamos celda, dispondremos, en la superficie, de m*n celdas o células. A cada célula le corresponderá una coordenada (x,y) en un espacio cartesiano o similar, en función de las coordenadas empleadas en el SIG o en el sistema de superficie que queramos modelar. Lo anterior es, *grosso modo*, la forma inicial de enlazar un Sistema de Información Geográfica o cualquier otro sistema de representación del terreno con la red de los autómatas celulares. Veamos ahora la definición estructural de los Autómatas Celulares.

Los autómatas celulares permiten modelar procesos dinámicos complejos empleando coordenadas temporales y espaciales discretas, e interacciones entre sus elementos sólo a nivel local. Fueron ideados y propuestos por Von

Neumann y por Stanislaw Ulam, bien cada uno por separado, bien conjuntamente (ambos científicos trabajaron en muchos proyectos conjuntos, especialmente en la bomba de Alamo Gordo, como se puede comprobar en la autobiografía de Stanislaw L. Ulam [Ulam, 2001]).

Desde un punto de vista más formal, un autómata celular es una red o matriz n-dimensional. En cada ubicación distinguible de dicha red tendremos una celda. Cada celda poseerá un estado, y el estado de dicha celda en la transición temporal siguiente dependerá de dicho estado y de los estados de las células de la vecindad.

Así, el Autómata Celular (en adelante, AC) se definirá mediante las siguientes propiedades [karafyllidis, 1997] [Quartieri, 2010] [Ljijlana, 2006]:

1. La estructura física de la red. La dimensión de la red puede ser 1 (ECA, Elementary Cellular Automaton, una línea de células), 2 (una red bidimensional o "parrilla") o superior. En general, la red será bidimensional, superpuesta a un espacio euclídeo. Las celdas suelen ser rectangulares o cuadradas, pero también pueden tener otras topologías (como hexagonal, [D'Ambrosio, 2006]). La anchura de cada dimensión suele ser finita.

2. Los estados de las celdas. Sea $\boldsymbol{S}$ un conjunto finito y no vacío de estados. Cada célula del AC poseerá uno y uno sólo de los estados $\boldsymbol{S_i} \in \boldsymbol{S}$. La Configuración Global del AC será el conjunto de los estados de todas las celdas del mismo.

3. La vecindad de las celdas. Cada celda poseerá un esquema de vecindad, de tal manera que el siguiente estado de dicha celda dependerá de su estado actual y de los estados de las celdas pertenecientes a dicha vecindad. En el espacio de dos dimensiones las vecindades más utilizadas son la de Von Neumann y la de Moore. En la Figura 2.1 se observan las vecindades de Von Neumann y de Moore para dos dimensiones, las utilizadas generalmente en los modelos de AC para la simulación de la extensión de incendios forestales.

4. La regla de transición de estados. La regla *f* se aplica localmente a cada célula, de forma que el estado de dicha célula cambia en cada paso discreto de la transición y dicho estado siguiente depende del estado actual de dicha celda y del estado actual de las celdas de su vecindad. Así, $\boldsymbol{S_{i,j}(t+1)=f(S_{i,j}(t),S_{vecindad(i,j)}(t))}$. Todas las celdas del AC aplicarán, de forma local, dicha función f, de manera que en el instante t+1 la configuración global del AC será el conjunto de los estados de cada celda del AC en dicho instante t+1.

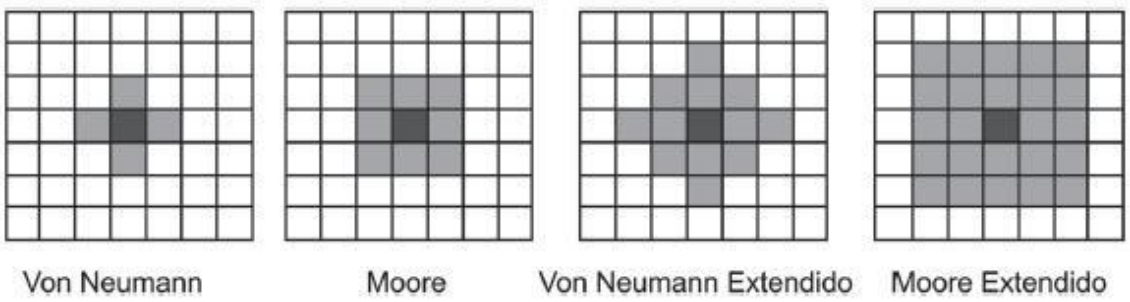


Figura 2. 1: Vecindarios de Von Neumann y de Moore

2.4 APLICACIÓN DE LOS AC A LA PREDICCIÓN DE INCENDIOS FORESTALES

Conforme a uno de los pioneros en el estudio de un modelo teórico de la propagación de la extensión de los incendios forestales [Fons, 1946], un modelo predictivo ha de considerar como principales los siguientes factores relacionados con las características del terreno sobre el que se va a realizar el modelo y con las condiciones meteorológicas de la atmósfera :

1. Tipo de la vegetación ("fuel type" o tipo del combustible vegetal). El combustible vegetal, es decir, lo que va a arder, posee unas características químicas que determinarán la cantidad de calor desprendida en su combustión, etc.. Igualmente, es importante la distribución de cada tipo de vegetación sobre cada unidad de superficie del terreno, bien se mida en densidad, bien en número de árboles o arbustos por unidad de superficie, etc..

2. Humedad de la vegetación (la cantidad de humedad contenida en dicho combustible vegetal).

3. Velocidad y dirección del viento.

4. Topografía del terreno: aquí se incluye la pendiente del mismo ("slope"), la presencia de barreras naturales no combustibles (ríos, rocas, etc.) así como la de elementos antropogénicos no combustibles (carreteras y otros).

A partir de las formulaciones de Fons, uno de los modelos empíricos más importantes y más utilizados en las simulaciones es el de Rothermel [Pastor, 2003], el cual, basándose en resultados de experimentos realizados en laboratorio, deduce una serie de ecuaciones dinámicas para calcular la velocidad de propagación del fuego en el perímetro del mismo. Las ecuaciones

del modelo de Rothermel ([Rothermel, 1972], [Rothermel, 1983]) han sido utilizadas en bastantes sistemas de simulación, los cuales, según la representación del espacio, se pueden clasificar en dos tipos [Alexandridis, 2008]:

1. Modelos basados en planos contiguos, donde se asume que el fuego avanza, en cada punto del perímetro, conforme a un sistema de ondas elípticas. La solución a estos modelos se realiza por medio de ecuaciones diferenciales parciales, las cuales son computacionalmente lentas.

2. Modelos basados en redes ("grid", cuadrícula o rejilla), los cuales son computacionalmente más rápidos.

Los modelos basados en redes se subdividen, a su vez, en modelos basados en percolación ("bond percolation") y modelos basados en Autómatas Celulares [Pastor]. Los modelos de percolación predicen, de manera eficiente, la forma del área quemada, pero no reproducen adecuadamente la dinámica del fuego. Esta debilidad de los modelos "bond percolation" se subsana mediante los modelos basados en autómatas celulares. Este hecho, sumado a la posibilidad de enlazar la representación espacial en los Sistemas de Información Geográfica de forma relativamente sencilla en la red n-dimensional de un AC hace que los AC sean un modelo empleado por varios sistemas de simulación de la extensión de los incendios forestales [Alexandridis, 2008].

2.5 DISCUSIÓN CRÍTICA SOBRE VARIOS MODELOS DE AC DE INCENDIOS

Partiendo de la base teórica de los AC, los diferentes modelos van desde simples simulaciones que no tienen en cuenta ni los fenómenos meteorológicos, ni el tipo de vegetación ni la topografía del terreno sobre el que se ejecutan, hasta modelos con reglas de transición de estados que sí tienen en cuenta meteorología, topografía y vegetación. Dentro de éstos últimos, se pueden distinguir modelos con reglas matemáticamente complejas, generalmente basadas en los estudios de Rothermel, de otros con reglas simplificadas mediante factores ponderativos y constantes empíricas, las cuales, a su vez, también suelen basarse en las fórmulas de Rothermel.

Realizaremos un breve repaso de los modelos que hemos estudiado, comenzando por los más sencillos y yendo hacia los más complicados.

2.5.1 MODELO DE QUARTIERI

[Quartieri, 2010]: es un modelo muy sencillo y fácilmente realizable matemáticamente.

1. Vecindad de Moore.

2. Cuatro estados posibles para cada celda (no combustible, combustible, ardiendo, quemada).

3. La regla de transición de estados es sencilla, basada en una probabilidad *p* que depende del número de células de la vecindad que están ardiendo.

4. El modelo no incluye variables meteorológicas ni topográficas, sólo la presencia de vegetación combustible o no combustible, indicando que se añadirán las variables anteriores en sucesivos refinamientos del estudio.

5. Realiza una simulación de una imagen de un terreno por satélite, convirtiendo dicha imagen a una red bidimensional y considerando como factores variables únicamente la presencia de materiales combustibles o no combustibles.

Es, pues, un modelo muy sencillo y necesitado de trabajo posterior.

2.5.2 MODELO DE LJILJANA

[Ljiljana, 2006]: modelo algo más complejo que el de [Quartieri] y que usa la tecnología SIG, con datos extraídos del CORINE Land Cover [Corine L.C.].

1. Cuatro estados posibles para cada celda (no combustible, combustible, ardiendo, quemada).

2. Emplea el Sistema de Información Geográfica GRASS GIS [Grass G.I.S.] para tratar la simulación, introduciendo datos extraídos del CORINE Land Cover, de la Agencia Europea del Medio Ambiente.

3. A los datos de la vegetación del CORINE se le aplican diferentes densidades de probabilidad de combustión, dividiendo dicha vegetación en no combustible, 30% combustible o 70% combustible, en función del tipo de vegetación.

4. Conforme a la densidad anterior, se divide el terreno en las células del autómata celular, y a cada célula se le aplica un estado, siendo éste el de combustible en función de la densidad de probabilidad de combustión anteriormente indicada.

5. La regla de transición de estados es sencilla, y sólo tiene en cuenta el factor de la dirección del viento.

Modelo matemáticamente sencillo y algo más elaborado que el de Quartieri [Quartieri, 2010], con el añadido de que incorpora la tecnología SIG y, además, hace uso del programa CORINE.

2.5.3 MODELO DE KARAFYLLIDIS

[Karafyllidis, 1997]: modelo matemáticamente más complejo que hace uso de factores de ponderación para simular la inclinación del terreno y la velocidad y dirección del viento.

1. Vecindad de Moore.

2. El estado inicial de cada celda del AC depende del área quemada de la misma respecto a su área total. Se parte de una matriz en la cual, a cada celda, se asigna dicho estado inicial, en función de las condiciones del terreno a simular.

3. La regla de transición de estados utiliza un factor geométrico en la propagación del fuego y factores de ponderación empíricos para simular la velocidad y dirección del viento, a la vez que un factor directo de multiplicación en función de la inclinación del terreno.

4. El tiempo t entre dos pasos sucesivos de la simulación depende de un campo escalar R de velocidades de expansión del fuego en áreas globales de la matriz de simulación. Este campo ha de venir de otro modelo, [Karafyllidis, 1997] no especifica de cuál.

El modelo posee un tiempo computacional rápido, pero adolece de la necesidad de encontrar un campo escalar R inicial de la velocidad de propagación del fuego, proveniente de un modelo externo, así como de la necesidad de unos factores de ponderación de la velocidad y dirección del viento que deberían estar más explicados en relación a la forma de su cálculo. La ventaja de este modelo es que no hace uso de ecuaciones matemáticas complejas, y sería factible de implementar haciendo uso de la tecnología SIG.

2.5.4 MODELO DE CLARKE

[Clarke, 1994]: modelo matemáticamente complejo que hace uso de la teoría de Agregación Limitada por Difusión y emplea datos en tiempo real.

1. Vecindad de Moore.

2. La regla de transición de estados se basa en factores de ponderación aplicados a las células de la vecindad para aplicar las variables de velocidad y dirección del viento, humedad relativa, temperatura del aire y contenido de humedad del combustible. A partir del tipo de vegetación de la célula y de las de su vecindad (tipo de combustible e inclinación del

terreno), se asigna una probabilidad de ignición de las células circundantes empleando el algoritmo de Monte Carlo.

3. La regla de transición anterior semeja una Agregación Limitada por Difusión (DLA, Diffussion Limited Aggregation), la cual es, en teoría, similar a los fractales.

4. Se emplean sensores de infrarrojos para calcular algunas de las variables anteriormente indicadas, otorgando la posibilidad de obtener datos en tiempo real. Una vez calibrados todos los datos anteriores para la matriz inicial, se considera que los datos de cada celda permanecerán constantes para todo el resto de la simulación (exceptuando, lógicamente, los estados de las celdas).

A nuestro entender, el modelo de Clarke [Clarke, 1994] es un modelo complicado, basado en las teorías de la Agregación Limitada por Difusión (al igual que en la de los fractales).

2.5.5 MODELO DE YASSEMI

[Yassemi, 2008]: modelo complejo que utiliza variables meteorológicas y topográficas en tiempo real junto con una regla de transición de estados afinada geométricamente, aplicándolo a una simulación basada en SIG.

1. Vecindad de Moore.

2. Los estados de cada célula varían de 0 (sin quemar) a 1 (quemada), pasando por todos los estados intermedios. Es, pues, un modelo con estados en una escala continua.

3. La regla de transición de estados se basa en la velocidad de avance del fuego (ROS) y en la dirección de avance del mismo (RAZ), obtenidas ambas en tiempo real, a partir del FBP (Fire Behaviour Prediction, del servicio meteorológico canadiense). Los parámetros anteriores, se calculan en el FBP a partir del tipo y distribución de la vegetación combustible en la zona de la simulación, la inclinación del terreno y el azimut (o acimut) cartográfico y la velocidad y dirección del viento, de manera empírica. Basándose en el RAZ y en el ROS, y utilizando fórmulas geométricas y factores de ponderación o variables discretas a tramos obtenidas del FBP, se calcula la proporción del área quemada de cada celda en función de las vecinas.

4. Se hace uso tanto de un SIG como de los datos del FBP, consiguiendo simulaciones adaptadas a las condiciones meteorológicas existentes respecto al tiempo cronológico de las mismas.

Modelo complejo pero, creemos, con una velocidad de computación rápida, y con la ventaja de utilizar información en tiempo real.

2.5.6 MODELO DE LOPES

[Lopes, 2002]: modelo basado en Rothermel que emplea SIG y presenta dos simulaciones posibles para los mismos datos de entrada, una más compleja y refinada, y otra más sencilla pero menos afinada. El sistema computacional de simulación realizado se denomina FIRESTATION.

1. Vecindad extendida a 16 celdas para corresponderse con un modelo elipsoidal de propagación del fuego.

2. Regla de transición de estados basada en el modelo de Rothermel, con datos obtenidos del FWI (Canadian Fire Weather Index). Puesto que la simulación del campo de vientos utiliza ecuaciones de Navier-Stokes (conjunto de ecuaciones en derivadas parciales no lineales que describen el movimiento de un fluido) y el cálculo computacional de las mismas conlleva un tiempo elevado, el modelo posee dos interfaces, la interfaz NUATMOS, que realiza, en un tiempo relativamente rápido, el cálculo mediante algoritmos de simplificación de dichas ecuaciones, y la interfaz CANYON, la cual realiza el cálculo canónico de las mismas, pero requiere un tiempo computacional mucho más elevado. Así, NUATMOS se utilizaría cuando fueran necesarias simulaciones en tiempo real, mientras que CANYON se emplearía en estudios donde no hay urgencia de obtención de los resultados.

3. Emplea SIG para la manipulación de las representaciones de la superficie.

Modelo matemático muy complejo con dos variantes, una sencilla y rápida y una compleja y lenta, con la ventaja de datos en tiempo real y la posibilidad de usar el simulador para pequeñas áreas o para la predicción en grandes superficies, como pueda ser el riesgo por zonas de incendios según la época del año en el área geográfica de todo Portugal.

2.5.7 MODELO DE ALEXANDRIDIS

[Alexandridis, 2008]: modelo muy completo para el cálculo de fuegos de superficie que incorpora el "spotting fire" (explicado más abajo).

1. Vecindad de Moore.

2. Cuatro estados posibles para cada celda (no combustible, combustible, ardiendo, quemada).

3. La regla de transición de estados emplea las siguientes variables: tipo y densidad de la vegetación, velocidad y dirección del viento, elevación del terreno y "spotting effect" (generación de nuevos puntos de ignición alejados del perímetro de avance del fuego debido a la caída de materiales ardiendo fuera de la zona de dicho perímetro, los cuales han sido transportados por las corrientes de convección creadas por el fuego). Los parámetros se introducen en la regla por medio de probabilidades calculadas empíricamente, y el efecto de "spotting" hace uso de una distribución de probabilidad de Poisson. El resultado es una técnica de optimización no lineal.

4. Emplea SIG.

Es un modelo avanzado y difícil matemáticamente, pero ya realizado en un simulador y con la ventaja de emplear un Sistema de Información Geográfica.

2.5.8 MODELO DE GUARISO

[Guariso, 2002]: modelo centrado en el área mediterránea y que incorpora al estudio de los fuegos de superficie los fuegos que se suceden en las copas de los árboles, iniciados por contagio con el fuego que asciende desde el suelo del terreno.

1. El modelo consta de dos capas superpuestas, una perteneciente a la superficie del terreno, y otra perteneciente a la zona de las copas de los árboles que nacen en dicha superficie.
2. Vecindad de Moore para la capa del suelo del terreno, junto con la celda correspondiente a la central (es decir, la célula de cuya vecindad tratamos) situada en el plano superior. Así, cada célula de la superficie del terreno posee 9 células vecinas, 8 en el plano inferior y 1 en el plano superior.
3. Cinco estados posibles para cada celda (no combustible, combustible en estado normal, combustible acumulando energía, combustible ardiendo y quemada). Se utilizan unos umbrales para pasar de normal a acumulando energía, y de acumulando energía a ardiendo.
4. La regla de transición de estados emplea una variante del modelo de Rothermel. Se aplican como variables el tipo y distribución de la vegetación combustible, la presencia de barreras al fuego, la inclinación del terreno, la temperatura del aire y la humedad y los efectos del viento.
5. Para los umbrales anteriormente mencionados y los tipos de combustibles a aplicar en el área mediterránea se emplean los trabajos de Valette [Valette, 1990].
6. Emplea SIG y la extracción automática de datos de mapas regionales. El modelo se completa con servicios de GPS para controlar la localización de unidades de combate contra el fuego y con lectura de los datos dados por las mismas, así como con el envío de mensajes de emergencia a uno o a varios usuarios.

Modelo complejo que incorpora dos capas, la superficial y la de las copas de los árboles, y que al utilizar factores constantes deducidos empíricamente en su formulación matemática, cuyos valores se pueden almacenar en hojas de cálculo, permite la reedición de dichos factores cuando se cambia de escenario y su almacenamiento en otra hoja de cálculo, con lo que la simulación se podría usar en otros escenarios, siempre y cuando se tuvieran los factores correspondientes a dichos escenarios. Incorpora, además, funciones relacionadas con la retroalimentación en los datos reales a través de las medidas tomadas, *in situ*, por las unidades que combaten el fuego, así como alarmas vía mensajes sms y de correo electrónico a dichas unidades y a las autoridades correspondientes.

2.5.9 MODELO DE PERRY

[Perry, 1999]: simulador PYROCART soportado por SIG.

1. Cinco estados posibles para cada celda (no quemada, quemada, quemada tras un paso de la simulación, quemada tras dos pasos de la simulación y punto de ignición inicial).
2. La regla de transición de estados se basa en Rothermel empleando un trabajo de campo bastante exhaustivo para la modelación de los tipos de la vegetación combustible en el área de estudio del modelo (Nueva Zelanda). Las variables utilizadas son topografía (elevación del terreno y, por tanto, la inclinación entre dos zonas de terreno separadas o "slope"), velocidad y dirección del viento y tipos de combustible vegetal (incluyendo su contenido en humedad y su distribución sobre el terreno). Los datos sobre la velocidad y dirección del viento se toman in situ, en estaciones situadas en la zona de la simulación.
3. Se utiliza un simulador basado en SIG para la presentación y manipulación del modelo.

Modelo adaptado a una localización exacta de Nueva Zelanda, con un estudio de la vegetación y de la combustibilidad de la misma exhaustivo y con unos esfuerzos de validación de los resultados importantes. El simulador adolece del hecho de ser dependiente del área especificada, pero, para zonas boscosas cuyas características de vegetación sean relativamente homogéneas, la forma en que la vegetación particular de una zona ha sido estudiada en este modelo sería extrapolable.

2.5.10 MODELO DE D'AMBROSIO

[D'Ambrosio, 2006]: simulador con células hexagonales y estudio de los fuegos de superficie y de las copas de los árboles.

1. Células hexagonales, cuya vecindad son las dos capas sucesivas de células que circundan a la célula central.
2. La función de transición de estados se basa en Rothermel e incluye como variables principales la velocidad y dirección del viento, la inclinación del terreno, el tipo de combustible vegetal y la temperatura y

humedad ambientales. El fuego a la altura de las copas de los árboles se basa en los estudios de Van Wagner [Van Wagner, 1969].
3. Simulación no basada en SIG.

Simulador con un juego de ecuaciones basado en Rothermel para los fuegos de superficie, y en Van Wagner para los fuegos contagiados a las copas de los árboles, con la peculiaridad de que las células del autómata celular son hexagonales, y con una vecindad de dos capas.

2.5.11 MODELO DE BERJAK

[Berjak, 2002]: simulador basado en un SIG adaptado a un área geográfica muy específica (la sabana sudafricana).

1. Tres estados posibles para cada celda (no quemada, ardiendo y quemada).
2. La regla de transición se basa en Rothermel y usa como parámetros principales el tipo y humedad del combustible vegetal, la dirección del viento (no considera para el terreno de estudio, en Sudáfrica, como importante la velocidad del viento) y la inclinación del terreno.
3. La simulación se realiza empleando un SIG.

Modelo complicado matemáticamente y aplicado al área específica de la sabana sudafricana. Trabajo importante de comprobación de la bondad de la simulación, puesto que se validan los resultados aplicando la simulación en dos escenarios diferentes, paisajes homogéneos y paisajes heterogéneos.

2.5.12 MODELO DE HARGROVE

[Hargrove, 2000]: simulación basada en Rothermel para fuegos en gran escala y en paisajes homogéneos y heterogéneos. El nombre del simulador es EMBYR.

1. Vecindad constituida por 8 células.
2. La regla de transición de estados se basa en formulaciones propias y toma como parámetros la velocidad y dirección del viento, el tipo de combustible vegetal y su contenido de humedad, desarrollando probabilidades de nuevos puntos de ignición en función de la inclinación del terreno. Para los efectos de "spotting" se basa en Rothermel.
3. El simulador utiliza un SIG.

Modelo bastante complejo, con un estudio previo muy trabajado y del cual se han extraído las diferentes formulaciones a usar en la propagación del fuego. Trabajo exhaustivo en relación a los tipos de vegetación combustible y a la validación de los datos resultantes de las simulaciones. Como elemento distintivo de otros modelos es de destacar la posibilidad de usar la simulación en fuegos de gran escala y en paisajes heterogéneos.

2.6 CONCLUSIONES SOBRE LOS MODELOS ANTERIORES

Partiendo del paradigma de los autómatas celulares, las diferencias entre los modelos se centran particularmente en los siguientes elementos:

1. Células rectangulares o hexagonales: casi todos los modelos expuestos disponen las celdas rectangularmente, salvo uno en el que las celdas son hexagonales.
2. Vecindades: la mayoría de los modelos usan el modelo de vecindad simple de Moore, aunque algunos presentan vecindades extendidas.
3. Número de estados posibles de las celdas: casi todos los modelos otorgan estados discretos a las celdas, 4 ó 5, aunque en algunos los estados son continuos.
4. Regla de transición de estados: la mayor discrepancia en los modelos se da en las reglas de transición de estados. En los modelos más simples las reglas de transición de estados no tienen en cuenta ninguna o muy pocas de las variables identificadas como determinantes en la propagación de los incendios forestales. En los modelos más complicados sí se emplean dichas variables (velocidad y dirección del viento, inclinación del terreno, humedad y temperatura ambiental, tipo y distribución de los combustibles vegetales, "spotting"). La aplicación de dichas reglas utiliza desde simplificaciones matemáticas hasta la resolución explícita de complicadas ecuaciones diferenciales y otras. Los modelos más complicados suelen ser semiempíricos, en los cuales las ecuaciones de Rothermel o de Anderson se refinan posteriormente haciendo uso de simulaciones y correlaciones estadísticas, a partir de las cuales se obtienen factores de ponderación que simplifican las ecuaciones.
5. Simuladores: en algunos casos los simuladores se apoyan en Sistemas de Información Geográfica. En los modelos que incorporan datos topográficos y meteorológicos, unos los obtienen de servicios meteorológicos oficiales y otros los recogen directamente sobre el terreno. En cuanto a la distribución y combustibilidad de las diferentes vegetaciones, se usan, en general, modelos derivados de Rothermel y Albini [Albini, 1976].
6. La mayor parte de los modelos estudian los fuegos de superficie (surface fires) mientras que unos pocos añaden también los fuegos de las copas de los árboles (crown fires) y/o el fenómeno del "spotting fire".

A reseñar, tal y como se indica anteriormente y tal y como aparece en la introducción de muchos de los modelos estudiados, que existe, entre otros, un paradigma alternativo ampliamente utilizado en lugar del de los autómatas celulares, y éste es el de la teoría de la propagación de las ondas ("wave propagation"). Por ejemplo, el simulador FARSITE [Finney, 1994], el cual aplica los principios de Huygens a las fórmulas de Rothermel. En cualquier caso, casi todos los modelos que aparecen en la literatura de la simulación de los incendios forestales, recogen, en mayor o menor medida, las aportaciones

de Fons [Fons,1946], Van Wagner [Van Wagner, 1969], Anderson [Anderson, 1982] y Rothermel [Rothermel,1972], [Rothermel,1983].

Para un compendio exhaustivo sobre distintos modelos y sus bondades remitimos a [Pastor, 2003].

2.7 AUTÓMATAS CELULARES Y EL FORMALISMO CELL-DEVS

Los autómatas celulares pueden ser de tiempo discreto o de eventos discretos.

2.7.1 AUTÓMATAS CELULARES DE TIEMPO DISCRETO

En ellos, en cada paso de tiempo cada célula sufre una transición de estado, con independencia de que el estado de la misma cambie o no. Suponiendo que, en general, en un espacio celular, en cada instante de la simulación sólo un número determinado del total de las células del espacio cambiarán de estado, inspeccionar todas las células del espacio conlleva una pérdida de eficiencia, ya que no todas cambiarán de estado.

2.7.2 AUTÓMATAS CELULARES DE EVENTOS DISCRETOS

Supongamos que conocemos el conjunto potencial de las células cuyo estado puede cambiar en el paso de la simulación en curso (como consecuencia, también conocemos cuáles son aquellas células cuyo estado no cambiará en la simulación en curso). Entonces, para lograr mayor eficiencia computacional que en los autómatas celulares de tiempo discreto, la simulación sólo inspeccionará aquel conjunto de células cuyo estado pueda cambiar, en lugar de inspeccionar al conjunto de todas las celdas. Tales simuladores se denominan de eventos discretos, pues se concentran en procesar los eventos (las células potencialmente cambiantes) en vez de examinar el conjunto de todas las células.

La lógica de este procedimiento es la siguiente [Urquía, 2011, p. 58]:

<< En una transición de estado deben señalarse aquellas células cuyo estado cambia. A continuación, establézcase el conjunto de todas las células vecinas de aquellas. Este conjunto contiene todas las células que pueden posiblemente cambiar su estado en el siguiente paso de tiempo. El estado de las células no pertenecientes a este conjunto permanecerá inalterado en el siguiente paso de tiempo>>

Partiendo del instante de tiempo inicial y aplicando esta regla, sólo aplicaremos la función de transición de estado a aquellas células cuyo estado es susceptible de cambiar, con lo que ganaremos en eficiencia computacional.

2.7.3 EL FORMALISMO CELL-DEVS

Entre los modelos celulares de eventos discretos un ejemplo es el formalismo CELL-DEVS, el cual permite construir modelos celulares basados en el paradigma de las Autómatas Celulares aplicando el formalismo DEVS.

2.7.3.1 MODELOS DEVS

El formalismo DEVS (Discrete Event System Specification), se aplica al modelado de eventos discretos. El modelo DEVS clásico fue propuesto por Zeigler [Zeigler, 1976] y, posteriormente, fue ampliado al modelo DEVS paralelo por Zeigler y Chow [Chow, 1994].

2.7.3.2 MODELO DEVS CLÁSICO

En el modelo DEVS clásico, los componentes pueden ser atómicos o acoplados. El modelo será acoplado si está compuesto de varios submodelos, cada uno de los cuales será un modelo atómico o, a su vez, acoplado.

Utilizando la definición de DEVS dada en [Wainer, 2009, p. 36] un modelo atómico DEVS se especifica mediante la tupla siguiente:

$M=< X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$, donde

X conjunto de todos los posibles valores que un evento de entrada puede tener.
Y conjunto de todos los posibles valores que un evento de salida puede tener.
S conjunto de los posibles estados secuenciales.
$\delta int : S \rightarrow S$ función de transición interna, que describe la transición de un estado al siguiente estado secuencial S.
$\delta ext : Q \times X \rightarrow S$ función de transición externa, la cual responde a eventos externos de entrada.
$\lambda : S->Y$ función de salida.
ta función de avance de tiempo.

Un modelo DEVS acoplado está compuesto de varios submodelos atómicos o acoplados. Se especifica mediante la siguiente tupla [Wainer, 2009, p. 37]:

$CM=< X, Y, D, \{M_d, d \in D\}, EIC, EOC, IC, select >$, donde

X conjunto de los eventos de entrada (eventos entrantes y puertos por los que entran).
Y conjunto de los eventos de salida (eventos salientes y puertos por los que salen).
D nombre de los componentes, donde, para cada d perteneciente a D, M_d es un modelo básico (atómico o acoplado).

EIC External Input Coupling: conexión entre las entradas externas al modelo a las entradas internas correspondientes a uno o varios de los submodelos.
EOC External Output Coupling: conexión entre las salidas de uno o varios de los submodelos a la salida externa del modelo acoplado.
IC Internal Coupling: conexión entre las salidas de uno o varios submodelos a entradas de uno o varios submodelos.
select función de prioridad, cuando hay varias transiciones planificadas para el mismo instante.

Otra definición diferente para un modelo DEVS acoplado puede verse en el mismo [Wainer, 2009, p. 39].

2.7.3.3 MODELOS CELL-DEVS

El formalismo CELL-DEVS nos permite aunar los Autómatas Celulares con la especificación DEVS. El formalismo CELL-DEVS se basa en los trabajos de [Giambiasi, 2002] y [Wainer, 1998].

Cada célula del autómata celular se define como un modelo atómico y, a continuación, se especifica la interacción entre las diferentes células.

El modelo atómico para una célula se especifica con la tupla siguiente [Wainer, 2009, p. 59]:

TDC= < X, Y, S, N, type, d, τ, δ_{int}, δ_{ext}, λ, D >, donde

X conjunto de los eventos externos de entrada.
Y conjunto de los eventos externos de salida.
S conjunto de los estados secuenciales de la célula.
$N \in X^n$ conjunto de los valores de entrada recibidos por la célula en cuestión desde las células de su vecindario.
d retardo.
type tipo del retardo d (de transporte, inercial u otra).
$\tau : N \rightarrow S$ función de cómputo local.
$\delta int : S \rightarrow S$ función de transición interna.
$\delta ext : Q \times X \rightarrow S$ función de transición externa.
$\lambda : S \rightarrow Y$ función de salida.
ta función de avance de tiempo.

Los dos tipos de retardo son el retardo de transporte y el retardo inercial. En el retardo de transporte, si el estado de la célula cambia, el resultado será transmitido a la salida después de una demora o plazo temporal dado d. En el retardo inercial, si el valor de entrada que llegó a la celda se mantiene durante el plazo d, el estado futuro será el resultado s, pero si llega otro valor de entrada antes de que el plazo d haya transcurrido, el resultado s se ignorará.

El modelo acoplado CELL-DEVS se define mediante la tupla siguiente [Wainer, 2009, p. 63]:

CM= < X, Y, D, {M_i}, {I_i}, {Z_{ij}}, select >, donde

X conjunto de los eventos externos de entrada.
Y conjunto de los eventos externos de salida.
D nombre de los componentes, donde, para cada d perteneciente a D, M_d es un modelo básico (atómico o acoplado).
I_i conjunto de componentes que son influenciados por el modelo i.
Z_{ij} función de traducción del modelo i al modelo j. Representa los acoplamientos entre las salidas y entradas de/a una celda con los modelos externos a su espacio celular.
select función de prioridad, cuando hay varias transiciones planificadas para el mismo instante.

Para un ejemplo explicativo de esta especificación remitimos al lector al libro [Wainer, 2009, pp. 64-67]. En [Wainer, 2001] se describe el mismo modelo, pero de una forma un poco más detallada.

2.7.3.4 "QUANTIZED" CELL-DEVS

Los modelos CELL-DEVS "cuantizados" utilizan un umbral de disparo. Sólo si el cambio en el estado de la célula supera el valor de dicho umbral se informará de dicho cambio a las células vecinas [Wainer, 2009].

2.7.3.5 MODELADO DE INCENDIOS FORESTALES Y CELL-DEVS

Existen diferentes campos de aplicación para el modelo CELL-DEVS. En [Wainer, 2009] se describen ejemplos para los ámbitos de la biología, usos militares y de emergencias, arquitectura y construcción, medio ambiente, física y química, redes y sistemas artificiales y, finalmente, modelado del tráfico urbano.

En el campo de las ciencias medioambientales, en el Capítulo 11 del libro antes citado [Wainer, 2009], Sección 11.7.2, el autor describe un modelo de extensión de incendios usando el modelo de Rothermel.

En [Wainer, 2006] se aplica un modelo similar al anterior, pero con una explicación mucho más prolija.

En [Muzy, 2005] se compara el formalismo DEVS con el CELL-DEVS, basándose en una simulación experimental en un laboratorio, para validar ambos formalismos y comparar sus respectivas eficiencias.

En [Innocenti, 2004] y [Muzy, 2006] se emplean diferentes variables de los formalismos DEVS anteriores, para simular incendios forestales.

En [MacLeod, 2006] y [Muzy, 2002] se utiliza el formalismo Quantized CELL-DEVS en la simulación de incendios forestales.

2.8 HERRAMIENTAS DE MODELADO

Existen diferentes herramientas de modelado para las simulaciones por ordenador, las cuales generalmente se basan en lenguajes de programación, (Java o C). Entre las diferentes herramientas estudiadas, mencionamos las siguientes:

1. Editores específicos del lenguaje de alto nivel. La simulación se realiza mediante un programa escrito directamente en el lenguaje de alto nivel. Tal es el caso de este trabajo, el cual se ha realizado en Java, utilizando el editor de Eclipse.
2. Simuladores específicos basados en un lenguaje de alto nivel, pero que se pueden ejecutar igualmente en IDEs de uso habitual, mediante el uso de "pluggins" en dichas IDEs. Tal es el caso, por ejemplo, de TerraME [TerraME], el cual es un entorno de programación para el simulado de dinámicas espaciales. TerraME, desarrollado en Brasil, está basado en los lenguajes de programación LUA [LUA] y C++. Los modelos se pueden programar directamente en el intérprete TerraME, o se pueden implementar a partir de otras IDEs que incluyen "pluggins" para extender el uso de TerraME a dichas IDEs, como Crimson o Eclipse. Otro caso similar es REPAST, Recursive Porous Agent Simulation Toolkit [REPAST]. Éste es un simulador basado en sistemas multiagente que también posee extensiones para simular Autómatas Celulares. REPAST se puede ejecutar independientemente o pueden emplearse sobre IDEs habituales, como Eclipse.
3. Como caso particular también tratado en este proyecto indicamos el simulador NetLogo [NetLogo]. Éste es un entorno de modelado de sistemas multiagente, el cual también puede ser utilizado para los autómatas celulares.

Para la realización de este PFC se ha empleado el editor "Eclipse IDE for Java Developers", versión Luna Service Release 1 (4.4.1).

Para el procesado de las imágenes se ha importado la librería "Java Advanced Image" versión 1.1.3.

Para el trabajo en NetLogo se ha empleado la versión NetLogo 5.1.0.

2.9 CONCLUSIONES

Entre las diferentes posibilidades para representar la dinámica de la propagación del fuego en la predicción de los incendios forestales, se encuentra la representación mediante autómatas celulares. Existen numerosos trabajos que realizan esta aproximación. Las diferencias entre unos y otros se basan, por un lado, en la estructura del autómata empleado (vecindarios, número de estados, morfología de las celdas, reglas de transición, tipos de fuegos a representar, etc.) y, por otro, en los apoyos empleados por el simulador (empleo de SIGs u otras técnicas). Aparte de estas diferencias,

existen extensiones al paradigma de los autómatas celulares, tales como el formalismo Cell-DEVS.

En este capítulo hemos expuesto un resumen suficientemente completo de lo anterior, de forma que poseamos suficientes conocimientos teóricos para proceder al estudio de un trabajo particular, el realizado por Alexandridis et Alter, y que se verá en el capítulo posterior.

3 DESCRIPCIÓN DEL MODELO IMPLEMENTADO

3.1 INTRODUCCIÓN

Entre los modelos reseñados en el capítulo anterior, el modelo de Alexandridis (Sección 2.5.7) desarrolla un autómata celular semiempírico.

Este PFC implementa dicho autómata celular, el desarrollado por A. Alexandridis, D. Vakalis, C.I. Siettos y G.V. Bafas en "A cellular automata for forest fire spread prediction: The case of the wildfire that swept through Spetses Island in 1990" [Alexandridis, 2008]. Dicho modelo se adaptará para corresponder con el algoritmo de clasificación de imágenes de Distancia Mínima que empleamos en nuestro PFC para clasificar la imagen Landsat original de la isla de Spetses.

En este capítulo realizaremos una descripción del trabajo de Alexandridis.

3.2 DEFINICIONES DEL AUTÓMATA CELULAR Y DE LOS PARAMETROS

Veamos, en primer lugar, la definición espacial del autómata celular y, a continuación, las variables que afectan a la extensión del incendio, así como su aplicación a las reglas de transición de estados.

3.2.1 DEFINICIÓN ESPACIAL DEL AUTÓMATA CELULAR

Se emplea una red bidimensional para representar la zona geográfica considerada, la isla de Spetses en Grecia. Cada celda de dicha red representa un área en el plano de la isla, siendo dicho área cuadrado. Se emplea el vecindario de Moore, consistente en una celda central y sus 8 vecinas. En la figura 3.1 vemos la celda central y sus 8 vecinas. Así, las flechas representan las posibles direcciones en las que el fuego de la celda central se puede propagar (el estado de la celda central influye en el de sus vecinas).

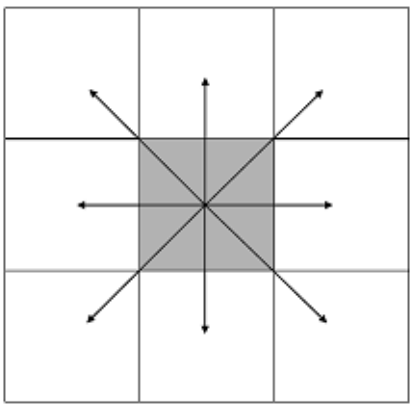


Figura 3. 1: Vecindario de Moore

3.2.2 ESTADO DE LAS CELDAS

Cada celda se caracteriza mediante un número finito de estados, el cual evoluciona en tiempo discreto. Los posibles estados son los siguientes:

1. Estado 1: la celda no contiene combustible vegetal y, por tanto, no puede arder. Tal es el caso, por ejemplo, de las zonas sin vegetación (el área urbana de la isla).
2. Estado 2: la celda contiene combustible (vegetación) que no se ha incendiado.
3. Estado 3: la celda contiene vegetación ardiendo.
4. Estado 4: la celda contiene vegetación que ha ardido completamente.

3.2.3 REGLAS DE TRANSICIÓN DE ESTADOS

En cada paso t de la simulación (los pasos son de tiempo discreto), dada la celda *(i,j),* tendremos que:

1. Regla 1: si estado de *(i,j,t)=1*, entonces estado de *(i,j,t+1)=1*. Es decir, el estado de las celdas que no contienen combustible (áreas urbanas) permanecerá siempre igual, dado que dichas celdas no pueden arder.
2. Regla 2: si estado de *(i,j,t)=3*, entonces estado de *(i,j,t+1)=4*. Es decir, si la celda arde en el paso actual de la simulación, en el siguiente paso de tiempo se considerará quemada.
3. Regla 3: si estado de *(i,j,t)=4*, entonces estado de *(i,j,t+1)=4*. Es decir, si la celda está quemada, en el siguiente paso permanece igual.
4. Regla 4: si estado de *(i,j,t)=3*, entonces estado de *(i+/-1,j+/-1,t+1)=3*, con una probabilidad de P_{burn}. Es decir, en el instante *t* consideramos la celda central *(i,j).* Si esta celda central está ardiendo, en el siguiente paso de la simulación la probabilidad de que las celdas de su vecindario (véase la Figura3.1) estén ardiendo es de P_{burn}. Dicha probabilidad se verá a continuación.

3.2.4 VARIABLES QUE AFECTAN A LA EXTENSIÓN DEL INCENDIO

Las variables que afectan a la extensión del incendio son las siguientes:

1. Densidad de la vegetación (factor P_{den}).
2. Tipo de la vegetación (factor P_{veg}).
3. Velocidad del viento (en el factor P_w).
4. Dirección del viento (en el factor P_w).
5. Elevación del terreno (factor P_s).

La probabilidad P_{burn} posee la siguiente fórmula:

$$P_{burn} = P_0(1 + P_{veg})(1 + P_{den})P_w P_s \qquad (3.1)$$

El factor P_0 es un factor empírico desarrollado por Alexandridis et Alter, el cual indica la probabilidad de que una celda vecina a una que esté ardiendo se

prenda en el siguiente paso de la simulación en condiciones de ausencia de fuego y con diferencia de elevación entre la celda central y la vecina nula.

El resto de los factores se verá a continuación.

3.2.5 EFECTO DEL TIPO DE LA VEGETACIÓN

Dicho efecto se modela mediante la probabilidad P_{veg}. En la isla se consideran 3 tipos de "vegetación", a cada uno de los cuales se asocia un diferente valor de P_{veg}, en función de su contribución a la extensión del fuego. Tales son:

1. Zonas agrícolas: P_{veg}= - 0.3
2. Zonas de matorrales: P_{veg} = 0
3. Zonas de pinos de Alepo: P_{veg} = 0.4

3.2.6 EFECTO DE LA DENSIDAD DE LA VEGETACIÓN

Dicho efecto se modela mediante la probabilidad P_{den}. En la isla se consideran 3 tipos de densidad de la vegetación, a cada uno de los cuales se asocia un diferente valor de P_{den}, en función de su contribución a la extensión del fuego. Tales son:

1. Densidad de vegetación escasa: P_{den} = - 0.4
2. Densidad de vegetación intermedia: P_{den} = 0
3. Densidad de vegetación alta: P_{den} = 0.3

3.2.7 EFECTOS DE LA VELOCIDAD Y DE LA DIRECCIÓN DEL VIENTO

Sea θ el ángulo formado entre la dirección de la propagación del fuego (la dirección en la cual se haya la célula vecina con respecto de la central) y la dirección del viento. Sea V la velocidad del viento. Entonces, el factor P_w se modela como:

$$P_w = exp(c_1 V) f_t \qquad (3.2)$$

donde

$$f_t = exp(V c_2 (\cos\theta - 1)) \qquad (3.3)$$

Las constantes c_1 y c_2 son parámetros empíricos desarrollados por Alexandridis et Alter.

3.2.8 EFECTO DE LA ELEVACIÓN DEL TERRENO

El fuego se propaga a mayor velocidad si va ladera arriba que ladera abajo. Para modelar este fenómeno, se considera la diferencia de altitud entre la celda central y la celda vecina a la cual se puede propagar el fuego. Sea E1 la elevación de la celda vecina y E2 la elevación de la celda central. Entonces,

$$P_s = exp(a\theta_s) \qquad (3.4)$$

donde θ_s posee diferente fórmula según que las celdas tengan disposición adyacente o diagonal. Así, si las celdas son adyacentes,

$$\theta_s = \tan^{-1}\frac{E_1 - E_2}{l} \qquad (3.5)$$

donde l es el tamaño de la celda, en metros. Si las celdas están en diagonal,

$$\theta_s = \tan^{-1}\frac{E_1 - E_2}{l\sqrt{2}} \qquad (3.6)$$

El parámetro a es una constante empírica desarrollada por Alexandridis et alter.

3.3 CASO DE ESTUDIO: EL FUEGO DE LA ISLA DE SPETSES DE 1990

La metodología anterior se aplica a un caso real, el fuego que sucedió en la isla griega de Spetses el 1 de agosto de 1990. El fuego se inició por causas desconocidas en un área boscosa del centro de la isla y se extendió hacia el sur, quemando una superficie total de alrededor de 6 km. cuadrados (algo menos de un tercio de la superficie de la isla). La duración del incendio fue de unas 11 horas.

A continuación, mostramos una imagen de la isla, donde el contorno del área quemada aparece en la zona central-sur, delimitada por las líneas blancas.

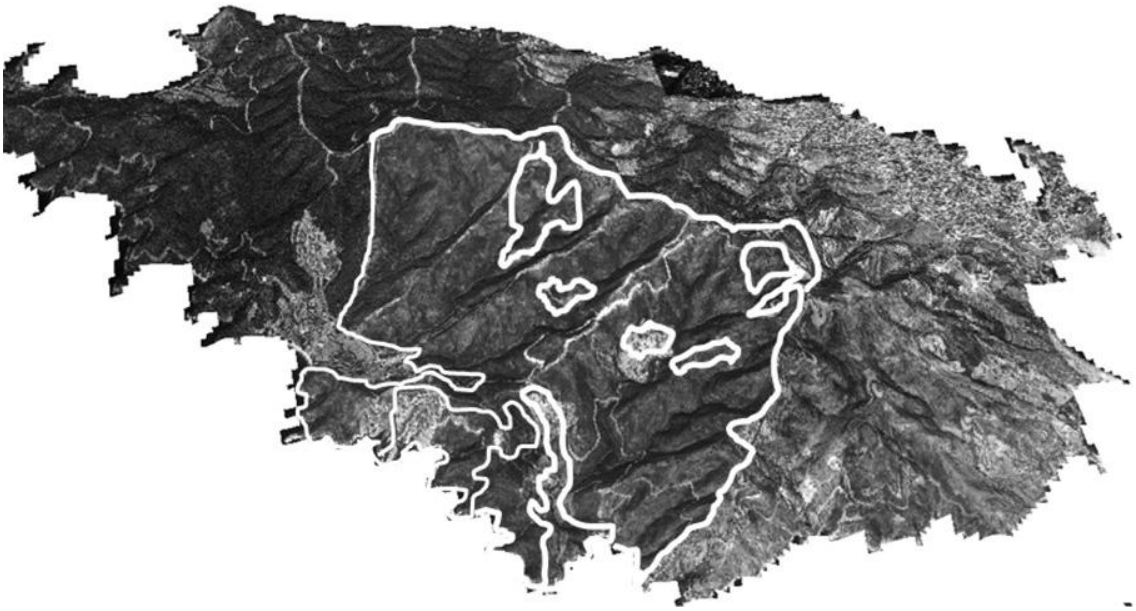


Figura 3. 2: Fuego de Spetses en 1990

El paisaje a modelar (es decir, la isla de Spetses) se representa mediante una red bidimensional, en cuadrículas de 5x5 metros cuadrados cada una. Cada cuadrícula constituye una celda del autómata celular, la cual poseerá unos parámetros de densidad de la vegetación, tipo de la vegetación y elevación.

Los valores de la densidad y tipo de la vegetación y de la elevación del terreno se determinan utilizando un SIG (sistema de información geográfica), concretamente Arc GIS 9.2 de ESRI, a partir de imágenes de la isla previas al incendio. A continuación, mostramos, en primer lugar, la imagen correspondiente a la densidad de la vegetación [Alexandridis, 2008] y, en segundo lugar, la correspondiente a la elevación del terreno [Russo, 2014]:

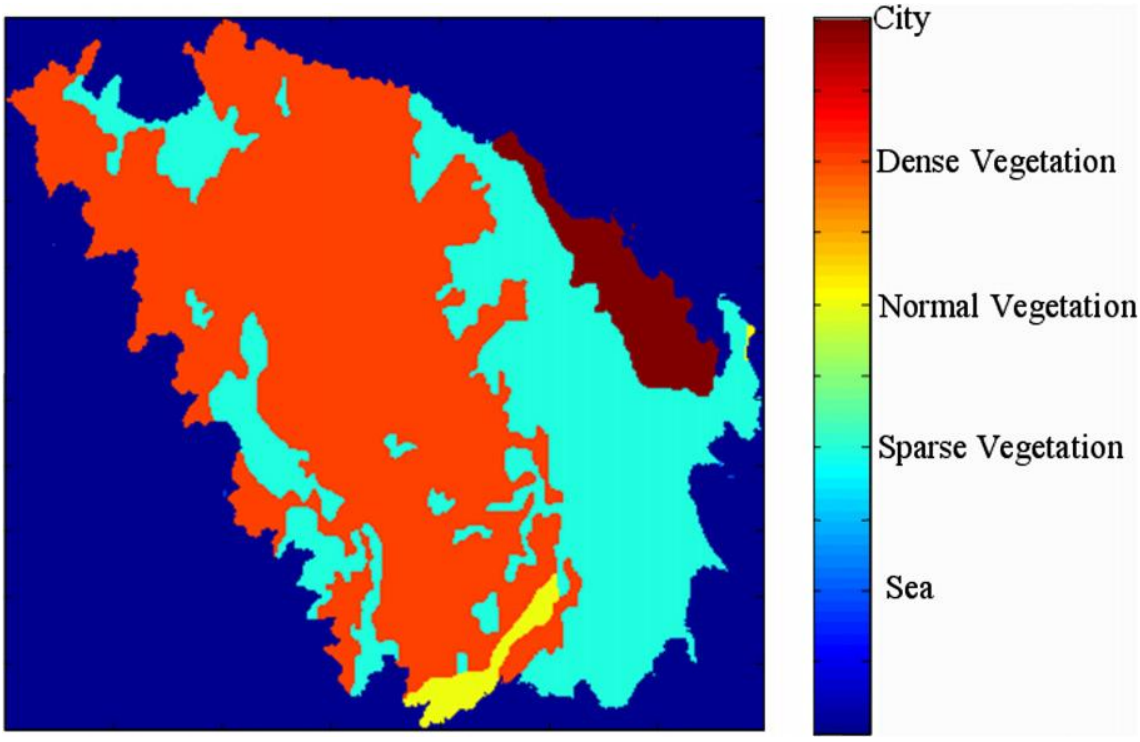


Figura 3. 3: Densidad de la vegetación

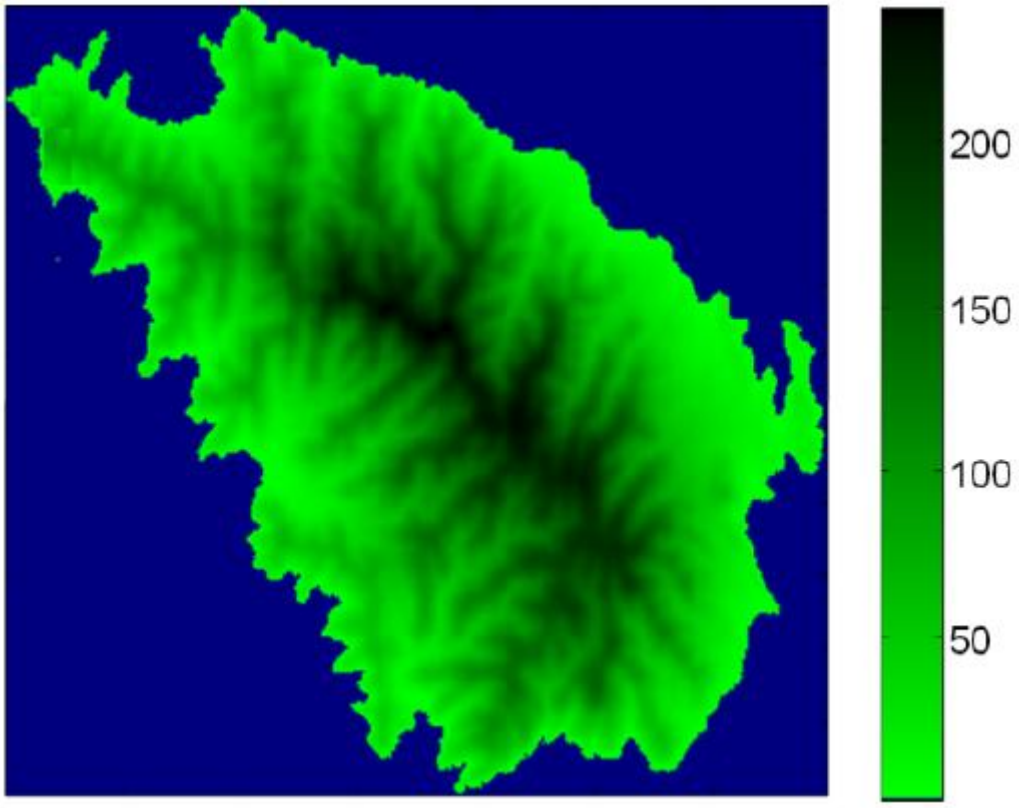


Figura 3. 4: Elevación del terreno

A partir de los datos extraídos se aplican, para cada celda del autómata, los parámetros relativos al tipo de la vegetación (ver Sección 3.2.5), a su densidad (ver Sección 3.2.6) y a su elevación (ver Sección 3.2.8). Dichos parámetros se obtienen, pues, empíricamente en el modelo de Alexandridis et alter

Del histórico del incendio se conocen los datos meteorológicos existentes en el día del mismo, los cuales se aplican a la simulación posterior en el autómata.

3.4 RESULTADOS DE LA SIMULACIÓN

Los parámetros de la probabilidad constante de propagación del fuego P_0, del coeficiente de elevación o "slope" a y de los coeficientes c_1 y c2 se determinan mediante técnicas de optimización no lineales ejecutando el simulador repetidas veces, de manera que la diferencia entre el número de celdas quemadas en la simulación y el número de celdas quemadas en el mapa del incendio real (teselado con la misma red bidimensional) sea la mínima. Se usa el algoritmo de optimización de Nelder-Mead. Los valores óptimos obtenidos son los siguientes:

1. $P_0 = 0.58$
2. $a = 0.078$
3. $c_1 = 0.045$
4. $c_2 = 0.131$

A continuación mostramos dos imágenes, la primera representa el área real quemada en el incendio, y la segunda representa el resultado de una simulación con los parámetros empíricos y los valores resultantes de la optimización anteriormente explicados [Alexandridis, 2008]:

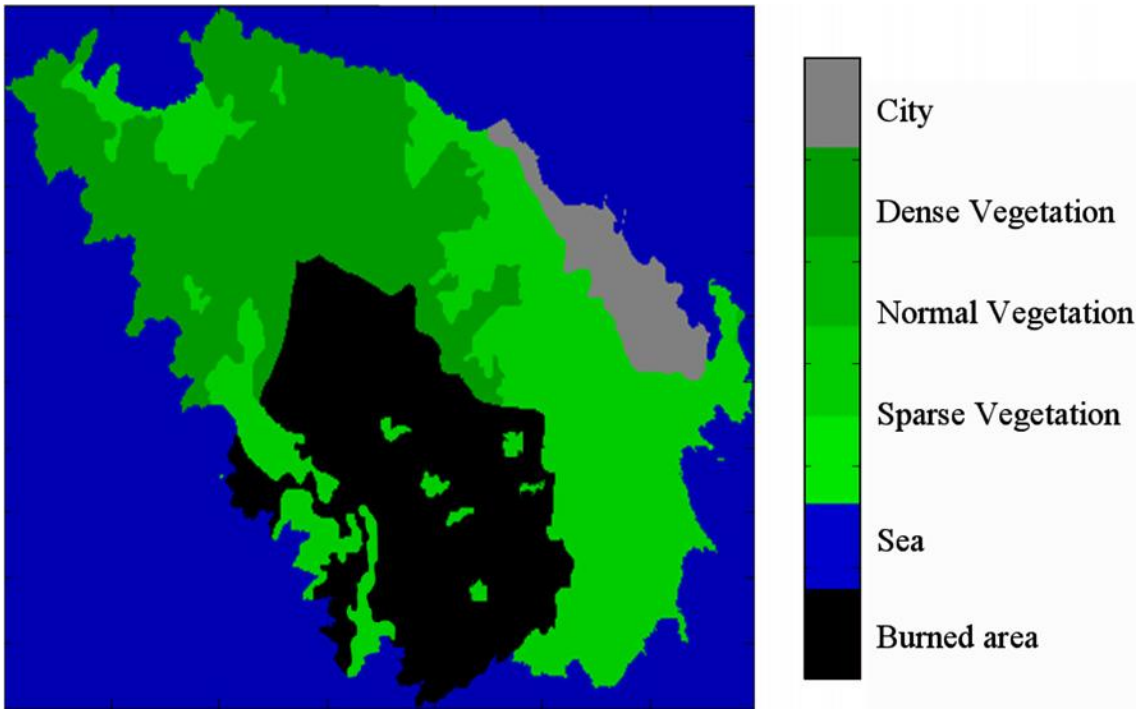


Figura 3. 5: Área quemada en el incendio (en color negro)

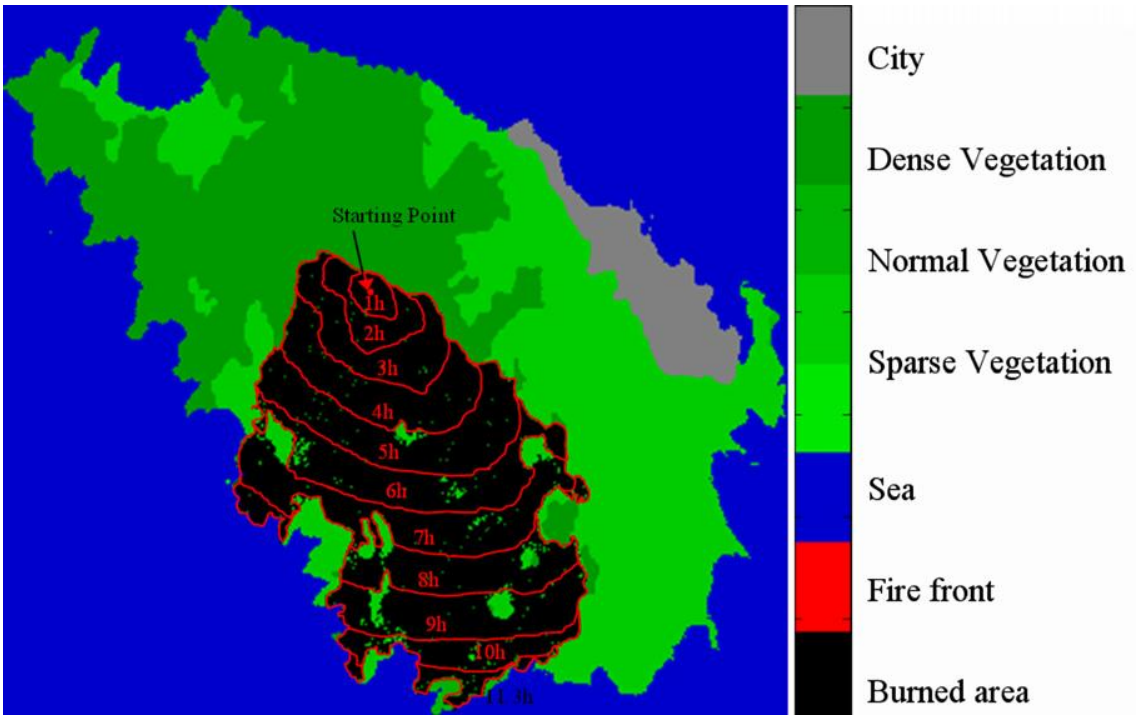


Figura 3. 6: Área quemada en la simulación (en negro)

Como vemos, el área simulada se aproxima bastante al área real. En la simulación, el fuego quemó un área total de 5.4 km. cuadrados en un tiempo de 11.3 horas, mientras que en el caso real, el área total quemada fue de 5.9 km. cuadrados en un tiempo de 11 horas.

Entre los factores que pueden producir diferencias respecto al modelo real están los siguientes:

1. Resolución limitada de las imágenes procesadas por el SIG.
2. Factores meteorológicos ambiguos: se conoce la velocidad y dirección general del viento en la isla el día del incendio real, pero no se poseen registros precisos sobre las mismas (ráfagas en distinta dirección, vórtices debidos al incendio, etc.).
3. Falta de información sobre las acciones humanas en relación a la lucha contra el incendio.

3.5 CONCLUSIONES

El modelo semiempírico de Alexandridis emplea la metodología de los autómatas celulares para la predicción dinámica de la extensión de un incendio forestal. Tiene en cuenta los valores meteorológicos de la velocidad y la dirección del viento. Como valores propios del paisaje donde se va a simular el incendio, toma en cuenta los valores de la densidad y el tipo de vegetación del terreno, así como los de la elevación del mismo. A partir de los datos anteriores y usando técnicas de optimización, obtiene coeficientes para aplicar a la fórmula general de la transición de estados en el autómata.

A partir de los datos existentes de un incendio real ocurrido en la isla de Spetses en 1990, se aplican los mismos valores meteorológicos a la simulación. Los resultados obtenidos se aproximan bastante a los del fuego real.

4 DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN

4.1 INTRODUCCIÓN

Una vez analizado detalladamente el trabajo de Alexandridis et Alter, hay que trasladarlo a una implementación en Java. Para ello, se plantean los siguientes pasos, no necesariamente sucesivos:

1. Diseño de la GUI.
2. Extracción de los datos de la imagen original.
3. Diseño del autómata.

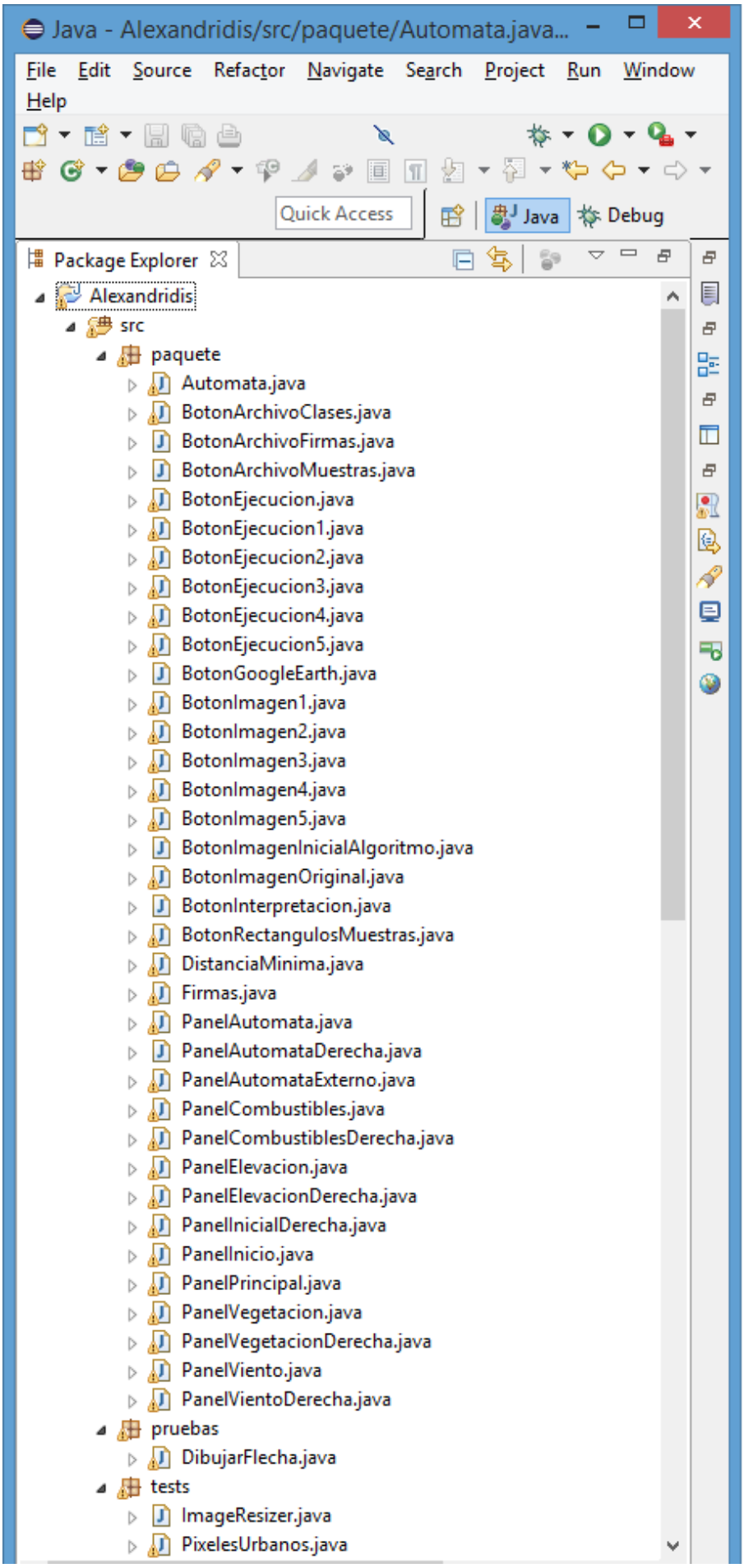
Veamos cada uno de los pasos, pero antes de ello estudiemos la distribución modular de la aplicación.

4.2 ESTRUCTURA DE LA APLICACIÓN

Mediante el método de refinamientos sucesivos, la aplicación ha quedado finalmente dispuesta de la siguiente forma:

1. Carpeta "imágenes": contiene todas las imágenes empleadas en la aplicación.
2. Carpeta "texto": contiene los archivos de texto plano usados en la aplicación.
3. Clases Java: las clases se estructuran en 3 paquetes, a saber:
    a. "paquete": contiene las clases principales del autómata.
    b. "tests": contiene clases auxiliares, empleadas principalmente para la ejecución del algoritmo de clasificación de la imagen original.
    c. "pruebas": en dicho paquete se iban probando las clases particulares que se iban desarrollando, a fin de no corromper la estructura principal de la aplicación.

A continuación presentamos una imagen con la estructura de archivos de la aplicación, dentro del editor Eclipse:


Java - Alexandridis/src/paquete/Automata.java...
File Edit Source Refactor Navigate Search Project Run Window
Help
Quick Access
Java
Debug
Package Explorer
Alexandridis
src
paquete
Automata.java
BotonArchivoClases.java
BotonArchivoFirmas.java
BotonArchivoMuestras.java
BotonEjecucion.java
BotonEjecucion1.java
BotonEjecucion2.java
BotonEjecucion3.java
BotonEjecucion4.java
BotonEjecucion5.java
BotonGoogleEarth.java
BotonImagen1.java
BotonImagen2.java
BotonImagen3.java
BotonImagen4.java
BotonImagen5.java
BotonImagenInicialAlgoritmo.java
BotonImagenOriginal.java
BotonInterpretacion.java
BotonRectangulosMuestras.java
DistanciaMinima.java
Firmas.java
PanelAutomata.java
PanelAutomataDerecha.java
PanelAutomataExterno.java
PanelCombustibles.java
PanelCombustiblesDerecha.java
PanelElevacion.java
PanelElevacionDerecha.java
PanelInicialDerecha.java
PanelInicio.java
PanelPrincipal.java
PanelVegetacion.java
PanelVegetacionDerecha.java
PanelViento.java
PanelVientoDerecha.java
pruebas
DibujarFlecha.java
tests
ImageResizer.java
PixelesUrbanos.java

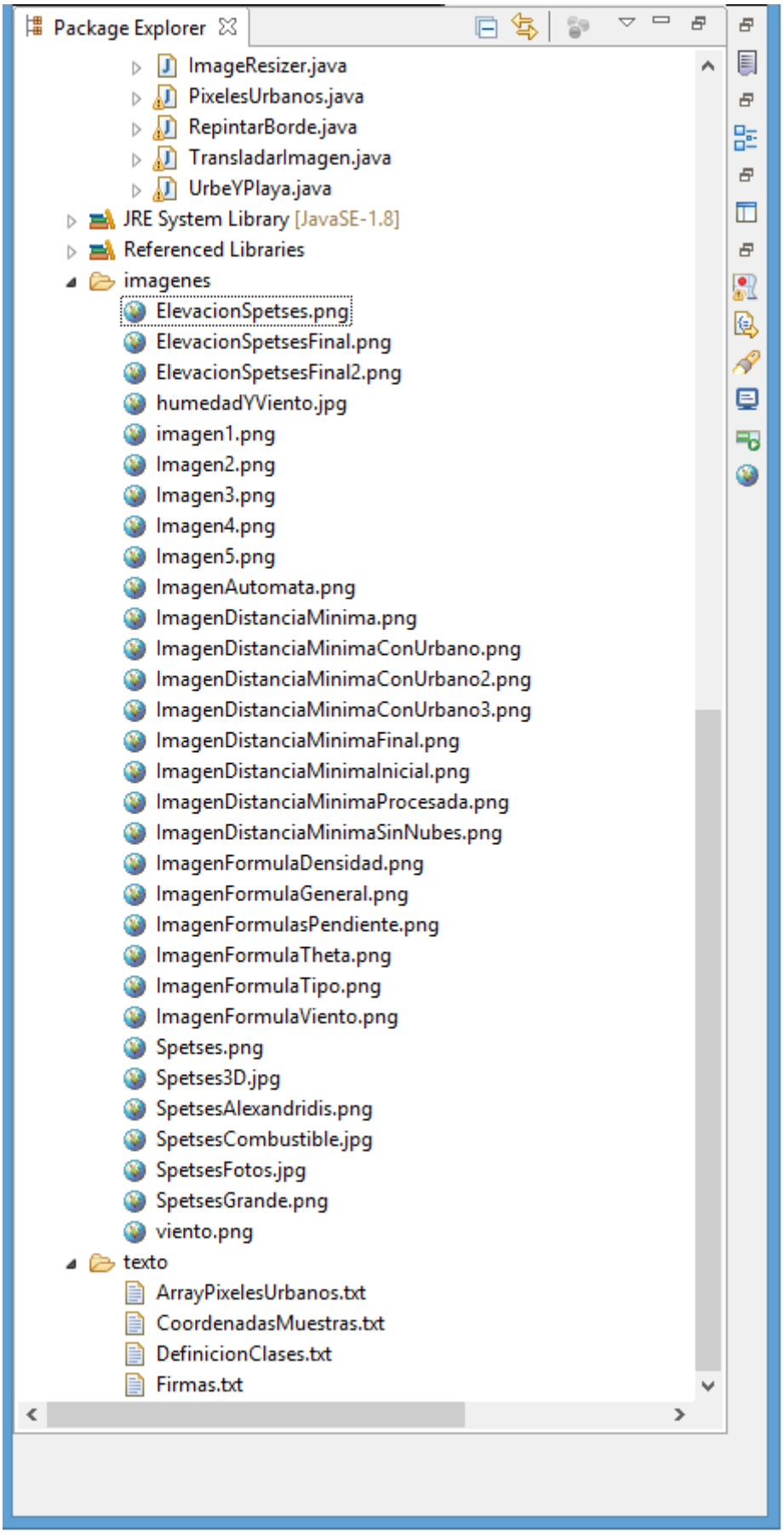


Figura 4. 1: Estructura de los archivos de la aplicación en Eclipse

4.3 DISEÑO DE LA GUI Y EXTRACCIÓN DE LOS DATOS DE LA IMAGEN ORIGINAL

Es evidente el requerimiento del paquete javax.swing para utilizar todo lo relativo a ventanas, paneles, botones, etc. Describimos a continuación las diferentes clases relacionadas con la GUI de la aplicación.

4.3.1 CLASE AUTOMATA

La clase principal de la aplicación es "Automata", la cual simplemente crea la ventana principal, de tamaño 1450x650 píxeles, e introduce en dicha ventana principal un panel, el panel "PanelPrincipal", el cual se desarrolla en otra clase Java. Al instanciar esta clase se ejecutan automáticamente las siguientes clases:

Firmas: calcula las firmas en el algoritmo de clasificación de la Distancia Mínima.

ImageResizer: se necesita transformar la imagen descargada de GoogleEarth al tamaño apropiado (700x550).

DistanciaMinima: el algoritmo de clasificación de la Distancia Mínima.

PixelesUrbanos: recoge la información sobre qué zona de la isla corresponde a áreas urbanas.

UrbeYPlaya: quita las nubes, pinta de negro las zonas urbanas y elimina defectos de las playas. Imagen final refinada en paint.

RepintarBorde: pinta de azul del mar el mapa de la elevación en la zona del borde, y cambia el color de las elevaciones a rojo, más apreciable.

Tales clases se explicarán más adelante, en las secciones correspondientes.

4.3.2 CLASE PANELPRINCIPAL

Divide la aplicación en dos zonas principales, la parte izquierda y la parte derecha. En la parte izquierda se aloja una panel de pestañas, "panelPestañas", y en la parte derecha se aloja otro panel, "panelDerecho". El panel de pestañas contiene 6 pestañas. Cada vez que se pincha en una de las pestañas el panel izquierdo alojará un nuevo panel interno. A su vez, el panel derecho alojará otro nuevo panel interno, relacionado con la pestaña seleccionada. Exponemos seguidamente unos extractos del código de la clase a fin de una mejor comprensión de lo aquí indicado:

```
//creamos el jtabbedpane
panelPestañas=new JTabbedPane();
panelPestañas.addTab("INICIO", panelInicio);
panelPestañas.addTab("VEGETACION", panelVegetacion);
panelPestañas.addTab("COMBUSTIBLES", panelCombustibles);
panelPestañas.addTab("VIENTO", panelViento);
panelPestañas.addTab("ELEVACION", panelElevacion);
```

```
panelPestañas.addTab("AUTOMATA", panelAutomata);


panelPestañas.addChangeListener(this);
add(panelPestañas);
add(panelDerecho);
setVisible(true);

JTabbedPane sourceTabbedPane = (JTabbedPane) e.getSource();
int index = sourceTabbedPane.getSelectedIndex();
System.out.println("Tab changed to: " + sourceTabbedPane.getTitleAt(index));
        switch (sourceTabbedPane.getTitleAt(index)){
            case "VEGETACION": this.remove(panelDerecho);panelDerecho= new
                   PanelVegetacionDerecha();this.add(panelDerecho);break;
             case "COMBUSTIBLES": this.remove(panelDerecho);panelDerecho= new
                   PanelCombustiblesDerecha();this.add(panelDerecho);break;
             case "VIENTO": this.remove(panelDerecho);panelDerecho= new
                   PanelVientoDerecha();this.add(panelDerecho);break;
             case "ELEVACION": this.remove(panelDerecho);panelDerecho= new
                   PanelElevacionDerecha();this.add(panelDerecho);break;
             case "AUTOMATA": this.remove(panelDerecho);panelDerecho=new
                   PanelAutomataDerecha();this.add(panelDerecho);break;

             case "INICIO": this.remove(panelDerecho);panelDerecho= new
                   PanelInicialDerecha();this.add(panelDerecho);break;
```

Código 4. 1: Extracto de código de la clase "PanelPrincipal"

Como vemos, el panel de pestañas contiene 6 pestañas, "INICIO", "VEGETACION", "COMBUSTIBLES", "VIENTO", "ELEVACION" y "AUTOMATA". A cada una de dichas pestañas le corresponderá su propio panel que se alojará en el panel derecho. Así, a la pestaña "VEGETACION" le corresponderá el panel "PanelVegetacionDerecha", etc.

1. Pestaña "VEGETACION": PanelVegetacionDerecha
2. Pestaña "COMBUSTIBLES": PanelCombustiblesDerecha
3. Pestaña "VIENTO": PanelVientoDerecha
4. Pestaña "ELEVACION": PanelElevacionDerecha
5. Pestaña "AUTOMATA": PanelAutomataDerecha
6. Pestaña "INICIO": PanelInicialDerecha

Inicialmente, antes de que se seleccione ninguna pestaña, la pestaña que se activa es "INICIO" y el panel que le corresponde en la parte derecha es "PanelInicialDerecha".

4.3.3 CLASE PANELINICIO

Contiene la imagen original Landsat de la isla de Spetses, y un MouseMotionListener, a fin de que al mover el ratón sobre la imagen aparezca en la parte inferior de la imagen la coordenada sobre la cual en ese momento se haya el puntero del ratón.

4.3.4 CLASE PANELINICIALDERECHA

Contiene texto (en JTextÁreas) y una imagen (en un JScrollPane). Explica el funcionamiento general del autómata en relación a los estados y las reglas de transición. La imagen es la fórmula general del autómata.

4.3.5 CLASE PANELVEGETACION

Contiene la imagen resultante de aplicar el algoritmo de clasificación de la Distancia Mínima. Al pasar el puntero del ratón, en la parte inferior de la imagen aparecen las coordenadas por las que pasa el puntero. Al pinchar en un punto de la imagen aparecen los valores RGB de dicho punto. Tales valores corresponderán a las diferentes densidades posibles, "escasa", "intermedia", "densa" o "área urbana". La explicación del algoritmo se indica en la siguiente clase.

4.3.6 CLASE PANELVEGETACIONDERECHA

El primer JTextÁrea explica el algoritmo de la Distancia Mínima [Santos]. Básicamente, el algoritmo es como sigue:

La imagen inicial (ver el botón "Imagen inicial") es procesada para clasificar el tipo y densidad de la vegetación existente. Para ello, a partir de los datos del terreno, sabemos que existen 5 tipos de clases de interés en el mapa: mar, vegetación escasa, vegetación intermedia, vegetación densa y, por último, suelos urbanos. Las clases se identifican en el archivo "DefinicionClases.txt" (ver el botón "DefinicionClases.txt"). La explicación de dicho archivo es que a la clase 1 le asignaremos en la imagen clasificada el color RGB cuyas coordenadas en el cubo RGB son las que vienen a continuación, y así hasta la clase 4. La zona urbana, debido a sus colores ocres, difíciles de distinguir de otros colores ocres del terreno, es necesario procesarla aparte.

A partir del espacio de colores de la imagen inicial (RGB), usamos el sistema de clasificación de la Distancia Mínima. Este consiste en emplear muestras rectangulares sobre áreas de la imagen, sobre aquellas zonas de la misma en las que identificamos zonas de color uniforme y que corresponderían al mismo tipo de substrato. Para cada tipo de substrato, se recogen varios rectángulos, cuyas coordenadas y clases correspondientes se guardan en el archivo "CoordenadasMuestras.txt" (ver el botón "CoordenadasMuestras.txt"). Así, para la clase1, tendremos varios rectángulos de muestras, y cada rectángulo se identificará a partir de 4 números, sus coordenadas (x,y) en la imagen original, y el ancho y el alto del rectángulo (vea el botón "Muestras") .

Para cada clase, se halla la media de color a partir de todas las muestras tomadas. Tal media se denomina la FIRMA de dicha clase. Las firmas se guardan en el archivo "Firmas.txt" (ver el botón "Firmas.txt"). La firma de cada clase está representada por sus coordenadas en el cubo RGB. Finalmente, el algoritmo de la Distancia Mínima clasificará los píxeles de la imagen en función de la mínima distancia euclidiana de cada píxel a las firmas, escogiendo cada píxel como perteneciente a aquella clase a la cual su distancia sea la inferior (la mínima distancia). El resultado es la imagen clasificada.

Así, debajo del JTextÁrea aparecen 5 botones:

1. Imagen inicial: enlace a la imagen Landsat original que va a ser clasificada. La clase "BotonImagenOriginal" implementa el enlace.

2. DefinicionClases.txt: enlace al archivo de definición de clases para el algoritmo de la Distancia Mínima. La clase Java "BotonArchivoClases" implementa dicho enlace.

3. Muestras: enlace a la imagen Landsat original con las áreas de muestras superpuestas. Para cada tipo de clase definida en el archivo "DefinicionClases.text", se toman varias muestras rectangulares sobre la imagen original. La clase "BotonRectangulosMuestras" implementa dicho enlace: carga la imagen original y dibuja los rectángulos de muestras sobre la misma. Posee igualmente un MouseMotionListener que permite ver las coordenadas sobre las que se mueve el puntero del ratón.

4. CoordenadasMuestras.txt: contiene un enlace a un archivo con las coordenadas de cada rectángulo de muestra para todas las muestras de cada tipo y para todos los tipos. Cada rectángulo se identifica con cuatro números, sus coordenadas (x,y) en la imagen original, y el ancho y el alto del rectángulo. La clase Java "BotonArchivoMuestras" implementa el enlace.

5. Firmas.txt: para cada clase se calcula la media de su color a partir de sus muestras correspondientes. La media será la FIRMA de dicha clase. El botón es un enlace al archivo donde se guardan las medias. La clase Java "BotonArchivoFirmas" implementa dicho enlace.

Debajo de los botones aparece otro JTextÁrea donde se explica el postprocesado de la imagen resultante del algoritmo. La razón del postprocesado es la siguiente: la imagen procesada inicialmente con el algoritmo de clasificación de la Distancia Mínima posee pequeños defectos en las zonas de costa de la parte derecha del mapa, donde el algoritmo ha producido como resultado zonas de "Vegetación intermedia" donde correspondería "Mar".

Esto es debido a que la imagen original descargada de GoogleEartht posee diferentes tonalidades para el mar. Igualmente, tampoco aparece en dicha imagen la zona correspondiente a los terrenos urbanizados. Para solucionar estos defectos, la imagen resultante del algoritmo es procesada en dos clases aparte. La primera clase, "PixelsUrbanos", nos devuelve un archivo de texto con las coordenadas de los píxeles correspondientes a la zona urbana. La segunda clase, "UrbeYPlaya" procesa dichos píxeles sobre la imagen defectuosa, introduciendo en la misma el color negro para la zona urbana. Igualmente, elimina las zonas defectuosas de la parte derecha de la imagen resultante inicial, donde aparecen zonas costeras de mar como pertenecientes a Vegetación Intermedia, y, a su vez, elimina las nubes que aparecen en la parte superior derecha de la misma.

En los botones que aparecen debajo de dicho JTextÁrea se pueden ver, respectivamente, la imagen inicial resultante del algoritmo antes de realizar el postprocesado, y la imagen tomada de GoogleEarth donde se aprecian las diferentes tonalidades del mar. La imagen final, una vez postprocesada, es la que aparece en el panel izquierdo. La clase "BotonImagenInicialAlgoritmo" desarrolla el primer enlace, y la clase "BotonGoogleEarth" el segundo.

Debajo de dichos botones hay otro JTextÁrea donde se explica la influencia de la densidad de la vegetación en el modelo de Alexandridis et alter, y una imagen con los valores del parámetro P_{den}.

4.3.7 CLASE PANELCOMBUSTIBLES

Contiene dos imágenes sucesivas de la isla de Spetses, a fin de apreciar los diferentes tipos de vegetación. La fuente de las imágenes aparece al pie del panel.

4.3.8 CLASE PANELCOMBUSTIBLESDERECHA

Contiene un JTextÁrea donde se explican las imágenes de la izquierda (las de "PanelCombustibles"), una imagen central con los valores del parámetro P_{veg} del modelo de Alexandridis para el tipo de vegetación y un JTextÁrea inferior donde se explican dichos parámetros.

4.3.9 CLASE PANELVIENTO

Contiene una imagen "de relleno". Al pie del panel se informa de la fuente de dicha imagen.

4.3.10 CLASE PANELVIENTODERECHA

Contiene un JTextÁrea donde se explica la fórmula que determina el factor P_w del modelo de Alexandridis et alter. Debajo de dicho JTextÁrea se coloca una imagen con las fórmulas indicadas. A continuación viene otro JTextÁrea donde se indica la dependencia del factor P_w con respecto al ángulo θ formado entre la dirección del viento y la dirección de propagación del incendio, y una imagen debajo con una gráfica donde se refleja dicha dependencia para unos parámetros dados. Dicha imagen se toma del trabajo de Alexandridis et alter.

4.3.11 CLASE PANELELEVACION

Contiene una imagen con la elevación del terreno en la isla de Spetses. Al pinchar sobre cualquier punto de la isla nos informa sobre la elevación de dicho punto.

4.3.12 CLASE PANELELEVACIONDERECHA

Contiene un primer JTextÁrea donde se explica la imagen de la izquierda (la del PanelElevacion), un segundo JTextÁrea donde se expone la influencia del parámetro de la pendiente o elevación del terreno

(P_s, s de "slope", inclinación o pendiente) conforme a el modelo de Alexandridis et alter, una imagen con las fórmulas de dicho parámetro, y, por último, otro JTextÁrea donde se explica muy someramente la influencia de la pendiente del terreno en la extensión de los incendios.

4.3.13 CLASE PANELAUTOMATA

Contiene la imagen clasificada. Si se pincha con el puntero del ratón, aparecen las coordenadas del píxel y el valor de la densidad del mismo (escasa, intermedia, densa o área urbana). Al pie del panel existen 4 JTextField, campoVelocidad, campoDireccion, campoPosicionX y campoPosicionY, para que el usuario introduzca los valores de la velocidad del viento, de la dirección del viento y de las coordenadas del foco inicial del incendio respectivamente. Debajo de estos campos existe un "Slider" para que el usuario introduzca la velocidad del simulador. Un botón "Iniciar" permite ejecutar el autómata deseado. Esta ejecución se lleva a cabo mediante la clase "BotonEjecucion", la cual explicaremos posteriormente.

4.3.14 CLASE PANELAUTOMATADERECHA

Un primer JTextÁrea nos indica la introducción de los valores en el panel de la izquierda (PanelAutomata). Un segundo JTextÁrea nos presenta 5 ejecuciones de ejemplo insertadas en la aplicación. Debajo de este JTextÁrea existen 2 filas de botones. Para cada "Ejecución x" se tendrá la correspondiente "Imagen x". Las clases que desarrollan las ejecuciones de ejemplo son "BotonEjecucionX", y las clases que enlazan a las imágenes resultantes de dichas ejecuciones son "BotonImagenX", donde, en ambos casos, la X tendrá valores de 1 a 5. Las clases "BotonEjecucionX" se explicarán posteriormente, y las clases "BotonImagenX" simplemente cargan la imagen final resultante de la ejecución del autómata con los valores de los parámetros indicados en la parte superior del panel.

A continuación aparece otro JTextÁrea con la interpretación del algoritmo del autómata, y, finalmente, un botón "Interpretación" en el cual la clase "BotonInterpretacion" desarrolla, en cinco JTextÁrea, una breve interpretación de los resultados de la ejecución de los autómatas de los ejemplos.

Una vez visto el diseño de la GUI nos centramos ahora en la extracción de los datos de la imagen original. En la Sección 4.3.1 vimos que la clase principal de la aplicación, la clase "Automata", invocaba a las siguientes clases:

1. Firmas.
2. ImageResizer.
3. DistanciaMinima.
4. PixelesUrbanos.
5. UrbeYPlaya.
6. RepintarBorde.

Veamos ahora, una por una, dichas clases.

4.3.15 CLASE FIRMAS

La clase de Java "Firmas" carga la imagen Landsat original y el archivo de texto con la definición de las clases.

```
BufferedImage input=ImageIO.read(new File("imagenes/Spetses.png"));
//leer el archivo con la descripción de las clases
BufferedReader br=new BufferedReader(new
      FileReader("texto/DefinicionClases.txt"));
```

A continuación lee el archivo de las clases y, para cada clase crea un "TreeMap" en el que almacena el identificador de la clase (key) y una tripleta con los valores RGB correspondientes a dicha clase (value).

Seguidamente, lee el archivo con los rectángulos de las muestras,

```
br=new BufferedReader(new FileReader("texto/CoordenadasMuestras.txt"));
```

Para cada clase, para cada rectángulo de muestras de dicha clase, y para cada píxel de dicho rectángulo, calcula la media de sus colores RGB sumando los valores respectivos (r, "red", g "green" y b "blue") y luego dividiendo por el número de píxeles de dicho rectángulo).

```
while(true){
                    String line = br.readLine();
                    if (line == null) break;
                    if (line.startsWith("#")) continue;
                    StringTokenizer st = new StringTokenizer(line);
                    if (st.countTokens() < 5) continue;
                    int classId = Integer.parseInt(st.nextToken());
                    int x= Integer.parseInt(st.nextToken()); //coordenada x
                    int y= Integer.parseInt(st.nextToken()); //coordenada y
                    int w= Integer.parseInt(st.nextToken()); //anchura
                    int h= Integer.parseInt(st.nextToken()); //altura
                    Color c=classMap.get(classId);
                    if (c!=null){//tenemos una región
                          double[] accum=avgMap.get(classId);
                          int count=countMap.get(classId);
                          //colocamos los valores de los píxeles
                          for (int row=0;row<=h;row++){
                                for (int col=0;col<=w;col++){
                                      int rgb=input.getRGB(x+col, y+row);
                                      int  r  =  (int)((rgb&0x00FF0000)>>>16);
// Red level
                                      int  g  =  (int)((rgb&0x0000FF00)>>>8);
// Green level
                                      int   b   =   (int)   (rgb&0x000000FF);
// Blue level
                                      //los añadimos a la media
                                      accum[0]=accum[0]+r;
                                      accum[1]=accum[1]+g;
                                      accum[2]=accum[2]+b;
                                      count=count+1;
                                }
                                //ponemos la media y el valor del contador de
vuelta en sus mapas correspondientes
```

```
                                    avgMap.put(classId,accum);
                                    countMap.put(classId, count);

      }//System.out.println(accum[0]);System.out.println(accum[1]);System.ou
t.println(accum[2]);System.out.println();System.out.println(count);
                    }

             }
```

Código 4. 2: Extracto de código de la clase "Firmas"

Calcula después la media de todos los rectángulos de cada clase, obteniendo así la firma correspondiente a dicha clase. Finalmente, escribe los resultados en el archivo "Firmas.txt".

4.3.16 CLASE IMAGERESIZER

La imagen Landsat original ("imagenes/SpetsesGrande.png") posee un tamaño de 1011x769 píxeles. Para poder trabajar con dicha imagen en la GUI de la aplicación, hay que reducir dicha imagen al tamaño de 700x550. Para que la reducción no produzca una pérdida en la calidad de la imagen, se utiliza el método de interpolación bicúbica en el "renderizado",

```
g2d.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                RenderingHints.VALUE_INTERPOLATION_BICUBIC);
```

Código 4. 3: Extracto de código de la clase "ImageResizer"

La imagen resultante es "imagenes/Spetses.png". Ésta será la imagen inicial sin clasificar, la entrada primera al algoritmo de clasificación.

4.3.17 CLASE DISTANCIAMINIMA

Esta clase Java es la encargada de clasificar cada píxel de la imagen de la isla de Spetses, mediante el algoritmo clasificador de la Distancia Mínima.

En primer lugar se carga la imagen inicial, que está sin procesar, "imagenes/Spetses.png".

```
BufferedImage input=ImageIO.read(new File("imagenes/Spetses.png"));
```

Después, se lee el archivo con la definición de las clases

```
BufferedReader br=new BufferedReader( new
      FileReader("texto/DefinicionClases.txt"));
```

Código 4. 4: Extracto de código de la clase "DistanciaMinima"

Un bucle recorre todos los píxeles de la imagen original y asocia a cada píxel el valor de su clase. Para ello, se elige aquella clase cuya firma es la más próxima a los valores RGB del píxel de la imagen original. Una vez asignado el píxel a

su clase se le pinta con el color RGB correspondiente a dicha clase (los valores de color de cada clase están en el archivo "texto/DefinicionClases.txt"). Finalmente, se dibuja la nueva imagen, ya clasificada, y se guarda en el archivo "imagenes/ImagenDistanciaMinimaInicial.png".

4.3.18 CLASE PIXELESURBANOS

Como hemos explicado en la Sección 4.3.6, correspondiente a la clase de Java "PanelVegetacionDerecha", la imagen resultante del algoritmo de la Distancia Mínima presenta ciertos errores y debe ser postprocesada.

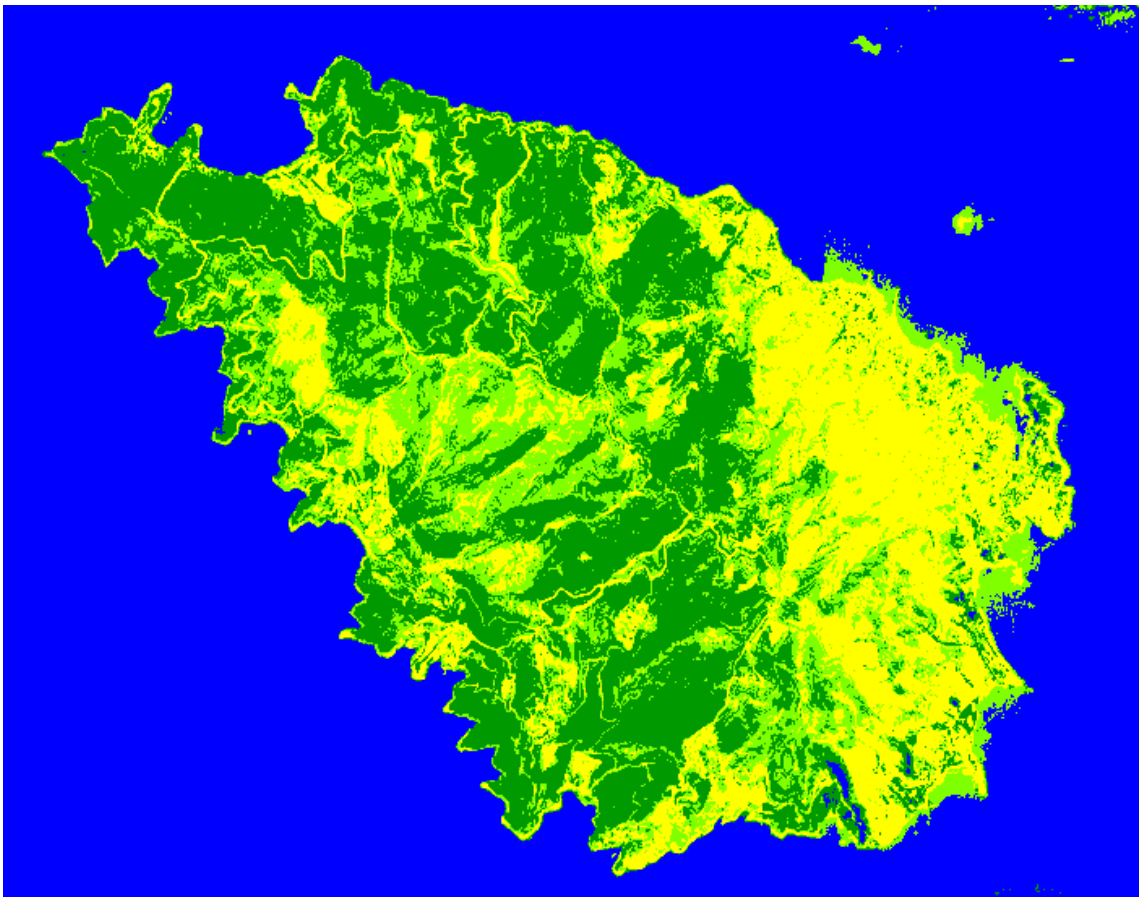


Figura 4. 2: ImagenDistanciaMinimaInicial.png

La clase de Java "PixelesUrbanos" escribe el archivo de texto "texto/ArrayPixelesUrbanos.txt", el cual contiene las coordenadas de los puntos correspondientes a terrenos urbanos.

Para ello usa la imagen "imagenes/SpetsesAlexandridis.png". De la misma, extrae, en función de sus valores RGB, las coordenadas de los píxeles correspondientes al área clasificada en dicha imagen como urbana. Dichas coordenadas las almacena en el archivo de texto plano mencionado en el anterior párrafo.

4.3.19 CLASE URBEYPLAYA

Para continuar el postprocesado de la imagen resultante del algoritmo de la Distancia Mínima, es necesario eliminar las zonas de las nubes y los errores en

las playas, tal y como se explica en la Sección 4.3.6. Igualmente, es necesario introducir en color negro los píxeles correspondientes a áreas urbanas.

La imagen de la Figura 4.2 se procesa eliminando las nubes, lo que produce la imagen "imagenes/ImagenDistanciaMinimaSinNubes.png", la cual vemos a continuación, en la Figura 4.3, donde se han eliminado las nubes:

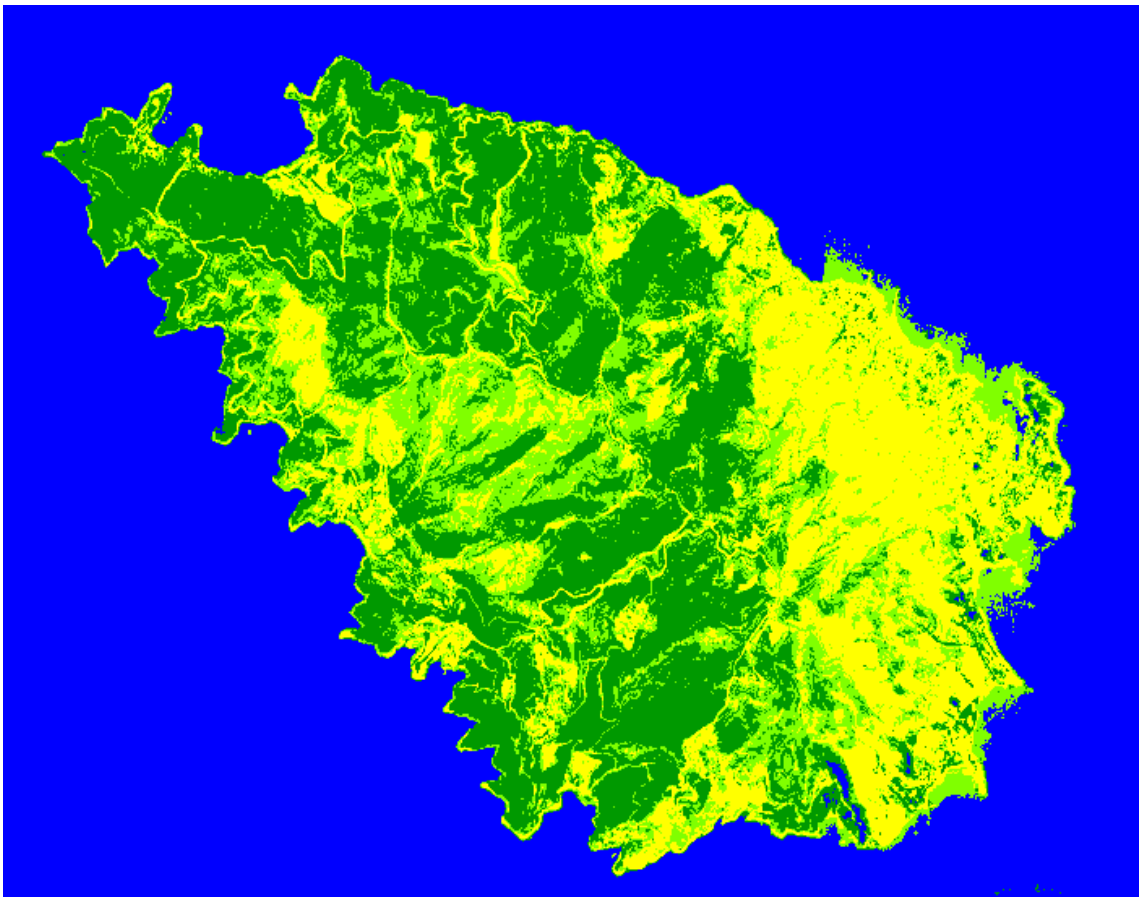


Figura 4. 3: ImagenDistanciaMinimaSinNubes.png

A partir de la lectura de los píxeles urbanos resultantes de la ejecución de la clase "PixelesUrbanos" (Sección 4.3.18), los cuales se extraen del archivo de texto "texto/ArrayPixelesUrbanos.txt", se pinta de negro en la imagen la zona urbana, resultando "imagenes/ImagenDistanciaMinimaProcesada.png", la cual vemos a continuación, en la Figura 4.4:

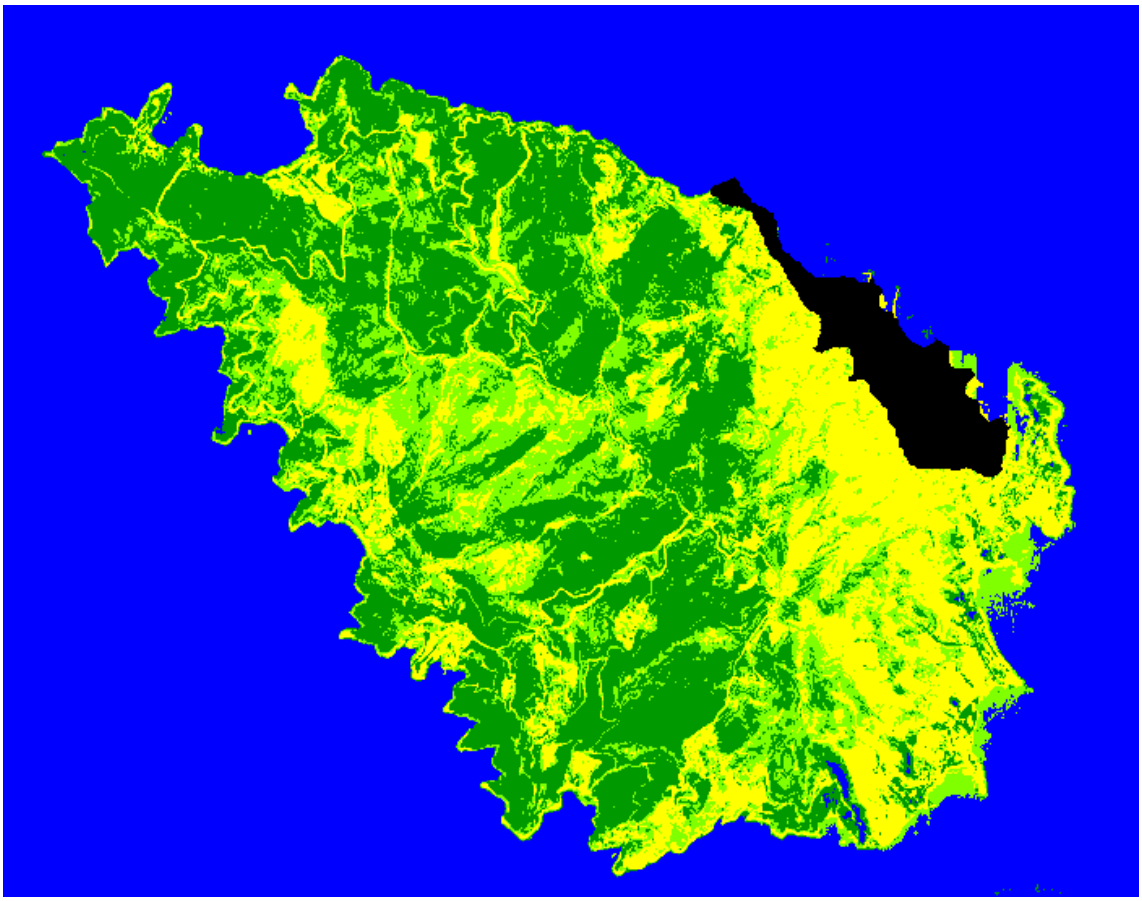


Figura 4. 4: ImagenDistanciaMinimaProcesada.png

Finalmente, dicha imagen se depura, para eliminar los errores en la parte suroeste de la isla, en el litoral, con paint, resultando la imagen "imagenes/ImagenDistanciaMinimaFinal.png", la cual vemos a continuación, en la Figura 4.5:

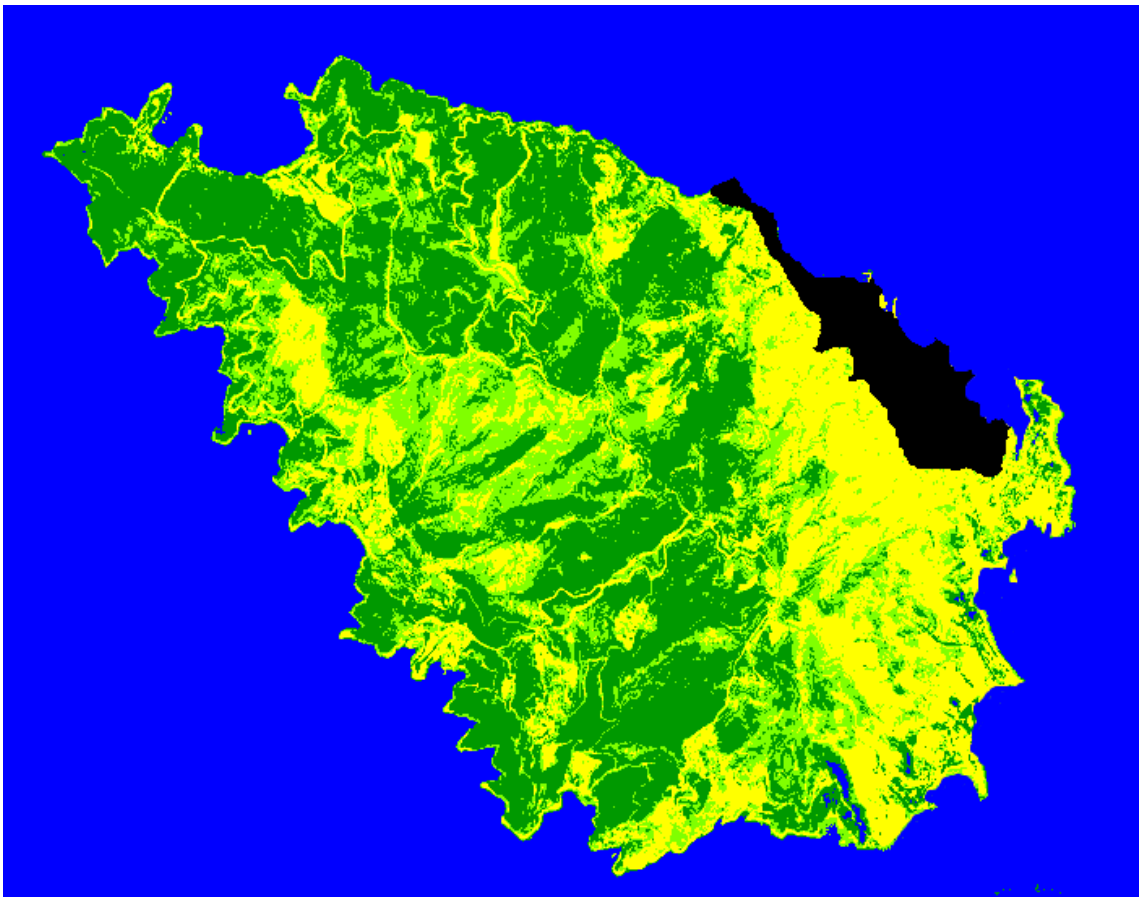


Figura 4. 5: ImagenDistanciaMinimaFinal.png

Esta es la imagen final postprocesada del algoritmo de la Distancia Mínima.

4.3.20 CLASE REPINTARBORDE

Los datos de la elevación del terreno los obtenemos de [Russo, 2014]. En dicho trabajo, el cual se basa en el autómata de Alexandridis et Alter, aparece una imagen de la isla con los valores de la elevación en forma de gradaciones de color (véase la Figura 4.6).Tal imagen, procesada y ampliada en paint y recolocada en la clase de Java "TransladarImagen", se halla en "imagenes/ElevacionSpetses.png".Dicha imagen se modifica a rojo, para obtener un mejor contraste.

A continuación se presentan las 2 imágenes:

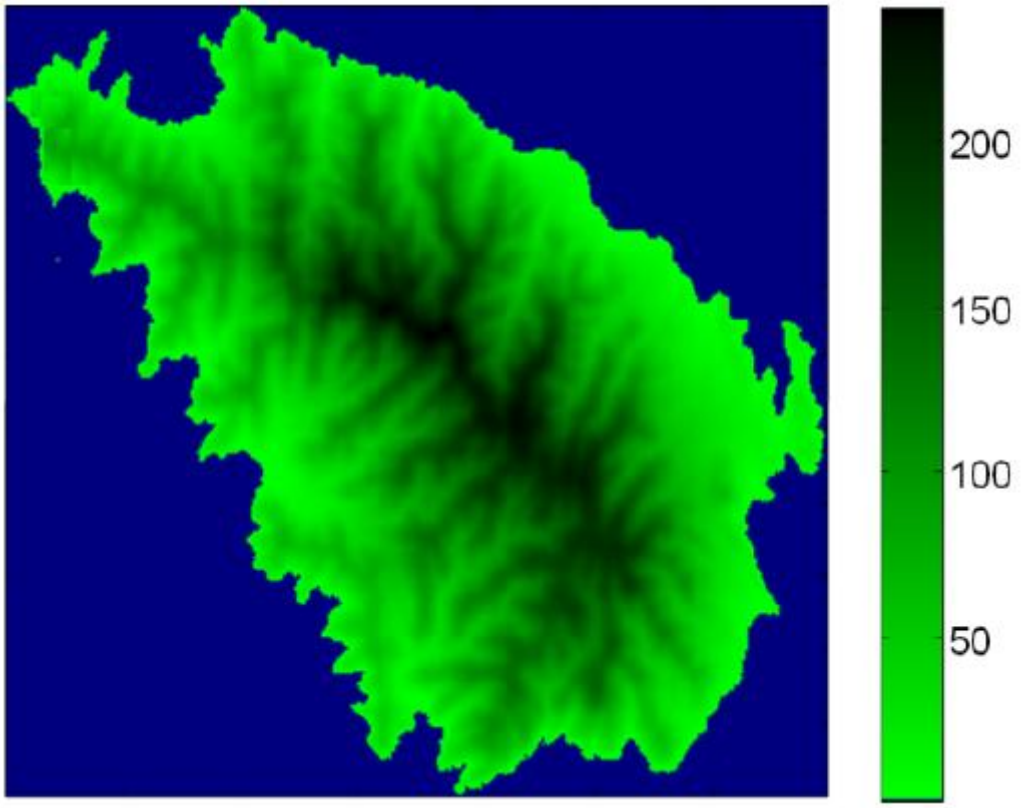


Figura 4. 6: Elevación de Spetses

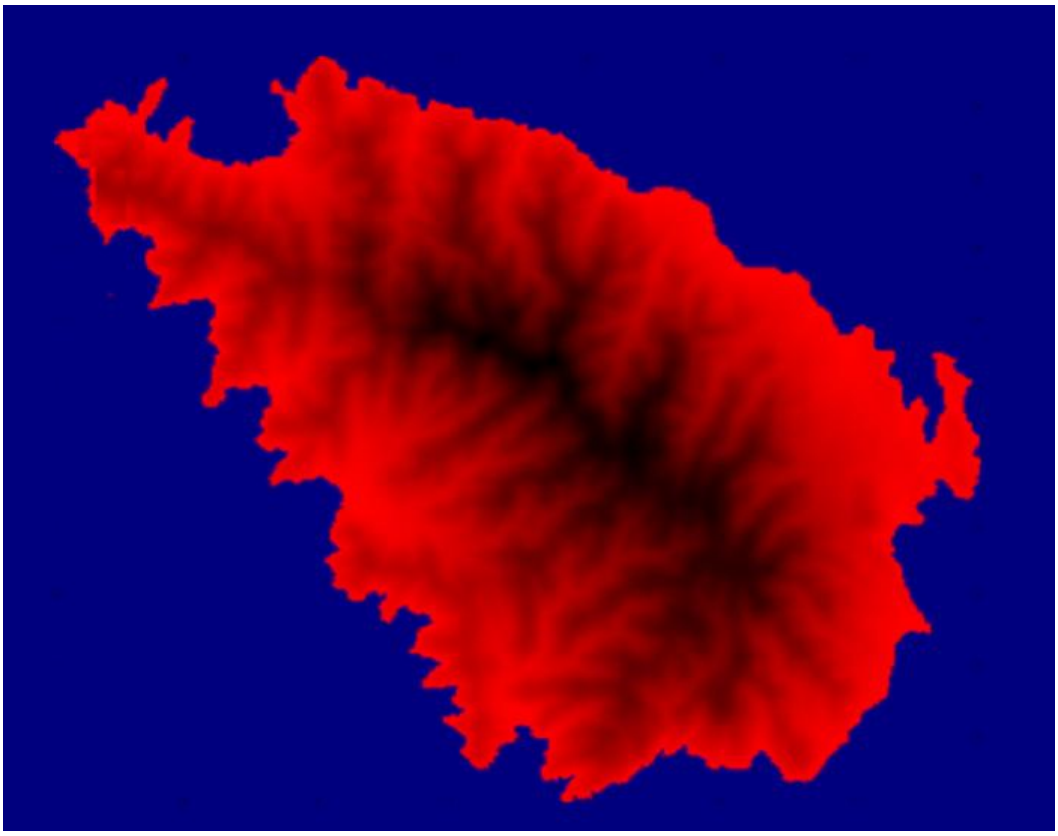


Figura 4. 7: ElevacionSpetsesFinal2.png

4.4 DISEÑO DEL AUTÓMATA

La clase "PanelAutomata" posee 4 JTextField para que el usuario introduzca los datos de la velocidad del viento, la dirección del viento y las coordenadas x e y del foco inicial del incendio. Igualmente, posee un "Slider" para regular la velocidad de ejecución del autómata. Al pulsar el botón "Iniciar", se llama a la clase "BotonEjecucion", la cual presenta le ejecución del autómata en un JFrame. Dentro de dicho Frame se halla un panel, el cual es una instancia de la clase "PanelAutomataExterno". Veamos ahora ambas clases.

4.4.1 CLASE BOTONEJECUCION

Desde la clase "PanelAutomata" se llama al constructor de la clase "BotonEjecucion" pasándole los siguientes parámetros: velocidad del temporizador, velocidad del viento, dirección del viento, posición x y posición y. En el Frame se ejecuta una nueva instancia de "PanelAutomataExterno", a la cual se le habrán reenviado los anteriores parámetros. En este panel es donde se ejecutará el autómata de Alexandridis et Alter propiamente dicho.

En "BotonEjecucion", debajo del panel se añaden dos botones, "Parar" y "Seguir". El primero para la ejecución de la instancia de "PanelAutomataExterno", llamando a un método estático de dicha clase, y el segundo la continúa, también invocando otro método estático.

En cada iteración de la clase "PanelAutomataExterno", la cual está controlada por un temporizador (Javax.swing.Timer) se modifica el texto de una etiqueta JLabel en la clase "BotonEjecucion", de forma que nos indica la iteración actual del autómata. Igualmente, en una etiqueta JLabel se nos indica su correspondencia con el tiempo "real".

Igualmente, en "BotonEjecucion" se presentan dos JTextField que nos permiten cambiar la velocidad del viento y su dirección durante la ejecución del autómata. Para ello, se crea en la clase "PanelAutomataExterno" un método estático, el cual permite modificar los valores de la velocidad y dirección del viento para un autómata en ejecución.

Cuando desde "BotonEjecucion" se invoca a "PanelAutomataExterno", a continuación se invoca también a la clase de Java "DibujarFlecha", contenida en el paquete "pruebas". En la invocación se pasan los parámetros de la velocidad y de la dirección del viento, de forma que la flecha resultante tendrá un tamaño acorde al módulo de la velocidad, y una orientación acorde a la dirección de la misma. La flecha se dibuja en un JFrame externo. El código de "DibujarFlecha" es una adaptación de circunstancia de un "snippet" obtenido en `http://www.bytemycode.com/snippets/snippet/82/`.

4.4.2 CLASES BOTONEJECUCION1…BOTONEJECUCION5

Estas clases simplemente presentan ejecuciones de ejemplo del autómata, con los parámetros preestablecidos. La ejecución se presenta en un JFrame y dos

botones en la parte inferior permiten al usuario "Parar" y "Seguir" la ejecución del autómata.

Los parámetros de estas ejecuciones de ejemplo son:

1. Ejecución 1: veloc. del viento: 9 m/s, dirección: 90 grados, píxel x: 300, píxel y: 300.
2. Ejecución 2: veloc. del viento: 9 m/s, dirección: 0 grados, píxel x: 410, píxel y: 439.
3. Ejecución 3: veloc. del viento: 9 m/s, dirección: 270 grados, píxel x: 81, píxel y: 107.
4. Ejecución 4: veloc. del viento: 9 m/s,dirección: 180 grados, píxel x: 346, píxel y: 237.
5. Ejecución 5: veloc. del viento: 9 m/s, dirección: 225 grados, píxel x: 249, píxel y: 307.

4.4.3 CLASE PANELAUTOMATAEXTERNO

Esta es la clase que desarrolla el autómata de Alexandridis et Alter. La clase es bastante compleja, así que la iremos explicando paso a paso, empleando fragmentos de código.

El constructor de la clase es invocado desde "BotonEjecucion", recibiendo los parámetros siguientes: velocidad del temporizador, velocidad del viento, dirección del viento, posición x y posición y.

```
public PanelAutomataExterno(int velocidadTemporizador,int velocidadViento,int
direccionViento,int posicionX,int posicionY
```

Se obtiene, a continuación, la imagen clasificada de la isla de Spetses, sobre la cual se ejecutará el autómata.

```
//obtenemos una planarImage a partir de una imagen preexistente
imagen=JAI.create("fileload", "imagenes/ImagenDistanciaMinimaFinal.png");
```

De dicha imagen se obtienen los datos de la densidad y el tipo de la vegetación de cada píxel, llamando al método "densidadYTipoVegetacion()", el cual veremos posteriormente.

```
densidadYTipoVegetacion();
```

Obtenemos igualmente, la imagen con la elevación del terreno:

```
imagenElevacion = leerImagen("imagenes/ElevacionSpetsesFinal2.png");
```

De dicha imagen obtenemos los datos de la elevación para cada píxel, llamando al método "elevacion()", el cual veremos posteriormente.

```
elevacion();
```

Dado que para actualizar el estado de cada celda en las iteraciones necesitamos conocer el estado en el instante actual (t) y en el siguiente (t+1) es necesario mantener dos arrays de celdas, el del estado actual y el del estado

siguiente. El estado inicial de las celdas será el obtenido en el método "densidadYTipoVegtacion()", y será como sigue: para todas las celdas de "mar" y "área urbana" el estado será 1, pues son no combustibles. Para todo el resto el estado será 2, pues son combustibles pero todavía no han ardido. Para el foco inicial del incendio el estado será 3 (la célula arde). El foco inicial ha sido pasado como parámetro al constructor.

```
//inicializamos los 2 arrays
celdas1 = new int[EJEX][EJEY];
celdas2 = new int[EJEX][EJEY];
for (int i = 0; i < EJEX; i++) {
      for (int j = 0; j < EJEY; j++) {
            if ((i==getPosicionX())&&(j==getPosicionY())){
                  celdas1[i][j] = 3;//celdas que arden inicialmente, cuyo
                                    estado inicial es 3
             }
             else{
                  celdas1[i][j] =arrayEstado[i][j];
             }
            //cells2[i][j] = 0;
       }
}
```

Código 4. 5: Extracto de código de la clase "PanelAutomataExterno", parte 1

Necesitaremos un nuevo array. Éste contendrá, al final de cada iteración, a las celdas que se han incendiado. Dicho array es necesario, pues la iteración se ejecuta celda por celda, y si una celda posee una probabilidad *p* tal que dicha celda se prendería, dicha celda no puede prenderse hasta el final de la iteración, pues si no, modificaría sobre la marcha los datos de dicha celda, y los datos de una celda han de permanecer constantes hasta que termine del todo la iteración. Tal es el propósito del array "arrayIncendiadas". Iniciaremos dicho array a false, excepto para la celda del foco inicial del incendio.

```
//inicializamos el array con las células que se van incendiando a false
for (int i = 0; i < EJEX; i++) {
      for (int j = 0; j < EJEY; j++) {
            arrayIncendiadas[i][j]=false;
arrayIncendiadas[getPosicionX()][getPosicionY()]=true;
      }
}
```

Código 4. 6: Extracto de código de la clase "PanelAutomataExterno", parte 2

Cada paso en el autómata viene controlado por un temporizador (javax.swing.Timer).

```
iteracion=0;
temporizador = new Timer(getVelocidadTemporizador(), this);
temporizador.start();

}
```

Código 4. 7: Extracto de código de la clase "PanelAutomataExterno", parte 3

Con lo anterior ya poseemos, para cada celda de la imagen clasificada de 700x550, todos los datos relevantes para el autómata, e igualmente tenemos los arrays inicializados. Antes de proseguir, veamos los métodos anteriormente citados, "densidadYTipoVegetacion()" y "elevacion()".

4.4.4 MÉTODO DENSIDADYTIPOVEGETACION

```
public static void densidadYTipoVegetacion() {
	for (int i=0;i<700;i++){
		for (int j=0;j<550;j++){
			int rgbColor=imagenDensidad.getRGB(i,j);
			int red = (rgbColor >> 16) & 0x000000FF;
			 int green = (rgbColor >>8 ) & 0x000000FF;
			 int blue = (rgbColor) & 0x000000FF;
//el factor Pden, "factorDensidad" depende de la densidad de la vegetación. Si escasa, -0.4. Si normal, 0. Si densa, 0.3. Para los dos
//tipos restantes, mar y urbana, damos un valor de 2, que luego eliminaremos con un condicional
//el factor Pveg, "factorTipoVegetacion" depende del tipo de la vegetación. Si escasa, -0.3. Si normal, 0. Si densa, 0.4. Mar y urbana 2, que luego eliminamos..
//inicialmente, estado 1(sin combustible) Mar y Urbano. estado 2(resto)
			double factorDensidad=0,factorTipoVegetacion=0;int
				estado=0;
			//mar
			if ((red==0)&&(green==0)&&(blue==255)){
				factorDensidad=2;factorTipoVegetacion=-
				2;estado=1;
			}
			//escasa
			if ((red==255)&&(green==255)&&(blue==0)){
				factorDensidad=-0.4;factorTipoVegetación=
				-0.3;estado=2;contadorCeldas++;

			}
			//intermedia
			if ((red==128)&&(green==255)&&(blue==0)){
				factorDensidad=0;factorTipoVegetacion=0;
				estado=2;contadorCeldas++;
			}
			//densa
			if ((red==0)&&(green==153)&&(blue==0)){
				factorDensidad=0.3;factorTipoVegetacion=0.4;
				estado=2;contadorCeldas++;
			}
			//urbana
			if ((red==0)&&(green==0)&&(blue==0)){
				factorDensidad=2;factorTipoVegetacion=-2;
				estado=1;contadorCeldas++;
			}
			//en funcion del tipo de la vegetacion daremos posteriormente valores al factor Pveg y al factor Pden
			arrayDensidadVegetacion[i][j]=factorDensidad;
			arrayTipoVegetacion[i][j]=factorTipoVegetacion;
			arrayEstado[i][j]=estado;
		}
```

```
        }
    }
```

Código 4. 8: Extracto de código de la clase "DensidadYTipoVegetacion"

Como vemos, mediante dos bucles recorremos los píxeles de la imagen clasificada. Para cada píxel, obtenemos sus valores en enteros RGB, y a partir de dichos enteros damos valor a los siguientes parámetros:

1. FactorDensidad.
2. FactorTipoVegetacion.
3. Estado.

A la salida de este método, poseemos los valores de los arrays "arrayDensidadVegetacion", "arrayTipoVegetacion" y "arrayEstado".

El factor P_{den}, "factorDensidad" depende de la densidad de la vegetación. Si escasa, -0.4. Si normal, 0. Si densa, 0.3. Para los dos tipos restantes, mar y urbana, damos un valor de 2, que luego eliminaremos, en su momento, con un condicional. La densidad de la vegetación se obtiene de la imagen clasificada a partir de los colores de la misma, los cuales están definidos en el archivo "DefinicionClases.txt". Vea la Sección 4.3.6, "PanelVegetacionDerecha", para una explicación más detallada.

El factor P_{veg}, "factorTipoVegetacion" depende del tipo de la vegetación. Si escasa, -0.3. Si normal, 0. Si densa, 0.4. . Para los dos tipos restantes, mar y urbana, damos un valor de 2, que luego eliminaremos, en su momento.

Inicialmente, estado 1(sin combustible) "Mar" y "Urbano". Estado 2 (resto).

La variable contadorCeldas sirve para contar las celdas de la isla. Dicha variable se usará al final del método "actionPerformed" para averiguar el porcentaje que se ha quemado de la isla, una vez se haya terminado el incendio. Ver la sección 4.4.7.

4.4.5 MÉTODO ELEVACION

```
public static void elevacion() {
      for (int i=0;i<700;i++){
            for (int j=0;j<550;j++){
                  int rgbColor=imagenElevacion.getRGB(i,j);
                  int red = (rgbColor >> 16) & 0x000000FF;
                 int green = (rgbColor >>8 ) & 0x000000FF;
                  int blue = (rgbColor) & 0x000000FF;
                  int elevacionInversa=red;
                  if (elevacionInversa>248){
                        elevacionInversa=248;
             }
             int elevacion;
             if ((red==0)&&(green==0)&&(blue==127)){
                  elevacion=0;
```

```
                    }else{
                         elevacion=248-elevacionInversa;
                    }

                    //en funcion del tipo de la vegetacion daremos
posteriormente valores al factor Pveg y al factor Pden
                    arrayElevacion[i][j]=elevacion;
                  }
            }
      }
```

Código 4. 9: Extracto de código de la clase "Elevacion"

A partir de la imagen de la elevación del terreno, obtenemos para cada píxel sus datos RGB, y, a partir de dichos datos, el valor de la elevación de cada píxel. A la salida del método poseemos los valores del array "arrayElevacion".

Veamos ahora el método paint(), el cual modifica los colores de la imagen sobre la que se desarrolla el autómata, en cada iteración.

4.4.6 MÉTODO PAINT

```
//este método pinta aquellos elementos de la imagen que cambian en cada paso
de la simulación. Aquí no ejecutamos ningún cambio de estado
public void paint(Graphics g) {
      super.paint(g);
      Graphics2D g2 = (Graphics2D)g;
//Graphics2D g2=imagen.getAsBufferedImage().createGraphics();
//sólo modificamos aquellos píxeles que cambian. El estado será: 3,rojo(arde)
y 4,violeta(quemada)
      for (int i = 0; i < EJEX; i++) {
          for (int j = 0; j < EJEY; j++) {
            switch(celdas1[i][j]){
            case 1: //si la celda no es combustible, no se cambia su
color(negro o azul)
                  break;
            case 2: //si la celda no arde, no se cambia su color(sea este
amarillo,verde claro o verde oscuro)
                  break;
            case 3: //si la celda arde, se cambia su color a rojo
                    g2.setColor(Color.red);
                 //imagen.getAsBufferedImage().setRGB(i, j, colorRojo);
                    g2.fillOval(i,j , 2, 2);
                   //imagenDensidad.setRGB(i, j, colorRojo);
                  break;
            case 4: //si la celda ha ardido, se cambia su color a naranja
                    g2.setColor(Color.orange);
                    //imagen.getAsBufferedImage().setRGB(i, j, colorVioleta);
                    g2.fillOval(i,j , 2, 2);
                    //imagenDensidad.setRGB(i, j, colorVioleta);
                    }
             }
          }
}
```

Código 4. 10: Extracto de código del método "paint"

Este método se ejecuta en cada iteración, regulado por el temporizador de javax.swing.Timer. El método recorre todos los píxeles de la imagen, modificando sólo aquellos píxeles cuyo estado se ha modificado. Tales píxeles sólo pueden ser aquellos con estado 3 ó 4. Si el estado se ha modificado a 3 (el píxel está ardiendo), el color del píxel se cambia a rojo, y si se ha modificado a 4 (el píxel ya ha ardido) su color se cambia a naranja. El estado a la salida de cada iteración se obtiene del array celdas1, pues dicho array se actualiza con el valor resultante de la iteración del autómata al final del método "actionPerformed", que es el que actualiza el autómata. Este método lo vemos a continuación.

4.4.7 MÉTODO ACTIONPERFORMED

```
//paso h de la simulación. Aquí se determina el valor de las celdas, que será
su estado. Los 4 estados posibles son:
       //1(celda no combustible),2(celda combustible),3(celda ardiendo) y
4(celda quemada)
public void actionPerformed(ActionEvent e) {
      celdas2=celdas1;
            for (int x = 0; x < EJEX; x++) {
            for (int y = 0; y < EJEY; y++) {

                switch (celdas2[x][y]){
                  case 1: break;//si la celda es no combustible, no hacemos
                        nada  temporalCells2[x][y]=1;
                  case 2: break;//si la celda es combustible, pero no arde,
                        no hacemos nada  temporalCells2[x][y]=2;
                  case 4: break;//si la celda ha ardido, no hacemos nada.
Opcionalmente, se puede cambiar su estado a cinco, y comprobar el repintado
```

Código 4. 11: Extracto de código del método "actionPerformed", parte 1

Este método recorre un array de 700x550. Lo que ha pintado el método paint es el array celdas1. Al entrar en este método, el array celdas2 coincide con el estado actual dibujado sobre la imagen. Se recorre toda la imagen, y se verifica el estado de cada celda de la misma:

1. Si el estado es 1, no se hace nada (la celda no es combustible).
2. Si el estado es 2, no se hace nada (la celda no arde).
3. Si el estado es 4, no se hace nada (la celda ya ha ardido).

Sólo se hace algo si el estado de la celda es 3, es decir, la celda está ardiendo. Cuando la celda arde, primero cambiamos su estado a 4 y después recorremos su vecindario, para calcular la probabilidad de que sus celdas vecinas ardan. Se recorre el vecindario (los límites son x1 x2 en sentido horizontal e y1 a y2 en sentido vertical) de la celda (x,y), que es la celda central, y se verifica cada celda (i, j) de dicho vecindario, tal y como vemos a continuación:

```
case 3//Calculamos la probabilidad de que sus celdas vecinas ardan
      //no se puede cambiar el estado de una celda hasta haber recorrido
      //todo el automata
```

```
arrayApagadas[x][y]=true;
System.out.println("\n************CALCULO CELDA CENTRAL:
"+x+","+y+"**********************************************************
****************");
int x1 = (x>0)?x-1:x;//por los limites x=0 y x=699
int x2 = (x<EJEX-1)?x+1:x;
int y1 = (y>0)?y-1:y;//por los limites y=0 y y=549
int y2 = (y<EJEY-1)?y+1:y;
//(x,y) es la celda central que arde. (i,j) serán sus vecinas
for (int i = x1; i <= x2; i++){
      for (int j = y1; j <= y2; j++){
```

Código 4. 12: Extracto de código del método "actionPerformed", parte 2

Para cada celda (i,j) del vecindario comprobaremos su estado. Si (i,j) coincide con (x,y) no hacemos nada. Si no, en el else, comprobaremos el estado de esta celda vecina:

1. Si su estado es 1, la celda no puede arder, no se hace nada.
2. Si su estado es 3, la celda ya está ardiendo, con lo cual no se hace nada.
3. Si su estado es 4, la celda ya ha ardido, no se hace nada.
4. Si su estado es 2, la celda es susceptible de arder. En este caso, calcularemos la probabilidad de que arda. Si P_{burn} es tal que consideramos que la celda vecina se ha contagiado del fuego, es decir, que arde, entonces añadiremos dicha celda al array que contiene las celdas incendiadas, "arrayIncendiadas".

```
System.out.println("\nCALCULO PROBABILIDAD VECINO "+i+","+j);
      //si la central y la vecina coinciden
      if ((i==x)&&(j==y)){
      System.out.println("la celda base y la vecina
            coinciden");
      System.out.println("Eliminado de la lista de
            pixeles ardiendo el pixel :"+x+", "+y);
      }else{
          switch(celdas2[i][j]){
            case 1: break;//celdas con mar o urbano, su estado
                  es 1, nunca arden. No hacemos nada
            case 2:
                  boolean ardeVecino=false;
                  ardeVecino = calculoProbabilidad(x,y,i,j);

                  //si el vecino arde, a la salida del bucle externo
debe quedar modificado el estado del pixel i,j a 3
                  if (ardeVecino){
                    System.out.println("Añadido a la lista de pixeles
                        ardiendo el pixel
                        **************************** :"+i+", "+j);
                        arrayIncendiadas[i][j]=true;

                  }
                  break;
            case 3: break;//celdas que arden. No hacemos nada
            case 4: break; //celdas quemadas. No hacemos nada
```

```
                }                                                    }


                                                }
                                            }
                                            break;
                                    case 4: break;//si la celda ha ardido, no hacemos
nada. Opcionalmente, se puede cambiar su estado a cinco, y comprobar el
repintado
                                }
                              }//for interno
                            }//for externo
                            //aquí cells2 representa la siguiente iteracion en el
automata, tras aplicar las reglas(es decir,en el paso h,
                            //la parrilla es cells1, y en h+1 es cells2, tras haber
aplicado las reglas a h
```

Código 4. 13: Extracto de código del método "actionPerformed", parte 3

Una vez que salimos del bucle, necesitamos un mecanismo para que el autómata pare su ejecución cuando no haya más celdas ardiendo. Para ello, usamos un contador. Si recorremos el estado de las celdas y ninguna tiene estado 3, paramos el autómata con temporizador.stop(). A continuación, calculamos el porcentaje total quemado de la isla. Éste será igual al número de celdas quemadas multiplicado por cien y partido por el número total de celdas de la isla.

```
int contador=0;
for (int x = 0; x < EJEX; x++) {
      for (int y = 0; y < EJEY; y++) {
            if (arrayApagadas[x][y]){
                celdas2[x][y]=4;arrayIncendiadas[x][y]=false;
                System.out.println("la celda "+x+","+y+" está apagada");
            }
            if (arrayIncendiadas[x][y]){//si la celda ha sido incendiada, se
                  cambia aquí
                celdas2[x][y]=3;
                System.out.println("la celda "+x+","+y+" se ha incendiado");
                contador=contador+1;
            }
      }
}
if (contador==0){
      System.out.println("El autómata ha terminado: ya no hay más celdas
            ardiendo.");temporizador.stop();
      System.out.println("Contador celdas: "+contadorCeldas);
      System.out.println("Contador celdas quemadas:
            "+contadorCeldasQuemadas);
      int porcentaje=(contadorCeldasQuemadas*100)/contadorCeldas;
      System.out.println("El porcentaje de la isla quemada es del
            "+porcentaje+" por ciento");}
```

Código 4. 14: Extracto de código del método "actionPerformed", parte 4

El estado resultante de la iteración en celdas2 se pasa a celdas1, convirtiéndose en el nuevo estado inicial.

```
celdas1=celdas2;//nuevo estado inicial en el paso h+1, es lo que se pinta en
      el método paint
```

Puesto que en la clase "BotonEjecucion" (y en las ejecuciones de ejemplo "BotonEjecucionX " donde X va de 1 a 5 y las cuales veremos posteriormente) indicamos en el campo de una etiqueta cuál es la iteración actual, necesitamos modificar dicho campo desde aquí. Igualmente, conociendo que en el modelo de Alexandridis una iteración equivale a 2.18 minutos de tiempo "real", calculamos igualmente cuál es el tiempo estimado que lleva el incendio ardiendo,

```
iteracion=iteracion+1;
//pasamos a las ejecuciones el valor de la iteración
                if (BotonEjecucion.existo){
                  BotonEjecucion.etiquetaIteracion.setText("Paso:
"+iteracion);
                  double                  tEstimadoDouble=iteracion*2.18;int
tEstimadoInt=(int)tEstimadoDouble;int          horas=(int)tEstimadoInt/60;int
minutos=tEstimadoInt%60;
                  BotonEjecucion.etiquetaTiempo.setText(", tiempo estimado:
horas: "+horas+", minutos: "+minutos);
                }
                if (BotonEjecucion1.existo){
                  BotonEjecucion1.etiquetaIteracion.setText("Paso:
"+iteracion);
                  double                  tEstimadoDouble=iteracion*2.18;int
tEstimadoInt=(int)tEstimadoDouble;int          horas=(int)tEstimadoInt/60;int
minutos=tEstimadoInt%60;
                  BotonEjecucion1.etiquetaTiempo.setText(", tiempo estimado:
horas: "+horas+", minutos: "+minutos);
                }
                if (BotonEjecucion2.existo){
                  BotonEjecucion2.etiquetaIteracion.setText("Paso:
"+iteracion);
                  double                  tEstimadoDouble=iteracion*2.18;int
tEstimadoInt=(int)tEstimadoDouble;int          horas=(int)tEstimadoInt/60;int
minutos=tEstimadoInt%60;
                  BotonEjecucion2.etiquetaTiempo.setText(", tiempo estimado:
horas: "+horas+", minutos: "+minutos);
                }
                if (BotonEjecucion3.existo){
                  BotonEjecucion3.etiquetaIteracion.setText("Paso:
"+iteracion);
                  double                  tEstimadoDouble=iteracion*2.18;int
tEstimadoInt=(int)tEstimadoDouble;int          horas=(int)tEstimadoInt/60;int
minutos=tEstimadoInt%60;
                  BotonEjecucion3.etiquetaTiempo.setText(", tiempo estimado:
horas: "+horas+", minutos: "+minutos);
                }
                if (BotonEjecucion4.existo){
                  BotonEjecucion4.etiquetaIteracion.setText("Paso:
"+iteracion);
                  double                  tEstimadoDouble=iteracion*2.18;int
tEstimadoInt=(int)tEstimadoDouble;int          horas=(int)tEstimadoInt/60;int
minutos=tEstimadoInt%60;
```

```
                    BotonEjecucion4.etiquetaTiempo.setText(", tiempo estimado:
horas: "+horas+", minutos: "+minutos);
                }
                if (BotonEjecucion5.existo){
                    BotonEjecucion5.etiquetaIteracion.setText("Paso:
"+iteracion);
                    double                  tEstimadoDouble=iteracion*2.18;int
tEstimadoInt=(int)tEstimadoDouble;int          horas=(int)tEstimadoInt/60;int
minutos=tEstimadoInt%60;
                    BotonEjecucion5.etiquetaTiempo.setText(", tiempo estimado:
horas: "+horas+", minutos: "+minutos);
                }
System.out.println("\nterminado action performed. Paso de la simulación:
      "+iteracion);
```

Hay que avisar a "paint()", para que pinte los cambios efectuados.

```
      repaint();
}
```

Código 4. 15: Extracto de código del método "actionPerformed", parte 5

4.4.8 MÉTODO CALCULOPROBABILIDAD

Este método calcula la probabilidad de que, dada la celda central (x,y) su celda vecina (i,j) se prenda.

```
//dada la célula ardiendo (x,y) hay que calcular la probabilidad Pburn de que
la célula vecina(i,j) se prenda
public boolean calculoProbabilidad(int x,int y,int i,int j) throws
      IOException{
      velocidadViento=getVelocidadViento();
      direccionViento=getDireccionViento();
      //final double c1=0.045;//Alexandridis tabla 4, pagina 199
      //final double c2=0.131;//Alexandridis tabla 4, pagina 199
      //calculo del angulo formado entre la direccion del viento y la de
            propagacion del incendio.Depende de la posición de la celda
            (i,j) respecto a la central (x,y)
```

Código 4. 16: Extracto de código del método "calculoProbabilidad", parte 1

Una vez obtenidas la velocidad y la dirección del viento, se continúa con otros cálculos. Para obtener la dirección de la propagación del viento se efectúa el método "calculoDireccionPropagacion", cuya estructura veremos posteriormente. Se continúa con el cálculo del ángulo θ (el ángulo formado entre la dirección de propagación del incendio y la dirección del viento).

```
      direccionPropagacion=calculoDireccionPropagacion(x,y,i,j);
      System.out.println("direccionPropagacion:
            "+direccionPropagacion);
      Theta=direccionPropagacion-direccionViento;
      System.out.println("theta: "+Theta);
      //Math.cos es en radianes, con lo que tenemos que pasar theta a
            radianes, Theta*2*Math.PI/360
```

```
FsubT=Math.exp(velocidadViento*c2*(Math.cos(Theta*2*Math.PI/360)-1));
System.out.println("FsubT: "+FsubT);//inicialmente vale 1
factorViento=FsubT*Math.exp(velocidadViento*c1);
System.out.println("factorViento: "+factorViento);//inicialmente vale
      1.4993025...
```

Código 4. 17: Extracto de código del método "calculoProbabilidad", parte 2

Tras calcular el factor viento, P_w, se requieren los valores del factor densidad de la vegetación, P_{den}, y los del factor tipo de la vegetación, P_{veg}. Tales valores se obtienen respectivamente de los arrays "arrayDensidadVegetacion" y "arrayTipoVegetacion", tal y como se explica en la Sección 4.4.4, donde se explica el método "densidadYTipoVegetacion".

```
//factor Pden, "factorDensidad", de la clase VegetacionYElevacion
      ,para la celda vecina
double factorDensidad=arrayDensidadVegetacion[i][j];
System.out.println("factor densidad: "+factorDensidad);
//factor Pveg, "factorTipoVegetacion", de la clase
      VegetacionYElevacion, para la celda vecina
double factorTipoVegetacion=arrayTipoVegetacion[i][j];
System.out.println("factor tipo veg: "+factorTipoVegetacion);
//factor Pcero, Alexandridis tabla 4, pagina 199
//final double Pcero=0.58;
//calculo del factor Pslope y, a continuacion, de la probabilidad
      Pburn. Necesitamos el vecindario de la celda (x,y). Para cada
      célula de dicho vecindario calcularemos su factor
//Pslope verificaremos primero el estado de la celda vecina, por si
      esta fuera no inflamable, etc. Inicialmente, el estado de las
      celdas tiene que ver con los valores del
//factorDensidad: si celda es mar o urbano(factorDensidad=2), su
      estado será 1; para el resto, su estado será 2; Después,
      a partir de la primera iteración, el estado
            de las celdas con estado 2 puede pasar a 3 y a 4
//celdas con material combustible pero que todavía no han
      ardido
//verificamos si para el slope, la disposición es en
      diagonal o en horizontal.(x,y) es la celda central.
      Sus diagonales son las sigu.:
//(x-1,y-1) (x+1,y-1) (x-1,y+1) (x+1,y+1). La celda vecina a
      comprobar es (i,j)
```

Código 4. 18: Extracto de código del método "calculoProbabilidad", parte 3

Para calcular el efecto de la inclinación del terreno (P_s, al que llamaremos factor Slope), dependiente de la diferencia de elevación entre la celda central y aquella vecina a la que se puede propagar el fuego, se requiere saber si la disposición entre ambas celdas es adyacente o diagonal (véase la Figura 3.1). Una vez consignemos dicha disposición, llamaremos al método "factorSlope", pasándole como parámetros las celdas central y vecina y la disposición mencionada. Explicaremos el método "factorSlope" posteriormente.

```
boolean diagonal;
if ((i==x-1)&&(j==y-1)||(i==x+1)&&(j==y-1)||(i==x-
      1)&&(j==y+1)||(i==x+1)&&(j==y+1)){
      diagonal=true;
}else{
```

```
            diagonal=false;
      }
      double factorSlope=calculoFactorSlope(x,y,i,j,diagonal
      //(x,y) es la celda base, (i,j) es la vecina
```

Código 4. 19: Extracto de código del método "calculoProbabilidad", parte 4

Una vez obtenido el factor Slope, tenemos ya el total de los factores, con lo que podemos proceder a calcular la probabilidad sin más, tal y como viene definida en el modelo de Alexandridis et Alter:

```
      //una vez obtenido el factorSlope, podemos calcular la
            probabilidad
      double                                                  probabilidad=
Pcero*(1+factorTipoVegetacion)*(1+factorDensidad)*factorViento*factorSlope;
            //fileWriter.write("PROBABILIDAD: "+probabilidad+"\n");
      System.out.println("PROBABILIDAD: "+probabilidad);

      //de cada 100 veces la celda se prenderá "probabilidad" veces
      return Math.random()<probabilidad;


}
```

Código 4. 20: Extracto de código del método "calculoProbabilidad", parte 5

4.4.9 MÉTODO CALCULODIRECCIONPROPAGACION

```
//(x,y) es la celda central que arde. (i,j) serán sus vecinas
      private int calculoDireccionPropagacion(int x, int y, int i, int j) {
            int direccion=0;
            if ((i==x)&&(j==y-1)){
                  direccion=0;
            }
            if ((i==x)&&(j==y+1)){
                  direccion=180;
            }
            if ((i==x-1)&&(j==y)){
                  direccion=90;
            }
            if ((i==x+1)&&(j==y)){
                  direccion=270;
            }
            if ((i==x-1)&&(j==y-1)){
                  direccion=45;
            }
            if ((i==x-1)&&(j==y+1)){
                  direccion=135;
            }
            if ((i==x+1)&&(j==y+1)){
                  direccion=225;
            }
            if ((i==x+1)&&(j==y-1)){
                  direccion=315;
            }
            return direccion;
      }
```

Código 4. 21: Extracto de código del método "calculoDireccionPropagacion"

Como vemos este método es muy sencillo. Simplemente calcula la disposición geométrica de la celda vecina (i,j) con respecto a la central (x,y), a fin de deducir cuál es la dirección de propagación necesaria para el cálculo de la probabilidad, conforme a el modelo de Alexandridis et Alter.

4.4.10 MÉTODO CALCULOFACTORSLOPE

```
//el vecindario de Moore de (baseX,baseY) es (x-1,y-1),(x,i-1),(x+1,i-1)  (x-
1,y),(  ,  ),(x+1,y)   (x-1,y+1),(x,i+1),(x+1,i+1). La célula (baseX,baseY) es
la que arde. Sólo se calcula
//el factor slope de aquellas vecinas cuyo estado es 2
public double calculoFactorSlope (int baseX,int baseY,int x,int y,boolean
              diagonal){

       //final double a=0.078;//factor a, Alexandridis tabla 4, pagina 199
       int elevacionVecino=arrayElevacion[x][y];
       //System.out.println("diagonal? :"+diagonal+" elevacion vecino: x,y
              "+x+" "+y+" : "+elevacionVecino);
       int    elevacionBase=arrayElevacion[baseX][baseY];
       //System.out.println("Elevacion base: "+elevacionBase);
       int d=11;//distancia de separación entre las celdas
       double ratio=0;
       if (diagonal==true){
              ratio=(elevacionVecino-elevacionBase)/(Math.sqrt(2)*d);
       }else {
              ratio=(elevacionVecino-elevacionBase)/d;
       }
       //System.out.println("ratio: "+ratio);
       //si la celda vecina es más alta que la central, beneficia al fuego.
              Si es más baja, lo perjudica
       //THETASlope es en radianes, no en grados
       double THETASlope=Math.atan(ratio);
       //System.out.println("Theta slope: "+THETASlope+" en radianes ");
       //System.out.println("factor slope: "+Math.exp(a*THETASlope));
       return Math.exp(a*THETASlope);
}
```

Código 4. 22: Extracto de código del método "calculoFactorSlope"

Una vez obtenidas la elevación de la celda central y de la vecina, calculamos el factor Slope conforme a las fórmulas indicadas en la Sección 3.2.8, "Efectos de la Elevación del Terreno".

4.5 CONCLUSIONES

Hemos desarrollado el modelo de simulación de la extensión de un incendio forestal en la isla de Spetses de Alexandridis et Alter en el lenguaje de programación de alto nivel Java. Dado que no hemos hecho uso de un Sistema de Información Geográfica para el procesado de las imágenes ha sido necesario realizar un procesamiento de la imagen Landsat inicial. Dicho

procesamiento se ha realizado usando el algoritmo de clasificación de imágenes de la Distancia Mínima. En el capítulo siguiente comprobaremos la bondad de la aplicación implementada con respecto a la teoría del modelo de Alexandridis et Alter.

5 PRUEBAS REALIZADAS AL SIMULADOR

5.1 INTRODUCCIÓN

Una vez implementado el simulador hay que realizar la verificación y validación del mismo. Para ello, comprobaremos, en primer lugar, que los cálculos matemáticos ejecutados en el algoritmo se corresponden correctamente con las fórmulas del autómata. En segundo lugar, verificaremos si el orden de ejecución de las reglas de transición de estados en el autómata es el correcto, tanto en relación al vecindario de una única celda como en relación al recorrido por todo el espacio del autómata. Finalmente, comprobaremos las ventanas de ejecución, para discernir si los pasos anteriores se dibujan correctamente en el mapa, y estableceremos una comparación con los resultados obtenidos por Alexandridis et Alter en su autómata.

5.2 VERIFICACIÓN MATEMÁTICA DEL ALGORITMO

Consideremos los siguientes datos de entrada en el simulador:

1. Velocidad del viento: 9 m/s.
2. Dirección del viento: 90 grados (es decir, en sentido de este a oeste).
3. Foco inicial del incendio: (300,300).

Veamos los cálculos de las primeras iteraciones, tal y como son listados en la consola al ejecutar el autómata en Eclipse (véase la Sección 6.9):

```
Firmas
ImageResizer
DistanciaMinima
7878
PixelesUrbanos
UrbeYPlaya
RepintarBorde
Tab changed to: AUTOMATA
velocidadViento 9 ,direccionViento 90 ,posicion x 300, posicion y 300
0.7071067811865475
0.7071067811865476
xx: 92
yy: 40

**********CALCULO                              CELDA                              CENTRAL:
300,300**************************************************************************
*****
```

El foco inicial del incendio es la celda (300, 300). Su vecindario está constituido por las celdas (299,299), (299,300), (299,301), (300,299), (300,300), (300,301), (301,299), (301,300) y (301,301).

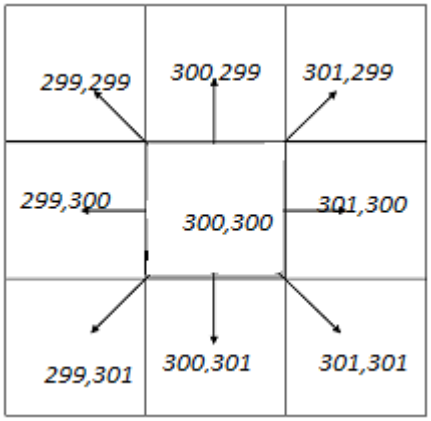


Figura 5. 1: Vecindario de Moore, foco inicial en la celda (300,300)

Conforme a el mecanismo de ejecución de un autómata celular, en cada iteración del mismo se examinan todas las celdas. Dado que la única celda ardiendo inicialmente es la (300,300), en la primera iteración sólo habrá que calcular la probabilidad P_{burn} en su vecindario.

Los cálculos que se realizan en cada celda son los siguientes, tal y como están explicados en el Capítulo 3:

1. DirecciónPropagacion: es la disposición en el vecindario de la celda vecina con respecto a la celda central.

2. Theta: es la dirección de propagación menos la dirección del viento.

3. FsubT: es el cálculo intermedio f_t en el factor viento, exp(V*c2*(cos(theta)-1)), donde V es la velocidad del viento y la constante c2 es 0.131. Véase la Eq. (3.3).

4. FactorViento, P_w: FsubT*exp(c1*V), donde c1 es 0.045. Véase la Eq. (3.2).

5. Factor densidad, P_{den}: si la vegetación posee densidad escasa, -0.4. Si intermedia, 0. Si densa, 0.3. Para las celdas sin vegetación (mar o áreas urbanas) no se calcula la probabilidad.

6. Factor tipo, P_{veg}: si la vegetación posee densidad escasa, consideramos zona agrícola, con factor -0.3. Si intermedia, consideramos zona de matorrales, con factor 0. Si densa, consideramos zona de pinares, con factor 0.4. Para las celdas sin vegetación (mar o áreas urbanas) no se calcula la probabilidad

7. Ratio, ThetaSlope y factor Slope. El factor de inclinación o factor Slope, P_s: es exp(a*thetaSlope), donde a es 0.078 y thetaSlope posee dos formulaciones. Si las celdas son adyacentes vertical u horizontalmente, ThetaSlope=arcotangente(E1-E2/l). Si las celdas son adyacentes en sentido oblícuo, thetaSlope=arcotangente(E1-E2/l*√2). E1 es la

elevación de la celda vecina y E2 es la elevación de la celda central, ambas expresadas en metros. La distancia de separación entre ambas celdas es l, equivalente a 7 metros. Véanse las Eqs. (3.4)- (3.6).

8. Probabilidad: $P_{burn} = P_0(1 + P_{veg})(1 + P_{den})P_w P_s$

Al principio del listado en la consola de Eclipse, obtenemos los datos para las celdas de este vecindario específico:

```
factorDensidad celda 299,299: -0.4
factorTipo celda 299,299: -0.3

factorDensidad celda 299,300: -0.4
factorTipo celda 299,300: -0.3

factorDensidad celda 299,301: -0.4
factorTipo celda 299,301: -0.3

factorDensidad celda 300,299: 0.0
factorTipo celda 300,299: 0.0

factorDensidad celda 300,301: -0.4
factorTipo celda 300,301: -0.3

factorDensidad celda 301,299: -0.4
factorTipo celda 301,299: -0.3

factorDensidad celda 301,300: 0.0
factorTipo celda 301,300: 0.0

factorDensidad celda 301,301: 0.0
factorTipo celda 301,301: 0.0

elevacion celda 299,299: 89
elevacion celda 299,300: 86
elevacion celda 299,301: 82
elevacion celda 300,299: 88
elevacion celda 300,300: 85
elevacion celda 300,301: 81
elevacion celda 301,299: 86
elevacion celda 301,300: 82
elevacion celda 301,301: 78
```

Verificamos los cálculos para la celda (299,299):

1. DireccionPropagacion: 45, tal y como se observa en la figura anterior.

2. Theta: 45-90= - 45.

3. FsubT: exp(9*0.131*(cos(45)-1))=0.707.

4. FactorViento: exp(9*0.045)*FsubT=1.06.

5. Factor densidad: -0.4. La celda (299,299) posee densidad escasa.
6. Factor tipo: -0.3.
7. Ratio: E1-E2/l*√2, disposición oblícua, 89-85/(12.47*1.4142)=0.2268.
8. ThetaSlope: arcTg(ratio), donde ratio es en radianes, arcTg(0.2268)=0.223.
9. Factor Slope: exp(0.078*0.223)=1.01755.
10. Probabilidad: 0.58*(1-0.3)(1-0.4)*1.06*1.01755=0.2627.

```
CALCULO PROBABILIDAD VECINO 299,299
direccionPropagacion: 45
theta: -45
FsubT: 0.7079929768924422
factorViento: 1.0614956402774713
factor densidad: -0.4
factor tipo veg: -0.3
ratio: 0.22685327685073947
ThetaSlope: 0.22307771627257436
factor Slope: 1.0175523247918554
PROBABILIDAD: 0.26311902404845716
Añadido a la lista de pixeles ardiendo el pixel
********************************* :299, 299
```

Vemos que los cálculos para dicha celda son correctos. Para las demás celdas, verificamos su dirección de propagación y el ángulo theta resultante, estando en todos los casos correcto. Comprobamos sus factores de densidad y tipo, y vemos que son correctos. Comprobamos también el ratio, para verificar la fórmula según sean las celdas adyacentes vertical/horizontalmente u oblicuas, y vemos que los cálculos son correctos. La celda (299,299) se ha incendiado. En definitiva, los cálculos de la probabilidad matemática son correctos.

Tal y como se define la regla de transición de estados número 4, P_{burn} indica la probabilidad de que la celda vecina se encienda. En la ejecución de la que hemos tomado estos datos, del vecindario inicial de la celda (300,300) se han prendido únicamente las celdas (299,299) y (300,299).

```
CALCULO PROBABILIDAD VECINO 299,300
direccionPropagacion: 90
theta: 0
FsubT: 1.0
factorViento: 1.4993025000567668
factor densidad: -0.4
factor tipo veg: -0.3
ratio: 0.08020474519777357
ThetaSlope: 0.08003342555989468
factor Slope: 1.0062621328752002
PROBABILIDAD: 0.3675172083612541

CALCULO PROBABILIDAD VECINO 299,301
direccionPropagacion: 135
```

```
theta: 45
FsubT: 0.7079929768924422
factorViento: 1.0614956402774713
factor densidad: -0.4
factor tipo veg: -0.3
ratio: -0.17013995763805462
ThetaSlope: -0.16852618047420484
factor Slope: 0.9869409766694256
PROBABILIDAD: 0.2552035313051932

CALCULO PROBABILIDAD VECINO 300,299
direccionPropagacion: 0
theta: -90
FsubT: 0.30758617103032765
factorViento: 0.4611647152086585
factor densidad: 0.0
factor tipo veg: 0.0
ratio: 0.2406142355933207
ThetaSlope: 0.2361256822330311
factor Slope: 1.0185884570312853
PROBABILIDAD: 0.2724474923069625
Añadido a la lista de pixeles ardiendo el pixel
********************************* :300, 299

CALCULO PROBABILIDAD VECINO 300,300
la celda base y la vecina coinciden
Eliminado de la lista de pixeles ardiendo el pixel :300, 300

CALCULO PROBABILIDAD VECINO 300,301
direccionPropagacion: 180
theta: 90
FsubT: 0.30758617103032765
factorViento: 0.4611647152086585
factor densidad: -0.4
factor tipo veg: -0.3
ratio: -0.3208189807910943
ThetaSlope: -0.31044567494002573
factor Slope: 0.9760760625692898
PROBABILIDAD: 0.10965211608192157

CALCULO PROBABILIDAD VECINO 301,299
direccionPropagacion: 315
theta: 225
FsubT: 0.13363021343002798
factorViento: 0.2003521130787603
factor densidad: -0.4
factor tipo veg: -0.3
ratio: 0.05671331921268487
ThetaSlope: 0.05665263203540484
factor Slope: 1.0044286830578066
PROBABILIDAD: 0.04902192005372668

CALCULO PROBABILIDAD VECINO 301,300
direccionPropagacion: 270
theta: 180
FsubT: 0.09460925260909793
factorViento: 0.14184788896532272
factor densidad: 0.0
factor tipo veg: 0.0
```

```
ratio: -0.2406142355933207
ThetaSlope: -0.2361256822330311
factor Slope: 0.9817507680329874
PROBABILIDAD: 0.08077037888262682

CALCULO PROBABILIDAD VECINO 301,301
direccionPropagacion: 225
theta: 135
FsubT: 0.13363021343002798
factorViento: 0.2003521130787603
factor densidad: 0.0
factor tipo veg: 0.0
ratio: -0.3969932344887941
ThetaSlope: -0.3779116534453976
factor Slope: 0.9709531035046501
PROBABILIDAD: 0.11282885347277141
la celda 299,299 se ha incendiado
la celda 300,299 se ha incendiado

terminado action performed. Paso de la simulación: 1
```

5.3 VERIFICACIÓN DE LOS PASOS DEL AUTÓMATA

Hasta aquí todo es correcto. Veamos ahora el funcionamiento no en cuanto a los cálculos matemáticos, sino en relación a la ejecución de los pasos del autómata. En esta primera iteración se han prendido las celdas (299,299) y (300,299). Por consiguiente, en la siguiente iteración, se deberá comprobar el vecindario de (299,299) y después el de (300,299). Para facilitar el seguimiento de estos resultados, en lugar de volver a presentar todo el listado presentamos únicamente los datos relevantes.

```
**********CALCULO                          CELDA                          CENTRAL:
299,299**************************************************************************
*****

CALCULO PROBABILIDAD VECINO 298,298
...
PROBABILIDAD: 0.265280989961118

CALCULO PROBABILIDAD VECINO 298,299
...
PROBABILIDAD: 0.8804510693806755
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
******************************** :298, 299

CALCULO PROBABILIDAD VECINO 298,300
...
PROBABILIDAD: 0.6102674919194931
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
******************************** :298, 300

CALCULO PROBABILIDAD VECINO 299,298
...
PROBABILIDAD: 0.2755520151222215
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
******************************** :299, 298

CALCULO PROBABILIDAD VECINO 299,299
```

```
la celda base y la vecina coinciden
Eliminado de la lista de pixeles ardiendo el pixel :299, 299

CALCULO PROBABILIDAD VECINO 299,300
...
PROBABILIDAD: 0.11028961093104038
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
********************************** :299, 300

CALCULO PROBABILIDAD VECINO 300,298
...
PROBABILIDAD: 0.11824387989534688

CALCULO PROBABILIDAD VECINO 300,299
celda ya ardiendo

CALCULO PROBABILIDAD VECINO 300,300
celda ya quemada

************CALCULO                          CELDA                          CENTRAL:
300,299**************************************************************************
*****

CALCULO PROBABILIDAD VECINO 299,298
...
PROBABILIDAD: 0.6316214046693285

CALCULO PROBABILIDAD VECINO 299,299
celda ya ardiendo

CALCULO PROBABILIDAD VECINO 299,300
...
PROBABILIDAD: 0.2563123466061871

CALCULO PROBABILIDAD VECINO 300,298
...
PROBABILIDAD: 0.2755520151222215

CALCULO PROBABILIDAD VECINO 300,299
la celda base y la vecina coinciden
Eliminado de la lista de pixeles ardiendo el pixel :300, 299

CALCULO PROBABILIDAD VECINO 300,300
celda ya quemada

CALCULO PROBABILIDAD VECINO 301,298
...
PROBABILIDAD: 0.04923763484625602

CALCULO PROBABILIDAD VECINO 301,299
...
PROBABILIDAD: 0.034128106570202564

CALCULO PROBABILIDAD VECINO 301,300
...
PROBABILIDAD: 0.11326905832972291
la celda 298,299 se ha incendiado
la celda 298,300 se ha incendiado
la celda 299,298 se ha incendiado
```

```
la celda 299,300 se ha incendiado

terminado action performed. Paso de la simulación: 2
```

Observamos que en esta segunda iteración, se han comprobado los vecinos de ambas celdas correctamente. Para la celda central (299,299) se han prendido sus vecinos (298,299), (298,300), (299,298), (299,300) y para la celda central (300,299) no se ha prendido ningún vecino. En la tercera iteración se calculará el vecindario de las 4 celdas resultantes de la segunda iteración:

```
***********CALCULO                        CELDA                        CENTRAL:
298,299*************************************************************************
*****

CALCULO PROBABILIDAD VECINO 297,298
...
PROBABILIDAD: 0.6364651704782586
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
******************************** :297, 298

CALCULO PROBABILIDAD VECINO 297,299
...
PROBABILIDAD: 0.890909513490086
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
******************************** :297, 299

CALCULO PROBABILIDAD VECINO 297,300
...
PROBABILIDAD: 0.6129528972496503

CALCULO PROBABILIDAD VECINO 298,298
...
PROBABILIDAD: 0.11509320731534119
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
******************************** :298, 298

CALCULO PROBABILIDAD VECINO 298,299
la celda base y la vecina coinciden
Eliminado de la lista de pixeles ardiendo el pixel :298, 299

CALCULO PROBABILIDAD VECINO 298,300
celda ya ardiendo

CALCULO PROBABILIDAD VECINO 299,298
celda ya ardiendo

CALCULO PROBABILIDAD VECINO 299,299
celda ya quemada

CALCULO PROBABILIDAD VECINO 299,300
celda ya ardiendo

***********CALCULO                        CELDA                        CENTRAL:
298,300*************************************************************************
*****

CALCULO PROBABILIDAD VECINO 297,299
...
```

```
PROBABILIDAD: 0.6364651704782586
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
******************************** :297, 299

CALCULO PROBABILIDAD VECINO 297,300
...
PROBABILIDAD: 0.885759887690463
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
******************************** :297, 300

CALCULO PROBABILIDAD VECINO 297,301
...
PROBABILIDAD: 0.6156674713609334
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
******************************** :297, 301

CALCULO PROBABILIDAD VECINO 298,299
celda ya ardiendo

CALCULO PROBABILIDAD VECINO 298,300
la celda base y la vecina coinciden
Eliminado de la lista de pixeles ardiendo el pixel :298, 300

CALCULO PROBABILIDAD VECINO 298,301
...
PROBABILIDAD: 0.10965211608192157

CALCULO PROBABILIDAD VECINO 299,299
celda ya quemada

CALCULO PROBABILIDAD VECINO 299,300
celda ya ardiendo

CALCULO PROBABILIDAD VECINO 299,301
...
PROBABILIDAD: 0.04776514703005617

**********CALCULO                        CELDA                        CENTRAL:
299,298**************************************************************************
*****

CALCULO PROBABILIDAD VECINO 298,297
...
PROBABILIDAD: 0.6316214046693285
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
******************************** :298, 297

CALCULO PROBABILIDAD VECINO 298,298
...
PROBABILIDAD: 0.3675172083612541
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
******************************** :298, 298

CALCULO PROBABILIDAD VECINO 298,299
celda ya ardiendo

CALCULO PROBABILIDAD VECINO 299,297
...
PROBABILIDAD: 0.2755520151222215
```

```
CALCULO PROBABILIDAD VECINO 299,298
la celda base y la vecina coinciden
Eliminado de la lista de pixeles ardiendo el pixel :299, 298

CALCULO PROBABILIDAD VECINO 299,299
celda ya quemada

CALCULO PROBABILIDAD VECINO 300,297
...
PROBABILIDAD: 0.11824387989534688

CALCULO PROBABILIDAD VECINO 300,298
...
PROBABILIDAD: 0.08175978496260354

CALCULO PROBABILIDAD VECINO 300,299
celda ya quemada

**********CALCULO                        CELDA                        CENTRAL:
299,300**************************************************************************
*****

CALCULO PROBABILIDAD VECINO 298,299
celda ya ardiendo

CALCULO PROBABILIDAD VECINO 298,300
celda ya ardiendo

CALCULO PROBABILIDAD VECINO 298,301
...
PROBABILIDAD: 0.2552035313051932

CALCULO PROBABILIDAD VECINO 299,299
celda ya quemada

CALCULO PROBABILIDAD VECINO 299,300
la celda base y la vecina coinciden
Eliminado de la lista de pixeles ardiendo el pixel :299, 300

CALCULO PROBABILIDAD VECINO 299,301
...
PROBABILIDAD: 0.10965211608192157

CALCULO PROBABILIDAD VECINO 300,299
celda ya quemada

CALCULO PROBABILIDAD VECINO 300,300
celda ya quemada

CALCULO PROBABILIDAD VECINO 300,301
...
PROBABILIDAD: 0.04776514703005617
Añadido     a     la     lista     de     pixeles     ardiendo     el     pixel
******************************** :300, 301
la celda 297,298 se ha incendiado
la celda 297,299 se ha incendiado
la celda 297,300 se ha incendiado
la celda 297,301 se ha incendiado
```

```
la celda 298,297 se ha incendiado
la celda 298,298 se ha incendiado
la celda 300,301 se ha incendiado

terminado action performed. Paso de la simulación: 3
```

Vemos que el recorrido se hace en el orden correcto y que se visitan los vecindarios correctos. En la primera celda central, (298,299) se originan tres nuevas celdas ardiendo, las (297,298), (297,299) y (298,298). Sin embargo, el estado de dichas celdas no cambiará hasta que termine la iteración del autómata, es decir, hasta que se recorran todas las celdas en la iteración 3. Así, observamos que para la segunda celda central, la (298,300) se vuelve a verificar el estado de la celda (297,299), puesto que, como todavía no ha terminado la tercera iteración, su estado sigue siendo 2. Sólo cuando termine la iteración tercera, cambiarán las celdas incendiadas al estado 3. En definitiva, el autómata parece seguir un funcionamiento correcto.

Como se observa, una verificación de N estados implicaría observar 9 elevado a la N celdas, lo cual es un trabajo irrealizable. Nos detenemos, por tanto, en este punto, y pasamos ahora a otro tipo de comprobaciones.

5.4 VERIFICACIÓN DE LAS VENTANAS DE EJECUCIÓN

La probabilidad de que una celda se prenda es P_{burn}. Ello implica que, para los mismos datos de entrada al autómata, dado P_{burn}, si tendiéramos a un número infinito de ejecuciones, en P_{burn} veces se prendería, y en (1-P_{burn}) no.

Evidentemente no podemos ejecutar el mismo autómata un número alto de veces, así que sólo lo ejecutaremos un número reducido de veces, con los mismos datos de entrada, para ver el grado de variación en las ejecuciones.

Ejecutamos 5 veces el autómata con idénticos datos de entrada. Tales ejecuciones son: velocidad del viento 9 m/s, dirección del viento 0 grados (de sur a norte), foco inicial (300,300). A continuación, mostramos las imágenes con los 5 resultados finales. En cada imagen indicamos en un rombo rojo el foco inicial, (300,300). A continuación, indicamos las imágenes con el porcentaje de celdas quemadas y el paso en el que el autómata se extingue:

1. Fig. 5.2: imagen de prueba 1: porcentaje: 66% (paso 680).
2. Fig. 5.3: imagen de prueba 2: porcentaje: 68% (paso 696).
3. Fig. 5.4: imagen de prueba 3: porcentaje: 49% (paso 501).
4. Fig. 5.5: imagen de prueba 4: porcentaje: 69% (paso 883).
5. Fig. 5.6: imagen de prueba 5: porcentaje: 35% (paso 294).

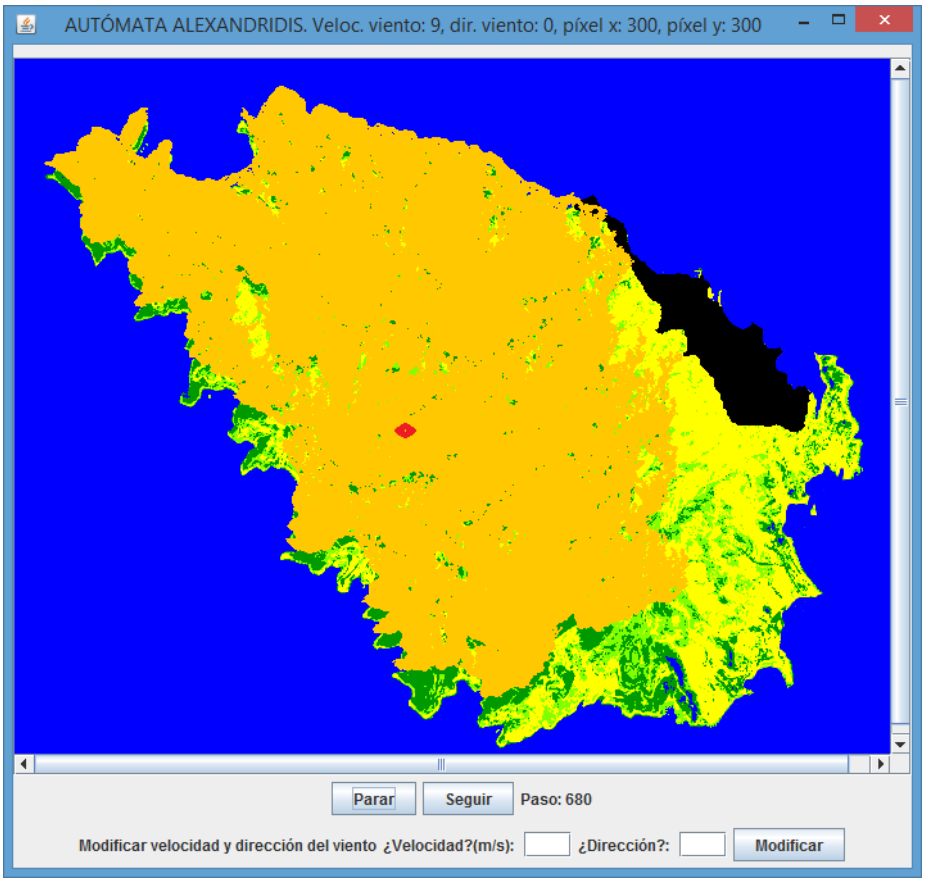


Figura 5. 2: Imagen de prueba 1

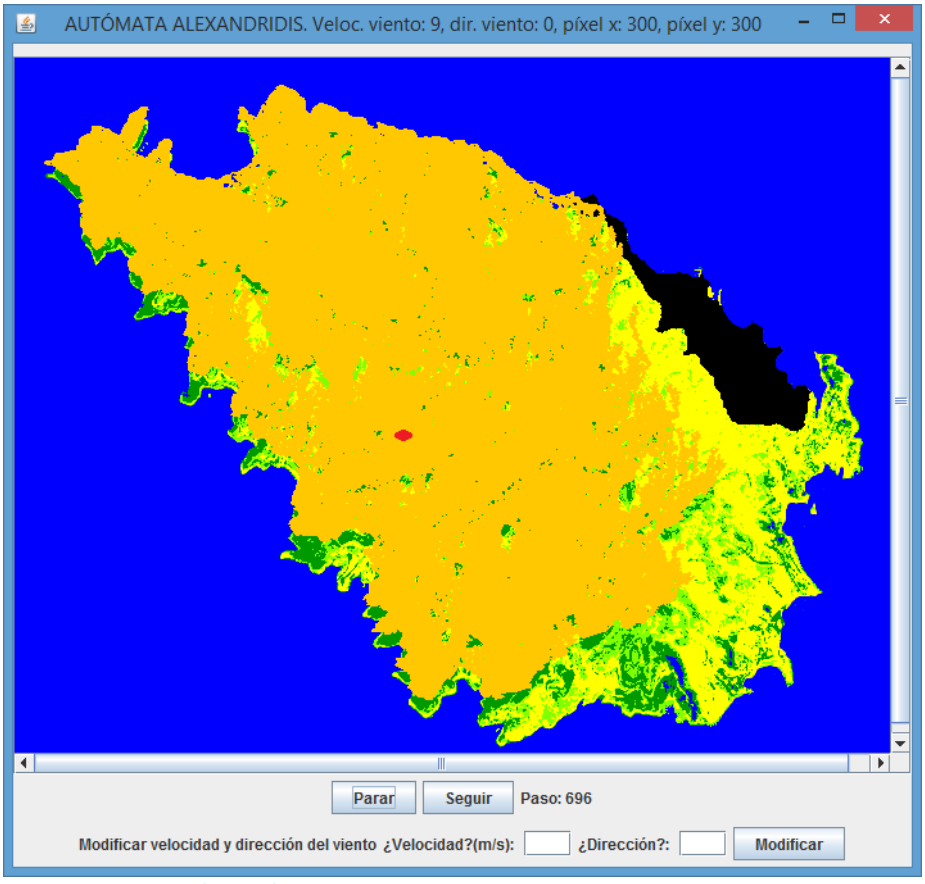


Figura 5. 3: Imagen de prueba 2

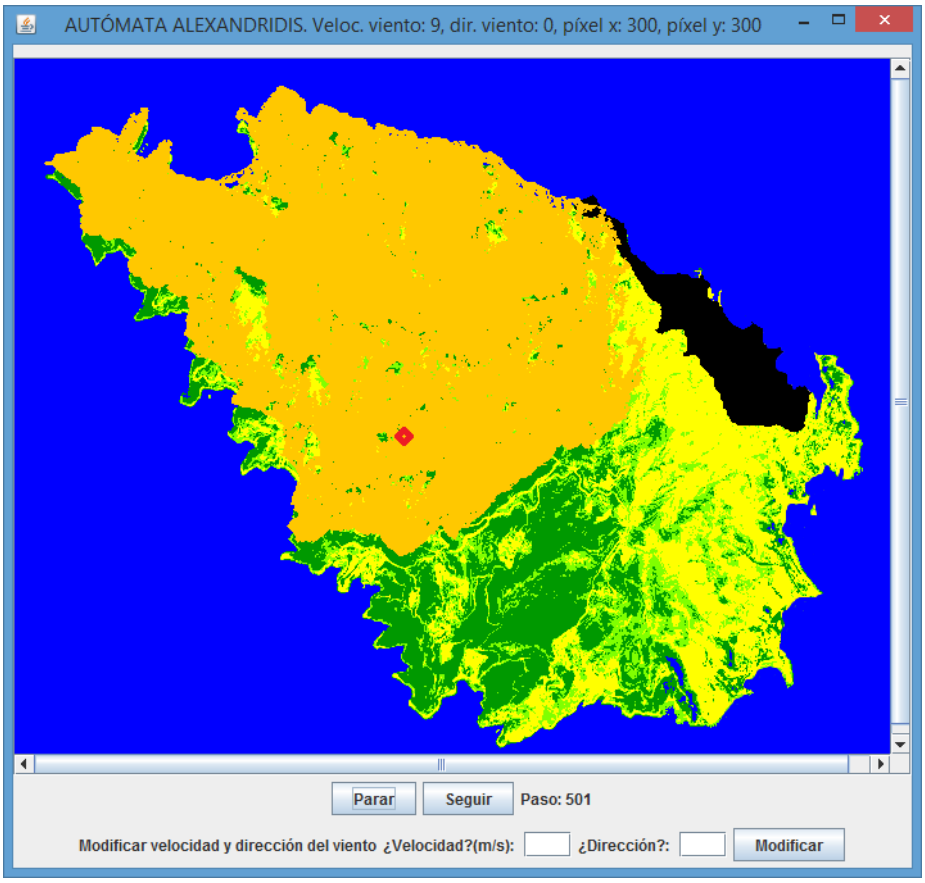


Figura 5. 4: Imagen de prueba 3

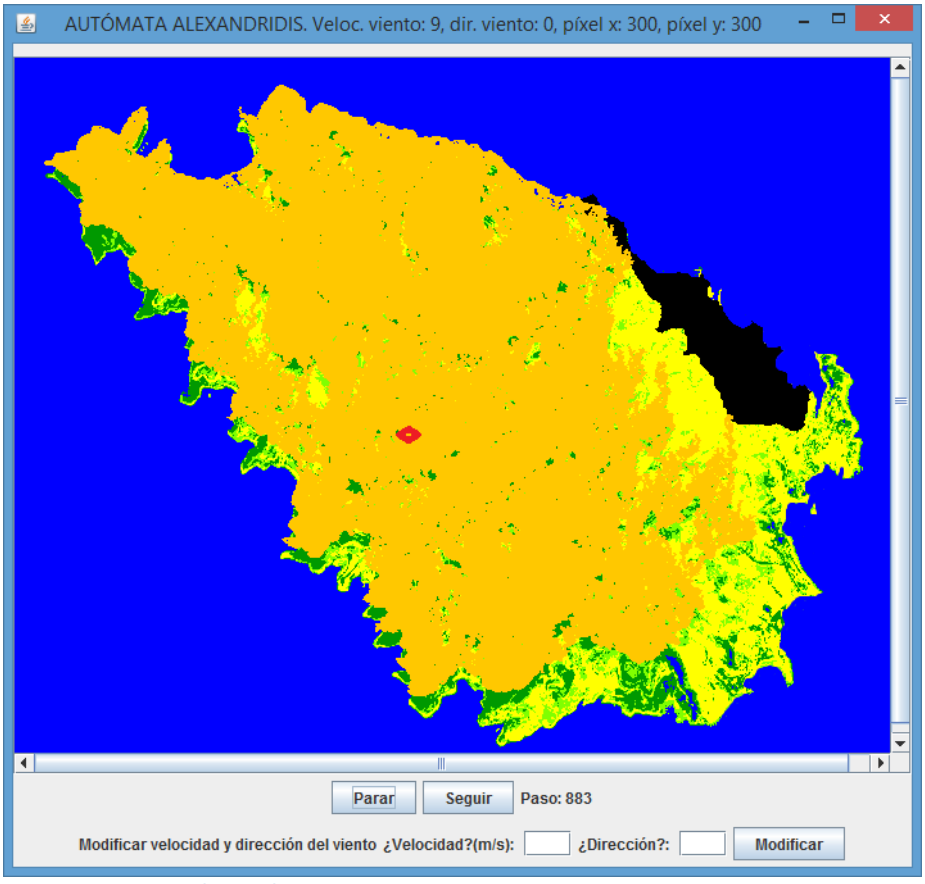


Figura 5. 5: Imagen de prueba 4

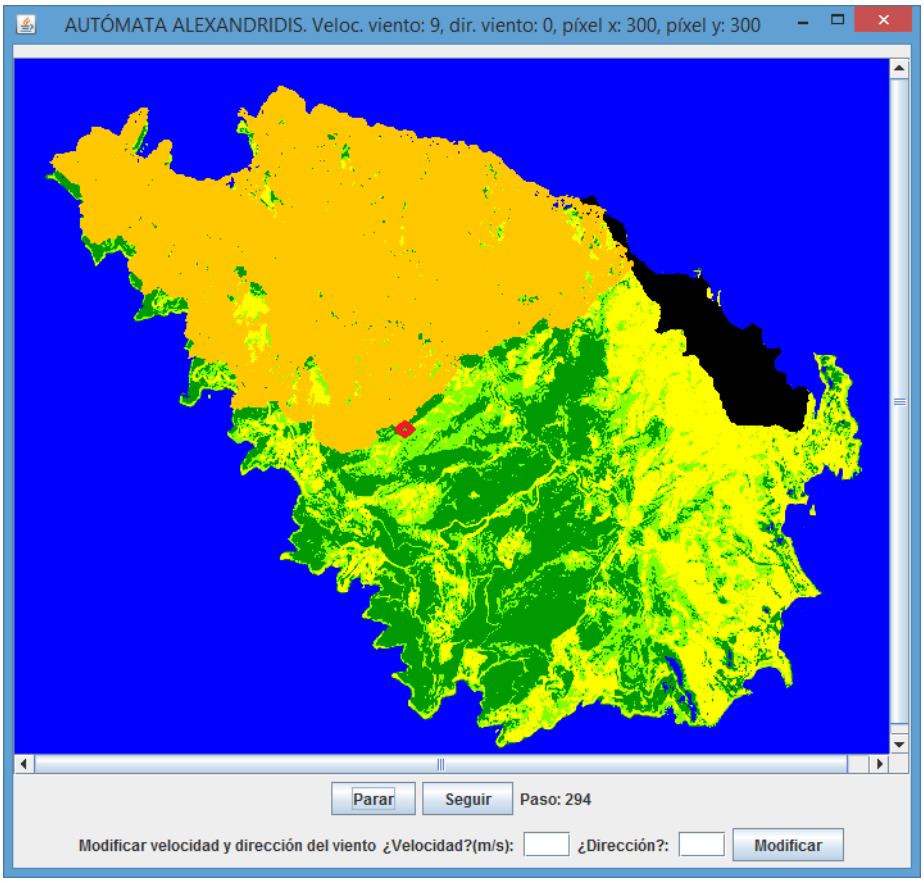


Figura 5. 6: Imagen de prueba 5

Como vemos, aunque a primera vista existe bastante variación, ésta tiene su explicación. En la Figura 5.6, el fuego se extingue en el paso 294, sin que apenas haya habido contagio a celdas en dirección contraria a la del viento. En los otros 4 casos, sin embargo, sí que ha habido este contagio, aunque en un caso, el fuego se haya extinguido poco a poco en las zonas en las que avanzaba en dirección contraria al viento (Figura 5.4). En los otros 3 casos, una vez que el fuego ha llegado a zonas con vegetación densa, aún cuando en estas zonas el incendio, en su avance, poseyera dirección contraria a la del viento, el fuego no se ha extinguido hasta llegar a la parte este de la isla, donde la vegetación es escasa. En los 5 casos, no obstante, el avance de la simulación en los pasos iniciales (hasta que quema la parte norte de la isla situada por encima del foco (300,300) fue muy similar.

Observamos, entonces, que la ejecución de las ventanas parece tener un funcionamiento lógico. Comprobémoslo nuevamente cambiando los parámetros de entrada al simulador. Usaremos las ejecuciones de ejemplo del lado derecho de la pestaña "AUTOMATA".

Comprobamos en primer lugar los 4 primeros ejemplos, en los que el parámetro que se modifica es el de la dirección del viento.

1. Ejemplo 1: velocidad 9, dirección 90 (hacia el oeste), foco inicial (300,300).

2. Ejemplo 2: velocidad 9, dirección 0 (hacia el norte), foco inicial (300,300).
3. Ejemplo 3: velocidad 9, dirección 180 (hacia el sur), foco inicial (300,300).
4. Ejemplo 4: velocidad 9, dirección 270 (hacia el este), foco inicial (300,300).

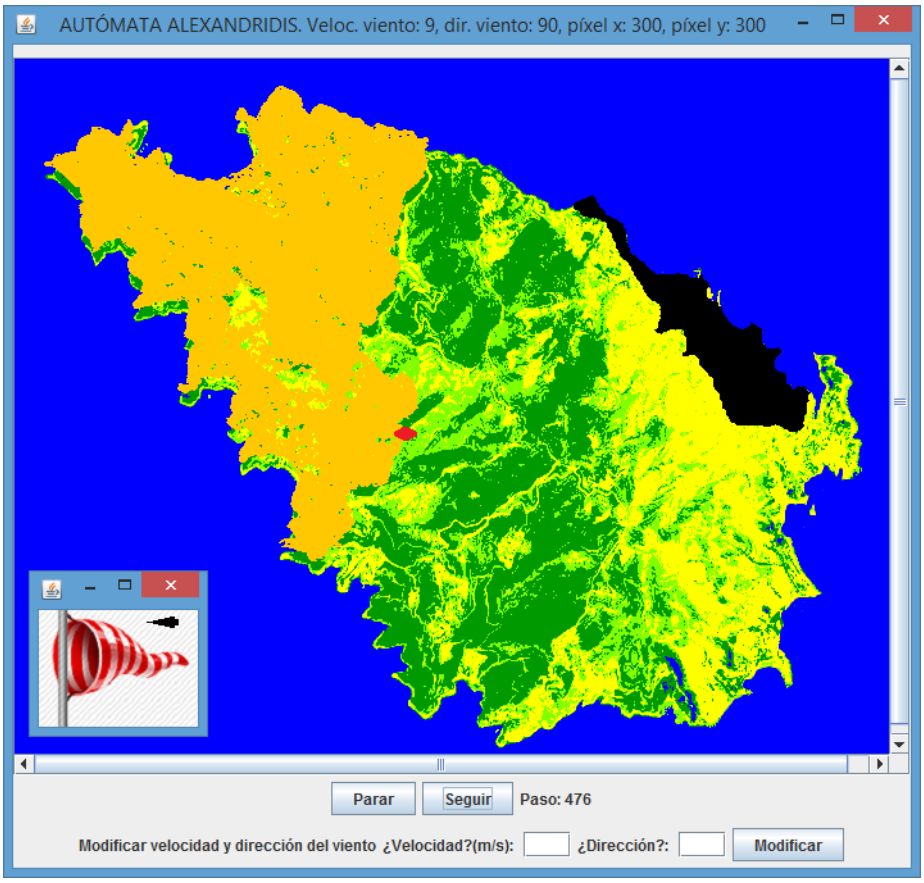


Figura 5. 7: Ejemplo 1, dirección 90 grados

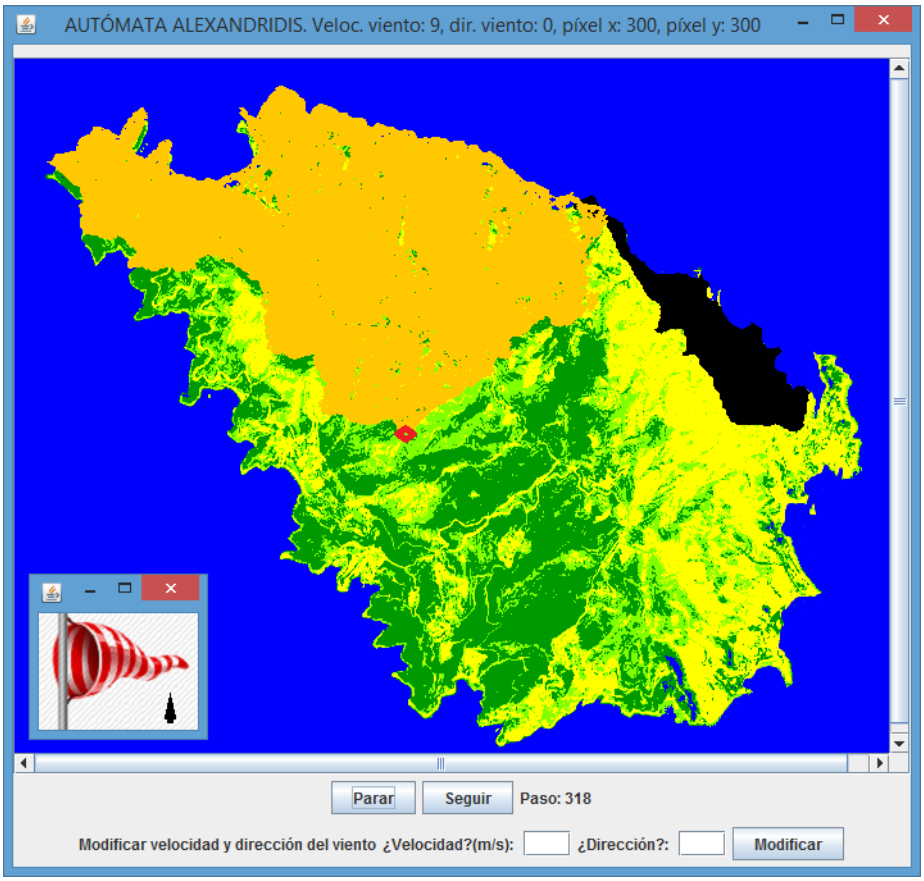


Figura 5. 8: Ejemplo 2, dirección 0 grados

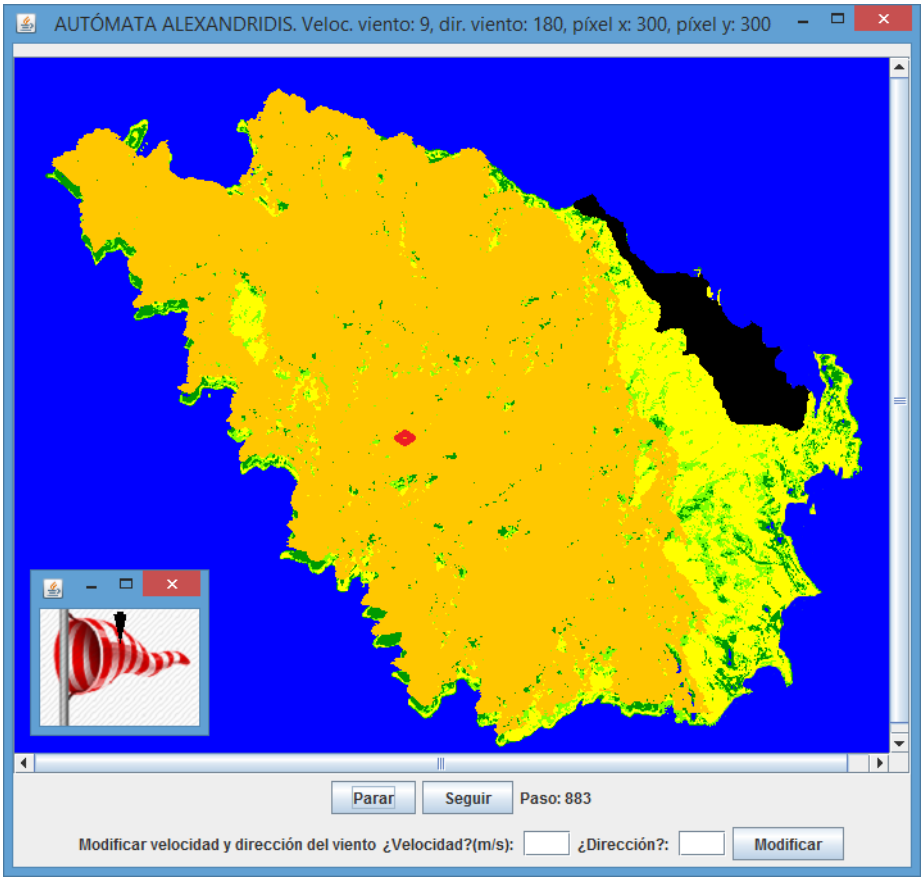


Figura 5. 9: Ejemplo 3, dirección 180 grados

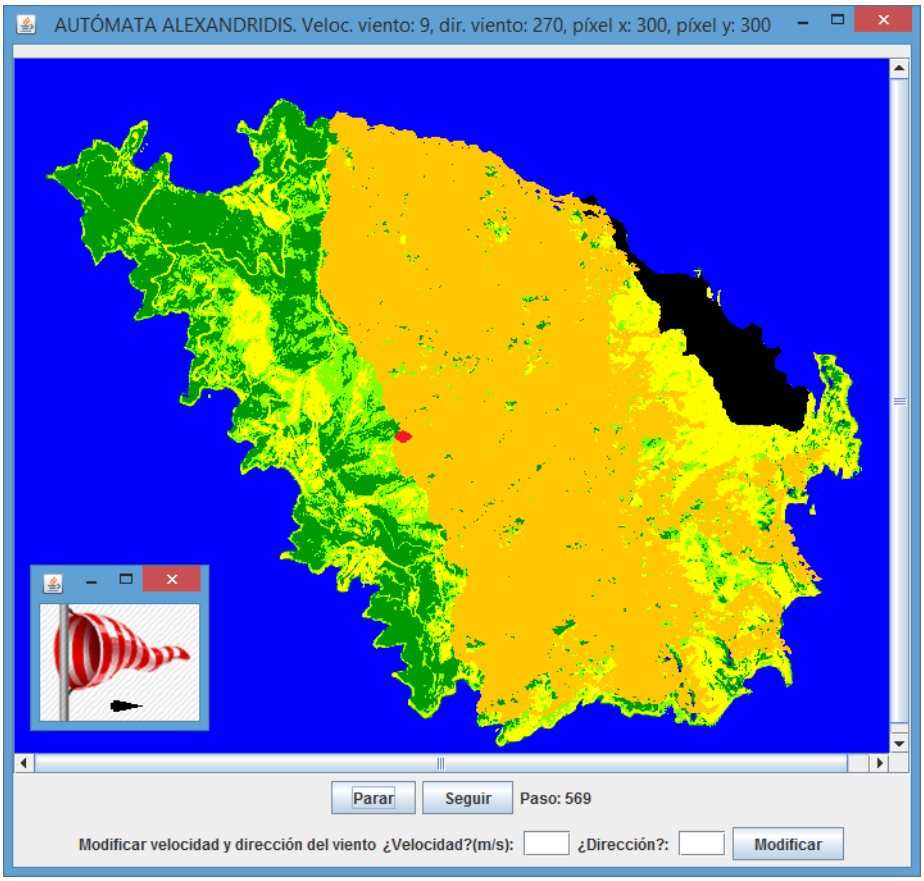


Figura 5. 10: Ejemplo 4, dirección 270 grados

Como se ve en las imágenes correspondientes, el área de la isla incendiada está en consonancia con la dirección del viento, salvo en la 3ª imagen, donde el viento es en dirección sur, y sin embargo también se ha incendiado la parte norte de la isla. Pero ya hemos comprobado en las imágenes anteriores a éstas que el incendio se puede extender en dirección contraria al viento. Por lo tanto, no vemos ninguna anomalía en estas ejecuciones.

Comprobamos ahora ejecuciones con datos similares de dirección y posición inicial del foco, pero con velocidades diferentes:

1. Ejemplo 1: velocidad 9, dirección 90 (hacia el oeste), foco inicial (300,300).

2. Ejemplo 5: velocidad 30, dirección 90 (hacia el oeste), foco inicial (300,300).

Como se ve en las imágenes que siguen, las dos ejecuciones siguen la dirección del viento. En la inferior, donde la velocidad del viento es mucho mayor (pasa de 9 m/s ó 32.4 km/h a 30 m/s ó 108 km/h) observamos que la forma del incendio es más estrecha, más definida conforme a la dirección del viento. Tampoco vemos anomalía en estas 2 ejecuciones.

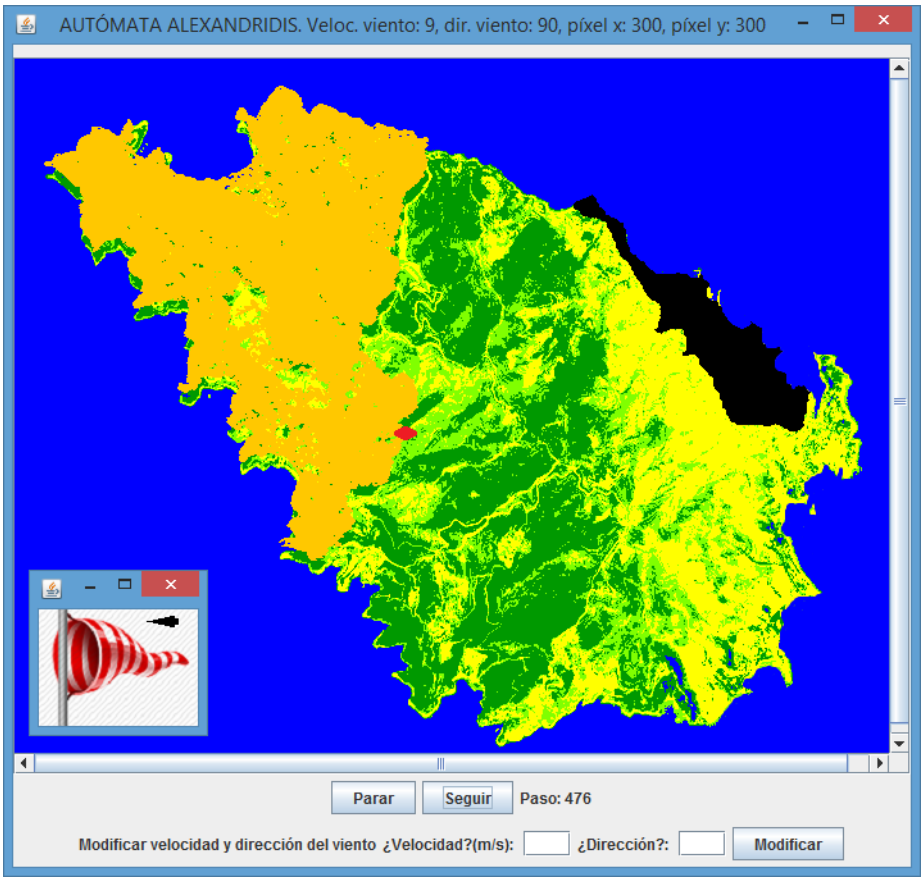


Figura 5. 11: Ejemplo 1, velocidad 9 m/s

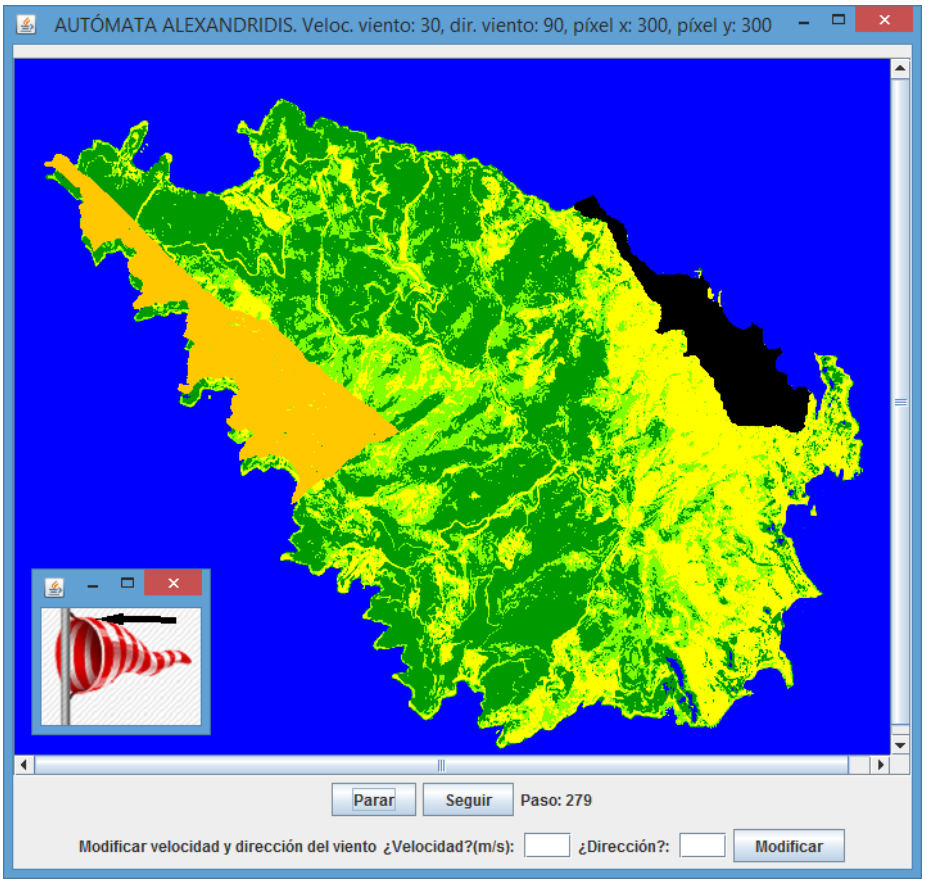


Figura 5. 12: Ejemplo 5, velocidad 30 m/s

5.5 COMPARACIÓN CON LAS FIGURAS 7 Y 9 DE ALEXANDRIDIS ET ALTER

Alexandridis et Alter, en su trabajo, presentan dos figuras, las números 7 y 8 que representan, respectivamente, el área real del fuego en la isla de Spetses en 1990 y el resultado de la simulación por ellos realizada. Presentamos seguidamente dichas 2 imágenes y, después de ellas, 2 simulaciones realizadas con nuestro autómata, simulando las condiciones iniciales del incendio:

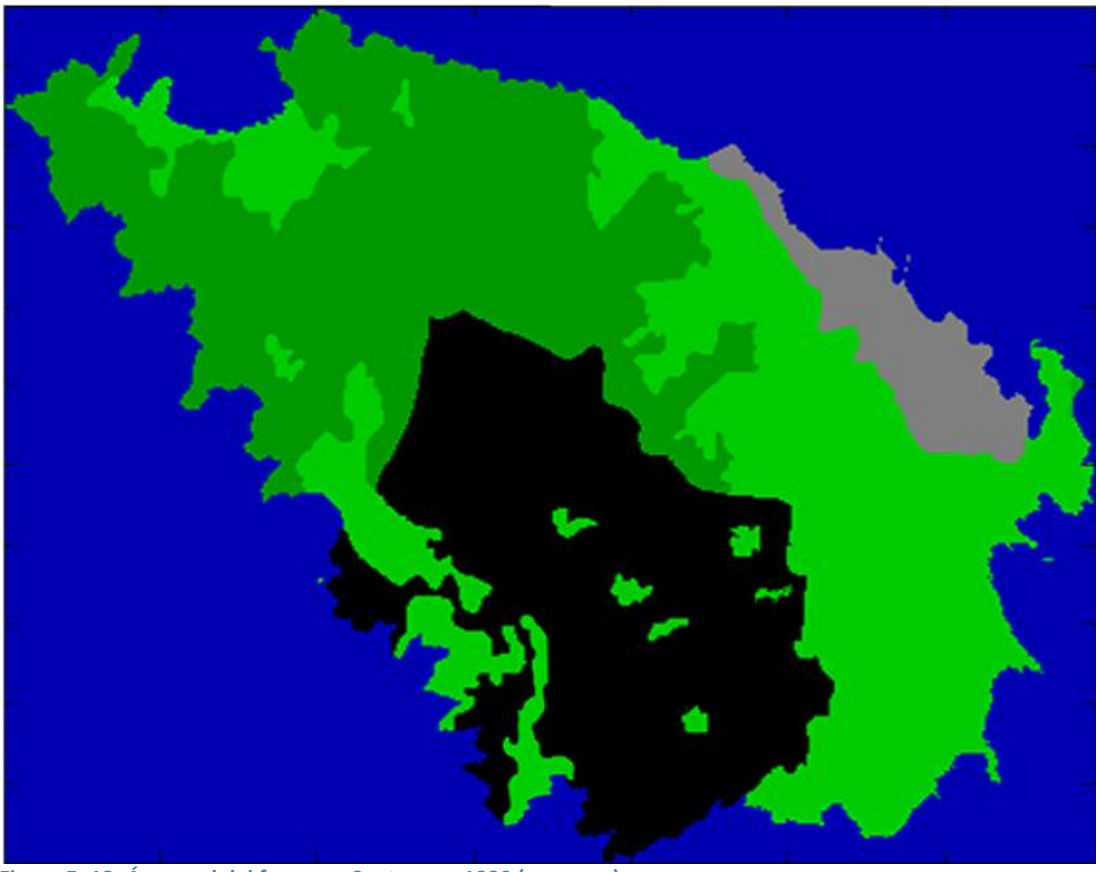


Figura 5. 13: Área real del fuego en Spetses en 1990 (en negro)

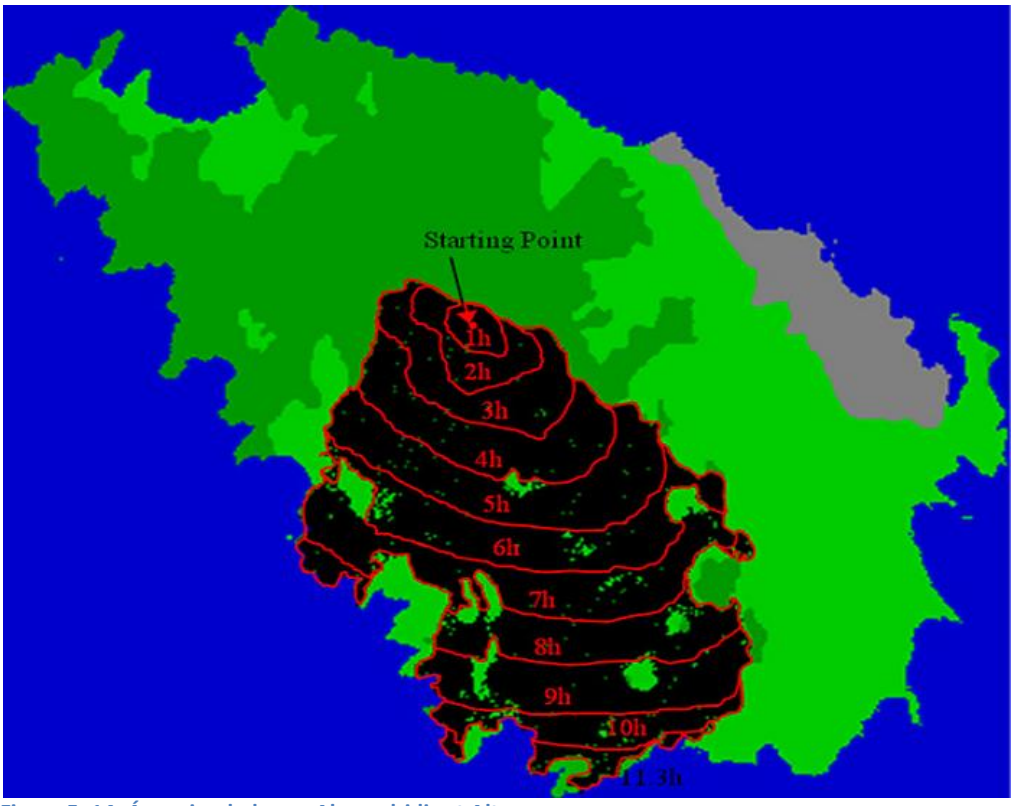


Figura 5. 14: Área simulada por Alexandridis et Alter

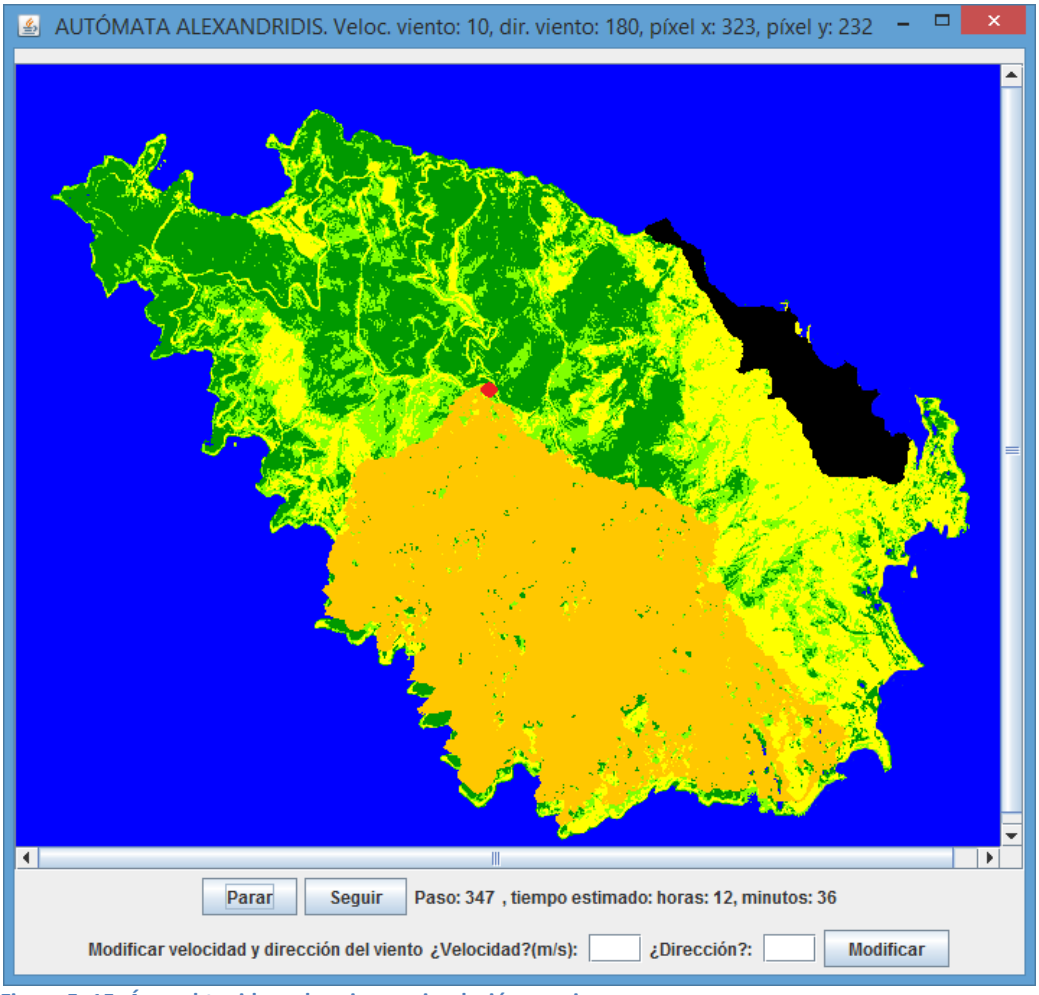


Figura 5. 15: Área obtenida en la primera simulación propia

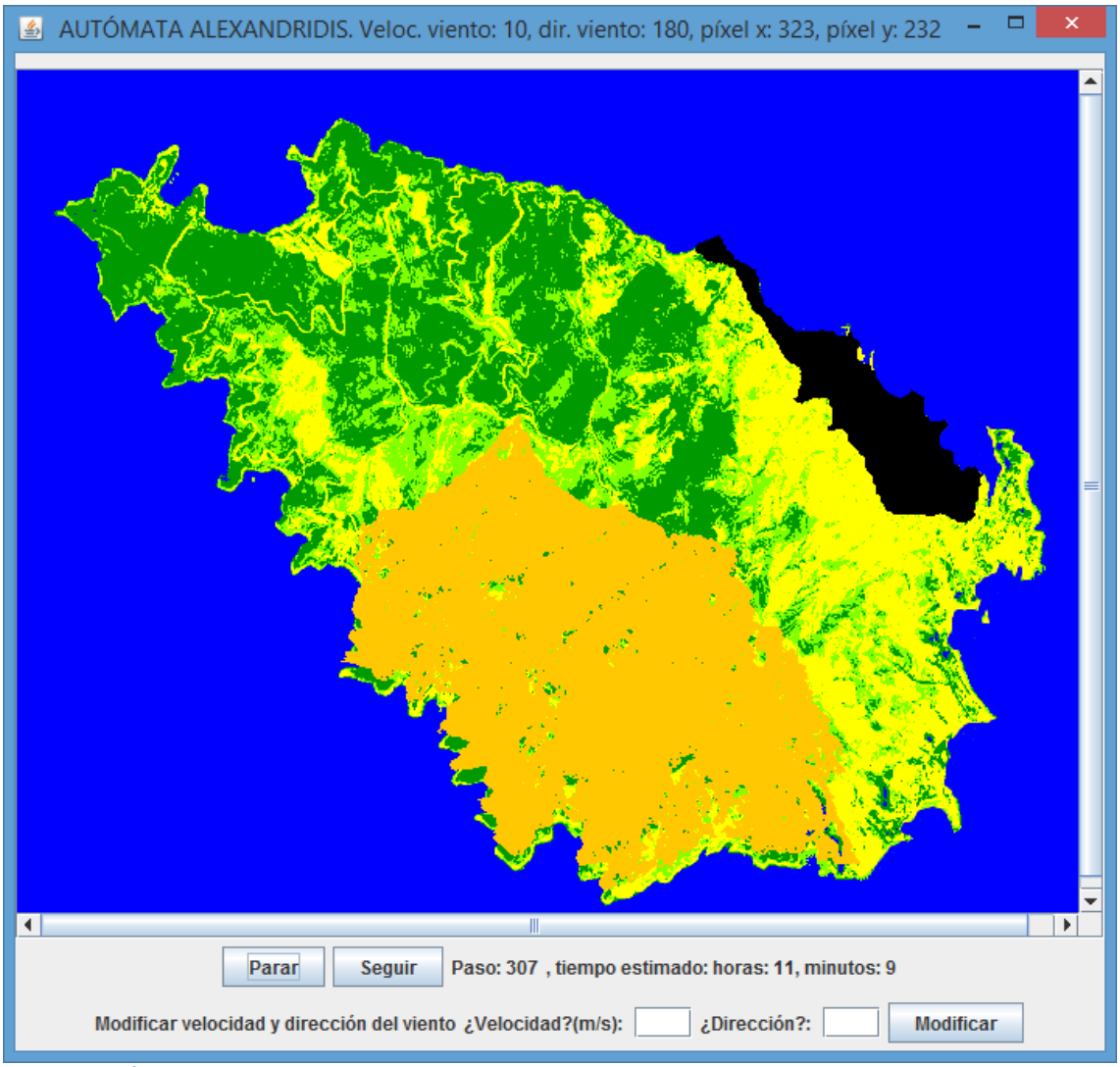


Figura 5. 16: Área obtenida en la segunda simulación propia

Como se ve claramente observando las primeras 2 imágenes con respecto a las 2 últimas, nuestro autómata presenta una extensión algo mayor del incendio. En las 2 primeras imágenes, el incendio se extingue al llegar a la zona de vegetación escasa. En nuestras ejecuciones, dado que la división de la densidad de terreno en las diversas zonas de la isla no es tan abrupta como en el modelo original de Alexandridis et Alter, existen islas con vegetación en la parte inferior de la isla que causan que el fuego continúe ardiendo un rato más en dirección sureste, y también, aunque en menor medida, en dirección suroeste.

Esto se puede comprender comparando la división de la isla en zonas respecto a la densidad de la vegetación, la cual es algo diferente en el trabajo original de Alexandridis que en el nuestro, dado que nosotros hemos empleado el algoritmo de clasificación de la Distancia Mínima para clasificar las zonas de la isla. Presentamos las imágenes con la clasificación de la densidad de la vegetación, la original de Alexandridis et Alter y la nuestra.

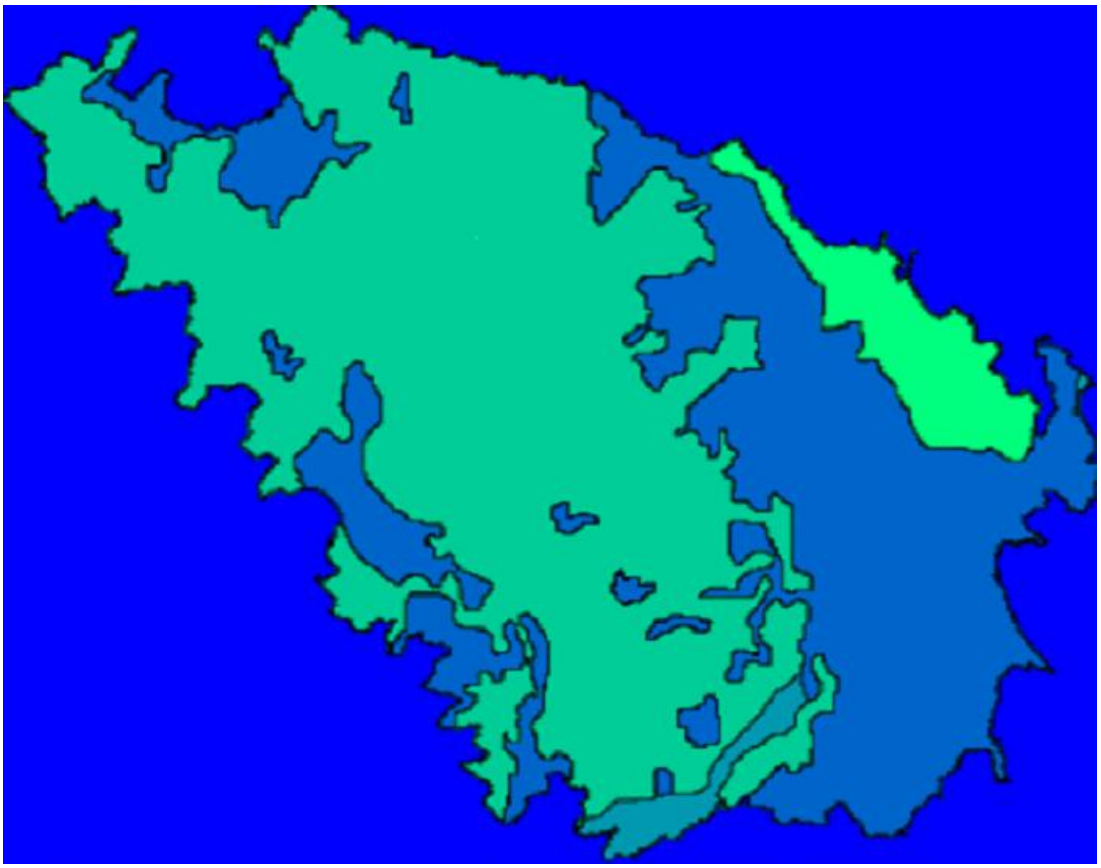


Figura 5. 17: Densidad de la vegetación en el modelo de Alexandridis et Alter

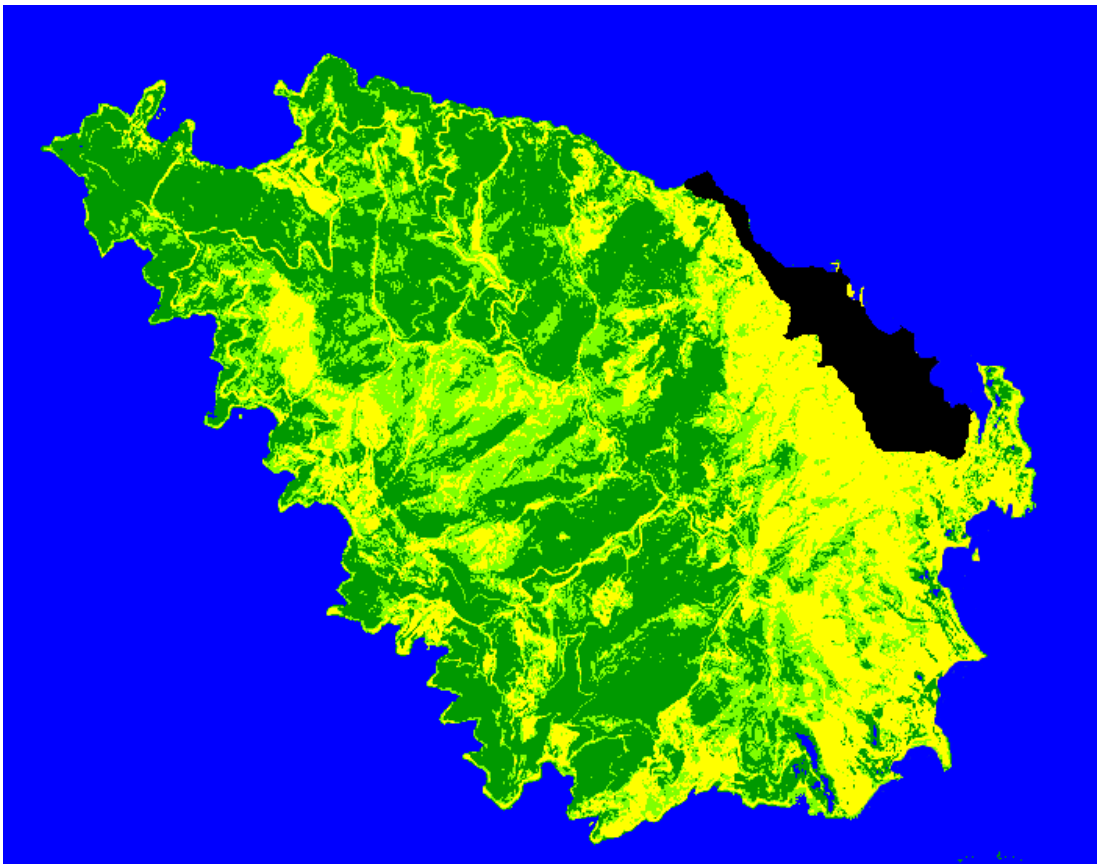


Figura 5. 18: Densidad de la vegetación conforme a el algoritmo de la Distancia Mínima

En la Figura 5.18 observamos cómo hay zonas con vegetación intemedia o densa en la parte sureste de la isla, las cuales en la Figura 5.17 corresponden a zonas de vegetación escasa. Así, en la ejecución de Alexandridis et Alter, la parte sureste de la isla representa una barrera definida en contra de la extensión del incendio, mientras que en nuestra propia ejecución, tal barrera existe pero no está tan definida.

Finalmente, hay que tener que el autómata es probabilístico, por lo que cada ejecución, aunque posea los mismos datos de entrada, presentará resultados diferentes.

5.6 CONCLUSIONES

Hemos realizado una verificación de la ejecución del algoritmo desde tres enfoques complementarios. Primero, hemos comprobado la validez de los cálculos matemáticos. Después, hemos revisado el orden correcto en el recorrido del autómata, tanto a nivel del vecindario de una única celda como en relación al recorrido de todas las celdas del autómata. Finalmente, hemos realizado varias ejecuciones para confirmar el dibujo de los dos pasos anteriores en las ventanas de ejecución. A su vez hemos comparado nuestros resultados con los obtenidos por Alexandridis. Deducimos una ejecución correcta y aproximada de nuestro simulador con respecto al simulador de Alexandridis et Alter y al incendio real.

Una verificación exhaustiva del recorrido del autómata es impracticable, con lo cual nos hemos limitado a unas pocas pruebas de ejemplo y sólo a los cálculos iniciales en cada recorrido.

6 MODO DE EMPLEO DEL SIMULADOR

6.1 INTRODUCCIÓN

En este capítulo explicaremos cómo usar el simulador. Comenzaremos indicando la distribución modular del programa (carpetas, archivos y ejecutable). Continuaremos por un recorrido, pestaña por pestaña, a través del simulador, una vez que se ha ejecutado el archivo que abre la aplicación. Finalmente, en la Sección 6.9, indicaremos cómo obtener una traza de la ejecución del autómata, a partir de la ejecución de la aplicación desde un editor de Java, concretamente desde Eclipse.

6.2 CONTENIDO DEL PAQUETE

La aplicación se distribuye en la carpeta AlexandridisEtAlter. Dentro de dicha carpeta existen 3 subcarpetas y el archivo java jar AlexandridisAutomata.

1. Imagenes: contiene las imágenes de la aplicación.

2. Texto: contiene los archivos de texto, con extensión txt, de la aplicación.

3. Src: contiene los archivos fuente de la aplicación. Para ver la distribución en paquetes de dichos archivos fuente véase la Figura 4.1.

4. AlexandridisAutomata: es el archivo jar de la aplicación.

A su vez, incluimos el archivo jai-1_1_3-lib-windows-i586: es el paquete "Java Advanced Image" de Sun Microsystems, el cual añadimos por si el usuario quiere ejecutar la aplicación desde un editor de Java.

Haciendo doble clic en el archivo AlexandridisAutomata, se abre la aplicación.

Al iniciarse la aplicación, se abre una ventana dividida en dos partes. En la parte izquierda, en el margen superior, aparece un panel de pestañas. Las pestañas son las siguientes:

1. INICIO: muestra la formulación usada en el autómata de Alexandridis.
2. VEGETACION: muestra la obtención de los valores de la densidad de la vegetación, a través del algoritmo de la Distancia Mínima.
3. COMBUSTIBLE: muestra la influencia del tipo de la vegetación.
4. VIENTO: muestra la influencia del viento.
5. ELEVACIÓN: muestra la influencia de la elevación del terreno.
6. AUTOMATA: realiza la ejecución del autómata conforme a los valores introducidos por el usuario y muestra ejecuciones de ejemplo.

6.3 PESTAÑA INICIO

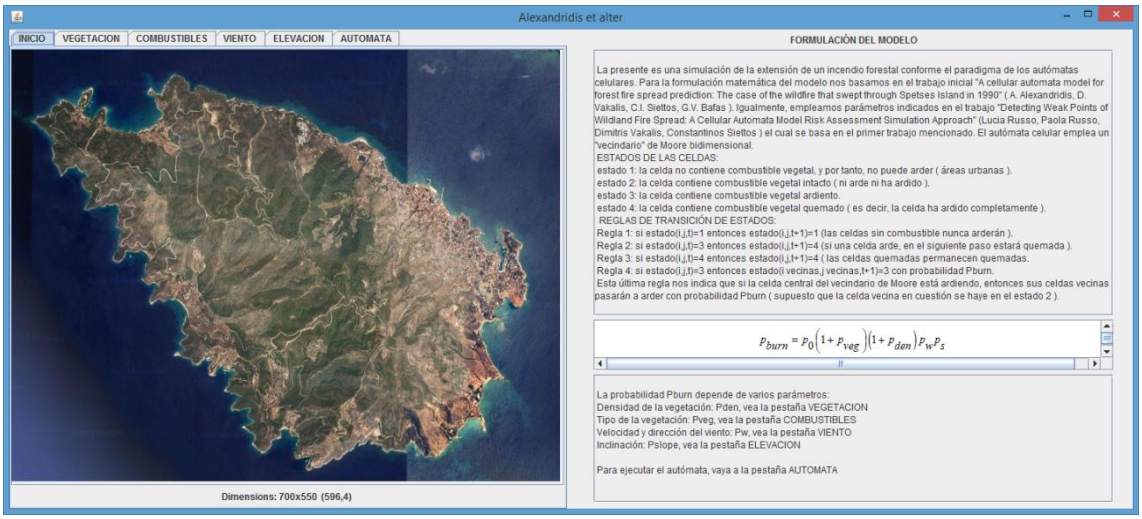


Figura 6. 1: Pestaña INICIO

En el lado izquierdo aparece una imagen Landsat de la isla de Spetses.
Ésta será la imagen original, la cual será clasificada usando el algoritmo de clasificación de imágenes de la Distancia Mínima:

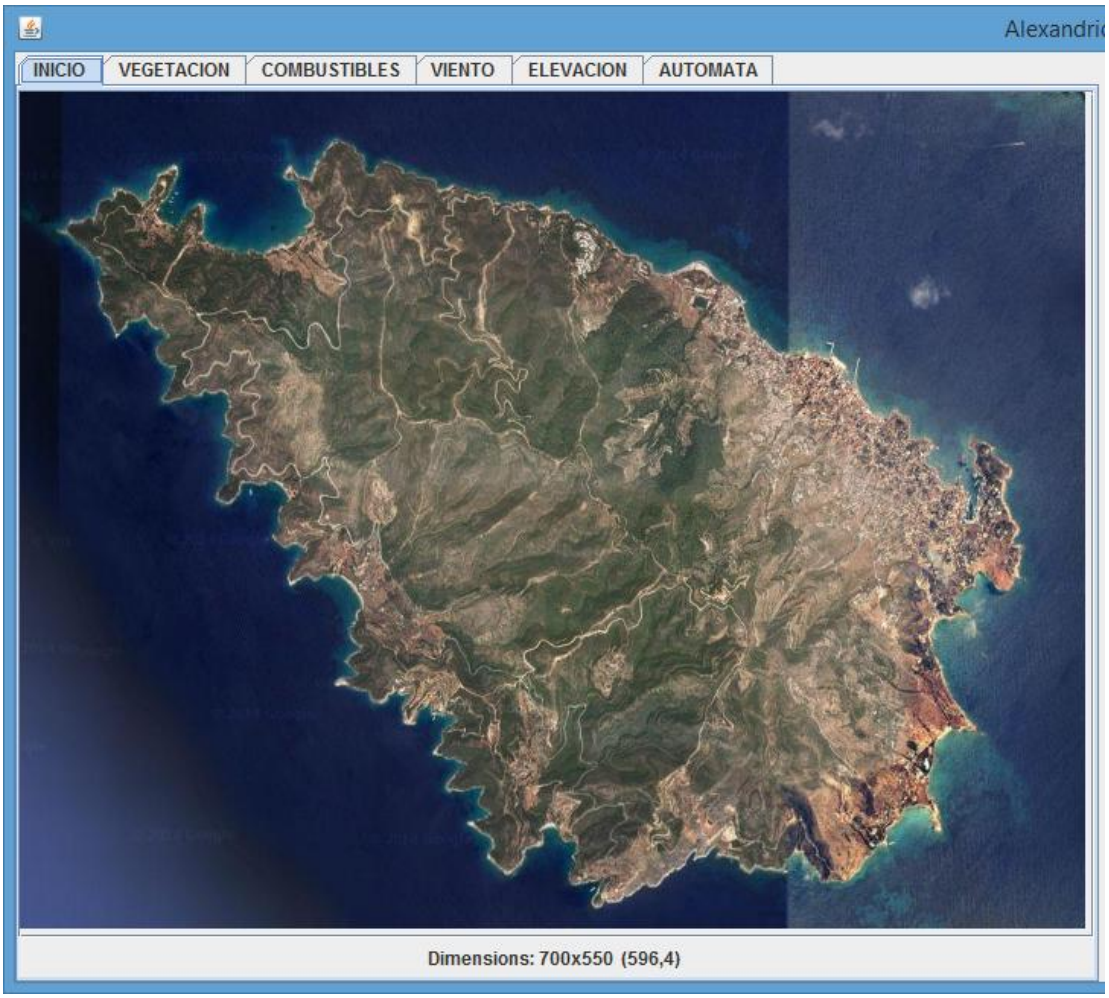


Figura 6. 2: Pestaña INICIO, lado izquierdo

En el lado derecho aparece una explicación general del Autómata de Alexandridis et Alter, con la fórmula general del mismo (Sección 3.2.4):

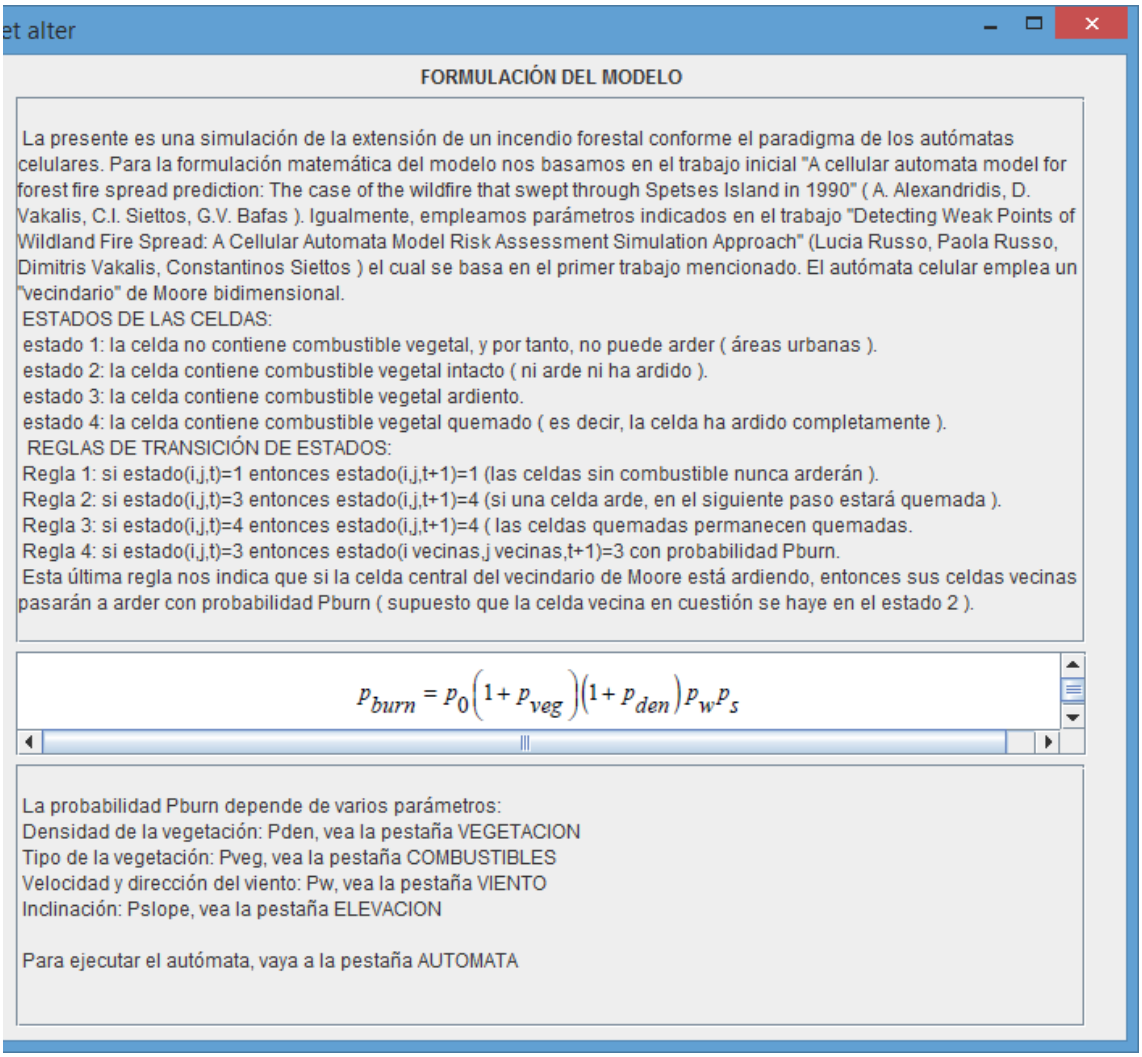


Figura 6. 3: Pestaña INICIO, lado derecho

6.4 PESTAÑA VEGETACION

A continuación se pulsa sobre la pestaña vegetación. Esta parte contiene la aplicación, a la imagen Landsat original, del algoritmo de clasificación de imágenes de la Distancia Mínima. Para una explicación prolija de dicho algoritmo, vea la Sección 4.3.6 Clase PanelVegetacionDerecha.

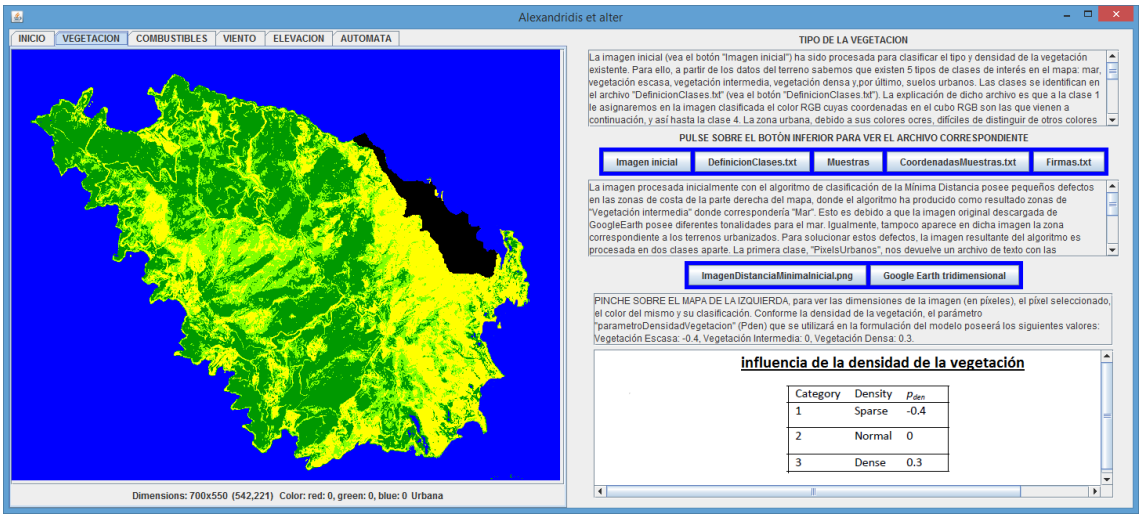


Figura 6. 4: Pestaña VEGETACION

En el lado izquierdo, aparece el resultado final de la clasificación de la imagen Landsat original. Pasando el cursor del ratón sobre dicha imagen aparecen al pie de la misma los datos relativos a la posición del cursor (coordenadas x e y del píxel), los valores RGB del píxel en cuestión y su clasificación en cuanto al parámetro "densidad de la vegetación", P_{den}.

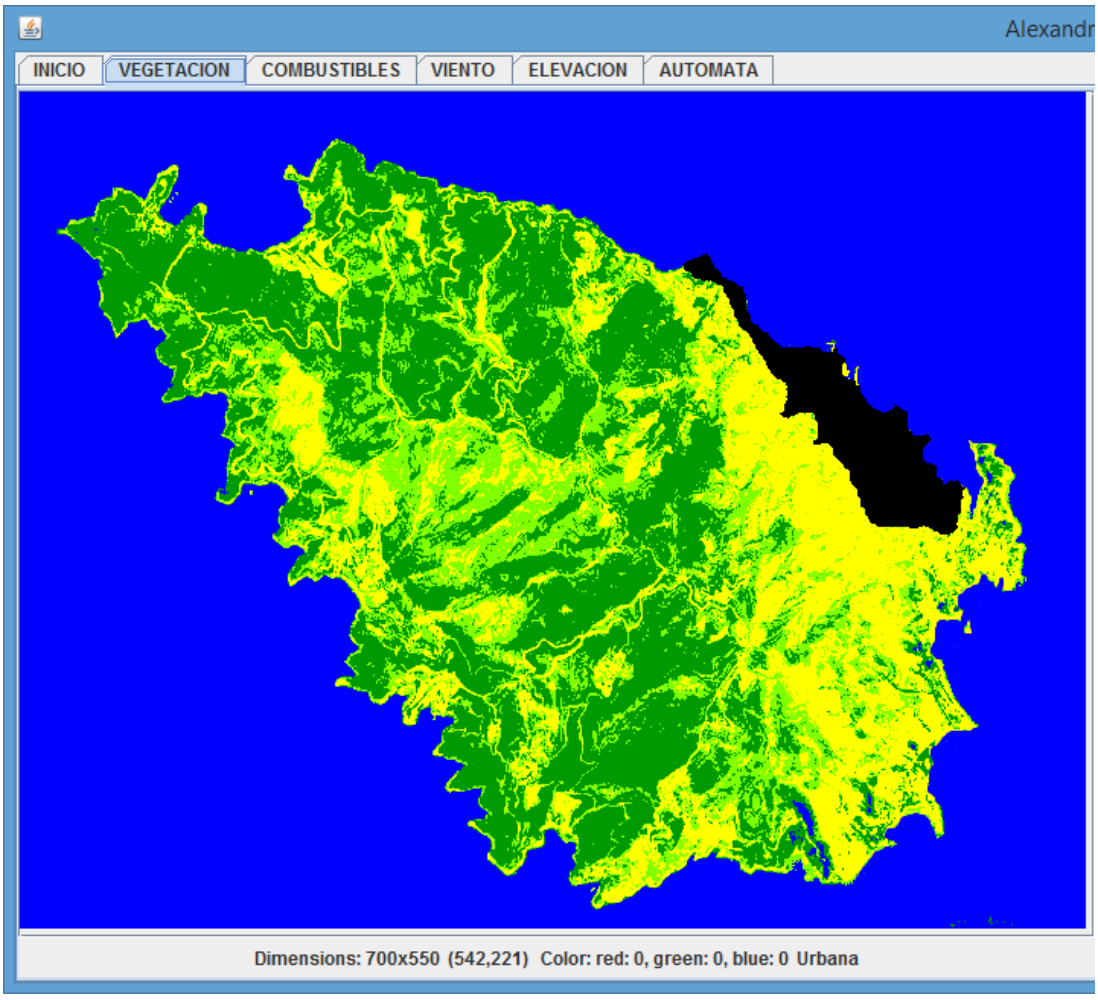


Figura 6. 5: Pestaña VEGETACION, lado izquierdo

En el lado derecho se explica, en primer lugar, el algoritmo de clasificación de imágenes de la Distancia Mínima. A continuación aparecen 5 botones:

1. Imagen inicial: al pulsarlo, aparece la imagen Landsat original, a fin de poder compararla con la imagen clasificada.

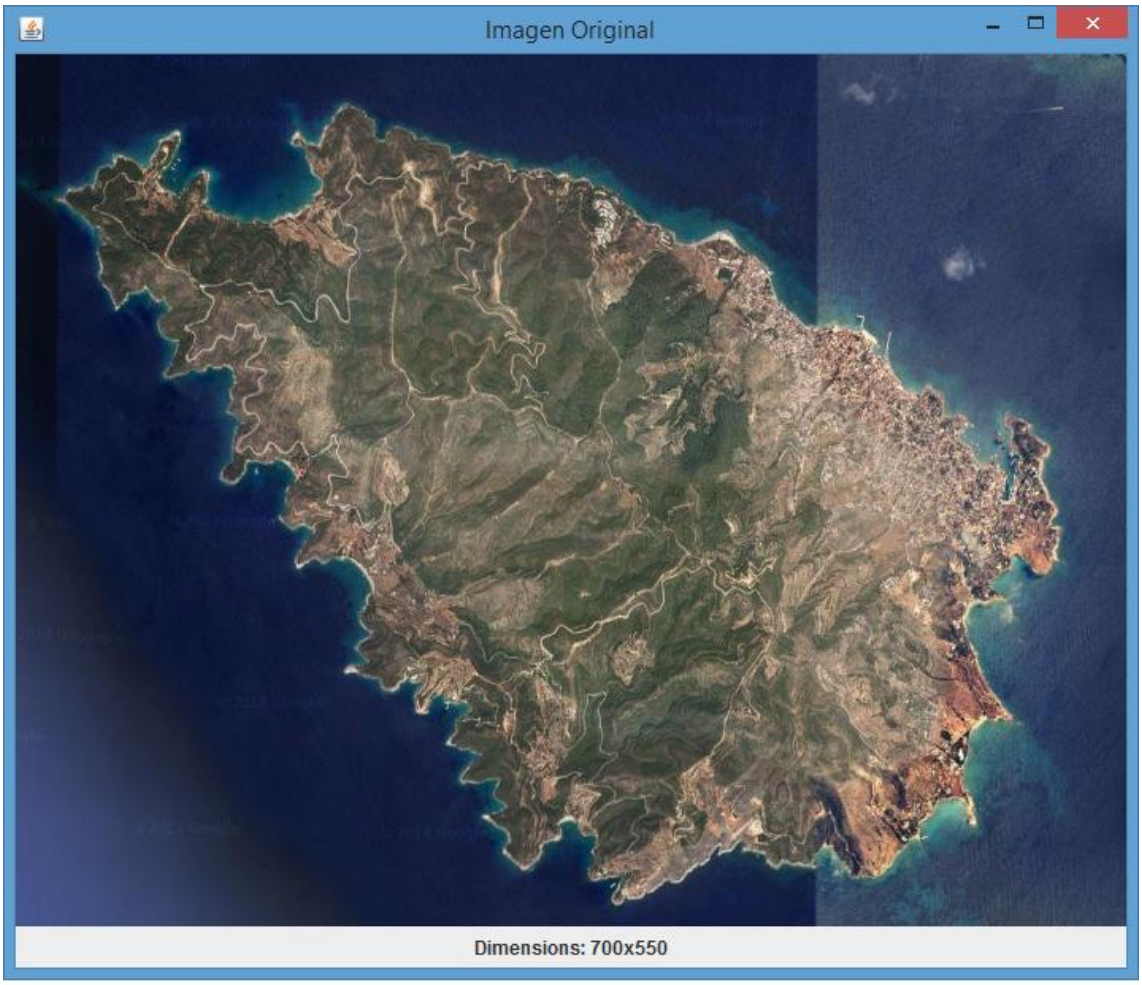


Figura 6. 6: Pestaña VEGETACION, lado derecho, botón "Imagen Inicial"

2. DefinicionClases.txt: al pulsarlo, aparece el archivo con las clases RGB empleadas en el algoritmo de clasificación de imágenes de la Distancia Mínima, explicado en la Sección 4.3.6.

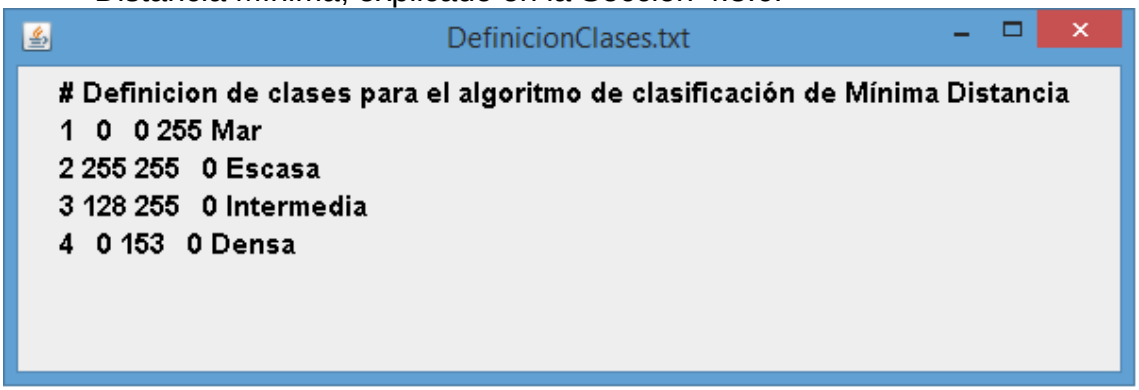


Figura 6. 7: Pestaña VEGETACION, lado derecho, botón "DefinicionClases.txt"

3. Muestras: al pulsarlo, aparece la imagen Landsat original con los rectángulos de lmuestras superpuestos a la misma. Debajo de la imagen se indica el color asociado a las muestras de cada clase.

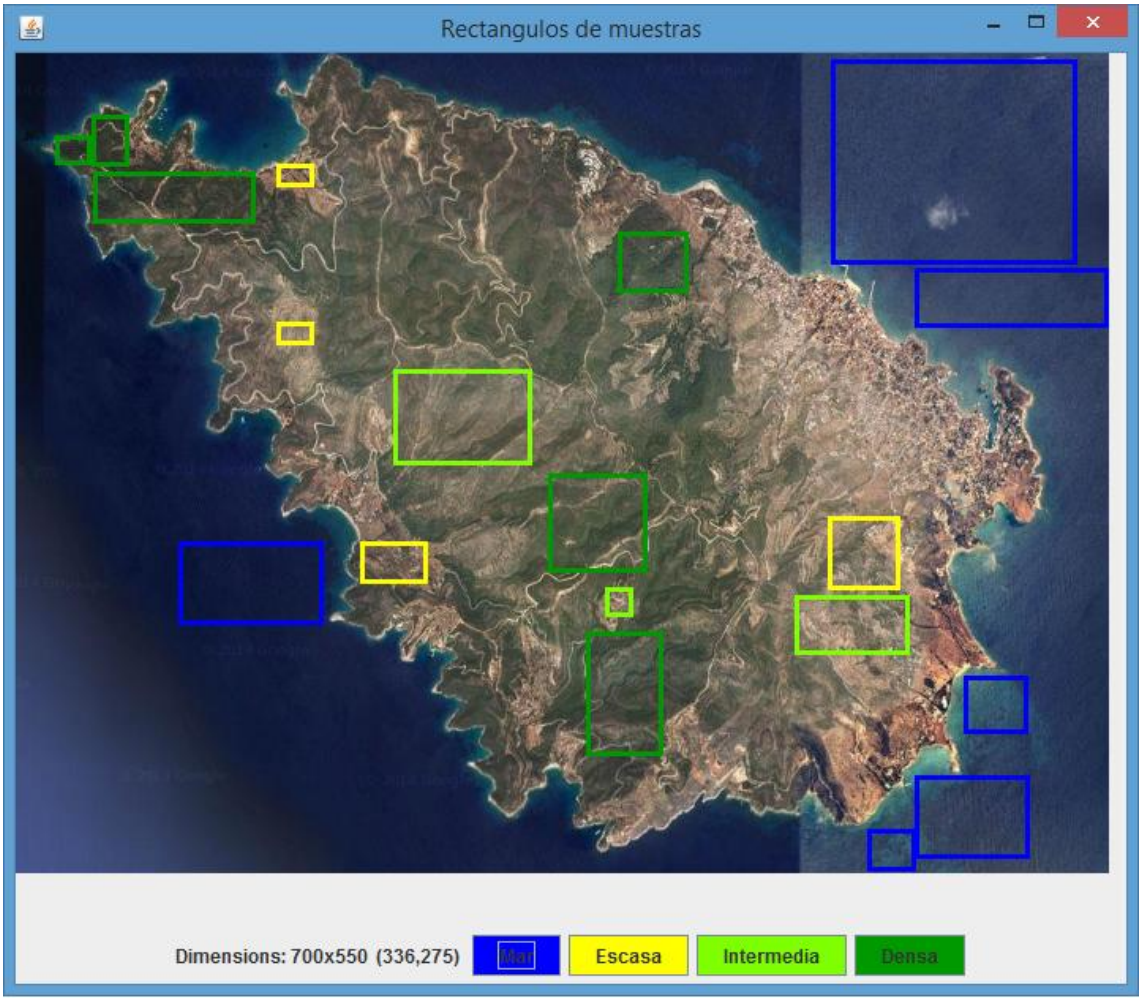


Figura 6. 8: Pestaña VEGETACION, lado derecho, botón "Muestras"

4. CoordenadasMuestras.txt: al pulsarlo, aparece una imagen del archivo que contiene las coordenadas de los rectángulos de las muestras para cada clase.

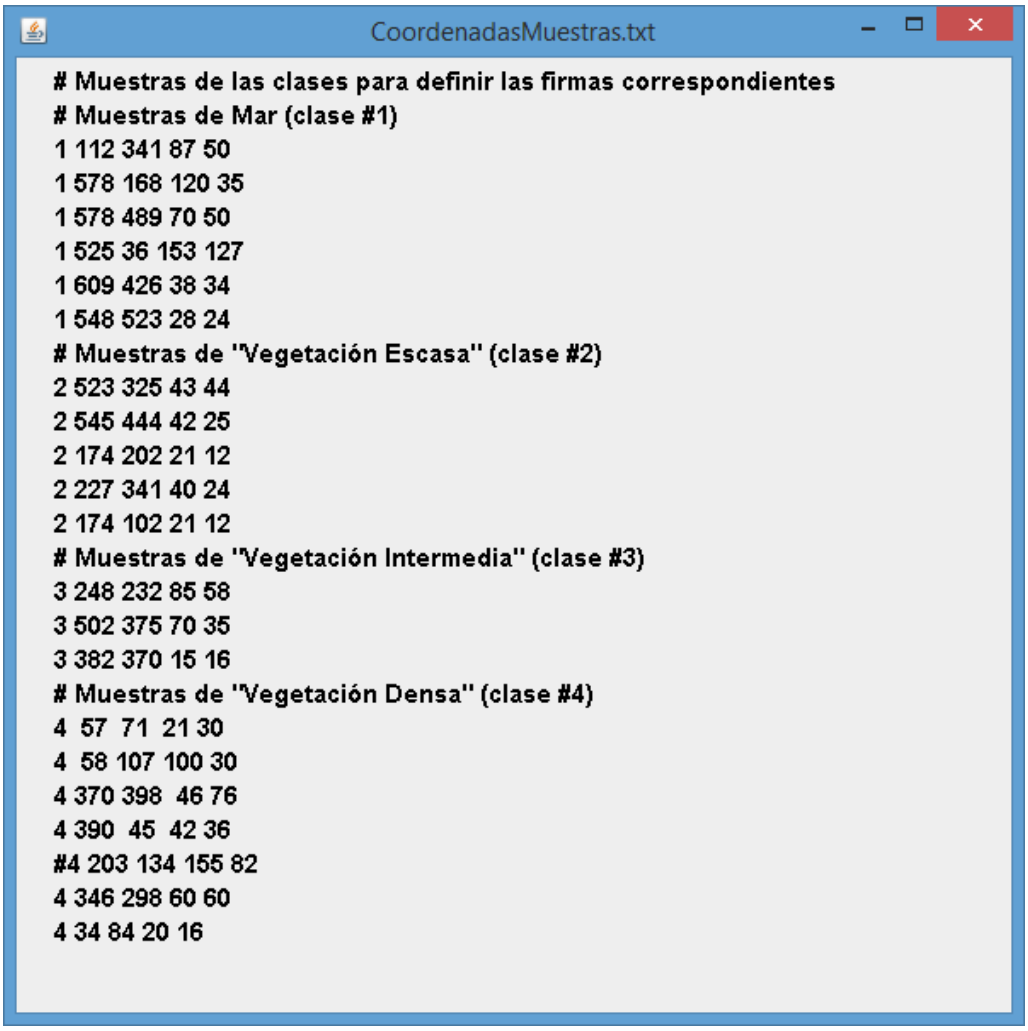


Figura 6. 9: Pestaña VEGETACION, lado derecho, botón "CoordenadasMuestras.txt"

5. Firmas.txt: al pulsarlo surge una imagen con las firmas de cada clase. El algoritmo de clasificación de imágenes de Distancia Mínima halla la media de los valores RGB para el conjunto de los rectángulos de muestras de cada clase. Tales valores medios constituyen la firma de la clase.

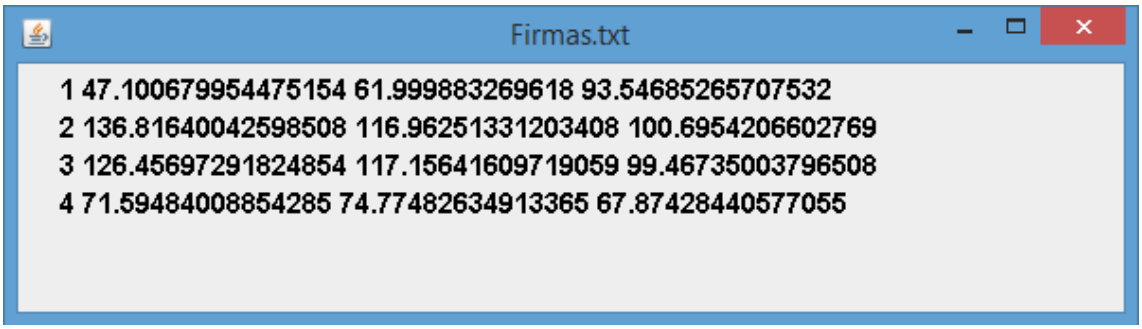


Figura 6. 10: Pestaña VEGETACION, lado derecho, botón "Firmas"

Debajo de dichos botones se explica que la imagen resultante del algoritmo ha de ser postprocesada (Sección 4.3.6). Los dos botones que aparecen a continuación muestran lo siguiente:

1. ImagenDistanciaMinimaInicial.png: al pulsarlo, aparece la imagen resultante del algoritmo, sin el postprocesado. En dicha imagen aparecen una serie de defectos, como las nubes de la esquina superior derecha, las zonas de playa de la parte derecha y la zona correspondiente al área urbana. La imagen puede compararse con la de la parte izquierda de la pestaña en curso, "VEGETACION".

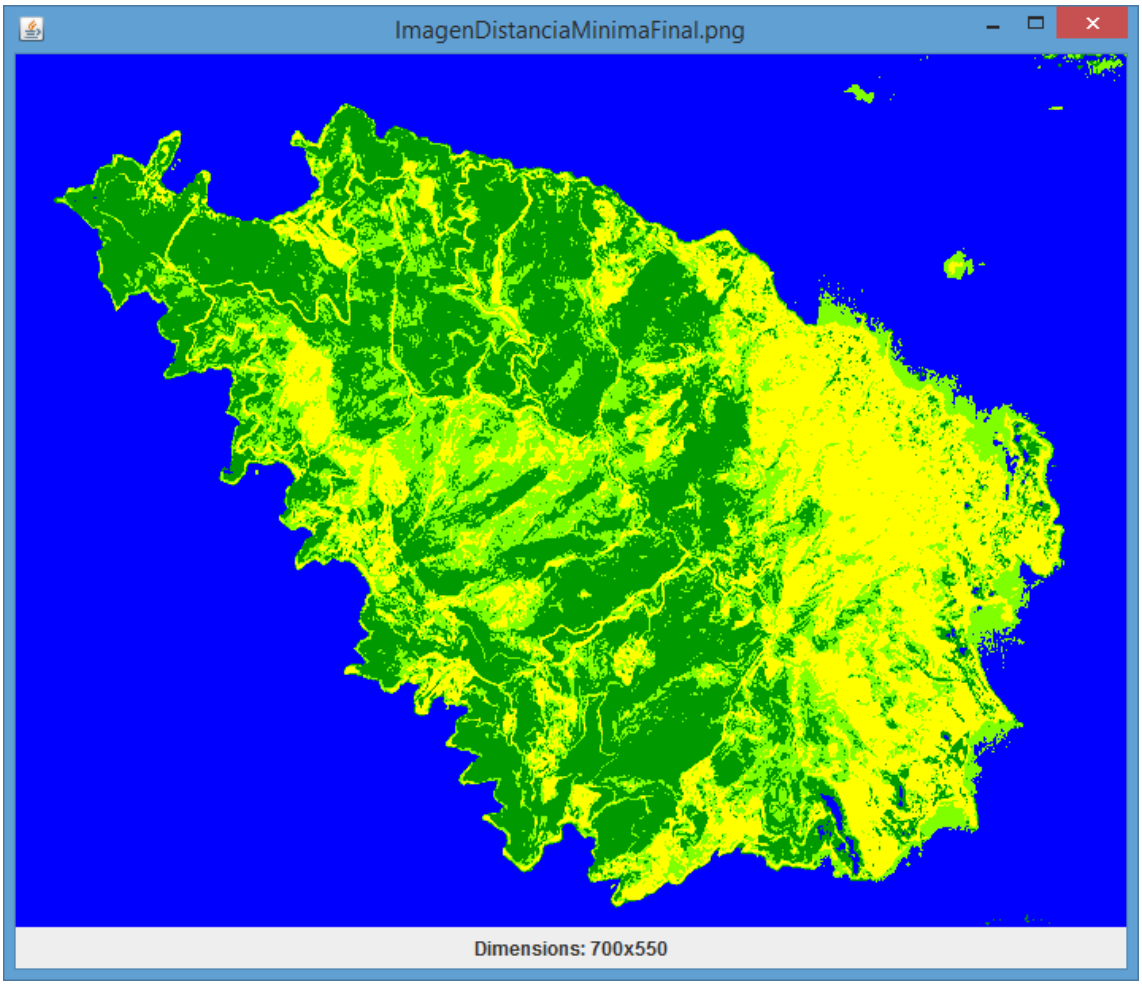


Figura 6. 11: Pestaña VEGETACION, lado derecho, botón "ImagenDistanciaMinimaFinal.png"

2. Google Earth tridimensional: al pulsarlo, aparece la imagen Landsat original, en tres dimensiones. Parte del postprocesado de la imagen anterior deriva de las diferentes tonalidades y sombras de la imagen Landsat original. Tales variaciones se aprecian mejor en la versión tridimensional de dicha imagen Landsat original, tal y como se puede apreciar en los diferentes tonos y brillos que aparecen rodeando a la isla y, especialmente, en la parte derecha de la misma.

Figura 6. 12: Pestaña VEGETACION, lado derecho, botón "GoogleEarthTridimensional"

Debajo de estos dos últimos botones aparece la información relacionada con el parámetro "densidad de la vegetación" del autómata de Alexandridis et Alter, junto con una imagen con la formulación del mismo. La correspondencia es: vegetación escasa (Sparse), vegetación intermedia (Normal) y vegetación densa (Dense).

6.5 PESTAÑA COMBUSTIBLES

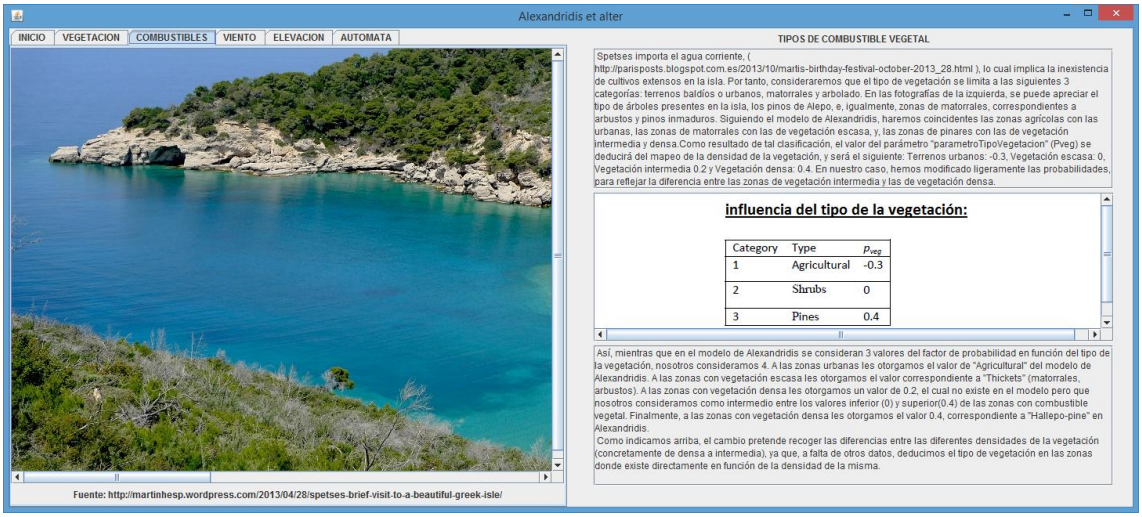


Figura 6. 13: Pestaña COMBUSTIBLES

En el lado izquierdo aparece un marco con dos imágenes. En la parte inferior de dicho marco se indica la fuente de Internet de la cual se han obtenido dichas imágenes. Para ver ambas imágenes, mueva con el cursor del ratón la barra deslizadora.

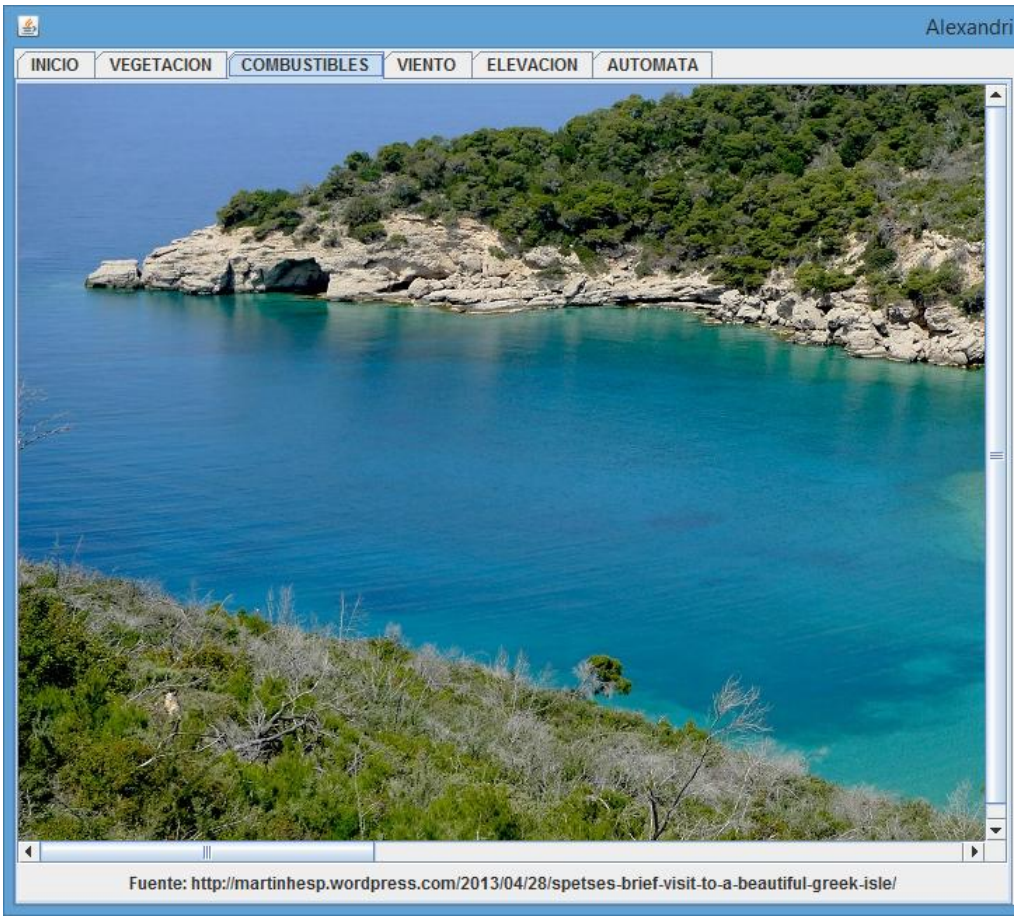


Figura 6. 14: Pestaña COMBUSTIBLES, lado izquierdo, imagen izquierda

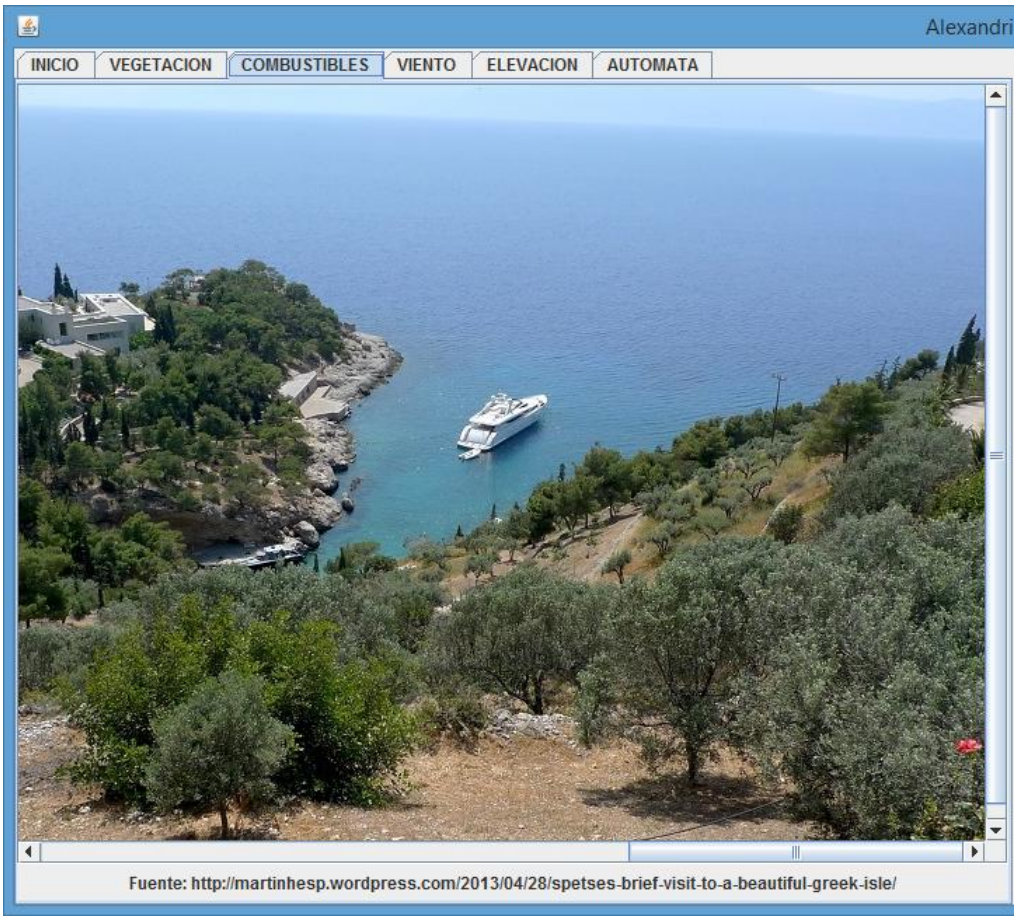


Figura 6. 15: Pestaña COMBUSTIBLES, lado izquierdo, imagen derecha

En el lado derecho se explica cómo se ha obtenido el parámetro "tipo de la vegetación", P_{veg}.

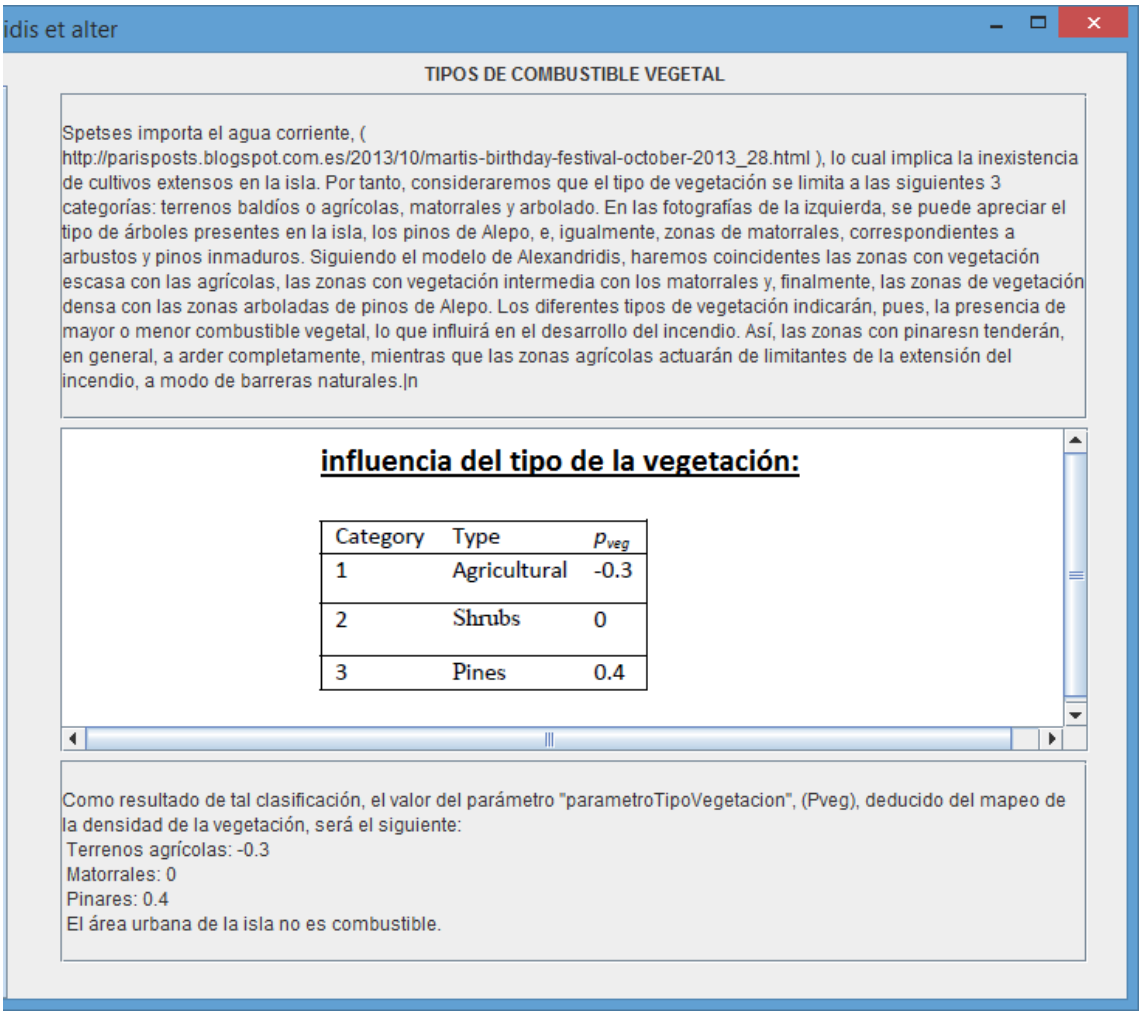


Figura 6. 16: Pestaña COMBUSTIBLES, lado derecho

6.6 PESTAÑA VIENTO

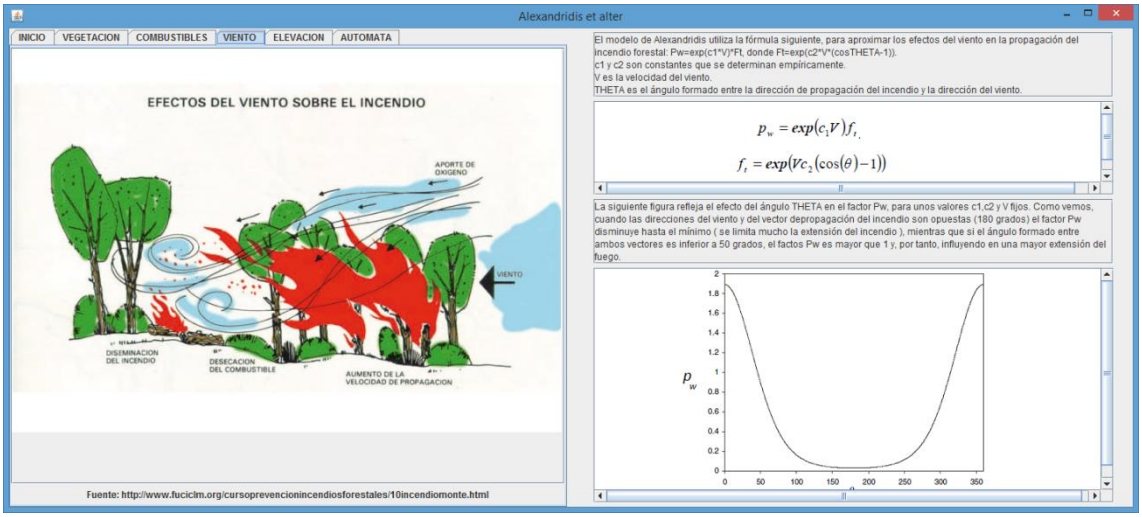


Figura 6. 17: Pestaña VIENTO

En el lado izquierdo, se añade una imagen "de relleno", mostrando los principales efectos del viento en la propagación de los incendios en zonas arboladas.

Figura 6. 18: Pestaña VIENTO, lado izquierdo

En el lado derecho se indica la formulación en el autómata de Alexandridis et Alter del parámetro que modela la influencia del viento, P_w. En la parte superior se muestran sus fórmulas, y en la parte inferior se muestra una gráfica donde se representa, para unos valores de c_1, c_2 y V fijos, la dependencia de P_w respecto al ángulo theta formado entre la dirección de propagación del incendio y la dirección del viento (Secciones 3.7 y 4.3.9).

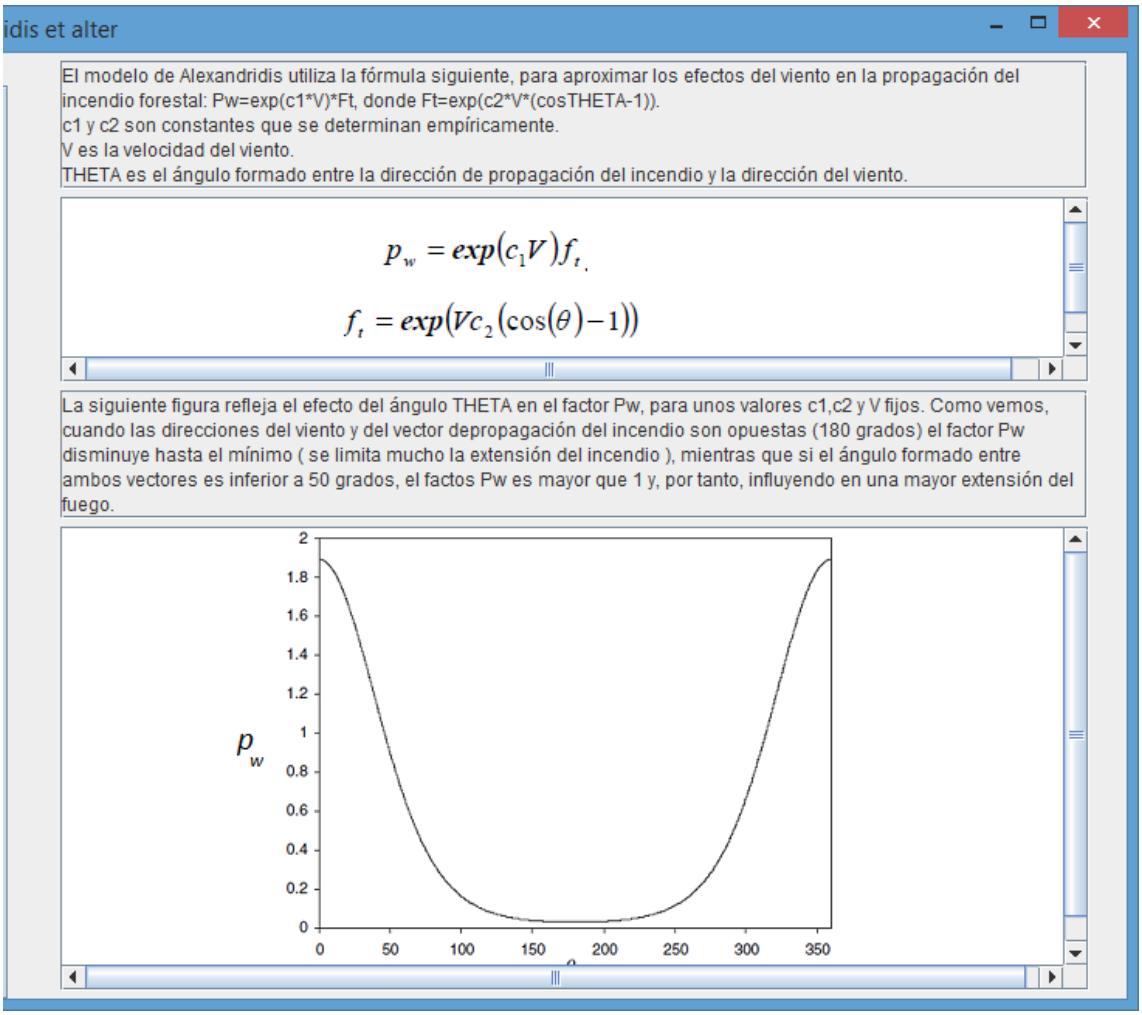


Figura 6. 19: Pestaña VIENTO, lado derecho

6.7 PESTAÑA ELEVACION

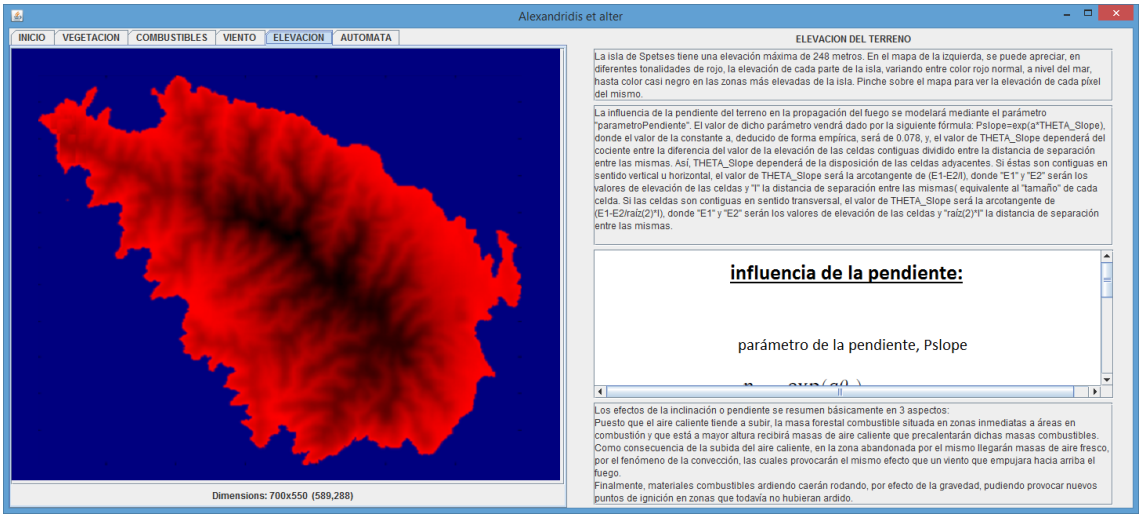


Figura 6. 20: Pestaña ELEVACION

En la parte izquierda aparece una imagen en color donde se indican, en colores rojos más oscuros, las partes de la isla más elevadas, y en colores rojos más

claros, las más próximas al nivel del mar. Pulse con el ratón sobre un punto de la isla para obtener la altitud de dicho punto.

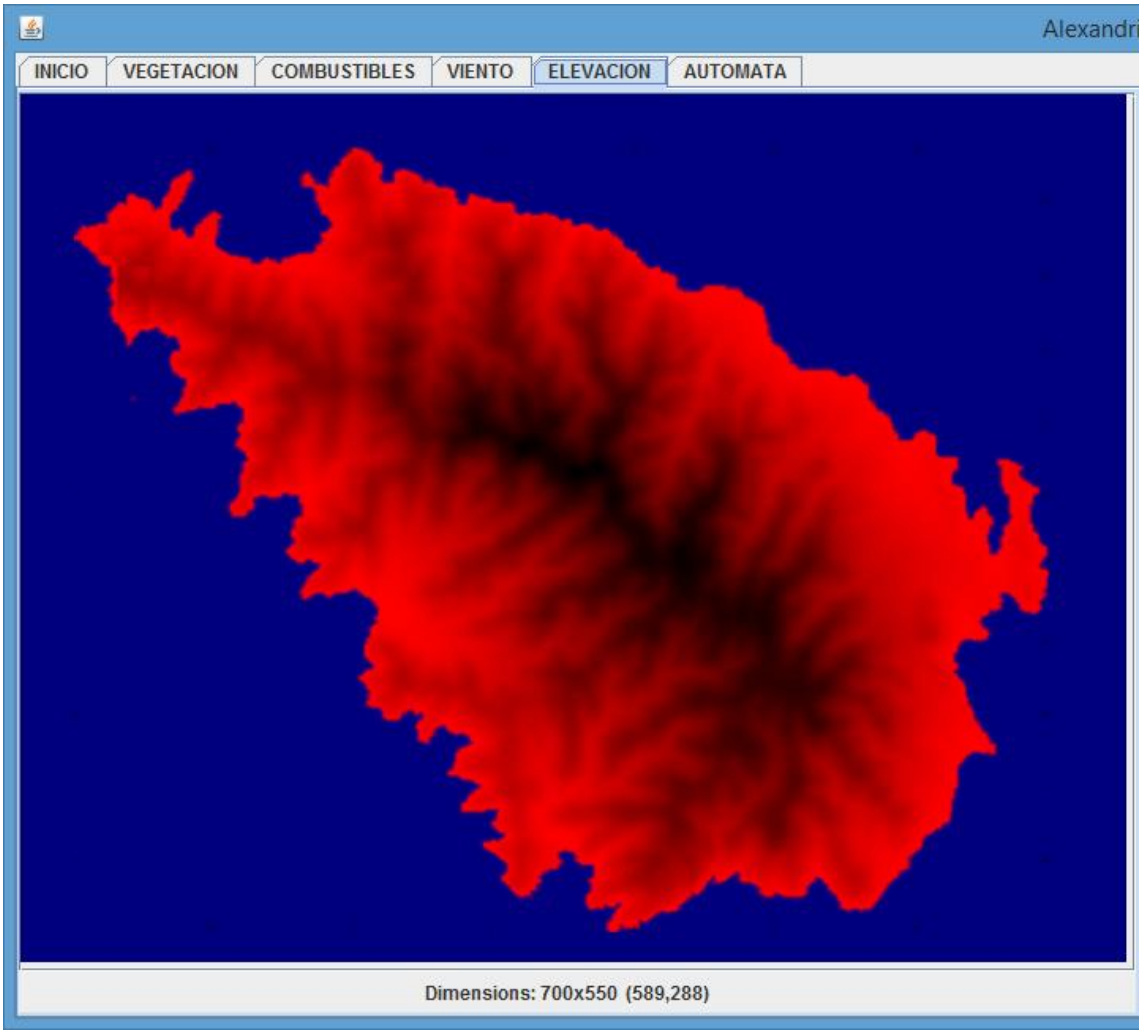


Figura 6. 21: Pestaña ELEVACION, lado izquierdo

En la parte derecha se indica la forma en que se obtiene el parámetro P_s que modela la influencia de la elevación del terreno en la forma de la extensión del incendio en el modelo de Alexandridis et Alter (Secciones 3.2.8 y 4.3.11). El último texto, en la parte inferior, describe someramente la influencia de la pendiente del terreno en los incendios.

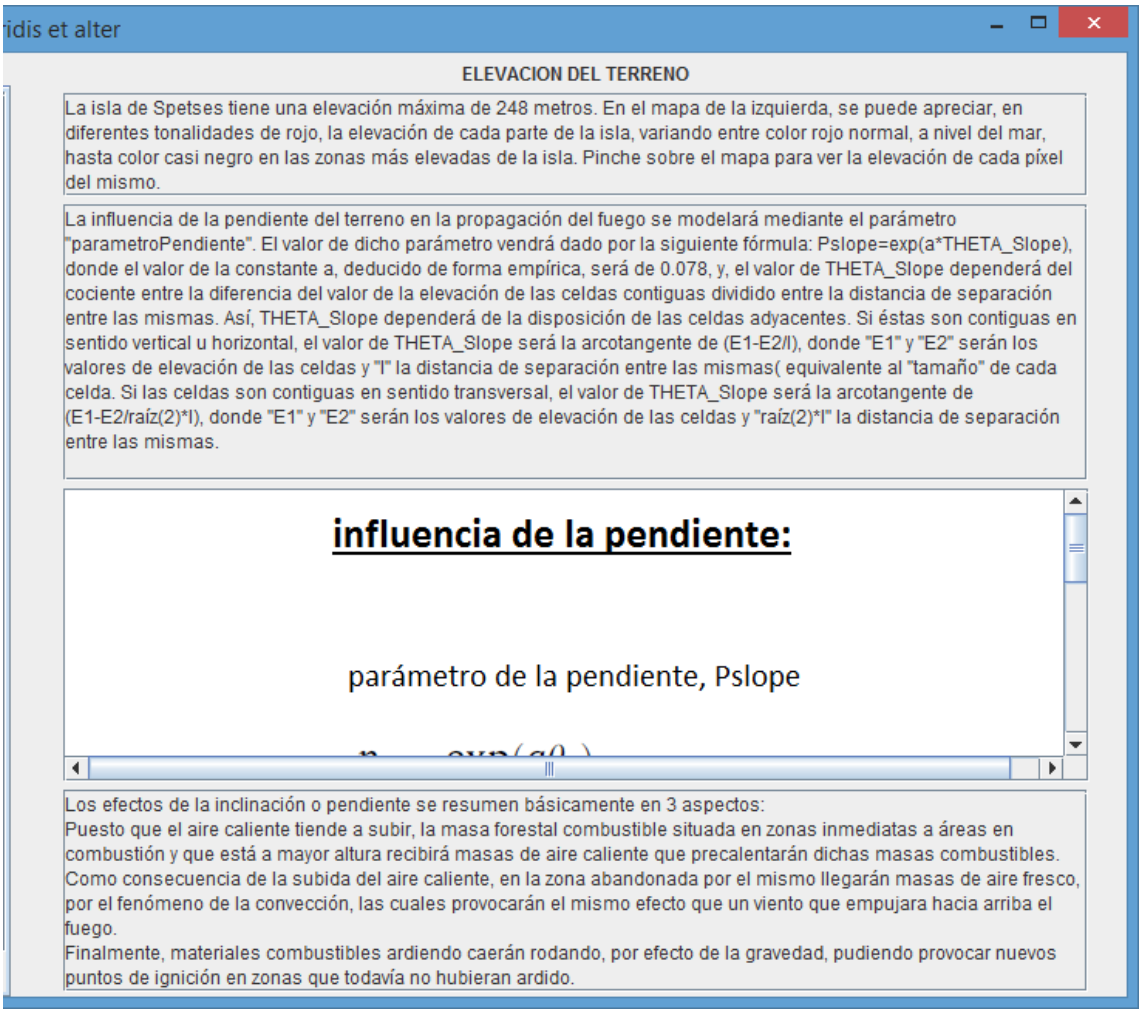


Figura 6. 22: Pestaña ELEVACION, lado derecho

6.8 PESTAÑA AUTOMATA

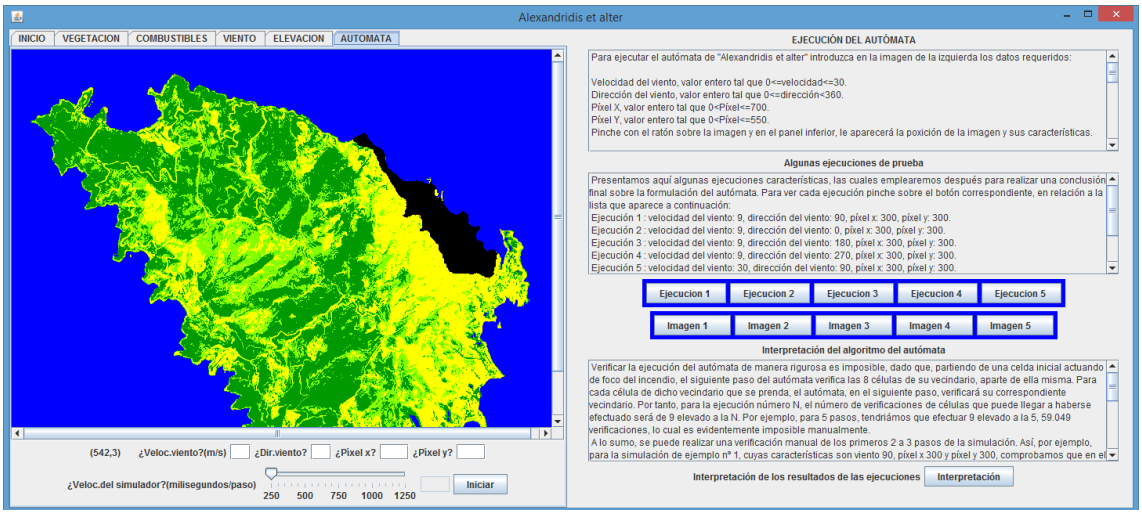


Figura 6. 23: Pestaña AUTOMATA

En la parte izquierda se muestra la ventana para la ejecución personalizada del autómata. La imagen inicial es la resultante del algoritmo de clasificación de

imágenes de la Distancia Mínima, incluyendo su post- proceso. Debajo de la misma se piden al usuario los siguientes parámetros:

1. Velocidad del viento, en metros por segundo (m/s). El rango varía entre 0 y 30 m/s. Algunas equivalencias en Km/h son 10 m/s igual a 36 Km/h, 20 m/s igual a 72 km/h, y 30 m/s igual a 108 km/h.
2. Dirección del viento, en grados. El rango varía entre 0 y 359 grados. Nótese, que para hacer coincidir el origen (0 grados) con el norte, se considera 0 grados (norte), 90 grados (oeste), 180 grados (sur) y 270 grados (este). El sentido es contrario al de las agujas del reloj.
3. Coordenadas x e y del foco inicial del incendio: pulse con el ratón sobre la imagen para obtener las coordenadas x e y de un punto de la isla.
4. Velocidad del simulador, en milisegundos por paso. Mediante la barra deslizadora, se podrá regular dicha velocidad. Caso de no modificar la posición inicial de la barra, la velocidad de ejecución no será de 250 milisegundos por paso, sino de 500, es decir, 2 pasos del simulador por segundo. Si se desea establecer otra velocidad, es necesario manipular la barra deslizadora.

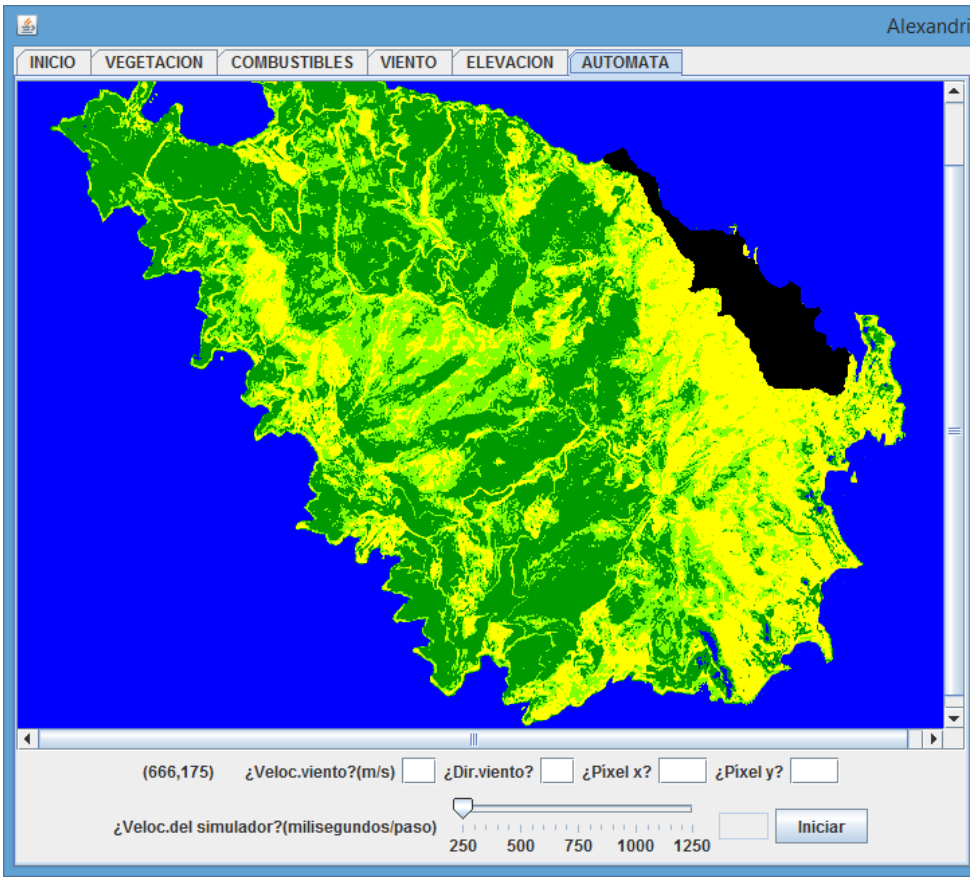


Figura 6. 24: Pestaña AUTOMATA, lado izquierdo

Una vez que pulse el botón "Iniciar", se abrirá una ventana con la ejecución del autómata.

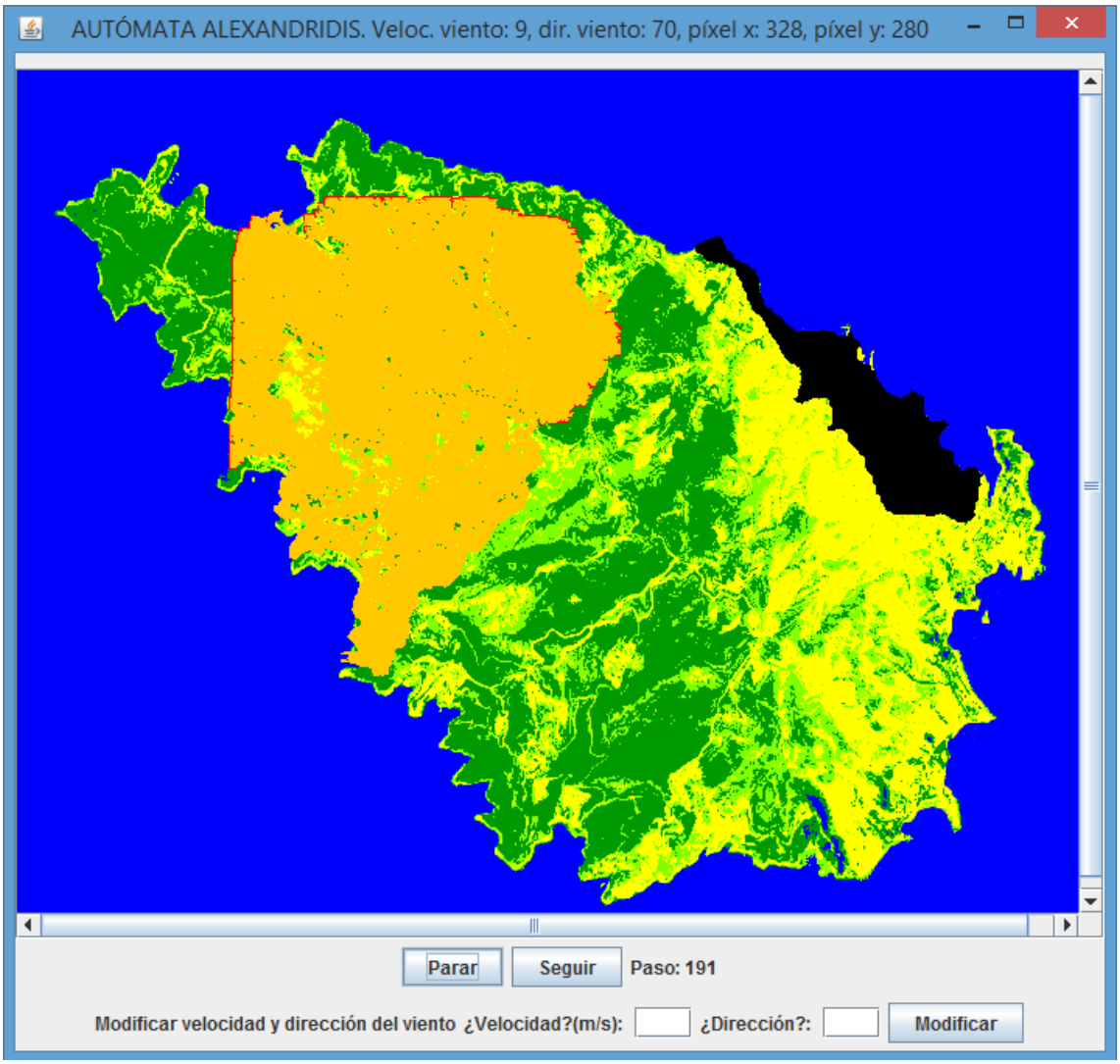


Figura 6. 25: Pestaña AUTOMATA, lado izquierdo, ejecución en curso

En la parte superior de dicha ventana, aparecerán los datos introducidos por el usuario, excepto el de la velocidad del simulador. En la ventana del autómata se irá ejecutando el mismo. Las células ardiendo tendrán color rojo, y así se podrá identificar el frente del incendio. Las células que hayan ardido tendrán color naranja, y así se podrá identificar la parte quemada por el fuego.

En la parte inferior aparecen los botones "Parar" para pausar la ejecución del autómata, y "Seguir" para continuar con la misma. A continuación de ambos botones se indica el paso de la simulación y la duración aproximada del fuego.

Debajo, se incluyen dos campos con los cuales el usuario puede cambiar la velocidad y la dirección de la ejecución en curso. Introduzca las nuevas velocidad y dirección y dele al botón "Modificar". El autómata continuará su ejecución con los nuevos parámetros. En caso de que se hubieran modificado la velocidad y dirección con el autómata pausado, será necesario pulsar el botón "Seguir", después de haber pulsado a "Modificar", para continuar con la ejecución.

Al abrirse la ventana del autómata, también se abrirá una pequeña ventana con un icono de una "bolsa de aire", a modo de indicador de la dirección del viento. En dicho icono la velocidad y la dirección del viento se representarán por medio de una flecha, cuyo módulo será proporcional a la velocidad del viento, y cuyo sentido será el de la dirección indicada, considerando el Norte como 0 grados.

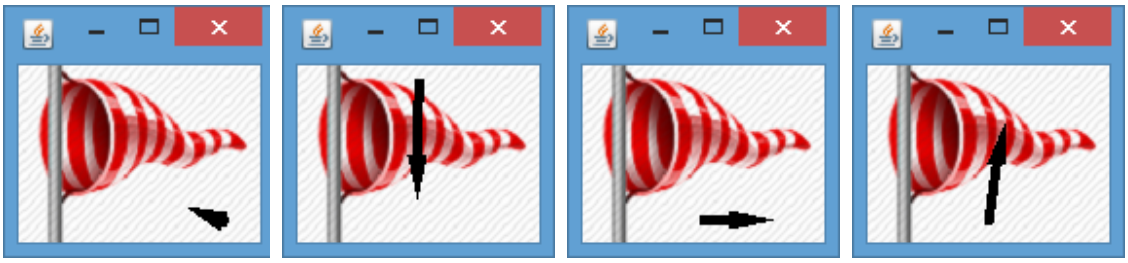


Figura 6. 26: "Bolsas de viento"

Aquí, la primera imagen corresponde a velocidad 9 m/s y dirección 70 grados, la segunda a 30 y 180, la tercera a 18 y 270 y la cuarta a 25 y 350.

En el lado derecho de la pestaña "AUTOMATA" se informa al usuario, en primer lugar, de lo relativo a la introducción de los datos para la ejecución del autómata. En segundo lugar se le presenta información sobre las ejecuciones de prueba, junto con 5 botones para ver las mismas y otros 5 para ver la imagen final resultante de cada una de ellas. Sigue una interpretación del algoritmo del autómata y, finalmente, un botón "Interpretación" que abre una ventana con la interpretación de las ejecuciones de ejemplo.

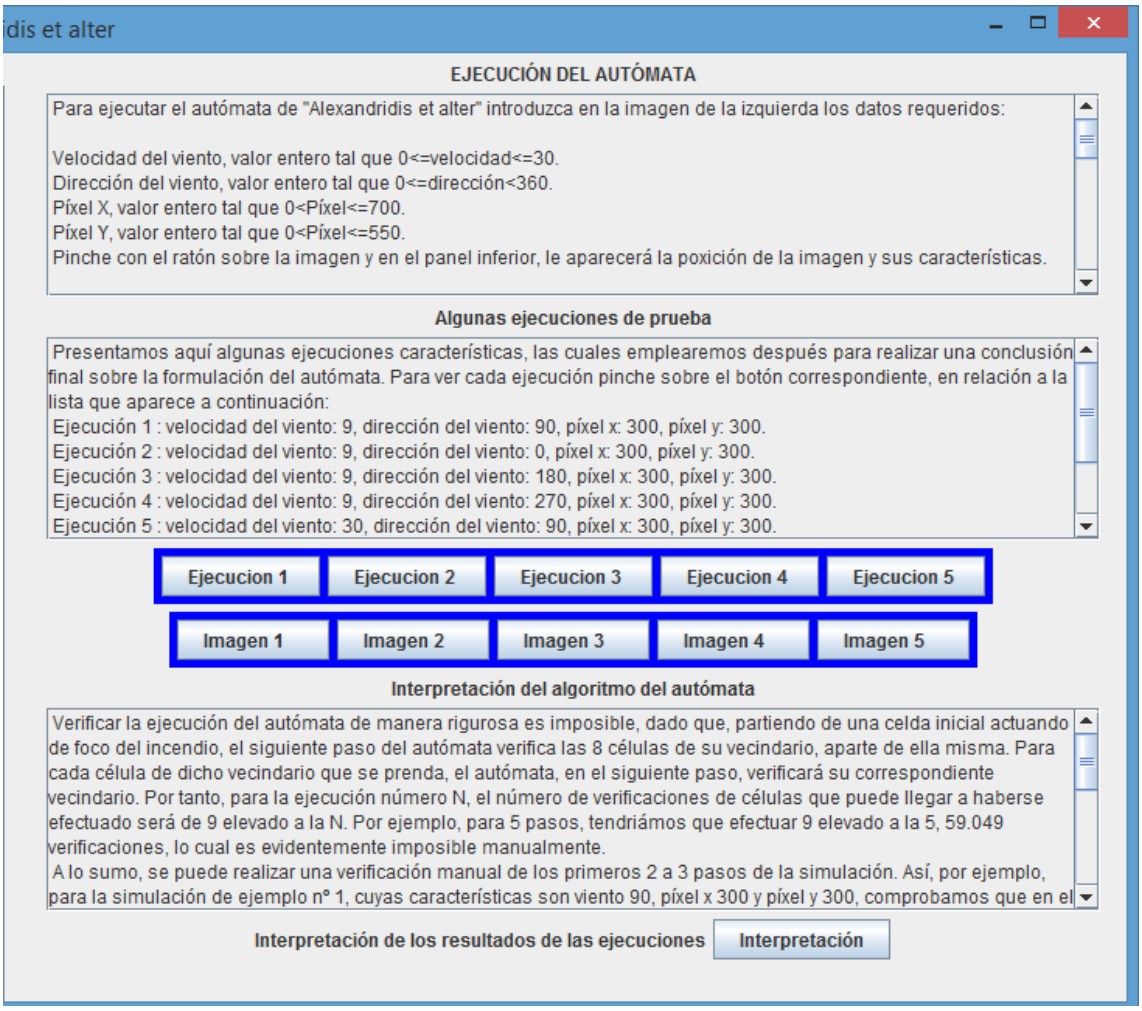


Figura 6. 27: Pestaña AUTOMATA, lado derecho

Veamos la primera ejecución de ejemplo, el resto sigue un esquema similar. Pulsando el botón "Ejecución 1" se abre una ventana, en la cual aparece la primera ejecución de ejemplo, la cual se inicia automáticamente. En la parte superior de la ventana se indican los parámetros de dicha ejecución. En la parte inferior de la ventana el usuario tiene la opción de pausar y continuar la ejecución, a la vez que se le indica el avance del autómata en cuanto a los pasos habidos del mismo y la duración del incendio en curso.

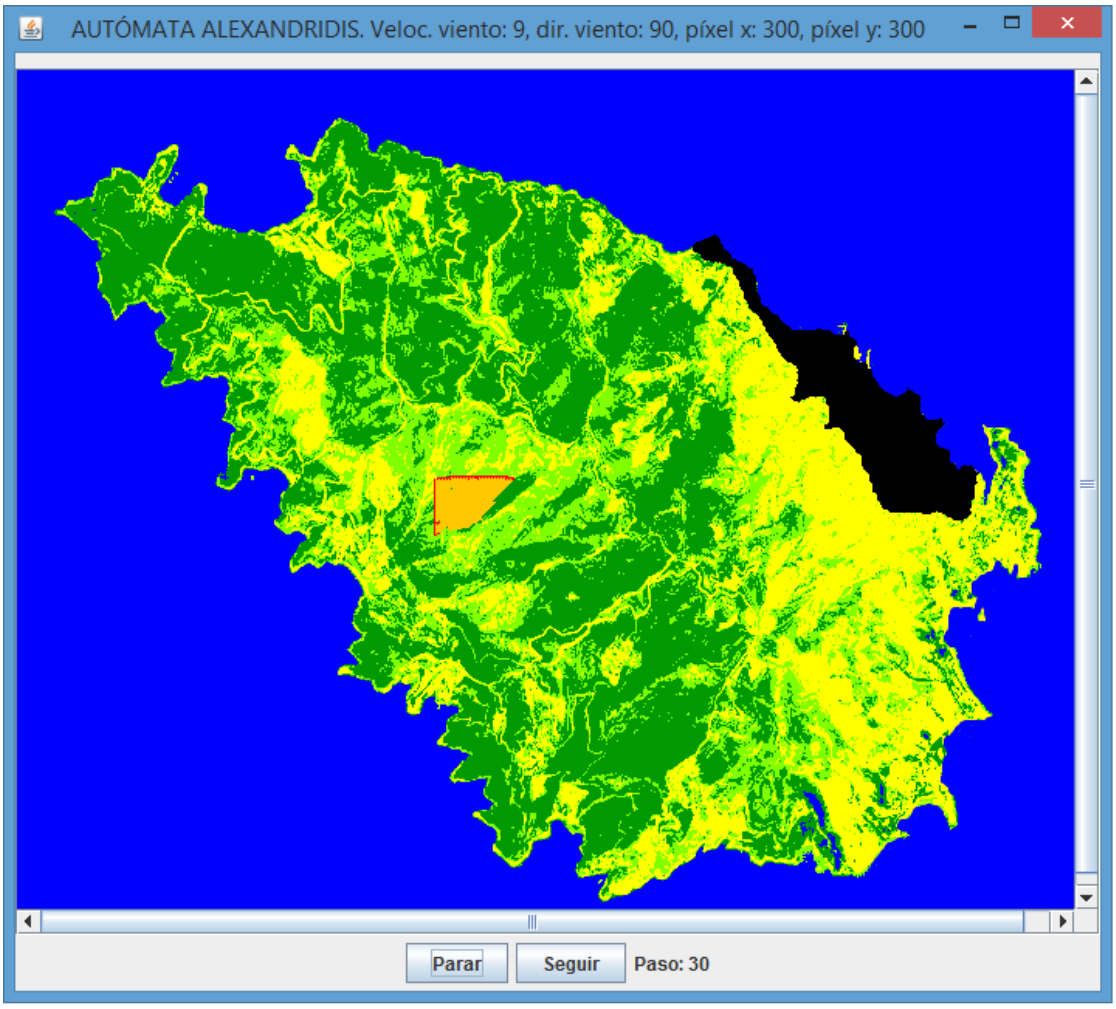


Figura 6. 28: Pestaña AUTOMATA, lado derecho, botón "Ejecución 1"

Si el usuario desea ver el resultado final de la ejecución del autómata, es decir, cómo queda una vez que ya no hay más celdas ardiendo, pulsando el botón "Imagen 1" se mostrará una ventana con la imagen del autómata una vez que éste ha terminado su ejecución. Hay que tener en cuenta que el autómata es probabilístico, por lo que diferentes ejecuciones de un mismo autómata (un autómata con los mismos datos iniciales) no tienen por qué dar el mismo resultado, y, de hecho, en general darán resultados diferentes, aunque con cierta afinidad entre ellos (véase la Sección 5.4).

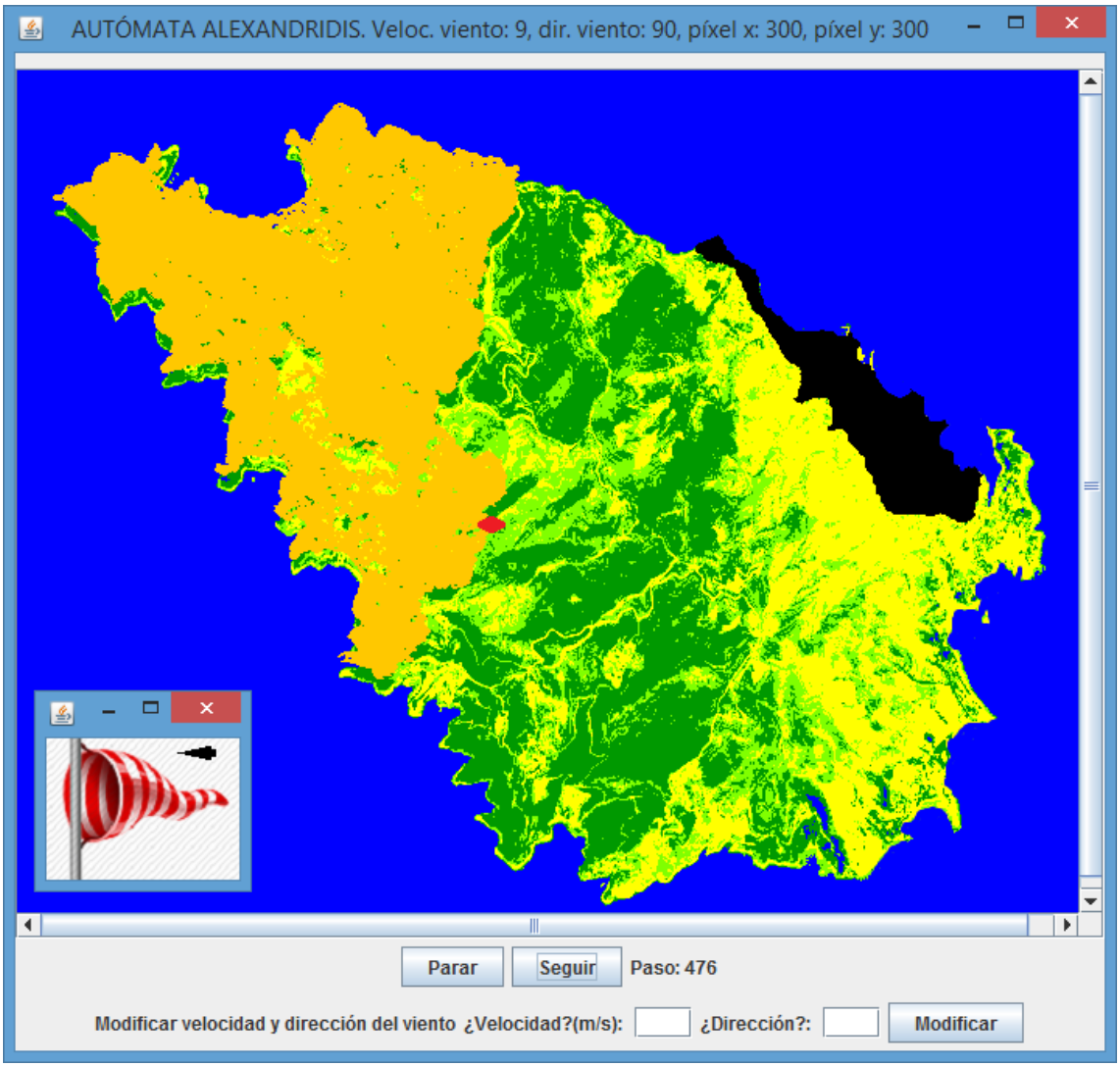


Figura 6. 29: Pestaña AUTOMATA, lado derecho, botón "Imagen 1"

Terminando con el lado derecho de la pestaña "AUTOMATA", en la parte inferior del mismo, pulsando el botón "Interpretación", se abre una ventana donde aparece una somera interpretación de los resultados de las 5 ejecuciones de ejemplo. En dicha imagen, los ejemplos están numerados de 1 a 5, correspondiendo a las ejecuciones de ejemplo mencionadas anteriormente.

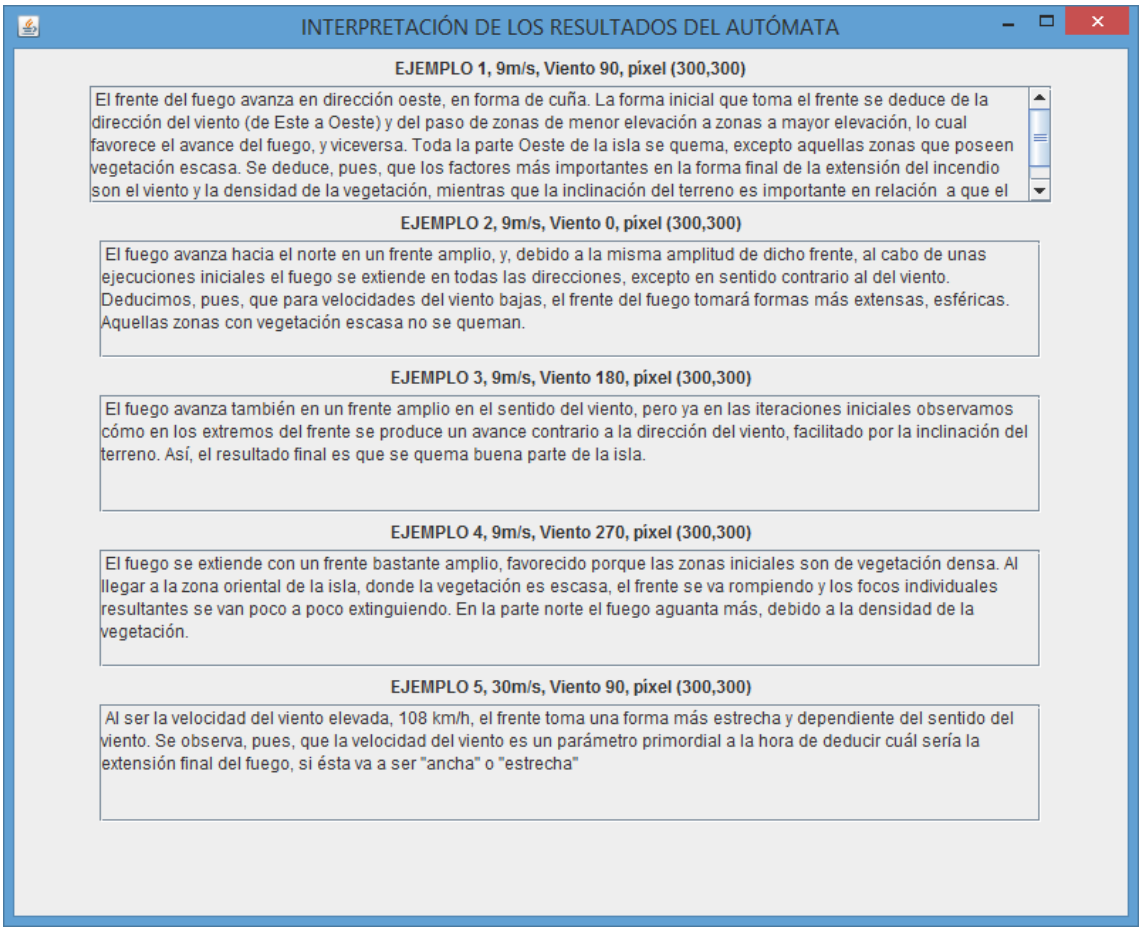


Figura 6. 30: Pestaña AUTOMATA, lado derecho, botón "Interpretación"

Para cerrar la aplicación, cierre la ventana principal de la misma, la que posee las pestañas.

6.9 TRAZA DE LOS RESULTADOS DEL AUTÓMATA

Indicamos finalmente que la mejor manera de seguir una traza de la ejecución del autómata sería el ejecutar el mismo desde un editor de Java. Utilizando el comando de Java `System.out.println`, en la consola del editor aparecerá información relativa a cada celda central que arde y a cada celda de su vecindario que se está estudiando.

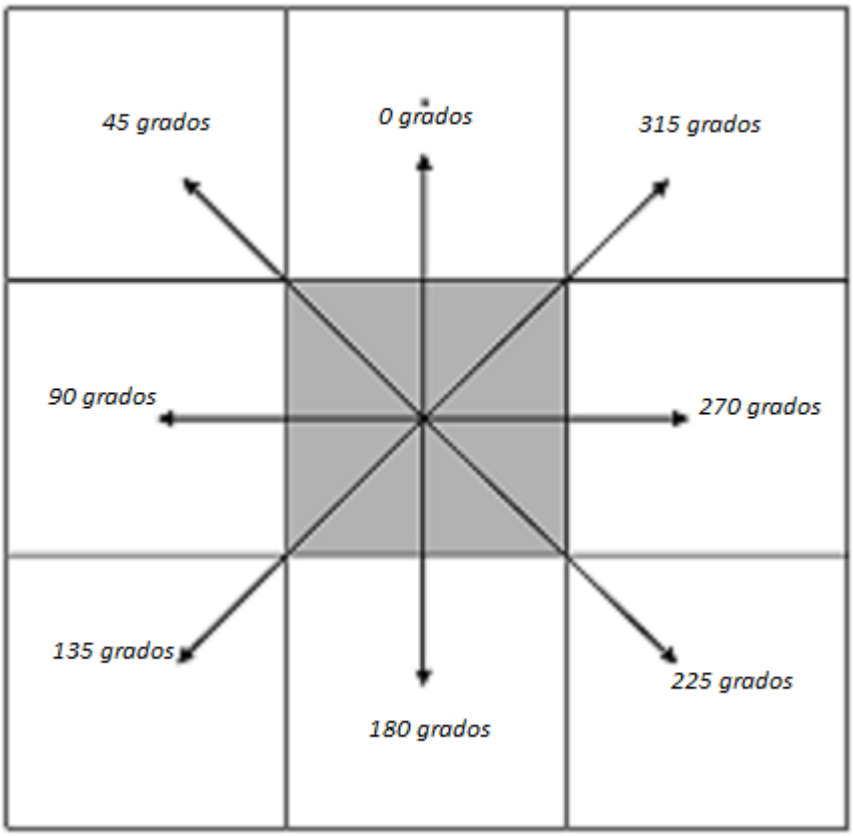


Figura 6. 31: Vecindario de Moore con direcciones de propagación

Para cada una de estas celdas, aparecerá la siguiente información sobre sus parámetros (véase la Sección 5.3):

1. Dirección de propagación: indica el ángulo que forma la celda central con respecto a dicha celda de su vecindario, medido en grados, considerando como 0 grados la celda en la posición superior a la celda central, 180 la inferior, 90 la de la izquierda, 270 la de la derecha, 45 la de la parte superior izquierda, 135 la de la parte inferior izquierda, 225 la de la parte inferior derecha y 315 la de la parte superior derecha, conforme a la Figura 6.31 del vecindario.
2. Theta, es decir, el ángulo formado entre la dirección del viento y la dirección de propagación del fuego de la celda central, la que arde, a su vecina.
3. FsubT: corresponde al cálculo parcial del parámetro P_w del viento, tal y como viene explicado en la Sección 3.2.7, donde FsubT equivale a f_t. FsubT también aparece en la imagen superior del lado derecho de la pestaña "VIENTO", siendo la segunda fórmula de dicha imagen.
4. Factor viento.
5. Factor densidad de la vegetación.
6. Factor tipo de la vegetación.
7. Ratio.
8. ThetaSlope.
9. Factor Slope.
10. Probabilidad

La celda vecina arderá o no en la siguiente iteración del autómata en función de la probabilidad obtenida, lo cual aparecerá igualmente indicado en la consola.

En la consola también se reflejarán los cambios realizados en la ventana de ejecución del autómata, como "Parar" el autómata, "Seguir" con su ejecución, y la modificación de la velocidad y la dirección del viento.

Si el usuario desea ejecutar la aplicación desde un editor de Java, lo único que debe hacer es importar el proyecto desde el editor de Java en cuestión.

Como ejemplo de la importación del proyecto, mostramos cómo hacerlo en el editor de Java Eclipse.

Una vez instalado Eclipse, hay que crear un nuevo proyecto, al que llamaremos, por ejemplo, Automata. Para ello, ejecutamos en Eclipse lo siguiente:

`File->New->Java Project`

En la ventana que se nos abre, damos el nombre al proyecto, y pulsamos `"Finish".` Tendremos lo siguiente:

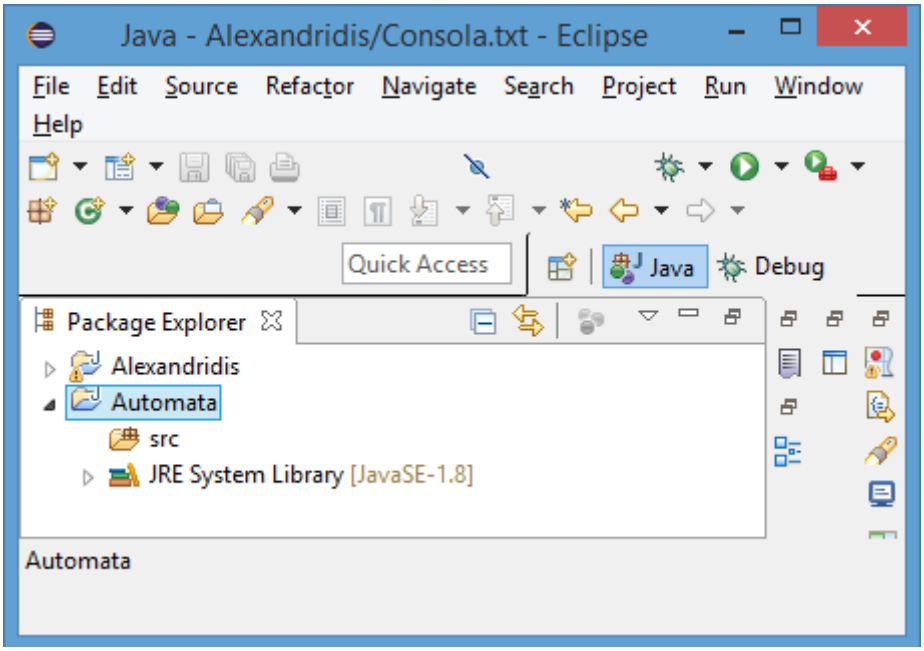


Figura 6. 32: Eclipse, nuevo proyecto

Seguidamente, pulsamos en el proyecto `"Automata"` con el botón derecho del ratón, y en el desplegable que nos sale pulsamos en `"Import".` Nos saldrá una ventanita:

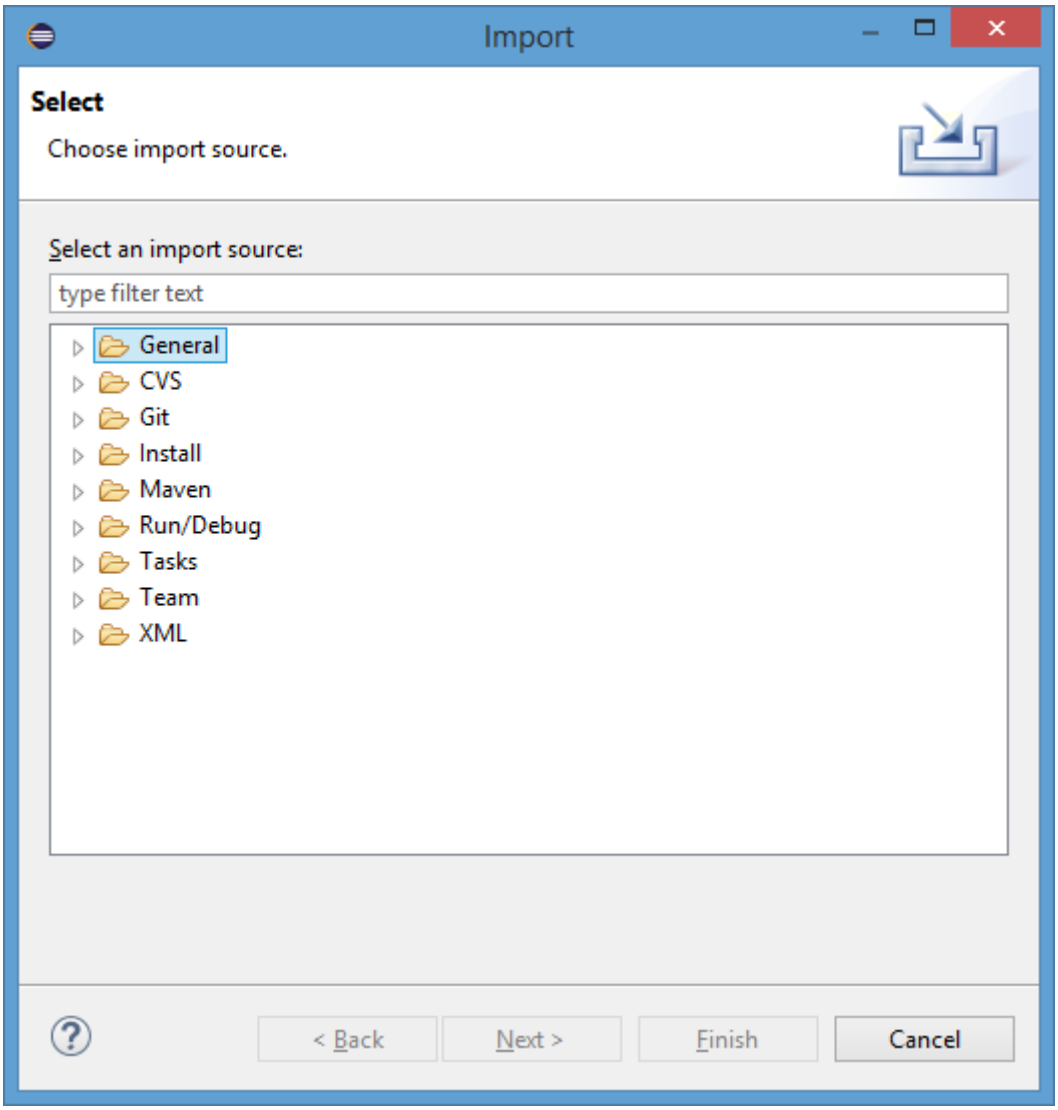


Figura 6. 33: Eclipse, ventana "import"

En dicha ventana, pulsamos en `"General"` y en las opciones resultantes en `"File system".` Nos saldrá una nueva ventana, en la cual nos pedirá la ruta al directorio raíz, donde dice `"From directory".` Allí, por medio de `"Browse"` seleccionaremos la carpeta que contiene la distribución, a la cual nosotros hemos llamado `"AlexandridisEtAlter".` Obtendremos la siguiente pantalla, en la que nosotros hemos seleccionado ya la casilla a la izquierda del nombre del directorio.

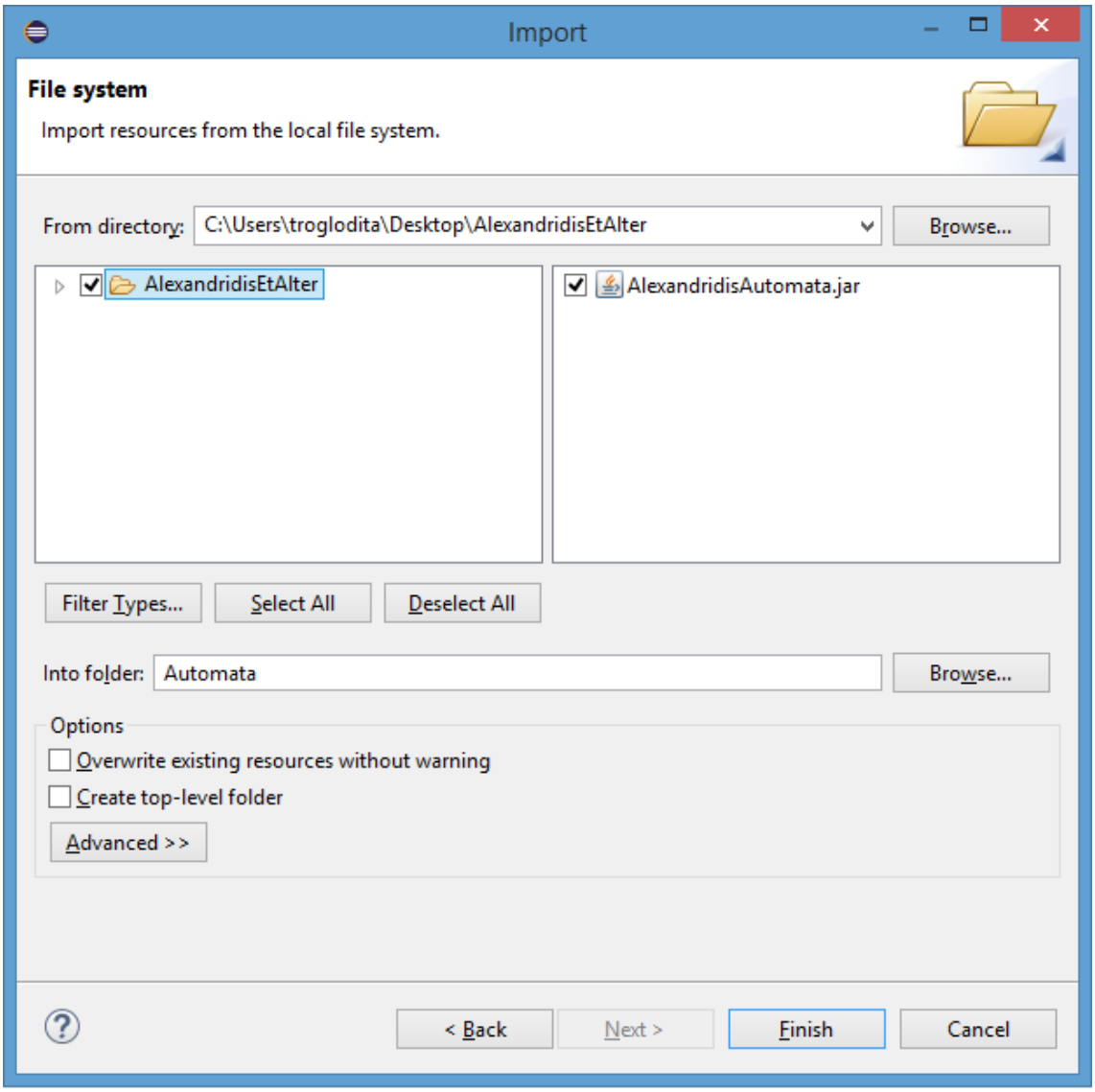


Figura 6. 34: Eclipse, ventana "import", "Browse"

Donde nos pide el directorio en el que hemos de colocar lo importado, le indicamos el nombre de nuestro nuevo proyecto, `"Automata".` Finalmente, pulsamos en `"Finish"` y ya tenemos el archivo importado. La estructura de los directorios será la siguiente o similar:

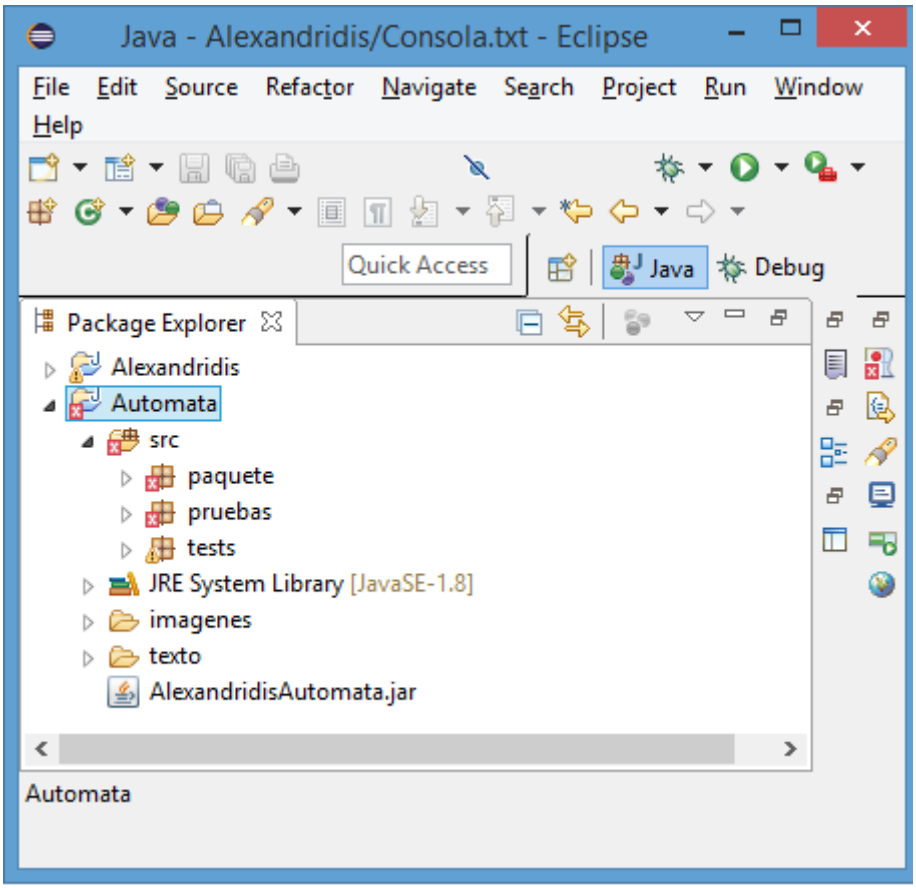


Figura 6. 35: Eclipse, estructura del proyecto

Sólo nos falta importar el paquete `"Java Advanced Image".` Nosotros le incluimos en la carpeta de la aplicación, jai-1_1_3-lib-windows-i586, aunque es evidente que no es nuestro, sino de Sun Microsystems. Para importar dicho paquete, primero es necesario instalarlo en la computadora (hablamos de la versión para Windows). Una vez instalado (lo hará, como regla general, en `"Archivos de Programa", "Sun Microsystems", "Java Advanced Imaging 1.1.3"`) tenemos que importar sus librerías en nuestro proyecto de Eclipse. Para ello, pulsamos en el proyecto con el botón derecho del ratón y, en el desplegable que nos sale, pulsamos en `"Build Path"` y, a continuación, en `"Configure Build Path".` Nos saldrá la siguiente ventana:

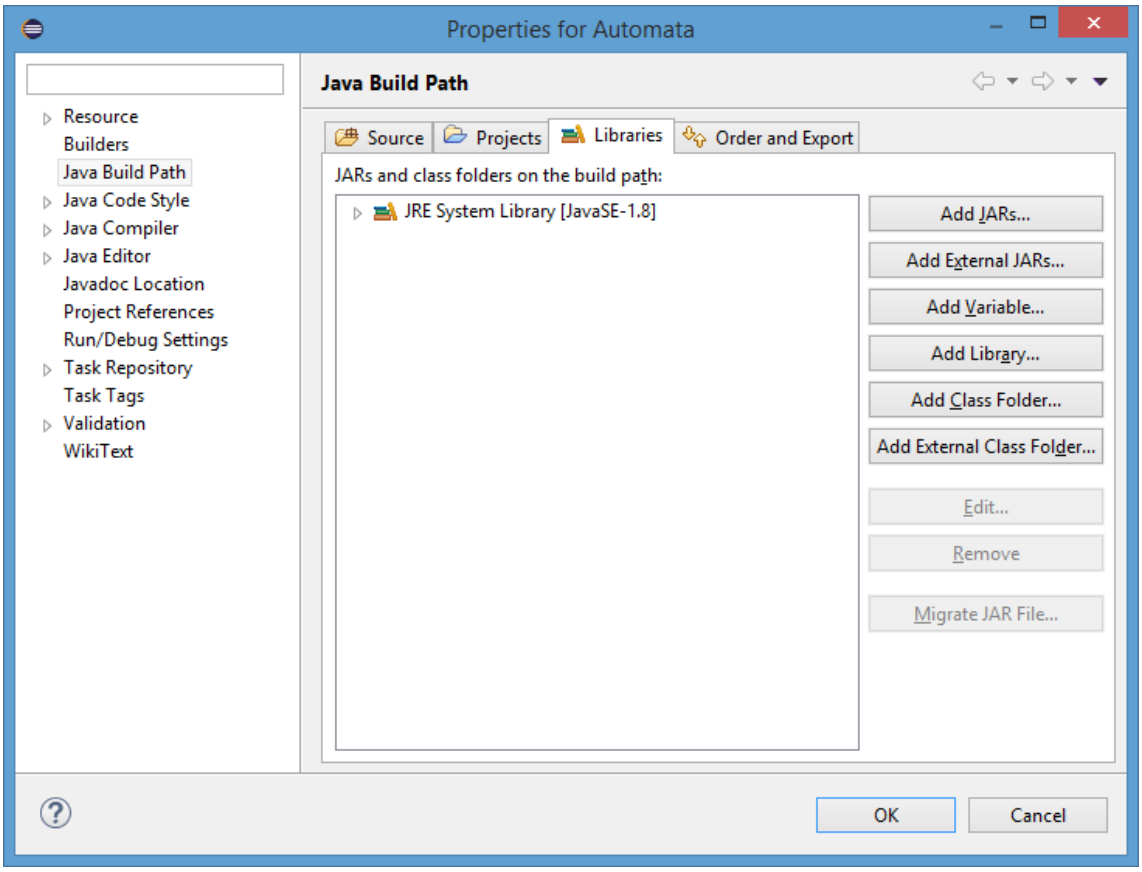


Figura 6. 36: Eclipse, "Build Path"

En ella, pulsamos, en la parte derecha, en el botón "Add External JARs...", y nos saldrá una ventana solicitándonos la ruta. Esta será algo parecido a "...\Archivos de Programa/Sun Microsystems\Java Advanced Imaging 1.1.3\lib". Nos saldrán 3 archivos jar, los cuales seleccionaremos,

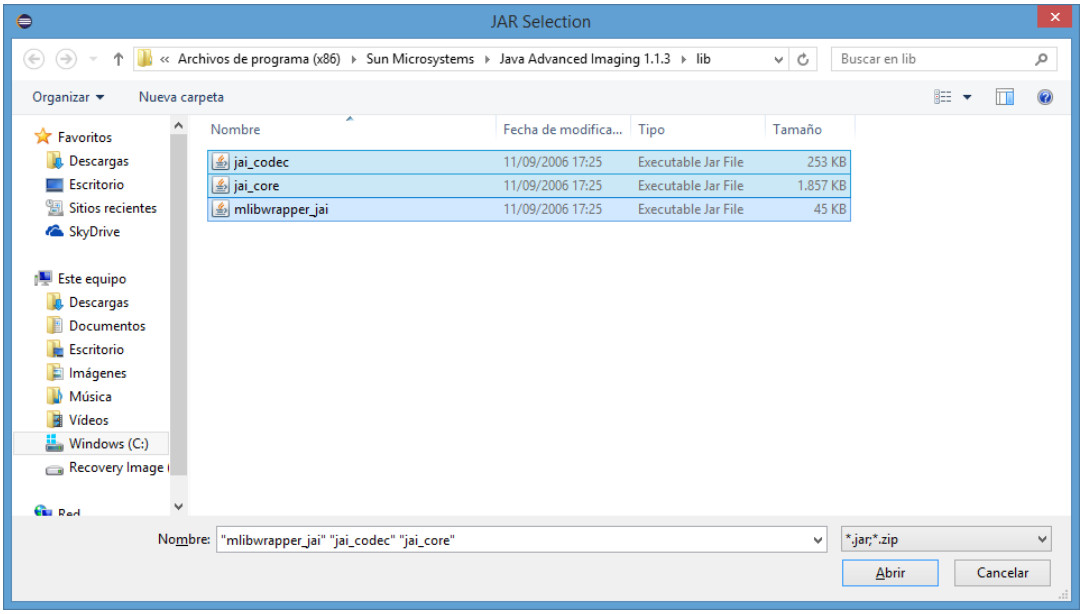


Figura 6. 37: Eclipse, "Add External JARs", ruta a JAI

Finalmente, pulsaremos `"Abrir"` y, con ello, ya tendremos la librería JAI importada en nuestro proyecto. Para comprobarlo, tendremos una estructura en el proyecto similar a la siguiente:

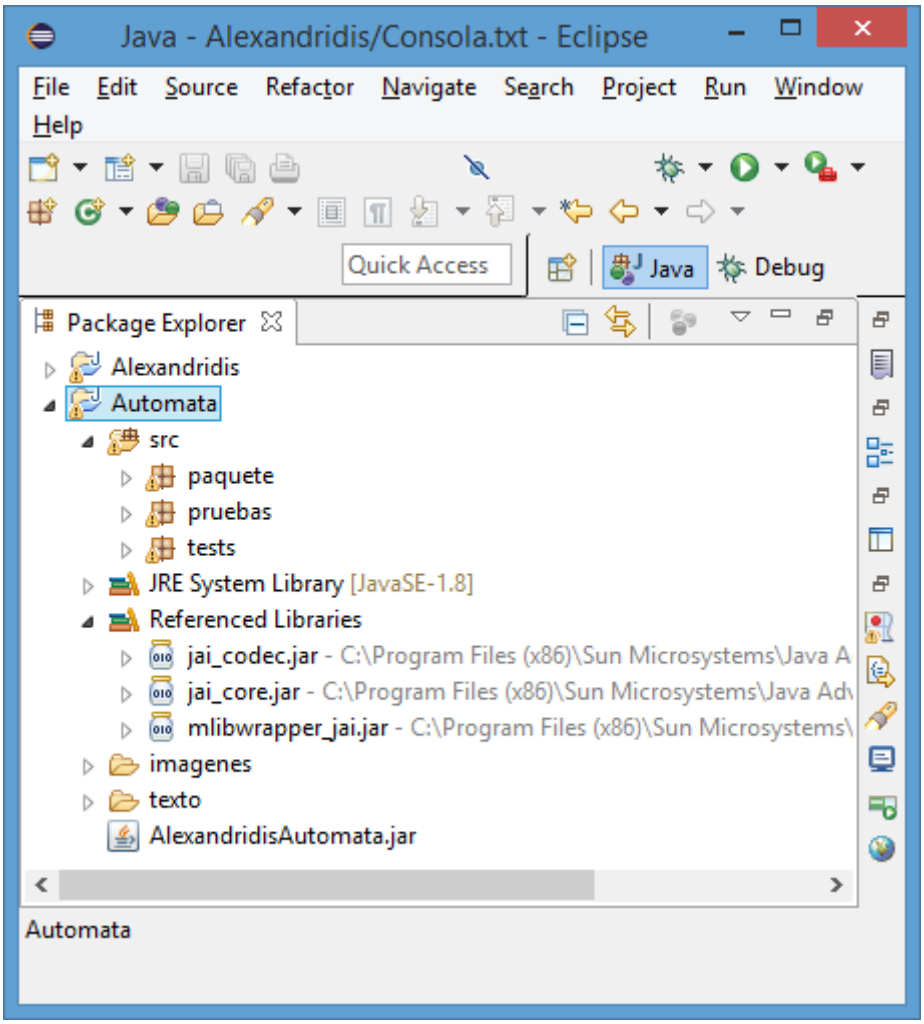


Figura 6. 38: Eclipse, estructura del proyecto con las librerías de JAI

En dicha imagen, observamos cómo tenemos importado los "jar" de JAI.

Ya estamos en condiciones de ejecutar el Autómata desde el editor de Eclipse. Para ello, pulsamos en el archivo de fuentes "paquete", el cual se ve en la imagen inmediatamente superior, y en dicho paquete, en el archivo "Automata". Una vez se nos abra la aplicación, iremos a la pestaña "AUTOMATA" y, una vez allí, podemos ejecutar nuestro propio autómata o uno de los ejemplos. En cualquier caso, podremos ver en la consola de Eclipse la información de la traza de ejecución del autómata, tal y como se ve en la siguiente imagen:

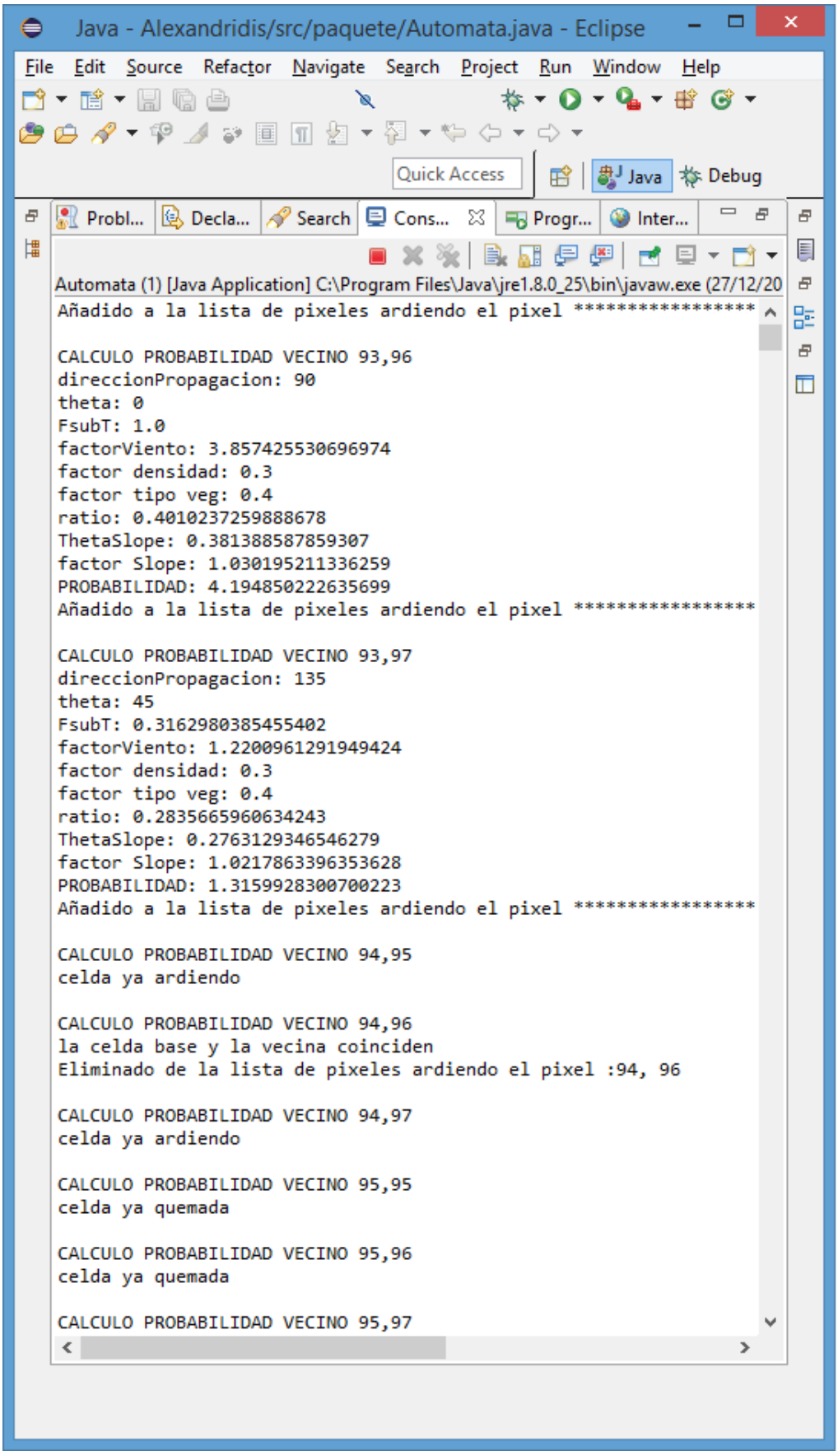


Figura 6. 39: Eclipse, traza de la ejecución del autómata

6.10 CONCLUSIONES

En este capítulo se ha descrito de forma exhaustiva el modo de empleo del simulador. A partir del paquete de la aplicación y siguiendo los pasos explicados en este manual, no debería de constituir ningún problema la ejecución del simulador, comenzando por la primera pestaña y terminando en la última, que es donde reside el núcleo de la aplicación, a saber, las ventanas de ejecución que permiten ejecutar las simulaciones del incendio propiamente dichas.

A fin de facilitar una traza de los resultados de la ejecución del autómata, a efectos de la verificación y validación del mismo, se ha añadido la Sección 6.9, donde se explica paso a paso la forma de ejecutar el autómata dentro del editor de Java Eclipse.

7 REPRESENTACIÓN EN NETLOGO

7.1 INTRODUCCIÓN

NetLogo es un entorno de programación para simular modelos de fenómenos físicos y sociales. Está orientado hacia los sistemas multiagente, donde cada agente es consciente de su entorno y puede interactuar con el mismo. Los agentes toman la forma de tortugas ("turtles"), parches ("patches"), enlaces entre ellos y la figura del observador ("observer") que es quien manipula el modelo. El programa se puede descargar de [Netlogo] y su instalación es sencilla.

En este capítulo realizaremos una simulación del modelo de Alexandridis et Alter utilizando como plataforma el entorno de NetLogo. Como la explicación del entorno de programación de NetLogo está fuera de los objetivos de este PFC, explicaremos la simulación realizada centrándonos, en primer lugar, en la interfaz de la aplicación realizada (a vez que explicamos su uso) para, después, explicar someramente el código de programación de la misma en relación a la simulación del autómata de Alexandridis et Alter. A continuación, realizaremos unas cuantas ejecuciones de prueba, para verificar los resultados obtenidos. Finalmente, compararemos nuestro modelo en NetLogo con las imágenes 7 y 8 del trabajo de Alexandridis et Alter.

En la carpeta "ProyectoNetLogo" está el ejecutable y los archivos de imágenes utilizados para la descripción en esta misma memoria de la implementación en NetLogo. Para poder abrir el archivo ejecutable, es necesario tener instalado NetLogo en la computadora (vea el archivo "instrucciones" adjunto en la carpeta).

7.2 INTERFAZ DE LA APLICACIÓN

NetLogo permite el empleo de botones, deslizadores, interruptores y selectores para controlar la ejecución del modelo. En nuestro caso, presentamos los siguientes elementos, ubicados a mano izquierda y de arriba a abajo en la interfaz gráfica:

1. Botón "setup": llama al comando del mismo nombre. Dicho comando, cuyo contenido explicaremos en la siguiente sección, donde examinaremos el código programado, sirve para cargar los parámetros iniciales del programa.
2. Botón "go": llama al comando del mismo nombre, el cual comienza la ejecución del simulador.
3. Botón "ver imagen elevación": permite ver la imagen que contiene, en colores RGB, los datos de la elevación del terreno de la isla de Spetses.

4. Botón "ver imagen tipo vegetación": permite ver la imagen que contiene, en colores RGB, los datos relativos al tipo de la vegetación, conforme a el modelo de Alexandridis et Alter.
5. Botón "ver imagen densidad vegetación": permite ver la imagen que contiene, en colores RGB, los datos relativos a la densidad de la vegetación.
6. Deslizador "velocidadViento": permite al usuario introducir, en el rango de 0 a 30 m/s, la velocidad del viento.
7. Deslizador "direccionViento": permite al usuario introducir, en el rango de 0 a 359 grados, la dirección del viento.
8. Interruptor "densidad": permite al usuario elegir entre dos imágenes de entrada para la densidad de la vegetación. Explicaremos ésto un poco más adelante.
9. Deslizador "pixelX": permite al usuario introducir, en el rango 1 a 422, la posición inicial del píxel X del foco de ignición del fuego.
10. Deslizador "pixelY": permite al usuario introducir, en el rango 1 a 400, la posición inicial del píxel Y del foco de ignición del fuego.

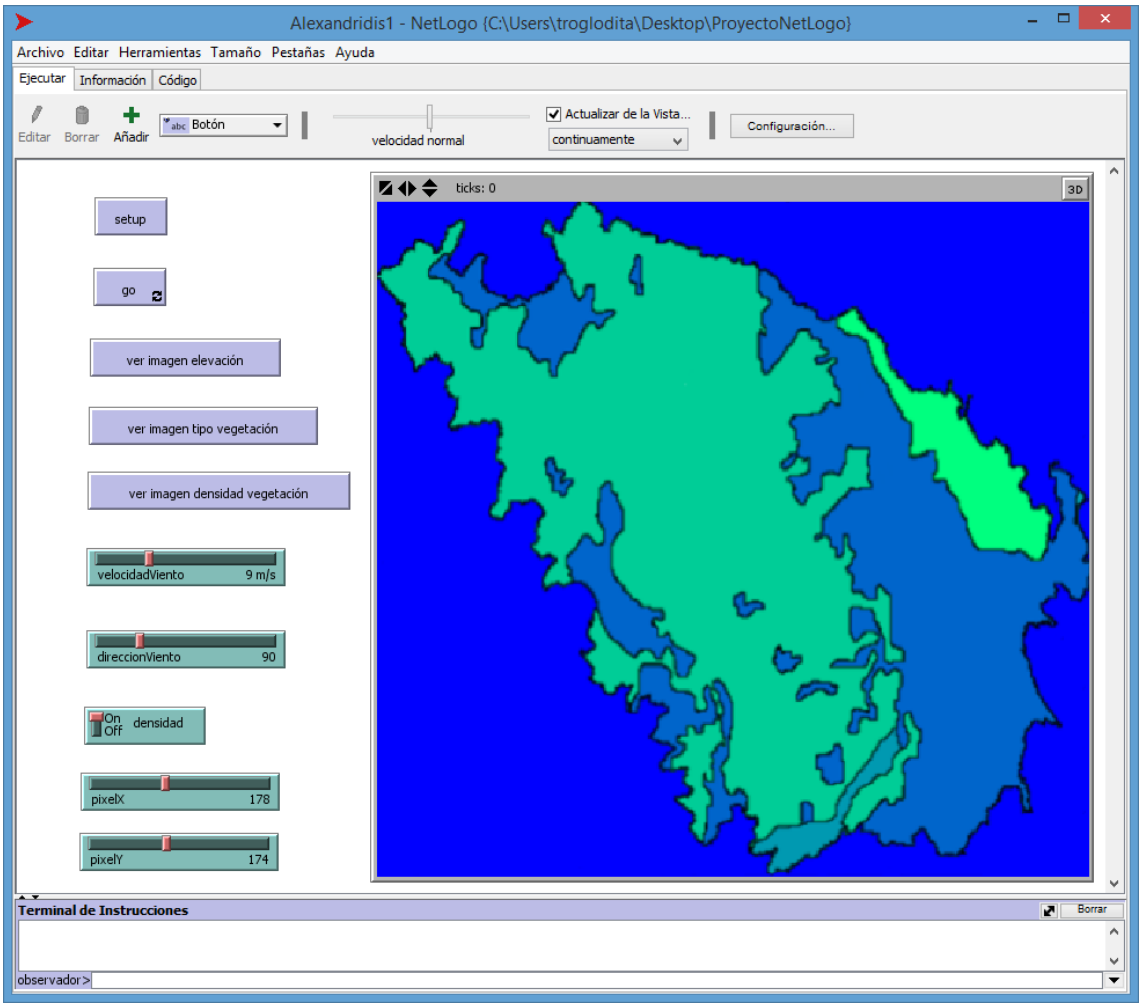


Figura 7. 1: Interfaz de NetLogo

En el lado derecho de la interfaz se ubica la pantalla o escenario donde se recogerá el resultado gráfico de la ejecución de la simulación, llamada en la

jerga de NetLogo "Mundo". Dicha pantalla se configura por medio del botón predefinido "Configuración", situado en la tercera barra de tareas a mano derecha, tal y como aparece en la imagen anterior. Al pulsar dicho botón, se abre una ventana que permite configurar dicho elemento:

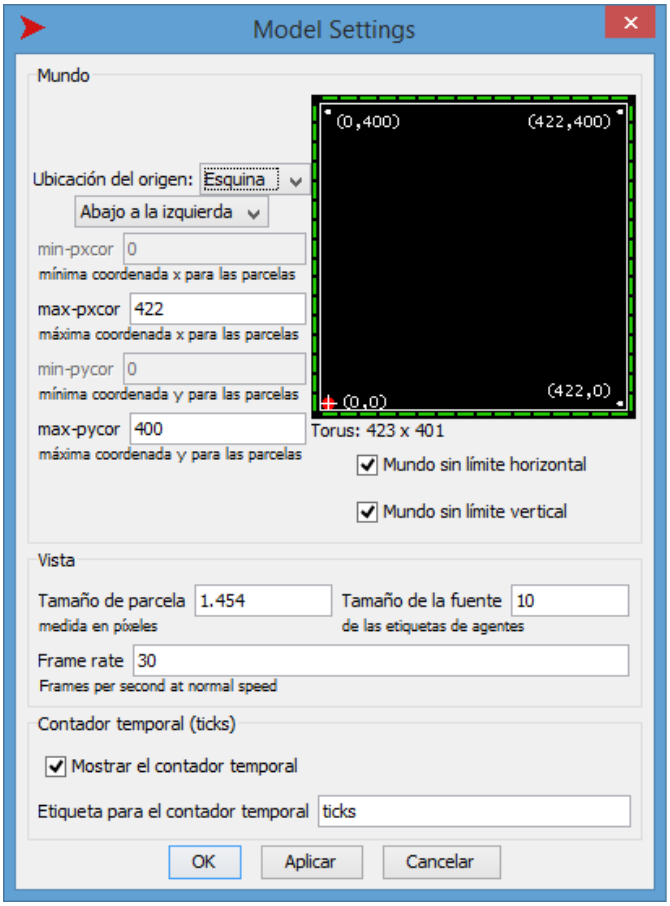


Figura 7. 2: Escenario o "Mundo"

Existen varias formas de configurar el "Mundo", tal y como se deduce de la Figura 7.2. En nuestro caso particular, hemos creado un mundo bidimensional de 422x400 parches ("patches"). Los patches equivalen a las celdas, con lo que nuestro autómata celular tendrá unas dimensiones de 422x400 celdas. 422 es la dimensión horizontal, y 400 es la vertical. Las coordenadas horizontales se leen en el eje de abscisas, de izquierda a derecha, y las verticales en el eje de ordenadas, de abajo a arriba. Tenemos, pues, una parrilla bidimensional de celdas algo diferente de la presentada en el caso de la simulación en Java.

En la parte inferior de la pantalla se muestra el "Terminal de Instrucciones", el cual sirve tanto para que el "observador" introduzca instrucciones fuera del código como para que se muestren resultados intermedios de texto conforme se va ejecutando el programa, a la manera de una consola de un editor de un lenguaje de programación. A partir de aquí, nos referiremos a dicho elemento como la consola.

Veamos ahora el uso de la interfaz gráfica.

Si pulsamos el botón "ver imagen elevación", se muestra, en el Mundo, la imagen RGB con los datos de la elevación. Es la misma imagen utilizada en nuestro desarrollo en Java, pero allí cambiamos los colores a tonos rojos, y aquí la presentamos en los tonos verdes originales:

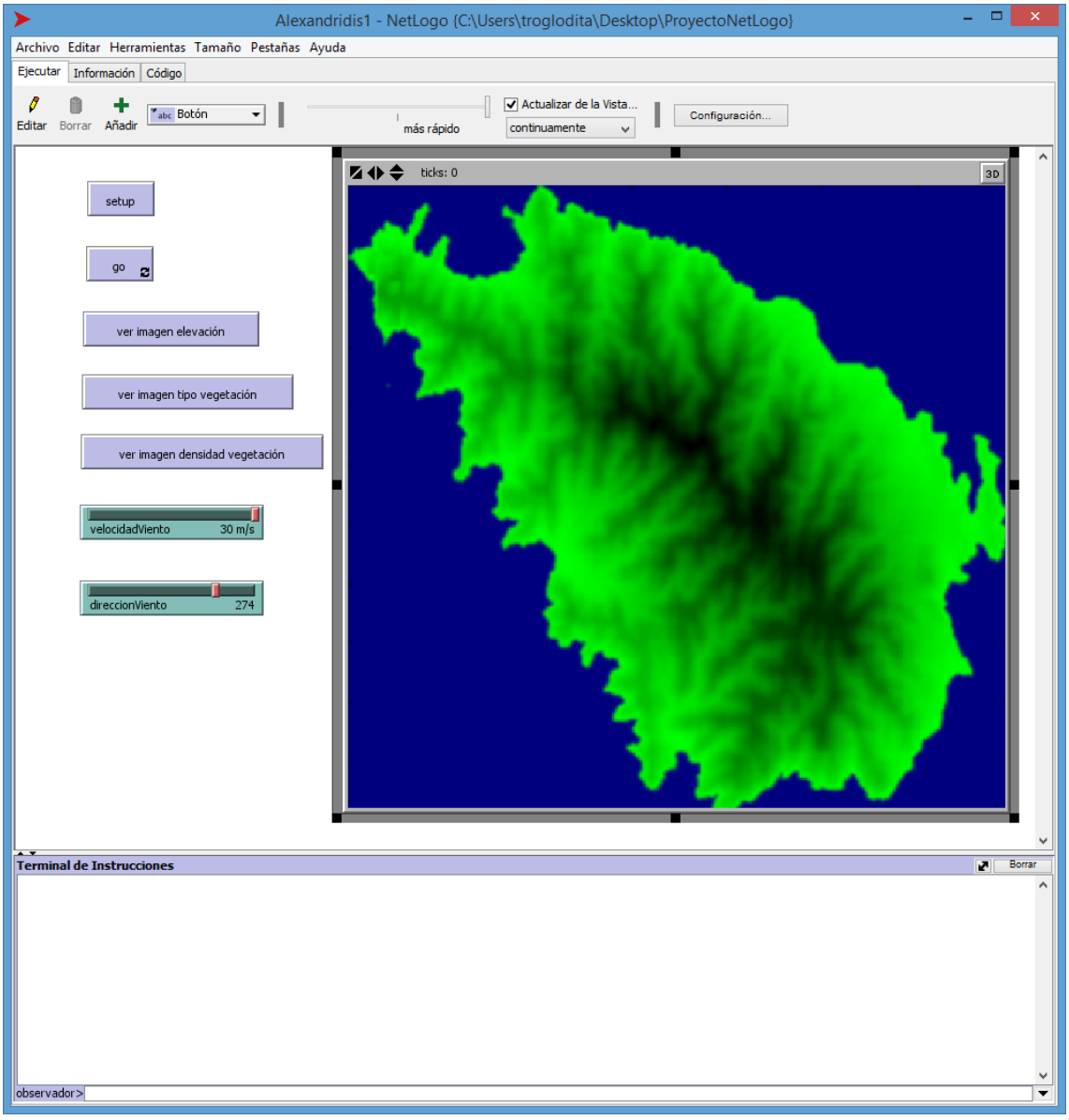


Figura 7. 3: Botón "ver imagen elevación"

Posteriormente, en la siguiente sección, explicaremos el código necesario para cargar dicha imagen.

Al pulsar el botón "ver imagen tipo vegetación", se muestra en el mundo la imagen RGB con los datos del tipo de la vegetación en el modelo de Alexandridis et Alter.

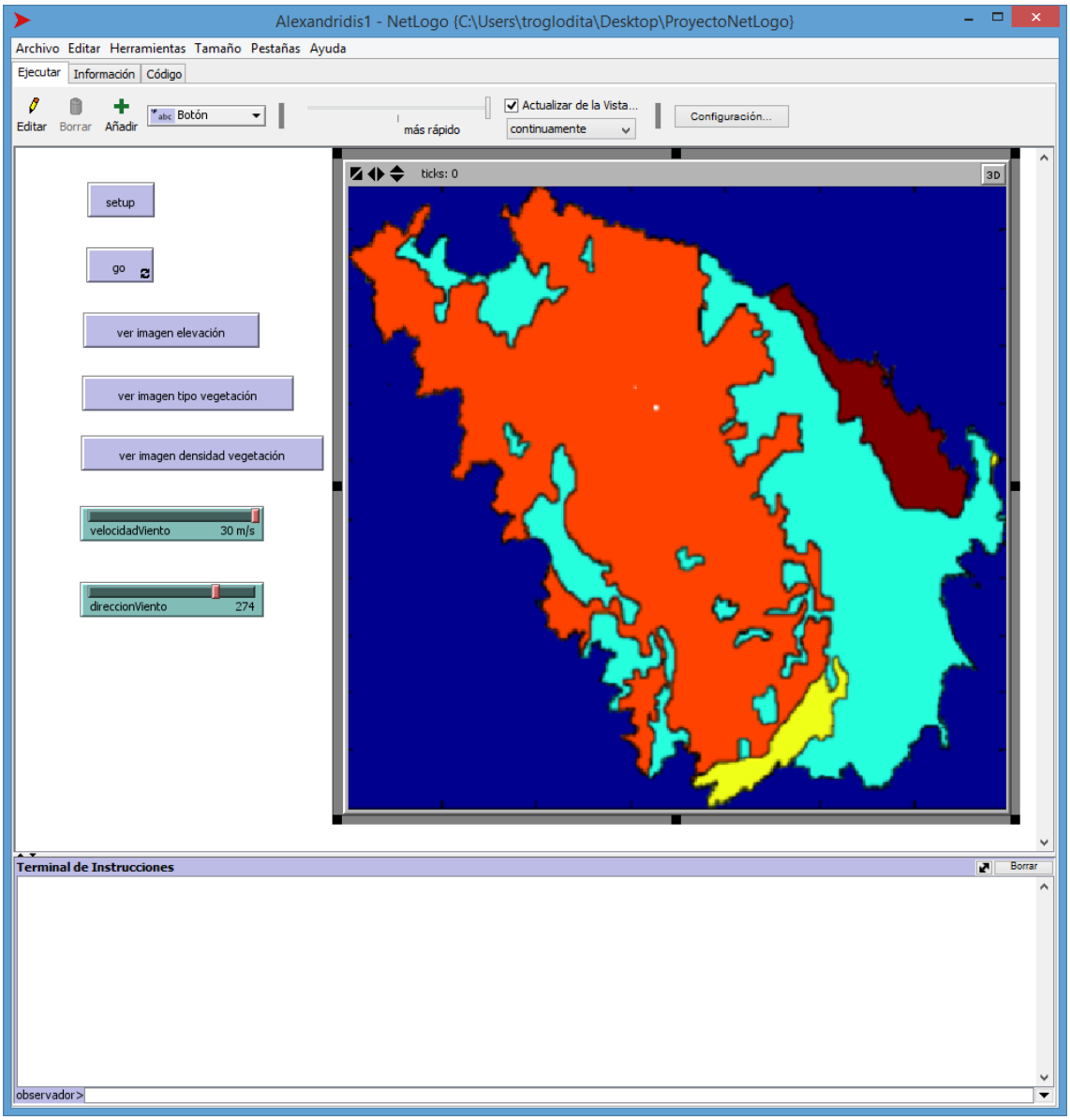


Figura 7. 4: Botón "ver imagen tipo vegetación"

Finalmente, al pulsar "ver imagen densidad vegetación" se abre la imagen RGB con los datos de la densidad de la vegetación. Este botón está relacionado con el interruptor "densidad". Dicho interruptor se maneja con el ratón. Si el interruptor está en "Off", al pulsar el botón "ver imagen densidad vegetación" se mostrará la imagen tratada con el algoritmo de clasificación de imágenes de la Distancia Mínima (véase la Sección 4.3.6).

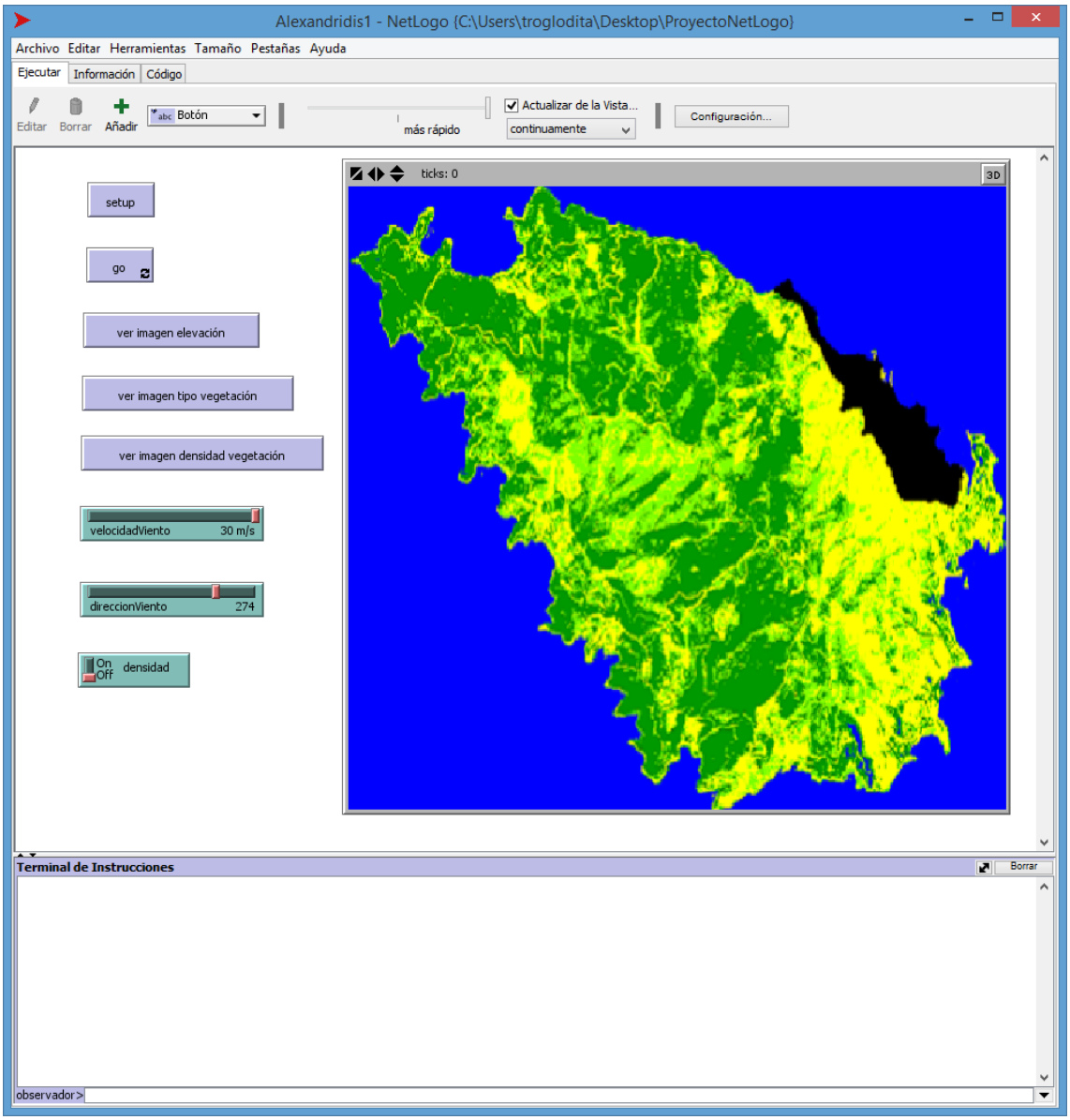


Figura 7. 5: Botón "ver imagen densidad vegetación", interruptor en "Off"

Si el interruptor está en "On" se mostrará la imagen de la densidad de la vegetación extraída del trabajo de Alexandridis et Alter, cuyo tratamiento digital es inferior al de la imagen clasificada con la Distancia Mínima.

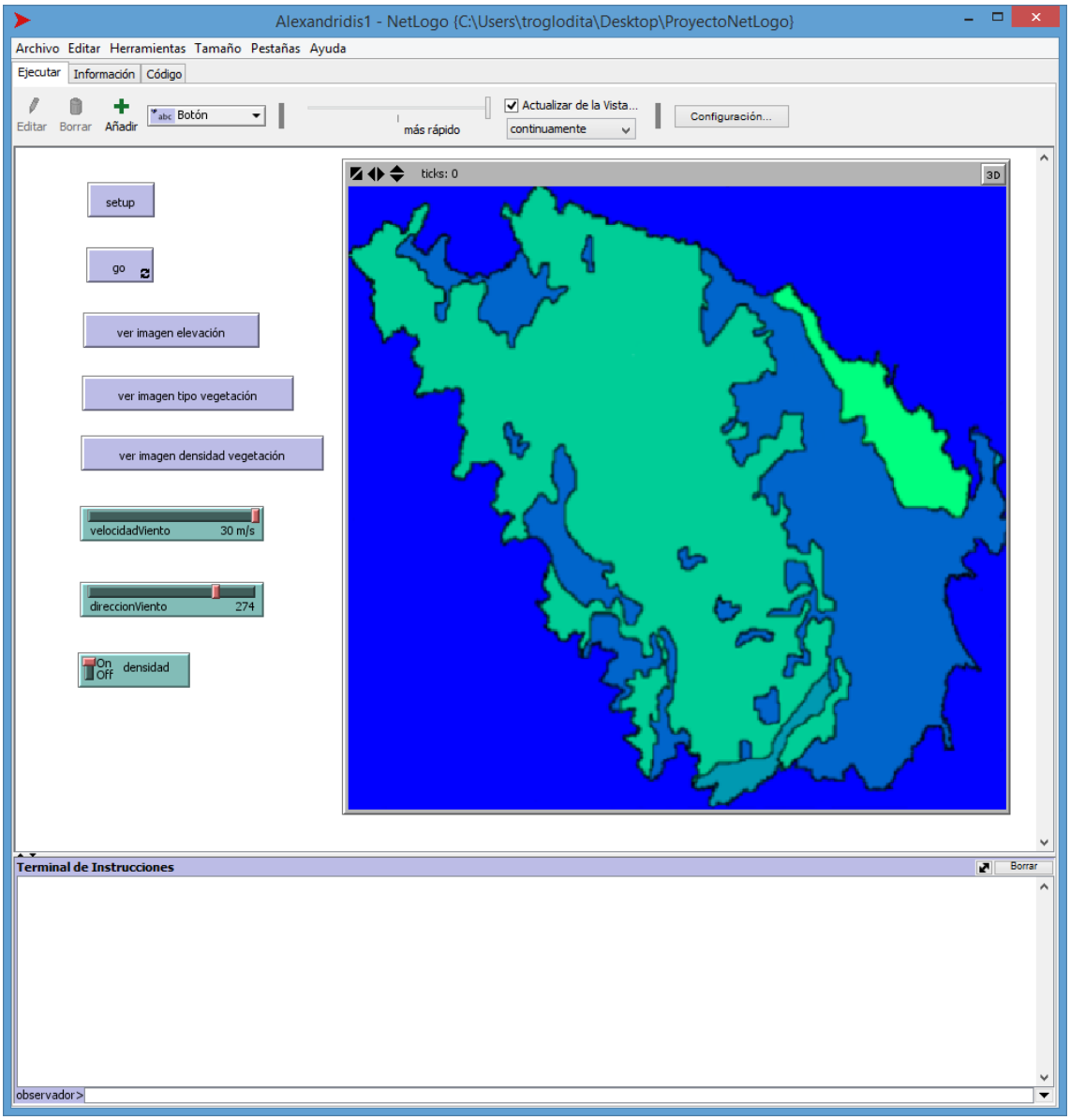


Figura 7. 6: Botón "ver imagen densidad vegetación", interruptor en "On"

Veamos ahora cómo se pone en marcha la aplicación. Al pulsar el botón "setup" se cargan en el Mundo sucesivamente las imágenes de la elevación y de la densidad de la isla de Spetses (la imagen de la densidad que se cargará será la seleccionada con el interruptor "densidad"). A continuación, aparecerá, sobre la imagen de la densidad un puntito rojo en el parche que contiene el foco inicial del incendio.

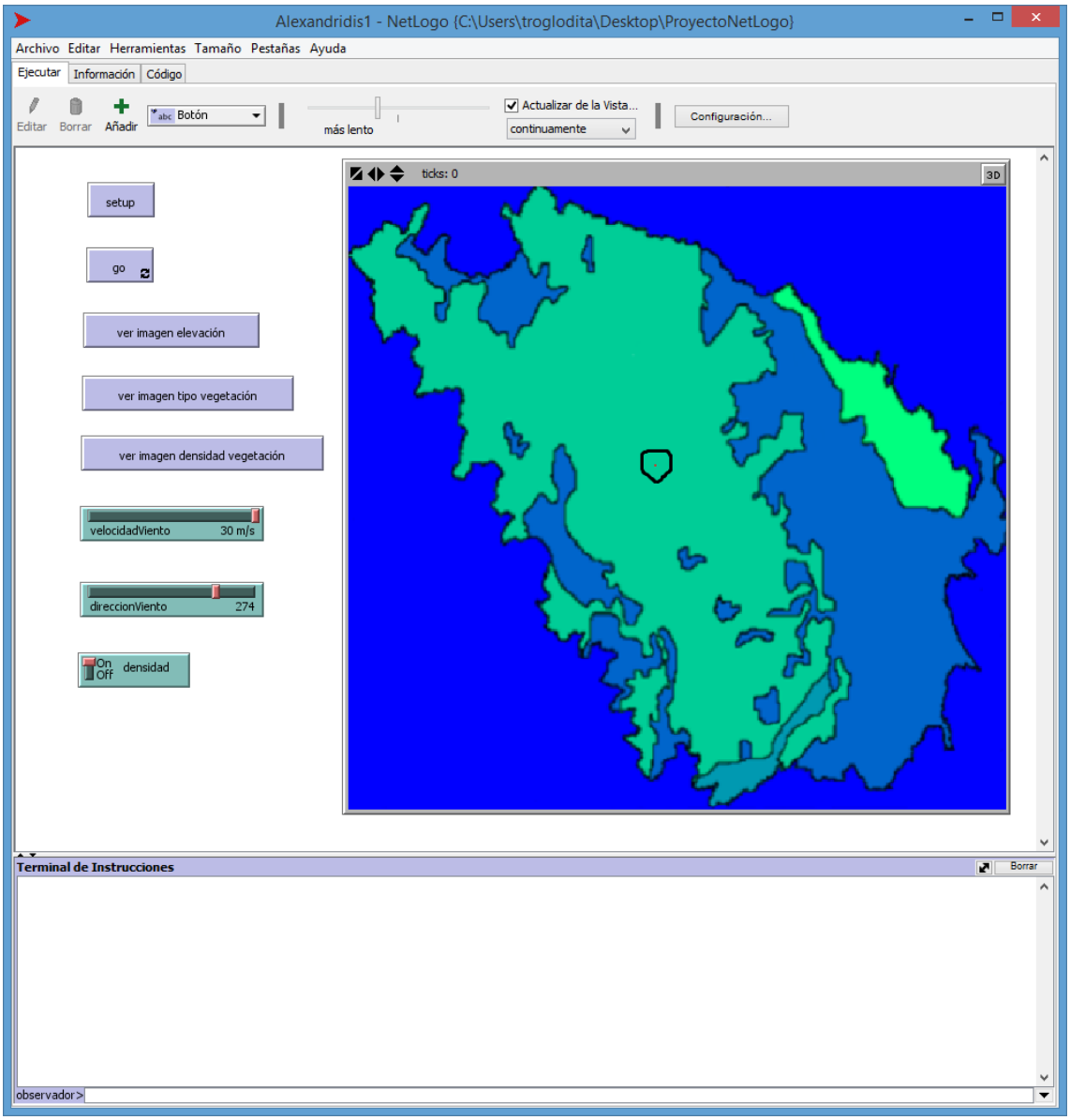


Figura 7. 7: Botón "setup", interruptor en "On"

Dicho punto, en esta ejecución particular, se aprecia en el Mundo aproximadamente en el centro de la isla. En las imágenes anterior y siguiente lo hemos resaltado en un perímetro negro, para los dos tipos de imágenes de densidad posibles.

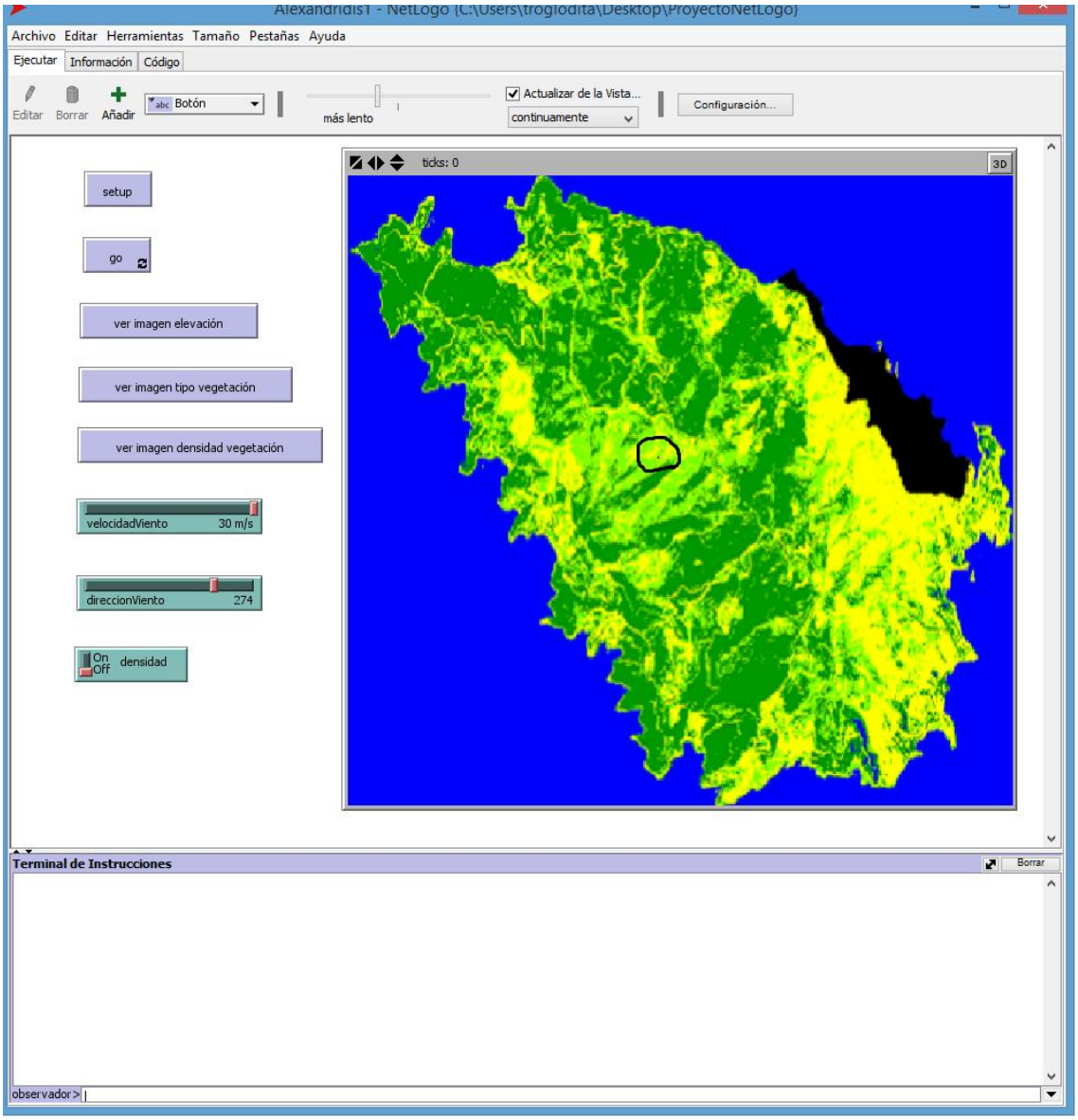


Figura 7. 8: Botón "setup", interruptor en "Off"

Una vez que se ha cargado dicha imagen, pulsaremos el botón "go" para poner en marcha la aplicación. En la consola aparecerán, si así se codifica especialmente, los cálculos matemáticos necesarios para comprobar el buen funcionamiento del código. Mostramos a continuación en la Figura 7.9 una imagen de una ejecución en curso. En el programa que incluímos en el paquete hemos eliminado la salida de resultados por la consola, porque ralentizaba en exceso la ejecución del programa.

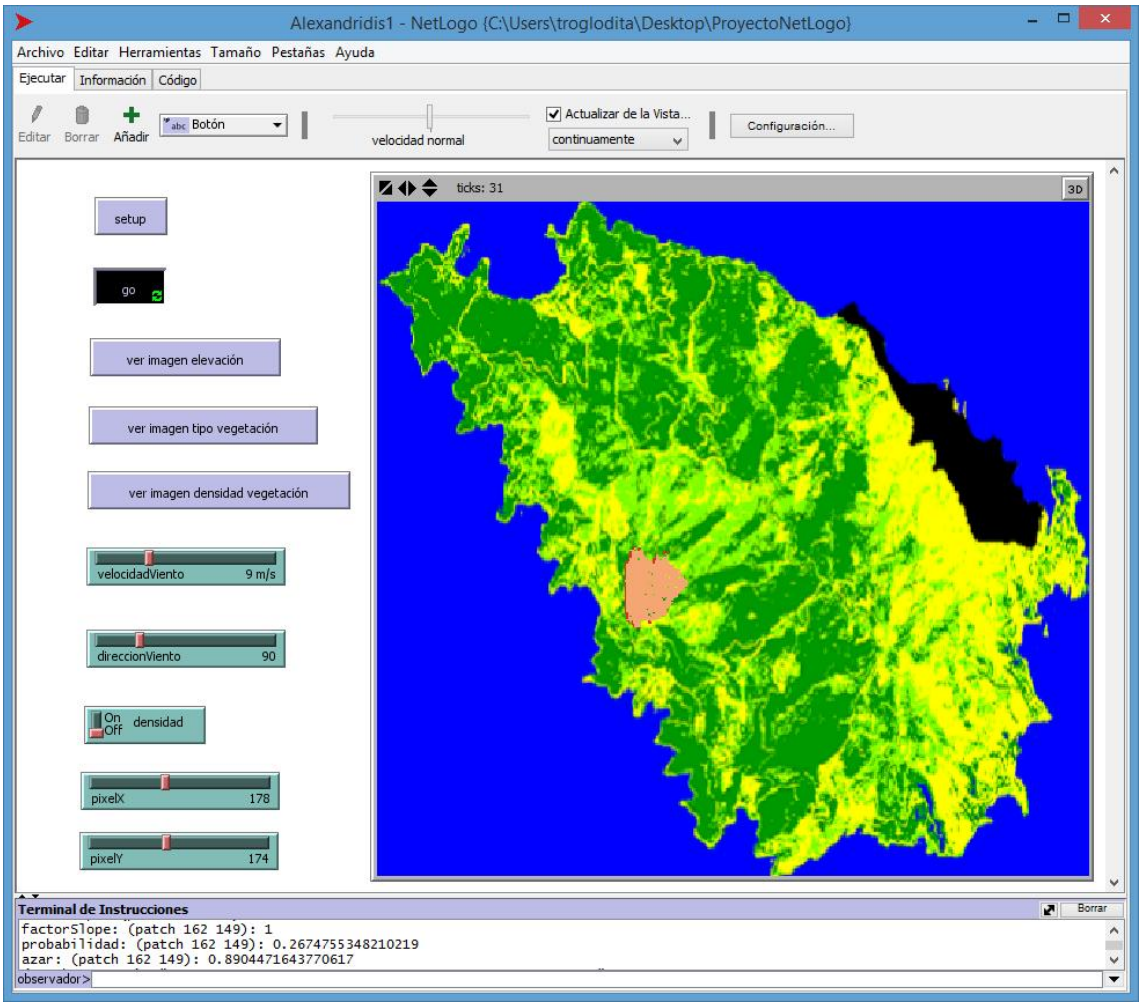


Figura 7. 9: Botón "go", ejecución

7.3 CÓDIGO DEL PROGRAMA

A continuación, iremos explicando, paso a paso, el código del programa. Para poder ver el código, basta con pulsar en la segunda barra de tareas en la pestaña "Código".

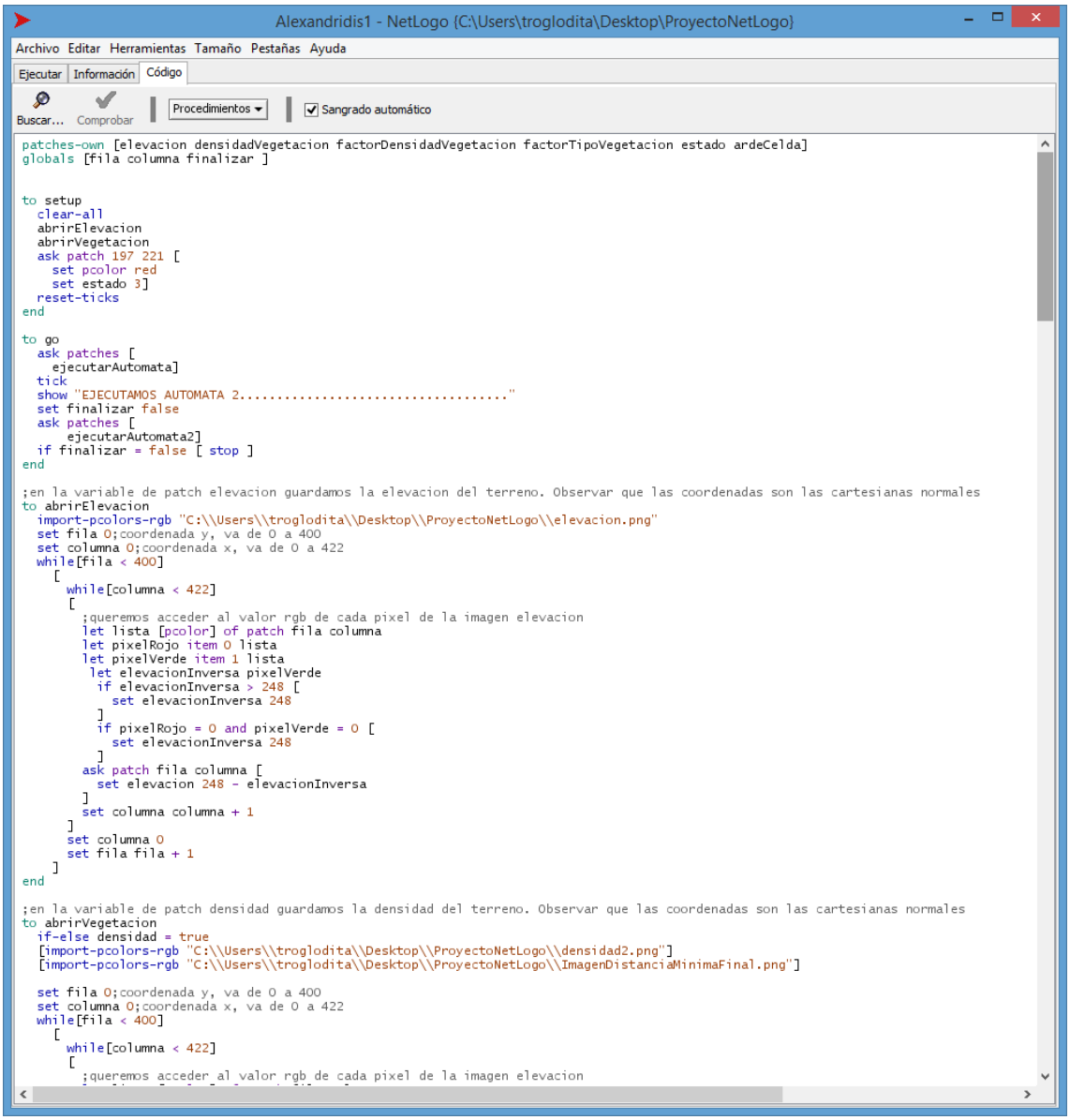


Figura 7. 10: Pestaña "Código"

Como hemos indicado anteriormente, el escenario (Mundo) se divide en celdas llamadas patches (en adelante, parches). Las tortugas (agentes) se sitúan sobre dichos parches. Las tortugas pueden interactuar con otras tortugas y, además, son conscientes de las características del parche en el que se hallan en cada momento.

A fin de simular un autómata celular en tal escenario, puesto que no vamos a tener agentes móviles que se muevan de un lado al otro de dicho escenario, sino que son simplemente los parches los que cambian sus características en cada iteración, vamos a prescindir de las tortugas.

7.3.1 VARIABLES GLOBALES Y VARIABLES DE PARCHE (PATCHES-OWN)

El programa en NetLogo comienza con la declaración de las variables. Estas pueden ser globales o privativas de cada parche o celda.

```
patches-own[elevacion densidadVegetacion factorDensidadVegetacion
factorTipoVegetacion estado ardeCelda]

globals [fila columna finalizar ]
```

Código 7. 1: Extracto de código en NetLogo. Variables

Así, las variables globales son únicas, mientras que las variables "patches-own" son privativas de cada parche, pero tienen idéntico nombre. Explicaremos después esta característica. Después de las variables, vienen una serie de procedimientos, que explicaremos en el mismo orden en que aparecen en el código.

7.3.2 PROCEDIMIENTO SETUP

```
to setup
  clear-all
  abrirElevacion
  abrirVegetacion
  ask patch pixelX pixelY [
    set pcolor red
    set estado 3]
  reset-ticks
end
```

Código 7. 2: Extracto de código en NetLogo. Procedimiento "setup"

Como indicamos cuando describimos el botón "setup", éste carga los parámetros iniciales de la aplicación. La primera línea, `clear-all`, reinicia todas las variables, las dos siguientes, `abrirElevacion` y `abrirVegetacion` ejecutan los procedimientos de dichos nombres, los cuales cargan, respectivamente, las imágenes con los datos RGB de la elevación y de la densidad de la vegetación en la isla de Spetses. Las imágenes se escalan al tamaño del escenario (Mundo), de forma que cada parche contendrá dos listas, una con los colores RGB correspondientes a la elevación, y otra con los correspondientes a la densidad. Esto puede verificarse pulsando sobre un punto de la imagen final (la de la densidad) con el botón derecho del ratón. Se abrirá un menú desplegable, tal y como vemos en la Figura 7.11:

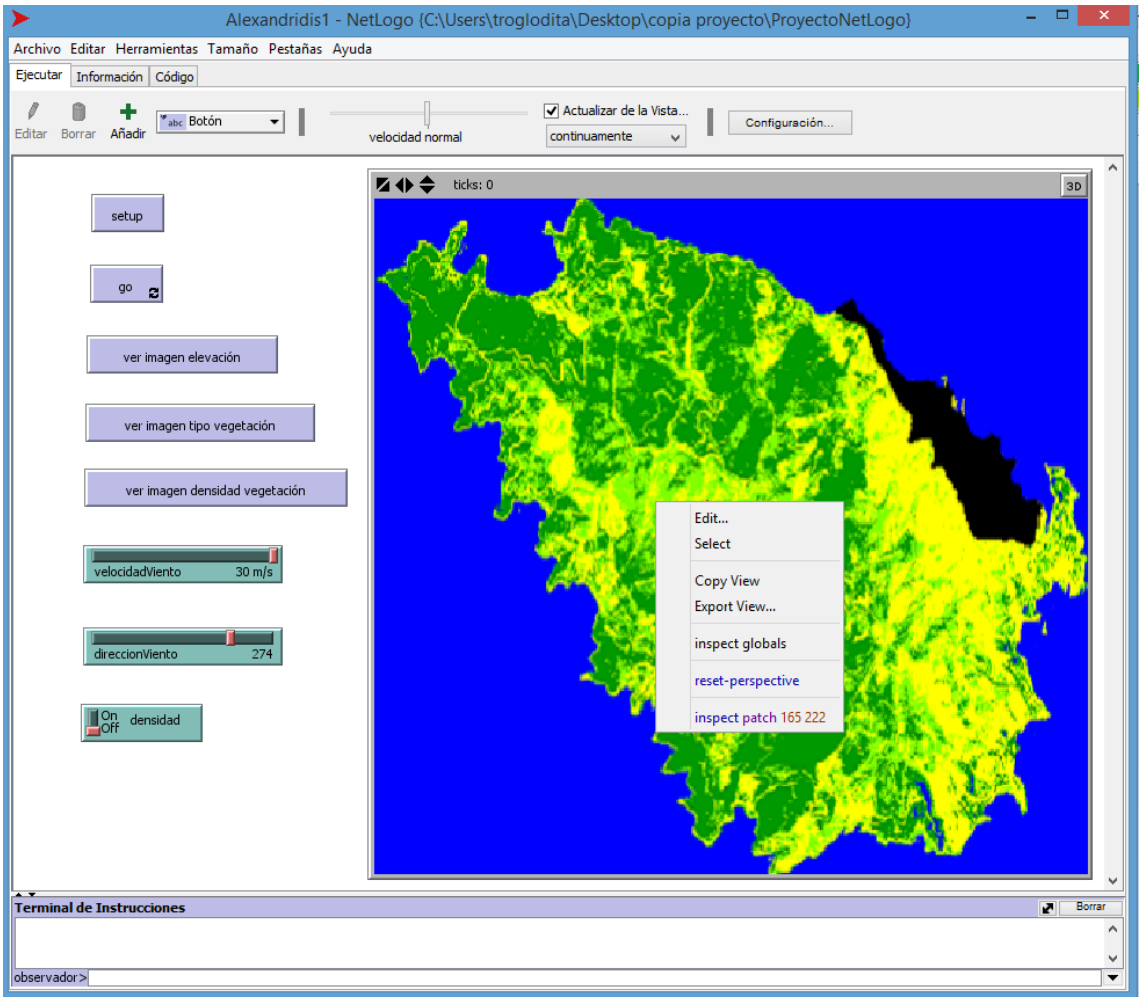


Figura 7. 11: Menú desplegable al pulsar en el escenario con el botón derecho del ratón

Si pulsamos en el punto "inspect patch xxx xxx" obtendremos una ventana emergente con los datos particulares del parche en cuestión. En dicha imagen observamos, entre otras, las siguientes variables del parche:

1. Pxcor: su coordenada x en el escenario.
2. Pycor: su coordenada y en el escenario.
3. Pcolor: sus valores RGB.
4. Elevacion: el valor de su variable de parche "elevacion", el cual se ha obtenido en el procedimiento "abrirElevacion".
5. DensidadVegetacion: el valor de su variable de parche "densidadVegetacion", el cual se ha obtenido en el procedimiento "abrirVegetacion". Este valor puede ser "mar", "escasa", "intermedia", "densa" o "urbana".
6. FactorDensidadVegetacion: el valor de su variable de parche "factorDensidadVegetacion", el cual se ha obtenido en el procedimiento "abrirVegetacion". Este valor es el valor numérico correspondiente a la densidad de la vegetación, conforme viene explicado en la Sección 3.2.6.
7. FactorTipoVegetacion: el valor de su variable de parche "factorTipoVegetacion", el cual se ha obtenido en el procedimiento

"abrirVegetacion". Este valor es el valor numérico correspondiente al tipo de la vegetación, conforme viene explicado en la Sección 3.2.5.

8. Estado: indica el estado de la celda, conforme se explica en la sección 3.2.2.
9. ArdeCelda: variable booleana que indica si la celda está ardiendo o no.

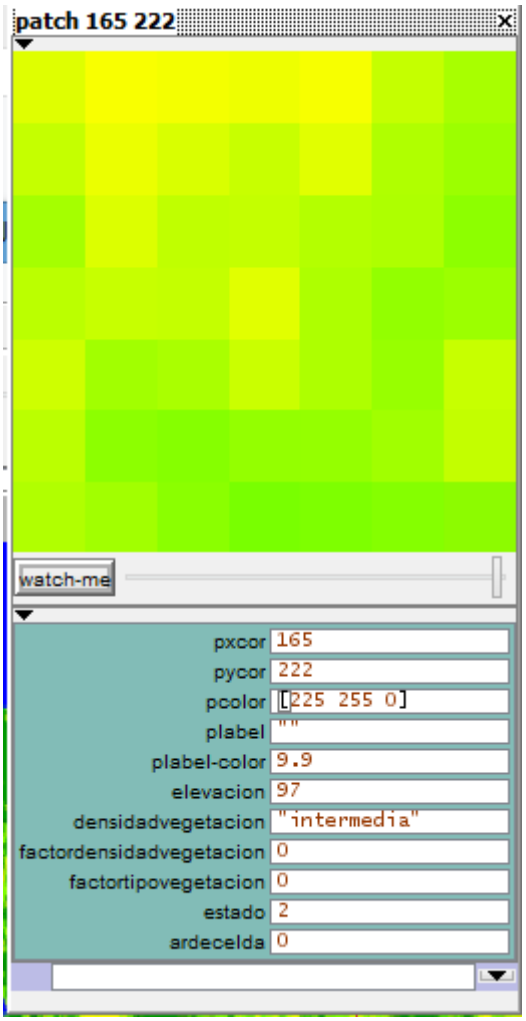


Figura 7. 12: Menú desplegable al seleccionar "inspect patch ..."

Continuando con el procedimiento "setup", vemos que se asignan los valores píxel X y píxel Y del parche que actuará de foco inicial del incendio. Esos valores son los introducidos en los deslizadores de la interfaz gráfica "pixelX" y "pixelY" respectivamente. Después, se cambia su color a rojo, con `set pcolor red` y `set estado 3`. Éste será el punto rojo que resaltamos en Figuras 7.7 y 7.8. La última línea, `reset-ticks`, reinicia los pasos del simulador a cero.

7.3.3 PROCEDIMIENTO GO

```
to go
  ask patches [ejecutarAutomata]
  tick
  show "EJECUTAMOS AUTOMATA 2.................................."
```

```
  set continuar false
  ask patches [ejecutarAutomata2]
  if continuar = false [ stop ]
end
```

Código 7. 3: Extracto de código en NetLogo. Procedimiento "go"

La primera línea `ask patches [ejecutarAutomata]` ordena a cada parche que ejecute el procedimiento "ejecutarAutomata", el cual veremos posteriormente.

La segunda línea, `tick` , indica que se dé un paso en la iteración.

La tercera línea muestra información en la consola.

La cuarta línea, `set continuar false`, es una variable de control para que la simulación termine si ya no hay más parches ardiendo.

La quinta línea, `ask patches [ejecutarAutomata2]` ordena a cada parche que ejecute el procedimiento "ejecutarAutomata2", el cual veremos posteriormente.

La última línea se corresponde con la cuarta, y se verá posteriormente cómo la variable de control "continuar" es modificada en el interior del procedimiento "ejecutarAutomata2".

En este procedimiento reside el funcionamiento del autómata celular en cuanto al recorrido de las celdas del mismo. Al ejecutar un comando "ask patches" todos los parches, es decir, todas las celdas del autómata ejecutan, cada una de ellas, el procedimiento que se indique. Así, en el procedimiento "go", los parches ejecutan primero el procedimiento "ejecutarAutomata", que es el que determina, para cada celda del autómata, cuál será su estado al terminar el recorrido por todas las celdas del autómata, sin que dicho estado se modifique hasta que dicho recorrido total se haya efectuado. El estado resultante dependerá del valor de la probabilidad, pero no será modificado hasta que se ejecute el procedimiento siguiente, "ejecutarAutomata2".

Una vez que se obtiene el estado de los patches al finalizar el recorrido por el autómata en "ejecutarAutomata" (¿ardeCelda?), se les ordena que ejecuten el procedimiento "ejecutarAutomata2". Este simplemente cambiará el estado de las celdas que arden.

Por tanto, este procedimiento "go" garantiza un correcto recorrido por las celdas del autómata.

7.3.4 PROCEDIMIENTO ABRIRELEVACION

```
to abrirElevacion
  import-pcolors-rgb "\\elevacion.png"
  set fila 0;coordenada y, va de 0 a 400
  set columna 0;coordenada x, va de 0 a 422
  while[fila < 400]
    [
      while[columna < 422]
      [
        ;queremos acceder al valor rgb de cada pixel de la imagen
elevacion
        let lista [pcolor] of patch fila columna
        let pixelRojo item 0 lista
        let pixelVerde item 1 lista
         let elevacionInversa pixelVerde
          if elevacionInversa > 248 [
            set elevacionInversa 248
          ]
          if pixelRojo = 0 and pixelVerde = 0 [
            set elevacionInversa 248
          ]
        ask patch fila columna [
          set elevacion 248 - elevacionInversa
        ]
        set columna columna + 1
      ]
      set columna 0
      set fila fila + 1
    ]
end
```

Código 7. 4: Extracto de código en NetLogo. Procedimiento "abrirElevacion"

Como hemos indicado anteriormente, este procedimiento carga la imagen "elevacion.png" y, a través de los valores RGB de la misma, asigna a la variable "elevacion" de cada parche su correspondiente valor.

7.3.5 PROCEDIMIENTO ABRIRVEGETACION

```
to abrirVegetacion
  if-else densidad = true
  [import-pcolors-rgb "\\densidad.png"]
  [import-pcolors-rgb "\\ImagenDistanciaMinimaFinal.png"]
```

```
  set fila 0;coordenada y, va de 0 a 400
  set columna 0;coordenada x, va de 0 a 422
  while[fila < 400]
    [
      while[columna < 422]
      [
        ;queremos acceder al valor rgb de cada pixel de la imagen
elevacion
        let lista [pcolor] of patch fila columna
        let pixelRojo item 0 lista
        let pixelVerde item 1 lista
        let pixelAzul item 2 lista
        let interruptor false
        ;mar
        if pixelRojo = 0 and pixelVerde = 0 and pixelAzul = 255 [
          ;if pixelRojo = 0 and pixelVerde = 0 and pixelAzul = 255 [
          ;es igual para las dos imágenes de la densidad
          ask patch fila columna [
            set densidadVegetacion "mar"
            set factorDensidadVegetacion 2
            set factorTipoVegetacion 2
            set estado 1
            set interruptor true
          ]
        ]
        ;escasa
        if-else densidad = true
       [if pixelRojo = 0 and pixelVerde = 101 and pixelAzul = 203 [
                    ask patch fila columna [
            set densidadVegetacion "escasa"
            set factorDensidadVegetacion -0.4
            set factorTipoVegetacion -0.3
            set estado 2
            set interruptor true
          ]
        ]]
       [if pixelRojo = 255 and pixelVerde = 255 and pixelAzul = 0 [
          ask patch fila columna [
            set densidadVegetacion "escasa"
            set factorDensidadVegetacion -0.4
            set factorTipoVegetacion -0.3
            set estado 2
            set interruptor true
          ]
        ]]
        ;intermedia
```

```
if-else densidad = true
[if pixelRojo = 0 and pixelVerde = 153 and pixelAzul = 177 [
            ask patch fila columna [
    set densidadVegetacion "intermedia"
    set factorDensidadVegetacion 0
    set factorTipoVegetacion 0
    set estado 2
    set interruptor true
  ]
]]
[if pixelRojo = 128 and pixelVerde = 255 and pixelAzul = 0 [
  ask patch fila columna [
    set densidadVegetacion "intermedia"
    set factorDensidadVegetacion 0
    set factorTipoVegetacion 0
    set estado 2
    set interruptor true
  ]
]]
;densa
if-else densidad = true
[if pixelRojo = 0 and pixelVerde = 205 and pixelAzul = 151 [
    ask patch fila columna [
    set densidadVegetacion "densa"
    set factorDensidadVegetacion 0.3
    set factorTipoVegetacion 0.4
    set estado 2
    set interruptor true
  ]
]]
[if pixelRojo = 0 and pixelVerde = 153 and pixelAzul = 0 [
  ask patch fila columna [
    set densidadVegetacion "densa"
    set factorDensidadVegetacion 0.3
    set factorTipoVegetacion 0.4
    set estado 2
    set interruptor true
  ]
]]
;urbana
if-else densidad = true
[if pixelRojo = 0 and pixelVerde = 255 and pixelAzul = 127 [
    ask patch fila columna [
    set densidadVegetacion "urbana"
    set factorDensidadVegetacion 2
    set factorTipoVegetacion 2
    set estado 1
```

```
            set interruptor true
          ]
        ]]
        [if pixelRojo = 0 and pixelVerde = 0 and pixelAzul = 0 [
          ask patch fila columna [
            set densidadVegetacion "urbana"
            set factorDdensidadVegetacion 2
            set factorTipoVegetacion 2
            set estado 1
            set interruptor true
          ]
        ]]
        ;ni chicha ni limoná (no coincide con ningún valor anterior)
        ;y líneas de separación, asignamos valores de vegetacion
intermedia
        if interruptor = false [
          ask patch fila columna [
            set densidadVegetacion "escasa"
            set factorDdensidadVegetacion 0
            set factorTipoVegetacion 0
            set estado 2
          ]
        ]

        set columna columna + 1
      ]
      set columna 0
      set fila fila + 1
    ]
end
```

Código 7. 5: Extracto de código en NetLogo. Procedimiento "abrirVegetacion"

Este procedimiento, al igual que el anterior, carga la imagen "densidad.png". Como indicamos al explicar el interruptor "densidad", podemos ejecutar la aplicación bien con la imagen propia de Alexandridis, "densidad.png", bien con la imagen resultante de ejecutar el algoritmo de clasificación de imágenes de la Distancia Mínima, "imagenDistanciaMinimaFinal.png". Esto lo hacemos a través de un if-else:

```
if-else densidad = true
  [import-pcolors-rgb "\\densidad.png"]
  [import-pcolors-rgb "\\ImagenDistanciaMinimaFinal.png"]
```

A partir de ahí se realiza un recorrido por todas las celdas o parches de la imagen, y a cada celda o parche se le asigna, en función de los valores RGB

de la imagen, los valores iniciales de sus variables "densidadVegetacion", "factorDensidadVegetacion", "factorTipoVegetacion" y "estado. Para distinguir entre los dos tipos posibles de imágenes base, se vuelve a utilizar un if-else, tal y como se ve en el siguiente fragmento de código, que asigna los valores para los parches cuyos colores corresponden con las características de la vegetación "escasa" (véanse las Secciones 3.2.5 y 3.2.6):

```
;escasa
        if-else densidad = true
       [if pixelRojo = 0 and pixelVerde = 101 and pixelAzul = 203 [
                    ask patch fila columna [
            set densidadVegetacion "escasa"
            set factorDensidadVegetacion -0.4
            set factorTipoVegetacion -0.3
            set estado 2
            set interruptor true
          ]
        ]]
       [if pixelRojo = 255 and pixelVerde = 255 and pixelAzul = 0 [
          ask patch fila columna [
            set densidadVegetacion "escasa"
            set factorDensidadVegetacion -0.4
            set factorTipoVegetacion -0.3
            set estado 2
            set interruptor true
          ]
        ]]
```

La última parte de este procedimiento asigna valores a dichas variables para aquellos parches de la imagen (sea ésta "densidad" o "imagenDistanciaMinimaFinal") cuyo color no está definido. A falta de otros, asignamos los valores intermedios a las variables, es decir, densidadVegetacion "intermedia", factorDensidadVegetacion "0" y factorTipoVegetacion "0".

```
;ni chicha ni limoná (no coincide con ningún valor anterior)
        ;y líneas de separación, asignamos valores de vegetacion
intermedia
        if interruptor = false [
          ask patch fila columna [
            set densidadVegetacion "intermedia"
            set factorDensidadVegetacion 0
            set factorTipoVegetacion 0
            set estado 2
          ]
        ]
```

Esto sucede porque cuando NetLogo importa las imágenes y las sitúa sobre el escenario desvirtúa los valores RGB de las mismas. Es decir, al contrario que en el caso de Java, en NetLogo las imágenes no poseen unos colores uniformes. Para entender ésto, si consideramos en Java la imagen resultante del algoritmo de clasificación de imágenes de la Distancia Mínima, tendremos que en la imagen resultante sólo existen 5 colores RGB diferentes, los cuales enumeramos a continuación:

1. ((red==0)&&(green==0)&&(blue==255)). Corresponde al mar.
2. ((red==255)&&(green==255)&&(blue==0)). Vegetación escasa.
3. ((red==128)&&(green==255)&&(blue==0)). Vegetación intermedia.
4. ((red==0)&&(green==153)&&(blue==0)). Vegetación densa.
5. ((red==0)&&(green==0)&&(blue==0)). Área urbana.

Sin embargo, en NetLogo, al cargar la imagen resultante de dicho algoritmo, los 5 colores anteriores se convierten en un sinnúmero de colores aproximados a dichos 5, pero no en únicamente 5. Esto se observa sobre la imagen en las Figuras 7.13, 7.14 y 7.15.

Como vemos, si bien el parche elegido, (165,222), posee el color definido (255,255,0), entre los parches próximos a él existen multitud de colores aproximados pero no exactamente con el valor (255,255,0). Verifiquémoslo sobre la propia ventana emergente de dicho parche, mediante "inspect-patch xxx xxx" tal y como se indica en la Sección 7.3.2. Pulsamos en la misma ventana emergente, en la parte superior, sobre los parches de alrededor del central (el (165,222)) y obtenemos las ventanas emergentes correspondientes a dichos parches:

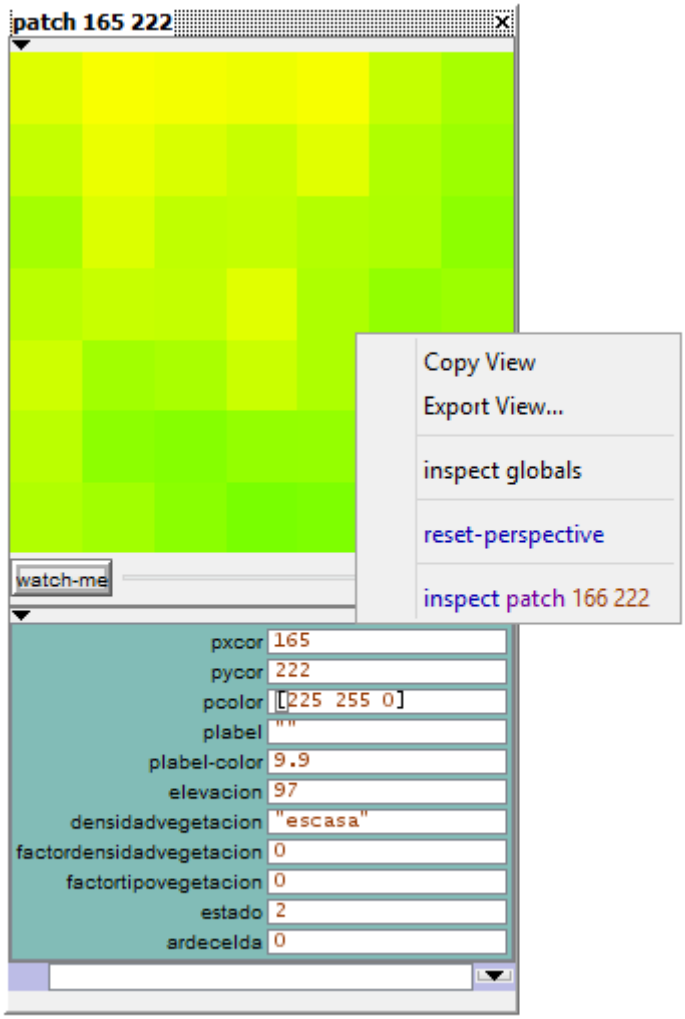


Figura 7. 13: Parche (166,222)

Veamos algunos de los parches que rodean al parche (165,222), a saber: (164,222), (165,223), (165,221) y (166,222).

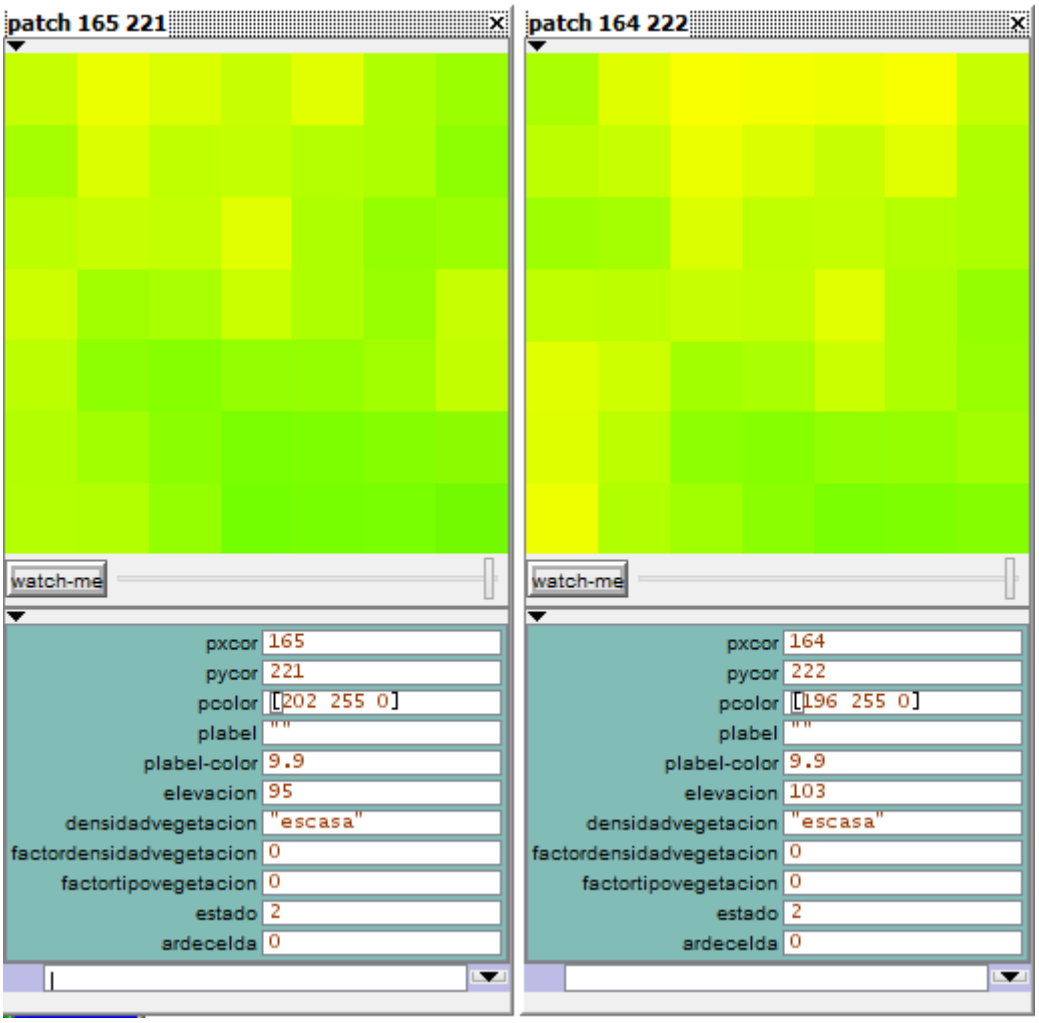


Figura 7. 14: Parches (165,221) y (164,222)

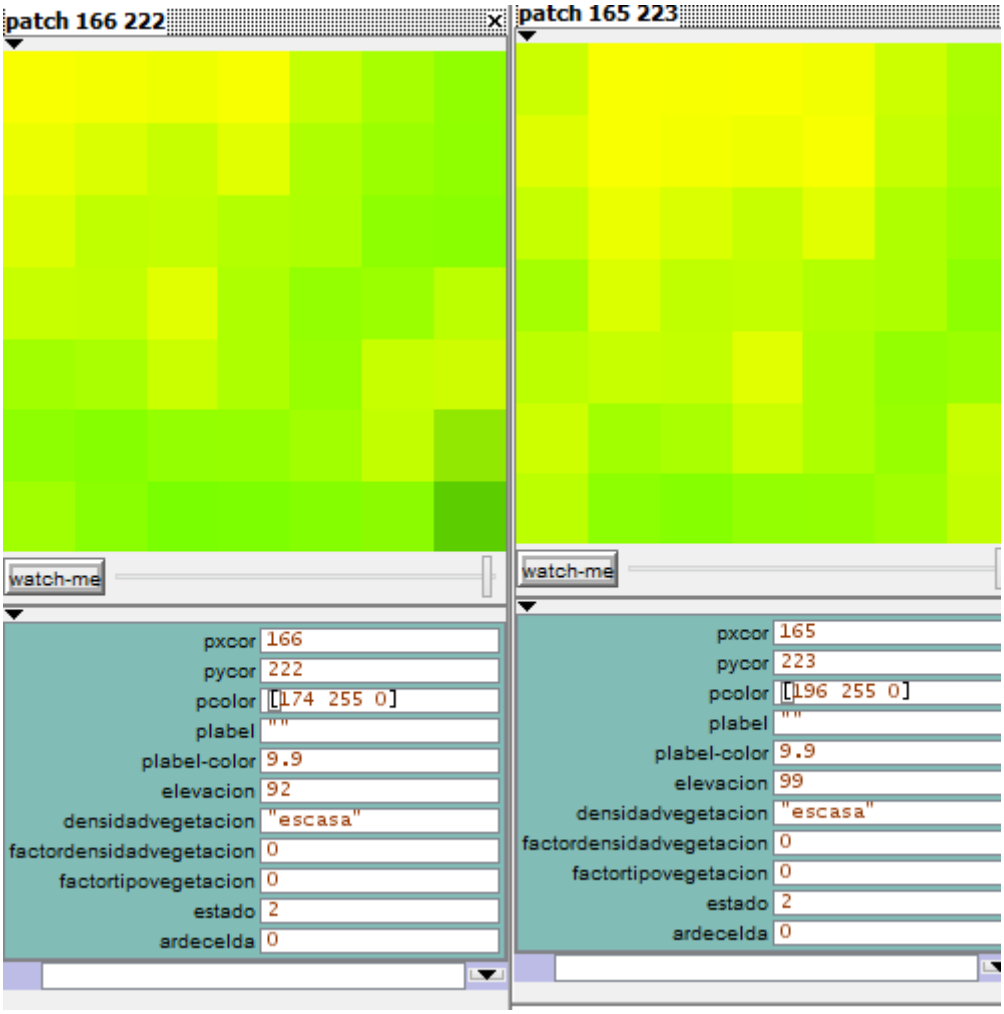


Figura 7. 15: Parches (166,222) y (165,223)

Como vemos, los colores RGB resultantes han sido los siguientes:

1. (164,222): r=196, g=255, b=0.
2. (165,223): r=196, g=255, b=0.
3. (165,221): r=202, g=255, b=0.
4. (166,222): r=174, g=255, b=0.

El color del parche central, (165,222) es r=255, g=255, b=0. Entonces, de los 4 parches que rodean al elegido como central, ninguno coincide con él ni con los otros 4 colores definidos.

Por esta razón es necesario un quinto if-else, para poder asignar valores intermedios a estos parches de colores "desvirtuados". Estos parches "desvirtuados" son los que se encuentran en las zonas fronterizas entre dos zonas de colores definidos diferentes. Así, la mayor parte de los parches tendrá uno de los 5 colores fundamentales, pero en las zonas fronterizas entre regiones con dichos colores fundamentales aparecerán estos parches desvirtuados. El otorgarles unos valores intermedios a sus variables es la única forma que vemos para poder subsanar esta irregularidad de NetLogo.

7.3.6 PROCEDIMIENTO EJECUTARAUTOMATA

```
to ejecutarAutomata
  if estado = 1 [];si la celda es no combustible, no hacemos nada
  if estado = 2 [];si la celda es combustible, pero no arde, no
hacemos nada  temporalCells2[x][y]=2;
  if estado = 3 [;si la celda arde, tenemos que cambiar su estado a 4.
Después, calculamos la probabilidad de que sus celdas vecinas ardan
    set estado 4
    set ardeCelda false
    show "************CALCULO CELDA CENTRAL: "

    let x1 pxcor - 1 show x1

    let x2 pxcor + 1 show x2

    let y1 pycor - 1 show y1

    let y2 pycor + 1 show y2

    let pxCeldaCentral pxcor
    let pyCeldaCentral pycor

    while [ x1 <= x2 ] [
      while [ y1 <= y2 ] [
        show "CALCULO PROBABILIDAD VECINO.....................: "

        ifelse pxcor = x1 and pycor = y1 [ set pcolor orange ];si la
central y la vecina coinciden ponemos a naranja la celda central
        [ask patch x1 y1 [
        if estado = 1 []
        if estado = 2 [ ;aquí es donde calcularemos la probabilidad de
que las celdas se prendan. La celda central es (pxcor,pycor), la
vecina es (x1,y1)
          ;hay que pasar los valores de la celda central, pxcor y
pycor a dos variables internas, pxCeldaCentral y pyCeldaCentral

          ;damos valores a las constantes
          let Pcero 0.58
          let c1 0.045
          let c2 0.131
          ;let velocidadViento 9

          ;calculamos la direccion de propagacion. Hay que tener en
cuenta que en netlogo las x crecen de izda a drcha(normal) pero las y
de abajo a arriba, al reves que en java
          ;por tanto, las direcciones están invertidas alrededor del
eje X de abscisas
```

```
          let direccion 0
          if x1 = pxCeldaCentral and y1 = pyCeldaCentral - 1 [ set
direccion 180 ]
          if x1 = pxCeldaCentral and y1 = pyCeldaCentral + 1 [ set
direccion 0 ]

          if x1 = pxCeldaCentral - 1 and y1 = pyCeldaCentral [ set
direccion 90 ]
          if x1 = pxCeldaCentral + 1 and y1 = pyCeldaCentral [ set
direccion 270 ]

          if x1 = pxCeldaCentral - 1 and y1 = pyCeldaCentral - 1 [ set
direccion 135 ]
          if x1 = pxCeldaCentral - 1 and y1 = pyCeldaCentral + 1 [ set
direccion 45 ]

          if x1 = pxCeldaCentral + 1 and y1 = pyCeldaCentral + 1 [ set
direccion 315 ]
          if x1 = pxCeldaCentral + 1 and y1 = pyCeldaCentral - 1 [ set
direccion 225 ]

          type "direccion: " show direccion
          ;calculamos el angulo theta
          ;let theta direccion - 90
          let theta direccion - direccionViento
          type "theta: " show theta
          ;calculamos             FsubT,             que             es
FsubT=Math.exp(velocidadViento*c2*(Math.cos(Theta)-1))
          ;en  netlogo  el  angulo  se  pone  en  grados,  con  lo  que
coseno(alfa) implica alfa en grados
          let cosTheta cos theta
          let cosThetaMenos1 cosTheta - 1; cos(theta)-1
          let expInterno velocidadViento * c2 * cosThetaMenos1
          let FsubT exp expInterno
          type "FsubT: " show FsubT

          ;calculamos          factorViento,          que          es
factorViento=FsubT*Math.exp(velocidadViento*c1);
          let expInterno2 velocidadViento * c1
          let factorViento FsubT * exp expInterno2
          type "factorViento: " show factorViento

          ;obtenemos factorDensidad de la celda vecina
          let factorDensidad [ factorDensidadVegetacion ] of patch x1
y1
          type "factorDensidad: " show factorDensidad
```

```
          ;obtenemos factorTipo de la celda vecina
          let factorTipo [ factorTipoVegetacion ] of patch x1 y1
          type "factorTipo: " show factorTipo

          ;calculo del factor slope
          let   factorSlope   calcularFactorSlope      pxCeldaCentral
pyCeldaCentral x1 y1
          type "factorSlope: " show factorSlope

          ;obtenemos la probabilidad con el factorSlope
          let parenteses1 1 + factorTipo
          let parenteses2 1 + factorDensidad
          let  probabilidad  Pcero  *  parenteses1  *  parenteses2  *
factorViento * factorSlope
          type "probabilidad: " show probabilidad
          ;*****************************************************
          ;obtenemos    la    probabilidad,    sin    el    slope,
probabilidad=Pcero*(1+factorTipoVegetacion)*(1+factorDensidad)*factorV
iento*factorSlope;
          ;let parenteses1 1 + factorTipo
          ;let parenteses2 1 + factorDensidad
          ;let  probabilidad  Pcero  *  parenteses1  *  parenteses2  *
factorViento

          ;en  funcion  de  la  probabilidad  determinamos  si  la  celda
vecina ha de arder
          ;if probabilidad < 0.55 [ set ardeCelda true ]
          let azar random-float 1
          type "azar: " show azar
          if azar < probabilidad
           [ set ardeCelda true ]
          if ardeCelda = true [
            type "se ha prendido la celda.....................: "
show patch x1 y1
          ]
          ;set ardeCelda true
          ]
        if estado = 3 []
        if estado = 4 []]]
       set y1 y1 + 1

      ];fin segundo while
      set y1 pycor - 1
      set x1 x1 + 1
    ];fin primer while
    ]
```

```
  if estado = 4 [];si la celda ha ardido, no hacemos nada

end
```

Código 7. 6: Extracto de código en NetLogo. Procedimiento "ejecutaraAutomata"

Este procedimiento es invocado dentro del procedimiento "go" y lo ejecuta cada uno de los parches del escenario. Así, para cada parche se comprueba su estado. Si el estado del parche es 1 (celda no combustible), 2 (la celda no arde) ó 4 (celda ya quemada) no hacemos nada. Sólo si el estado de la celda es 3, es decir, si la celda está ardiendo, realizamos el cuerpo del procedimiento. En el mismo, se calcula, para cada celda de su vecindario (véase la Figura 3.2) la probabilidad de que dicha celda se prenda, como consecuencia del fuego emanado por la celda central. Esta probabilidad sólo se calcula para las celdas que pueden arder, aquellas cuyo estado es 2.

Entonces, para aquellas celdas del vecindario cuyo estado es 2 se calcula su probabilidad conforme a la fórmula de la Sección 3.2.4. Para el cálculo de dicha probabilidad se invocará a la función "calcularFactorSlope", la cual veremos en breve. Una vez obtenida la probabilidad, se determinará si la celda finalmente arde mediante la función `random-float`. Si el número devuelto por esa función es inferior a la probabilidad, consideraremos que la celda del vecindario se ha prendido. Para la verificación, lo indicaríamos en la consola mediante "type".

```
;en funcion de la probabilidad determinamos si la celda vecina ha de
arder
          let azar random-float 1
          type "azar: " show azar
          if azar < probabilidad
           [ set ardeCelda true ]
          if ardeCelda = true [
            type "se ha prendido la celda.....................: "
show patch x1 y1
          ]
          ]
```

7.3.7 PROCEDIMIENTO EJECUTARAUTOMATA2

El resultado final del procedimiento anterior, "ejecutarAutomata", realizado por todos los parches o celdas es que, para cada celda, se determina si al final del recorrido del autómata dicha celda se ha prendido. Almacenará dicha información en su variable booleana "ardeCelda". Después, en el procedimiento "ejecutarAutomata2" se utilizará dicha variable para cambiar su estado a 3. Dicho cambio tendrá efecto cuando se haya realizado el recorrido

completo de todo el autómata, tal y como indicamos en la Sección 7.3.3, al explicar el procedimiento "go".

```
to ejecutarAutomata2
   if ardeCelda = true [
   set pcolor red
   set estado 3
   set continuar true
  ]
end
```

Código 7. 7: Extracto de código en NetLogo. Procedimiento "ejecutarAutomata2"

Se pinta la celda de rojo, para que se vea en el escenario, durante la ejecución de la simulación, que dicha celda está ardiendo, y se modifica el valor de la variable de control "continuar" a cierto, indicando que el autómata no debe terminar a la salida de este procedimiento, pues todavía hay celdas que arden. Dicha variable de control "continuar" será examinada en el procedimiento "go", que es donde continúa el programa a la salida de "ejecutarAutomata2". Si la variable tiene valor falso, significa que ya no hay ninguna celda ardiendo, por lo que el autómata terminará su ejecución.

7.3.8 FUNCION CALCULARFACTORSLOPE

Esta función es llamada por "ejecutarAutomata" para calcular el factor "slope" o de inclinación del terreno (véase la Sección 3.2.8).

```
to-report calcularFactorSlope [ x y i j ]
  let a i = x - 1 and j = y - 1
  let b i = x + 1 and j = y - 1
  let c i = x - 1 and j = y + 1
  let d i = x + 1 and j = y + 1
  let diagonal false
  if a or b or c or d
    [ set diagonal true ];la celda central y la vecina están en
diagonal
    ;[ set diagonal false ];la celda central y la vecina no están en
diagonal
  let elevacionVecino [ elevacion ] of patch i j;elevacion de la celda
vecina
  let elevacionBase [ elevacion ] of patch x y; elevacion de la celda
base
  let ratio 0;?????????????????????????????????? ha de ser ratio
global o con un let
```

```
  ;la fórmula cambia, pues las celdas no son cuadradas. d ya no vale
como en Java. La isla mide 6980 metros de ancho y 5190 metros de alto
  ;la distancia de separación entre celdas adyacentes, en sentido
horizotal, es de 6980/422=16.54 metros
  ;la distancia de separación entre celdas adyacentes, en sentido
vertical, es de 5190/400=12.97
  ;por tanto, la separación en diagonal es la hipotenusa del triángulo
rectángulo de dichos lados, raíz-
cuadrada(16.54*16.54+12.975*12.975)=21.022
  ifelse diagonal = true
    [ let numeradorRatio elevacionVecino - elevacionBase
      let denominadorRatio sqrt 2 * 21.022
    set ratio numeradorRatio / denominadorRatio]
    [ let numeradorRatio elevacionVecino - elevacionBase
      let denominadorRatio 16.54
      set ratio numeradorRatio / denominadorRatio ]
    type "ratio: " show ratio
  ;hay que obtener la arcotangente de ratio. Se usa la siguiente
fórmula: arcTg(ratio)=atan(ratio)*2*pi/360
  ;let arcoTangenteEnGrados atan ratio 1
  ;let thetaSlope arcoTangenteEnGrados * 2 * PI / 360

  ;hay que obtener la arcotangente de ratio en radianes.
  let menosRatio ratio * -1
  let arcoTangenteEnGrados atan menosRatio 1
  let thetaSlope arcoTangenteEnGrados * -2 * PI / 360
  type "thetaSlope: " show thetaSlope
  let expInterno 0.078 * thetaSlope
  report exp expInterno ;Math.exp(a*THETASlope);

end
```

Código 7. 8: Extracto de código en NetLogo. Función "calcularFactorSlope"

En función de la situación de la celda central, (x,y), con respecto a la vecina, (i,j), tendremos una disposición adyacente o diagonal.

Debido a que las dimensiones del escenario en NetLogo difieren de las dimensiones del espacio en Java, hay que modificar el valor de las fórmulas (3.5) y (3.6) con respecto a Java. Si en Java el espacio estaba constituido por celdas cuadradas de dimensiones 12.5 x12.5 metros cuadrados, aquí en NetLogo el escenario está constituido por parches rectangulares de dimensiones 16.54x12.97 metros cuadrados. Por tanto, se modifica el código acorde a esta variación.

A su vez, dado que en NetLogo no existe la función "arco tangente", hay que realizar una serie de trucos para obtener el valor correspondiente, a partir de la función primitiva de NetLogo "atan", la cual no devuelve una arco tangente, sino la orientación de un punto vecino con respecto al punto central (0,0).

7.3.9 PROCEDIMIENTO VERELEVACION

Responde al botón de la interfaz "ver imagen elevación".

```
to verElevacion
  import-pcolors-rgb
"C:\\Users\\troglodita\\Desktop\\ProyectoNetLogo\\elevacion.png"
end
```

Código 7. 9: Extracto de código en NetLogo. Procedimiento "verElevacion"

Este procedimiento no tiene efecto sobre la ejecución del autómata. Simplemente muestra en la interfaz la imagen RGB de la elevación de la isla de Spetses.

7.3.10 PROCEDIMIENTO VERTIPOVEGETACION

Responde al botón de la interfaz "ver imagen tipo vegetación".

```
to verTipoVegetacion
  import-pcolors-rgb "\\tipo.png"
end
```

Código 7. 10: Extracto de código en NetLogo. Procedimiento "verTipoVegetacion"

Similarmente al anterior, este procedimiento muestra la imagen RGB con los datos del tipo de vegetación de la isla de Spetses.

7.3.11 PROCEDIMIENTO VERDENSIDADVEGETACION

Responde al botón de la interfaz "ver imagen densidad vegetación".

```
to verDensidadVegetacion
  if-else densidad = true
  [import-pcolors-rgb "densidad.png"]
  [import-pcolors-rgb "ImagenDistanciaMinimaFinal.png" ]
end
```

Código 7. 11: Extracto de código en NetLogo. Procedimiento "verDensidadVegetacion"

De forma similar a los dos anteriores, este procedimiento muestra en pantalla la imagen elegida mediante el interruptor "densidad" para ver los valores RGB de la densidad de la vegetación en la isla de Spetses.

7.4 PRUEBA DE LAS EJECUCIONES

Al igual que hicimos con la implementación en Java, realizamos inicialmente 5 ejecuciones con valores similares de entrada, para comprobar cómo influye la probabilidad en el resultado final del incendio:

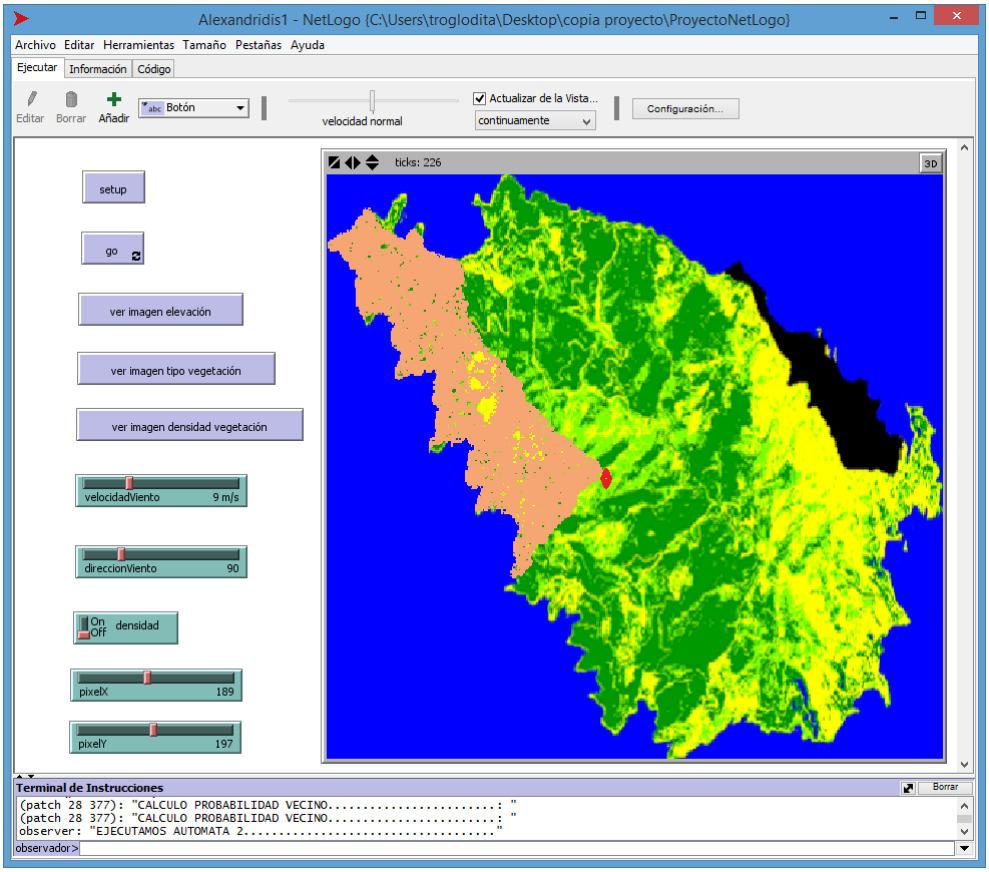


Figura 7. 16: Primera ejecución con velocidad 9 m/s y dirección 90 grados

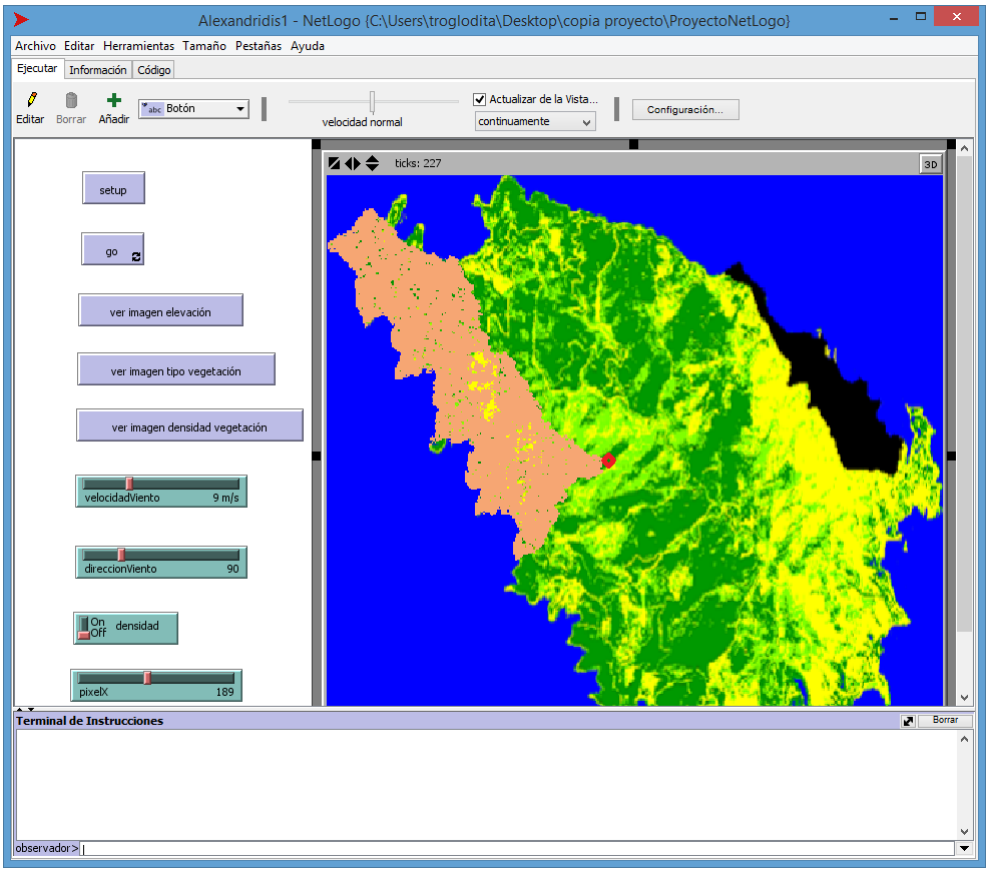


Figura 7. 17: Segunda ejecución con velocidad 9 m/s y dirección 90 grados

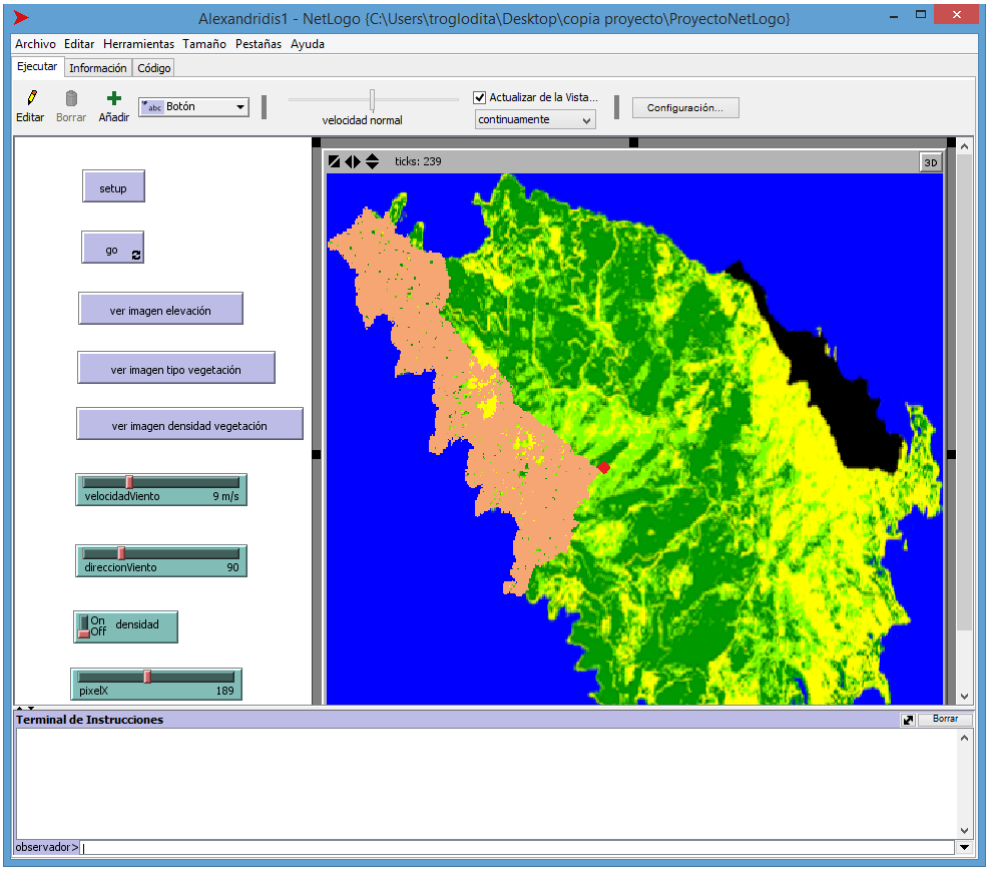


Figura 7. 18: Tercera ejecución con velocidad 9 m/s y dirección 90 grados

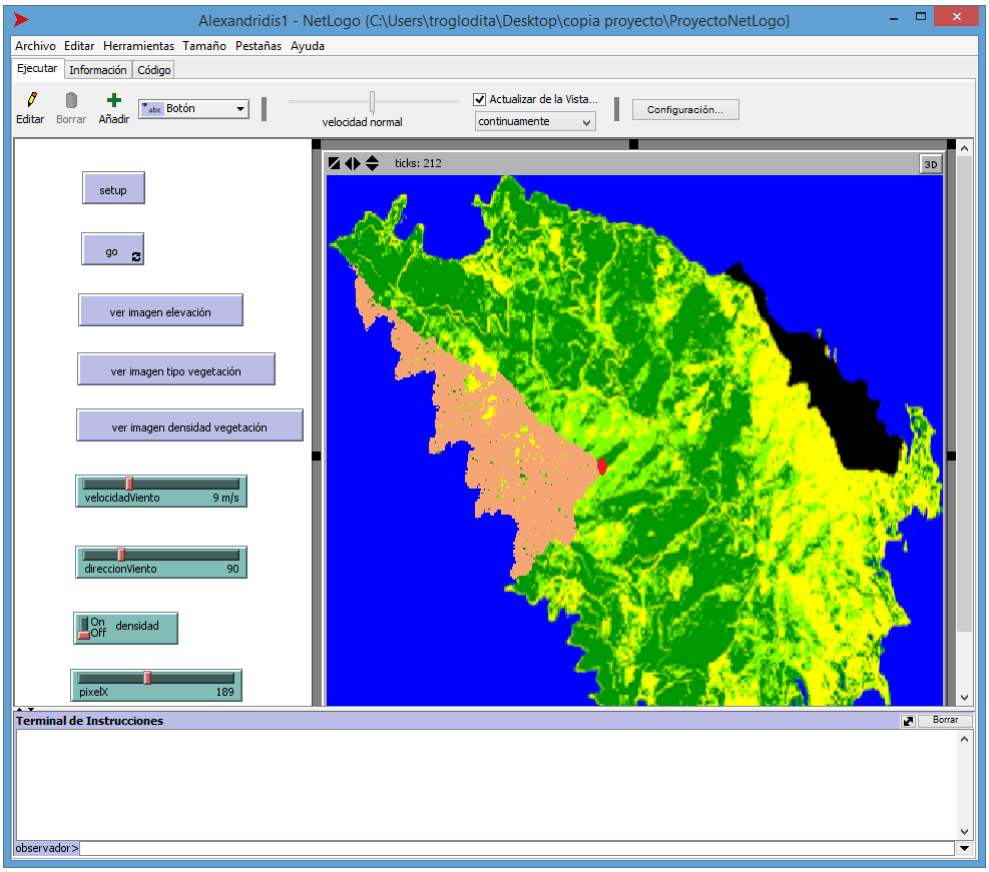


Figura 7. 19: Cuarta ejecución con velocidad 9 m/s y dirección 90 grados

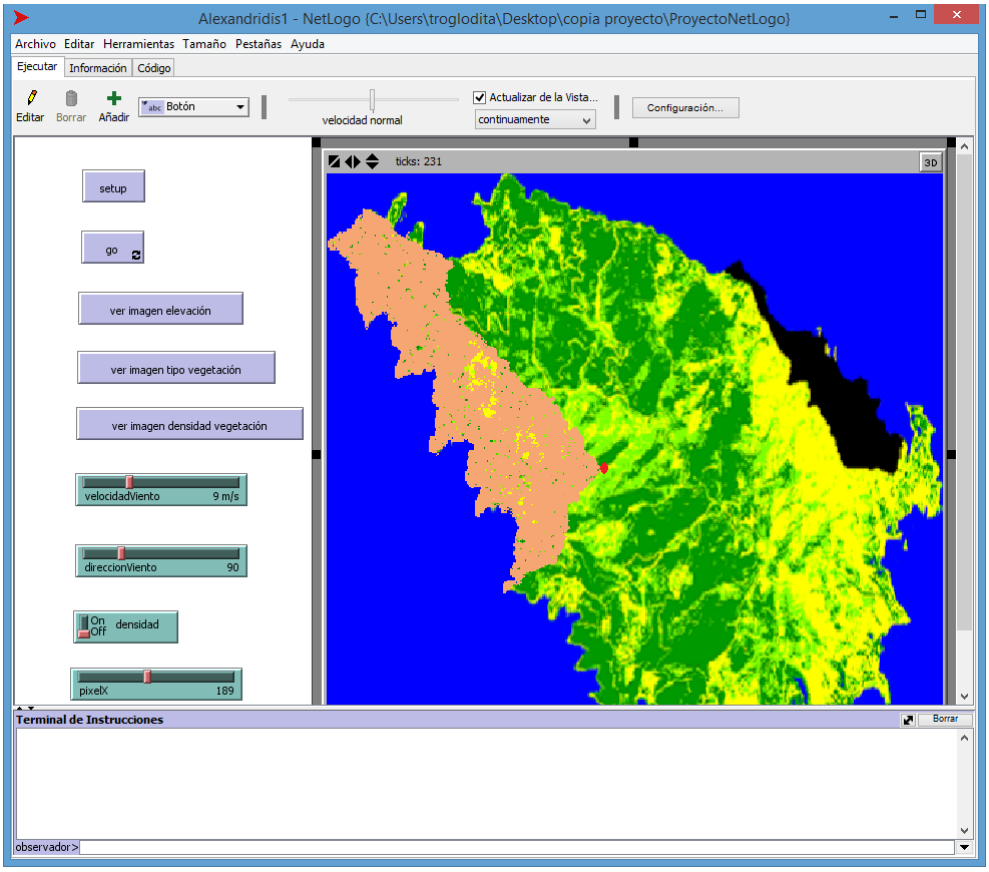


Figura 7. 20: Quinta ejecución con velocidad 9 m/s y dirección 90 grados

Como vemos, ninguna de las ejecuciones es exactamente igual a otra, aunque los resultados finales reflejen cierta proximidad. En los cinco casos el viento posee dirección oeste (90 grados) con una velocidad de 9 m/s.

Realizamos ahora cuatro ejecuciones con diferentes datos de entrada, centradas en el mismo foco inicial:

1. Ejecución A: velocidad del viento 9 m/s, dirección oeste (90 grados).
2. Ejecución B: velocidad del viento 9 m/s, dirección norte (0 grados).
3. Ejecución C: velocidad del viento 9 m/s, dirección sur (180 grados).
4. Ejecución D: velocidad del viento 9 m/s, dirección este (270 grados).

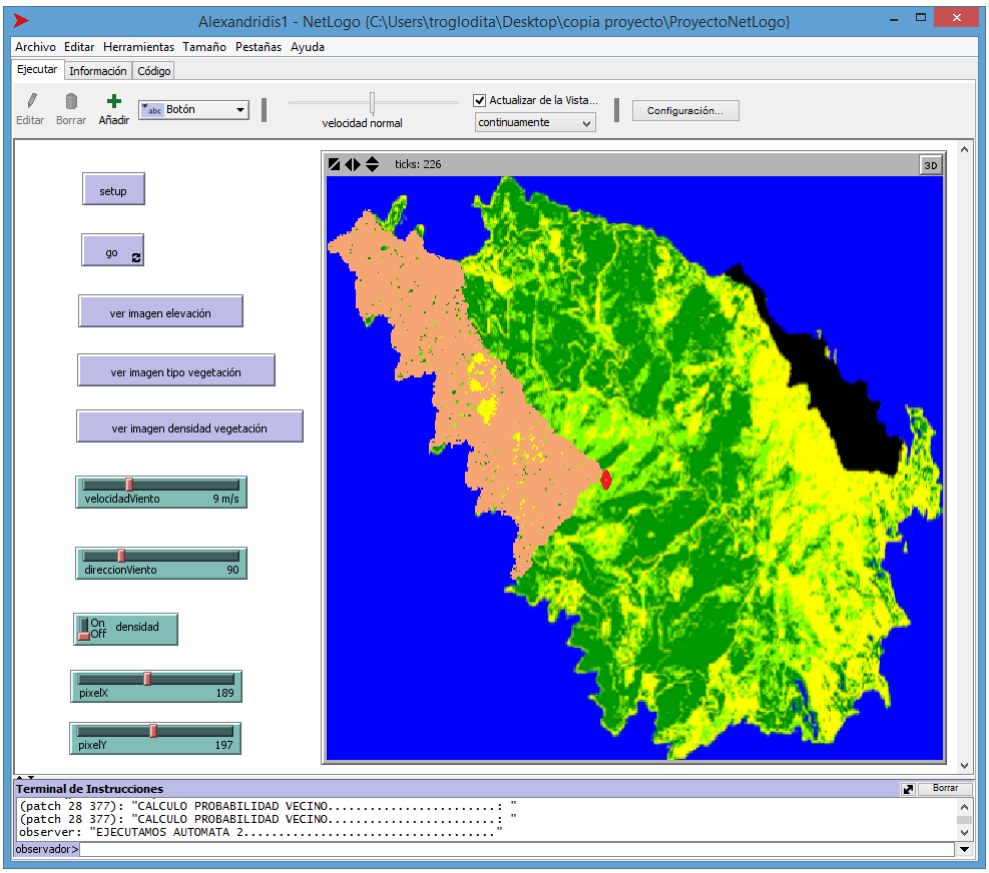


Figura 7. 21: Ejecución A, con velocidad 9 m/s y dirección 90 grados

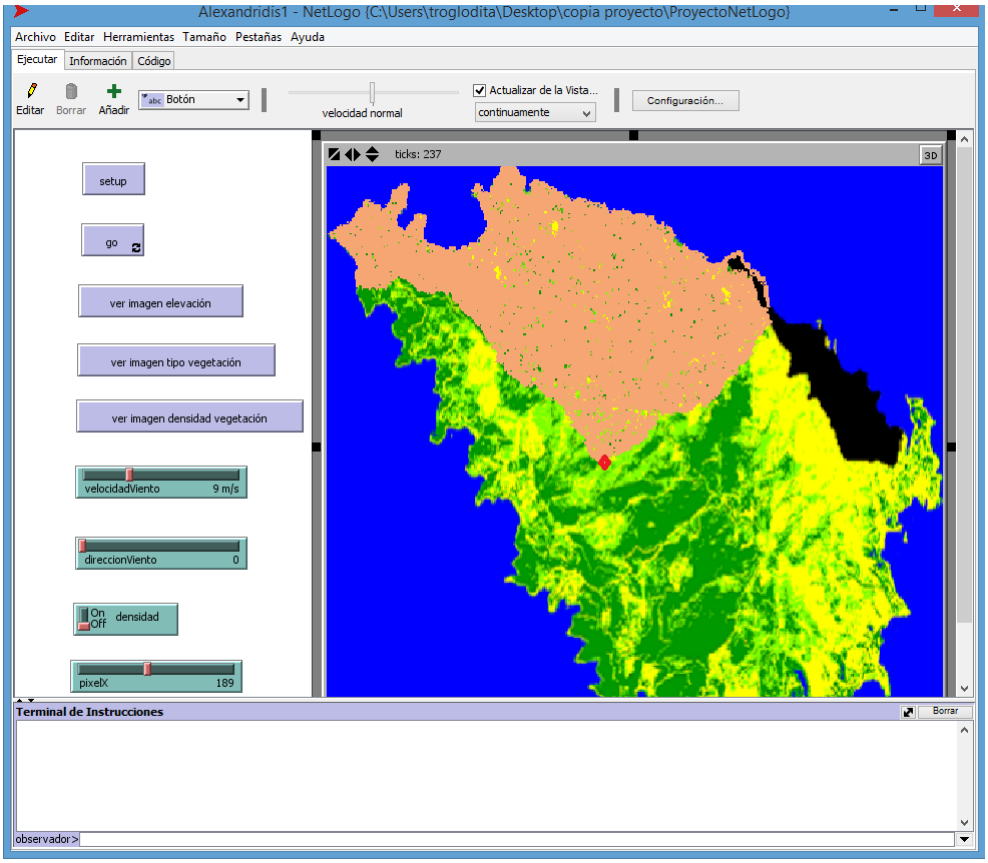


Figura 7. 22: Ejecución B, con velocidad 9 m/s y dirección 0 grados

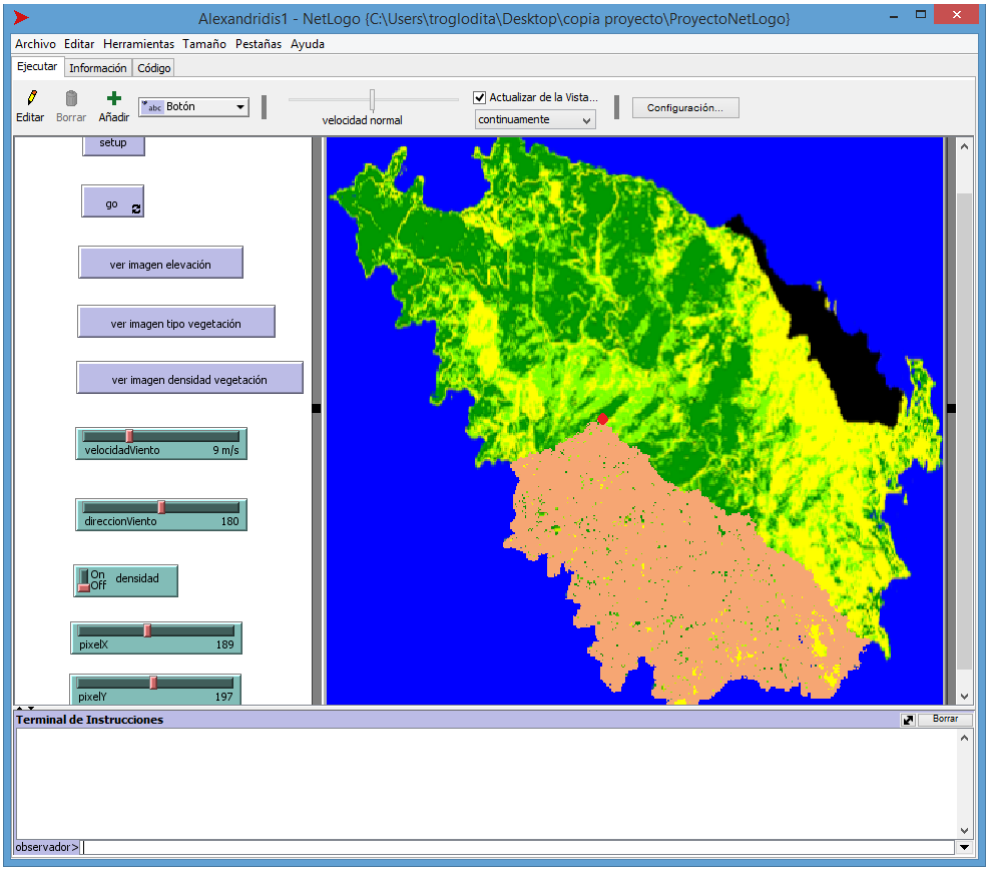


Figura 7. 23: Ejecución C, con velocidad 9 m/s y dirección 180 grados

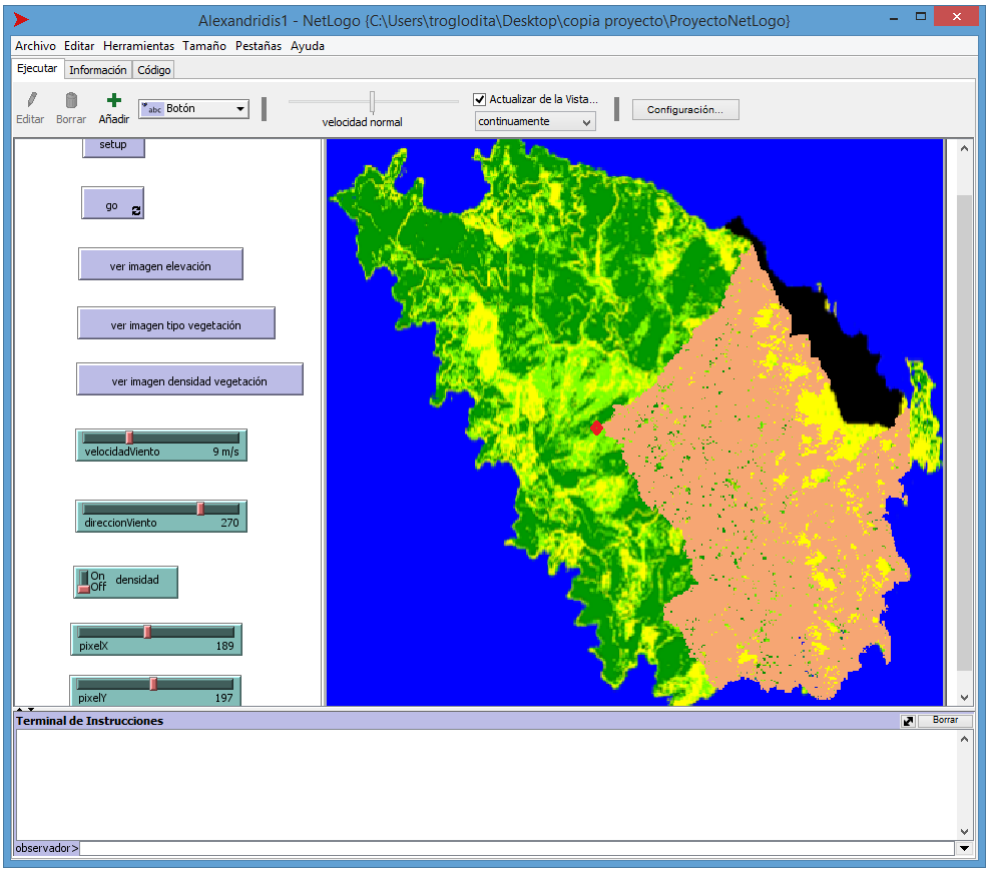


Figura 7. 24: Ejecución D, con velocidad 9 m/s y dirección 270 grados

Como se aprecia en las imágenes correspondientes, las zonas incendiadas de la isla incendiada están en consonancia con la dirección del viento.

Realizamos ahora dos ejecuciones con el mismo foco inicial y dirección del viento, pero con velocidad del viento diferente:

1. Ejecución E: velocidad del viento 9 m/s, dirección oeste (90 grados).
2. Ejecución F: velocidad del viento 30m/s, dirección oeste (90 grados).

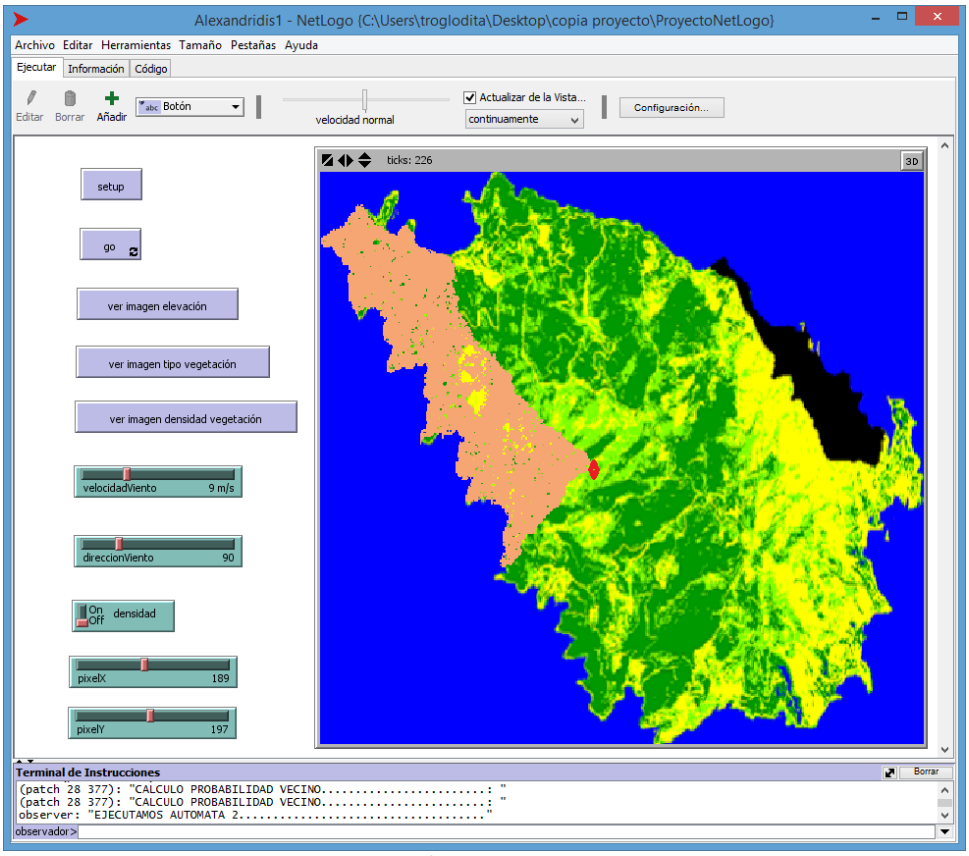


Figura 7. 25: Ejecución E, con velocidad 9 m/s y dirección 90 grados

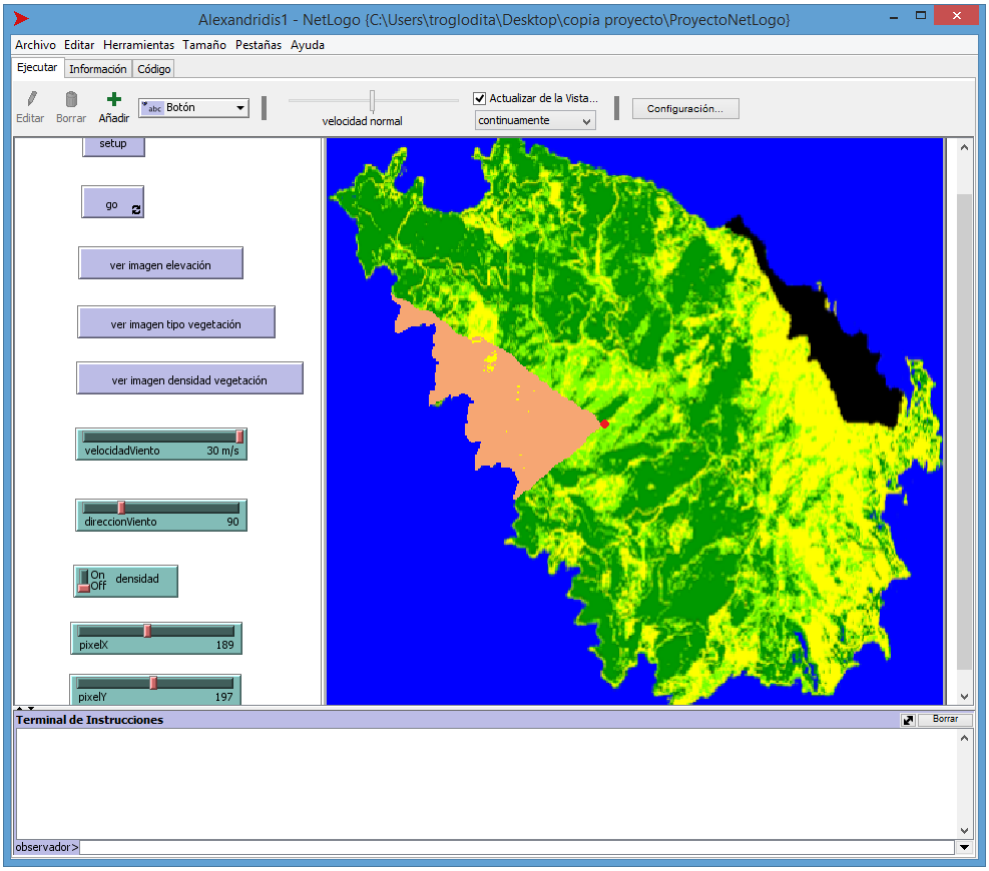


Figura 7. 26: Ejecución F, con velocidad 30 m/s y dirección 90 grados

Vemos que cuando la velocidad del viento es mayor (pasa de 9 a 30 m/s) la forma del fuego es más definida.

En resumen, en las comparaciones realizadas en esta sección hemos comprobado que la forma de la extensión del incendio sigue un desarrollo lógico, en función de los diferentes parámetros de velocidad y dirección del viento.

7.5 COMPARACIÓN CON LAS IMÁGENES DE ALEXANDRIDIS ET ALTER

Como indicamos en el Capítulo 5 de este PFC, en el trabajo de Alexandridis et Alter, las figuras 7 y 8 del mismo representan, respectivamente, el área real del incendio ocurrido en la isla de Spetses en 1990 y el resultado de la propia simulación de dichos autores.

Presentamos a continuación dichas imágenes:

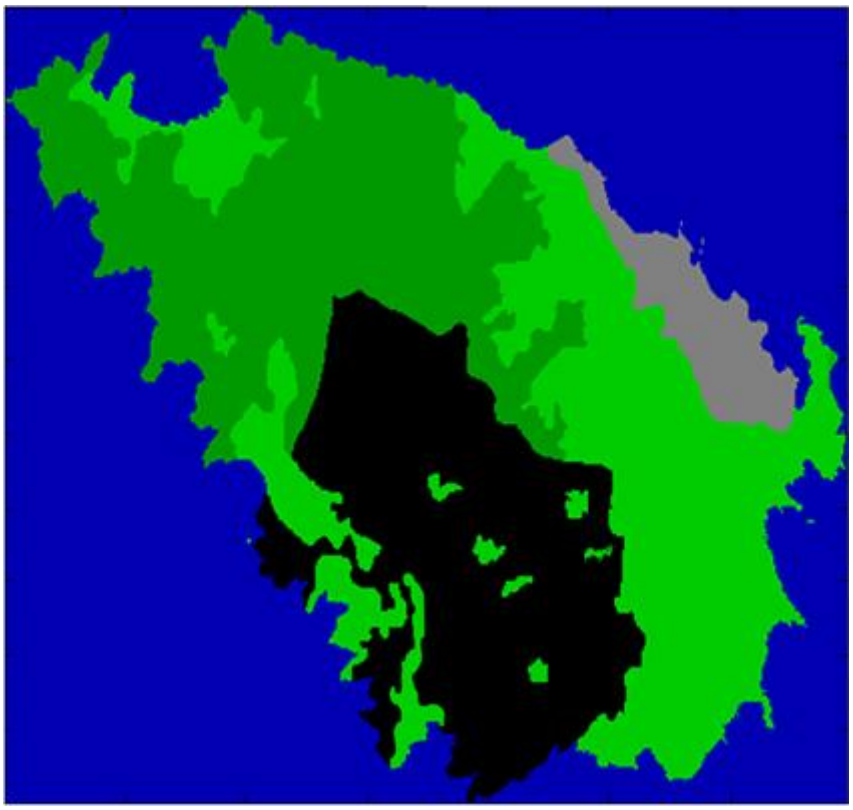


Figura 7. 27: Área del fuego real en Spetses en 1990 (en negro)

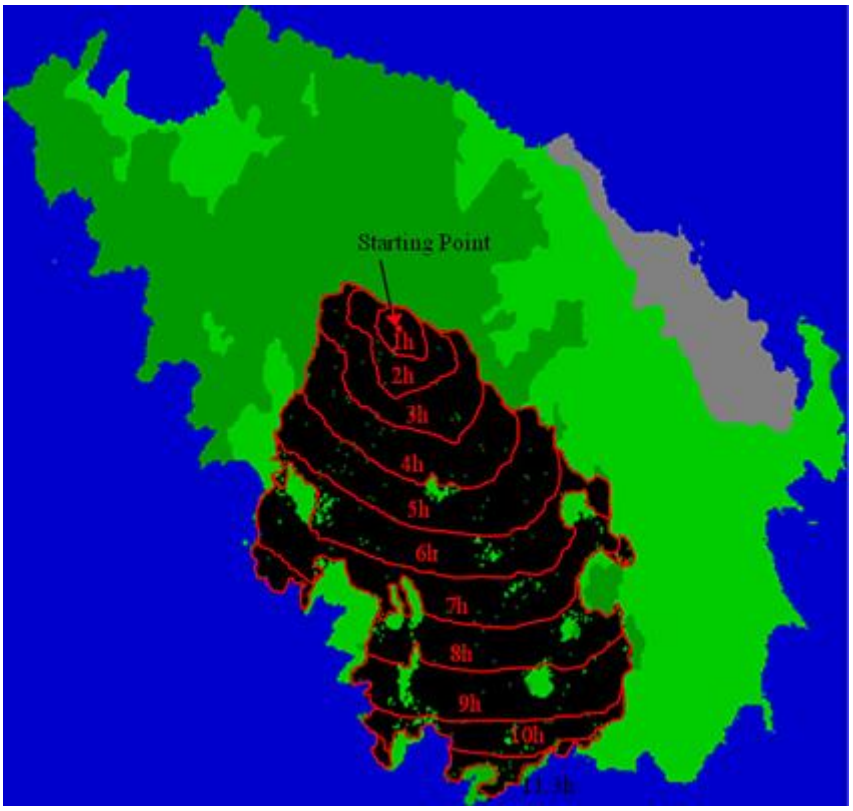


Figura 7. 28: Área del fuego en la simulación de Alexandridis et Alter (en negro)

Mostramos ahora dos ejecuciones de nuestro modelo, con los parámetros de Alexandridis:

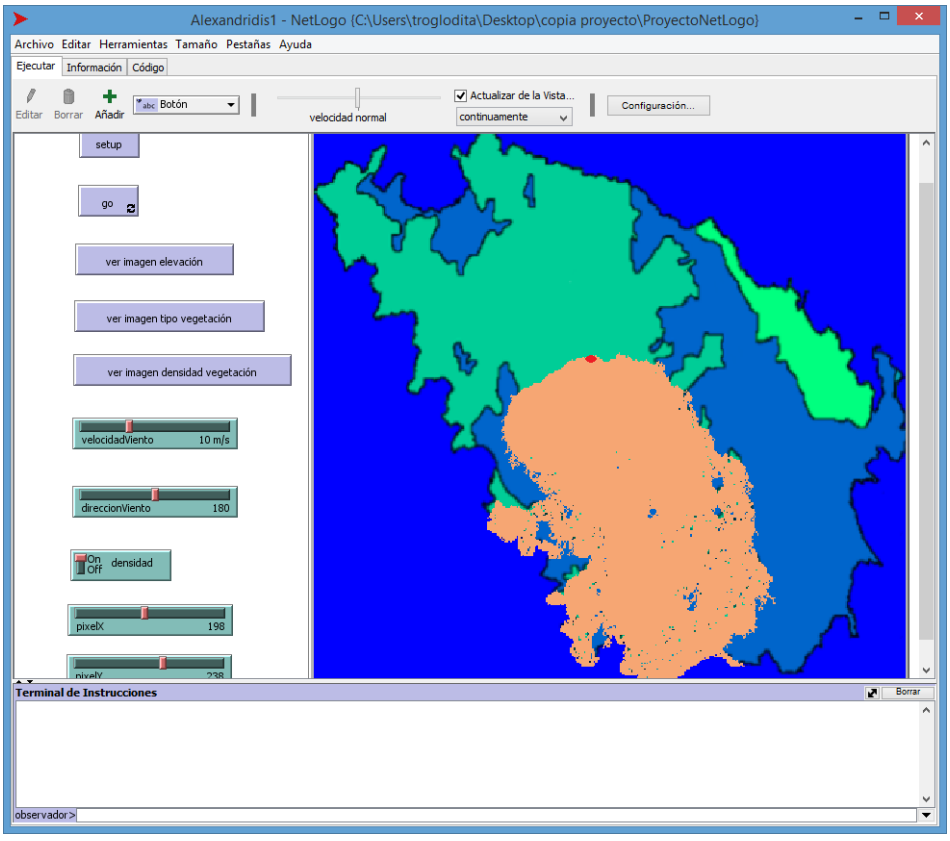


Figura 7. 29: Área del fuego en la primera simulación propia

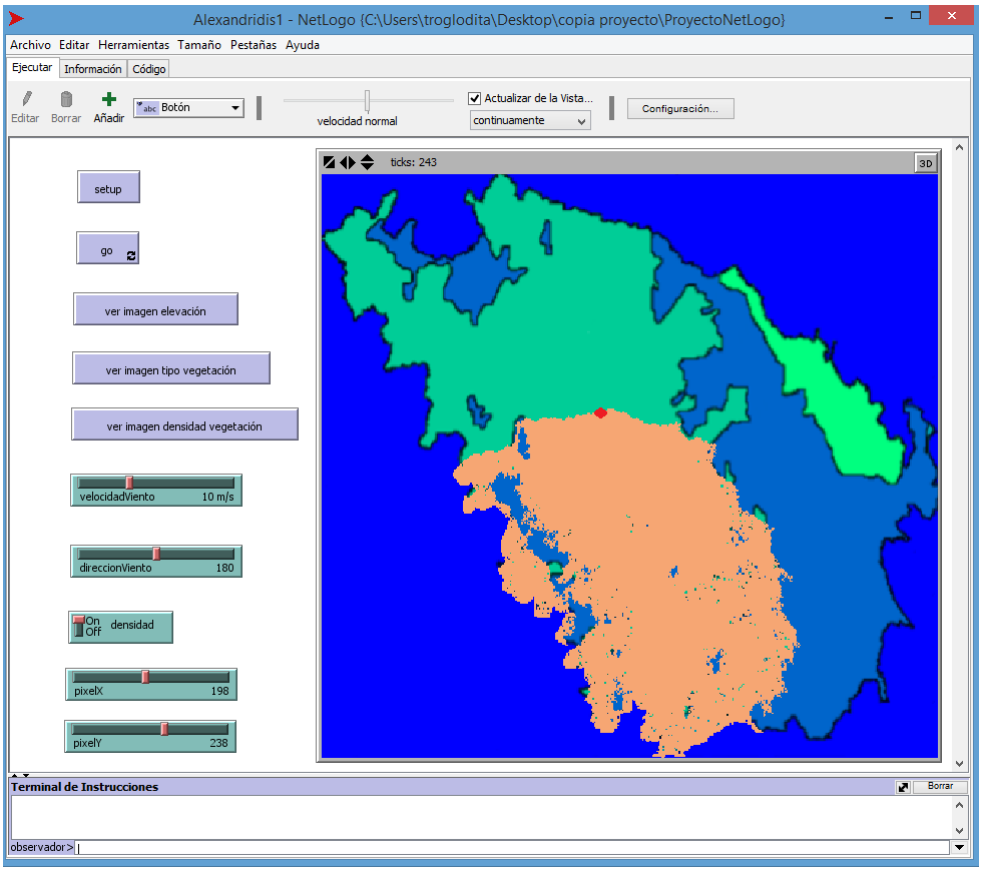


Figura 7. 30: Área del fuego en la segunda simulación propia

Como se aprecia, las ejecuciones en NetLogo difieren en poco de las figuras 7 y 8 del trabajo de Alexandridis et Alter. En primer lugar, hay que considerar que las simulaciones se basan en el empleo de números aleatorios. A su vez, hay que tener en cuenta lo expuesto en la Sección 7.3.4. NetLogo desvirtúa los colores RGB de las imágenes cargadas, lo cual nos obliga a establecer unos valores intermedios para aquellos píxeles cuyos colores "desvirtuados" no se corresponden con ninguno de los usados para clasificar la vegetación. En nuestro caso, hemos empleado los valores propios de la vegetación intermedia en las zonas fronterizas entre dos colores diferentes. Esta forma de subsanar el error de NetLogo provoca que haya unos pocos cambios en las probabilidades calculadas en nuestro modelo en NetLogo con respecto a las del modelo original de Alexandridis et Alter. Finalmente, hay que considerar que el número de celdas en el modelo original de Alexandridis et Alter es de 1400x1000 (tal y como indica [Russo, 2014]), mientras que en nuestro escenario de NetLogo es de 422x400.

7.6 CONCLUSIONES

Hemos implementado el modelo de autómata de Alexandridis et Alter en el entorno de programación de sistemas multiagentes de NetLogo.

Tal implementación presenta la ventaja de que determinadas operaciones sobre las imágenes (importación de las mismas, escalado al escenario) son más sencillas de realizar que en Java. Por contra, presenta la desventaja de que el procesamiento de las imágenes presenta, en las zonas fronterizas entre diferentes colores principales, los valores RGB de dichos parches (que equivaldrían a píxeles) cambiados con respecto a la imagen original (véase la Sección 7.3.4.).

La ejecución del autómata es bastante lenta si se quieren ver los cálculos intermedios por la consola. Sin embargo, si se elimina la salida por la consola, la ejecución es rápida.

Otra ventaja con respecto a Java es que NetLogo posee elementos de interfaz (botones, deslizadores, interruptores y otros) predefinidos y de uso bastante sencillo.

En definitiva, NetLogo se presenta como una herramienta útil para este tipo de trabajos relacionados con los autómatas celulares, siempre y cuando la desvirtuación producida en algunos valores RGB al importar imágenes al escenario no perjudique excesivamente los resultados en los cálculos de las reglas de transición del autómata.

8 PLANIFICACIÓN Y PRESUPUESTO

8.1 INTRODUCCIÓN

La planificación de las fases de desarrollo de un proyecto informático y el cálculo de los costes asociados es parte fundamental en el mismo, de cara a la realidad laboral.

En el caso de un PFC, dado que éste se realiza, en general, por una única persona, y, además, el PFC es un desarrollo académico y no laboral, el cumplimiento de los hitos iniciales previstos presentará, por lo general, ciertas dificultades, aun cuando, y en la medida de lo posible, el alumno intentará cumplir con lo proyectado.

8.2 PLANIFICACIÓN

A continuación, mostramos un diagrama de Gantt en el que esquematizamos la planificación prevista en el anteproyecto y lo que ha constituido el desarrollo real del PFC.

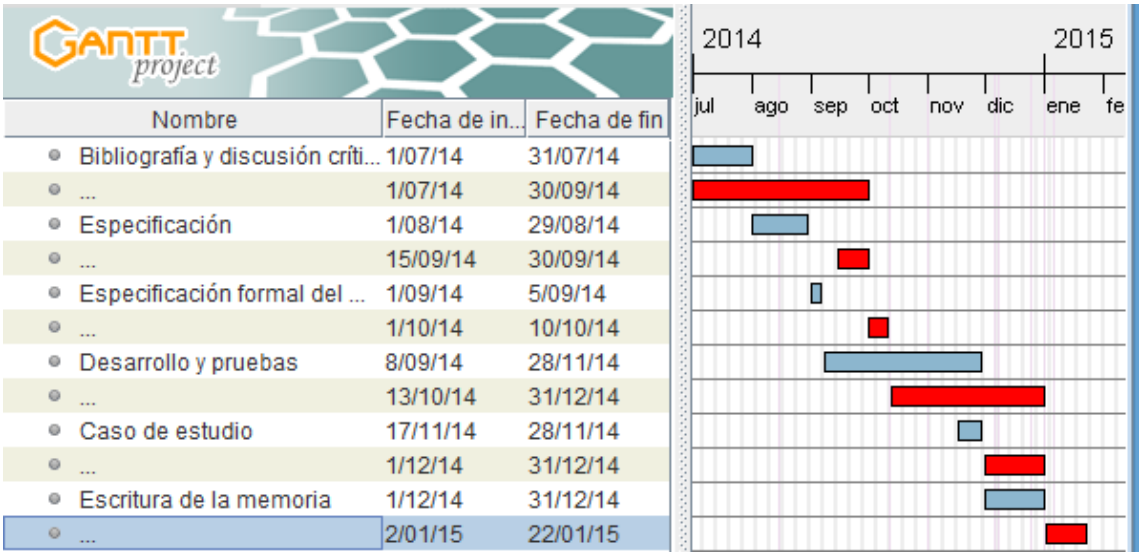


Figura 8. 1: Planificación: fases previstas (en azul) y fases reales (en rojo)

En azul aparecen las fases de realización, según estaban previstas, y en rojo (debajo de su fase azul correspondiente) las fases tal y como han sucedido.

La divergencia entre las fechas previstas para cada fase y las fechas reales se deben, principalmente, a cuestiones laborales y de estudios ajenos al proyecto, no previstas inicialmente.

Existen solapamientos entre las tareas, lo cual se debe, principalmente, a la implementación extra en NetLogo.

8.3 PRESUPUESTO

Puesto que el PFC se ha realizado en el propio domicilio del autor y con su ordenador personal consideraremos que los costes relacionados con los materiales, consumibles, electricidad, conexión a internet, etc., se trasladan a un precio-hora, que es el que cobraría un ingeniero informático que trabajase "por horas" por cuenta propia, en lugar de con un sueldo fijo y en un lugar de trabajo empresarial.

El precio-hora que utilizaremos en estos cálculos será de 60 euros por hora de trabajo.

Para calcular el número de horas consideraremos una jornada diaria de 2 horas hasta el 30 de septiembre y de 3 horas desde el 15 de octubre en adelante. Excluimos los fines de semana. Esta media se calcula considerando que, debido al desempeño de otros trabajos ajenos al PFC, ha habido días de más horas de trabajo y días de menos o ninguna, pero esas medias son las que el autor se ha obligado a conseguir, en medio de otras faenas.

Así, calculando, para los periodos en los que hay solapamiento, mitad del tiempo para cada tarea, tendríamos las siguientes horas y costes por fase:

FASE	**FECHA DE INICIO**	**FECHA DE FIN**	**NUMERO DE HORAS**	**COSTE EN EUROS**
Bibliografía y discusión	1/7/14	30/9/14	118	7.080
Especificación inicial	15/9/14	30/9/14	12	720
Especificación formal	1/10/14	10/10/14	24	1.440
Desarrollo y pruebas	13/10/14	31/12/14	136	8.160
Caso de estudio	1/12/2014	31/12/2014	32	1.920
Escritura de la memoria	2/1/2015	22/1/2015	60	3.600
TOTAL			382	22.920

Figura 8. 2: Costes del proyecto por fase y coste total

Así pues, el coste total sería de 22.920 euros.

8.4 CONCLUSIONES

La realización ha llevado un tiempo total de 382 horas, equivalente a unas 51 jornadas laborales (considerando 7.5 horas diarias), es decir, unos 3 meses si se hubiera trabajado a tiempo completo.

Considerando un precio-hora de 60 euros, los costes totales serían 22.920 euros.

9 CONCLUSIONES Y TRABAJOS FUTUROS

9.1 INTRODUCCIÓN

En el campo de la simulación de la extensión de los incendios forestales, en relación a la técnica utilizada para modelar la propagación del fuego en el paisaje se suele elegir entre una de las 3 técnicas siguientes:

1. Redes regulares basadas en autómatas celulares ("cellular automata") o en percolación ("bond percolation").
2. Representación en plano contiguo, mediante propagación en ondas ("wave propagation").

Decidir sobre la mayor o menor bondad de cada una de las técnicas escapa totalmente al conocimiento de un informático. En nuestro caso particular, nos decidimos por el modelo de Alexandridis dado que su formulación era comprensible y asequible en el dominio de la realidad a simular.

El tratamiento de imágenes complejas en Java se ha realizado, principalmente, mediante el uso de los conceptos de Java2D y de las clases Java.awt.Image y Java.awt.image.BufferedImage. La aplicación final de las fórmulas del autómata a los datos obtenidos de la imagen representa, también, una parte importante del desarrollo de la aplicación, pues es necesaria la comprobación de que el autómata funciona específicamente como un autómata celular, es decir, que los cambios realizados sobre cada celda no tienen efecto hasta el final del recorrido en el conjunto de las celdas, y de que las fórmulas empleadas en la transición de los estados se han implementado correctamente.

La implementación desarrollada con NetLogo ha tenido como objetivo la verificación de las capacidades de dicha herramienta para la implementación de simuladores basados en autómatas celulares.

A continuación examinaremos las conclusiones sacadas de la realización de este trabajo, y propondremos, igualmente, posibles mejoras futuras.

9.2 CONCLUSIONES

Como indica el nombre del proyecto, el objetivo ha sido la realización de un simulador de la propagación de incendios forestales mediante autómatas celulares.

Para poder adentrarse en el dominio de la realidad a simular, se ha realizado en el Capítulo 2 de este PFC un breve repaso el modelado de los incendios forestales. Se han clasificado estos mismos en teóricos, empíricos y

semiempíricos. A continuación, se ha revisado la teoría principal sobre los autómatas celulares y se ha presentado una discusión crítica sobre varios modelos de autómatas celulares de propagación de incendios forestales. Finalmente, en este capítulo se ha presentado una extensión al paradigma de los autómatas celulares, el formalismo llamado Cell-DEVS. La conclusión que extraemos de este capítulo es la de que existe suficiente literatura que demuestra la factibilidad del empleo de los autómatas celulares para la simulación de los incendios forestales. La abundante documentación existente y la aquí presentada así lo justifican.

En el Capítulo 3 del PFC se ha presentado el modelo a simular, el autómata desarrollado por A. Alexandridis, D. Vakalis, C.I. Siettos, y G.V. Bafas en el trabajo "A cellular automata model for forest fire spread prediction: The case of the wildfire that swept through Spetses Island in 1990" [Alexandridis, 2008]. Se ha referido la definición del autómata (vecindario, estados y reglas de transición) y se han detallado cada una de las variables meteorológicas y propias del terreno que afectan a la extensión del incendio. Se ha explicado el "caso de estudio" del modelo de Alexandridis et Alter, el incendio que asoló la isla de Spetses en 1990, y, finalmente, se han mostrado los resultados obtenidos por dichos autores con su simulador. La conclusión obtenida de este capítulo es que el autómata de Alexandridis et Alter constituye un buen ejemplo de la utilización del paradigma de los autómatas celulares para la simulación de la propagación de los incendios forestales, ya que el trabajo explica con detalle la definición del autómata, el caso de estudio y los resultados obtenidos, los cuales se aproximan bastante a los efectos del fuego real que ocurrió en Spetses.

En el Capítulo 4 del PFC se ha presentado la aplicación realizada. En primer lugar, se ha examinado la estructura modular de la aplicación, y, a continuación, se ha explicado el diseño y codificación de la interfaz. Ésta ha desarrollado el autómata de Alexandridis et Alter en una red o parrilla de 700 x 550 metros (el paisaje donde se va a simular el incendio). Se ha realizado un tratamiento de las imágenes "reales" de dicho paisaje, para la extracción de los datos necesarios para calcular las reglas de transición de estados del autómata (densidad de la vegetación, tipo de combustibles de la misma y elevación o inclinación del terreno, a los cuales el usuario añade posteriormente los datos relativos a las variables velocidad y dirección del viento y foco inicial del incendio). Con los datos ya extraídos, se ha implementado el autómata celular. La conclusión hallada es que la realización del autómata celular en lenguaje Java es posible, por medio del empleo de los paquetes de tratamiento de imágenes Java Advanced Image y Java2D. Como añadido al trabajo de Alexandridis et Alter se ha empleado el algoritmo de clasificación de imágenes de la Distancia Mínima para clasificar una imagen Landsat de la isla de Spetses en cuanto a su vegetación, comprobando la

posibilidad del empleo de este algoritmo para clasificar imágenes obtenidas por satélite.

En el Capítulo 5 del PFC se ha realizado una verificación del algoritmo desde tres aspectos necesarios: la validez de la aplicación de las fórmulas que se aplican a las reglas de transición del autómata, el recorrido por el autómata y las ventanas de ejecución del simulador. La conclusión principal de este capítulo es que una verificación matemática exhaustiva del autómata es impracticable, pues conllevaría cálculos de orden exponencial, pero una verificación de la aplicación correcta en cuanto a los recorridos del autómata sí es posible, extrapolando a partir de unos pocos pasos iniciales.

En el Capítulo 6 del PFC hemos presentado un manual o tutorial del uso de nuestro simulador, a la vez que hemos indicado cómo obtener una traza de los pasos ejecutados por el autómata mediante la ejecución del mismo desde un editor de Java. Así, hemos explicado la instalación del autómata en Java Eclipse, a fin de poder observar en la consola del editor los pasos sucesivos que va realizando la clase principal del autómata (PanelAutomataExterno), lo cual nos permite realizar una verificación del modelo, tal y como la hemos presentado en el Capítulo 5.

En el Capítulo 7 del PFC hemos presentado una implementación del mismo autómata de Alexandridis et Alter en el lenguaje para sistemas multiagente NetLogo. NetLogo presenta ventajas e inconvenientes respecto de Java, tal y como se explica en la Sección 7.6, pero representa claramente una opción útil para el desarrollo de autómatas celulares. El hecho de que los parches se puedan equiparar conceptualmente a las celdas de un autómata disminuye en buena medida el tiempo de desarrollo de un autómata en NetLogo. Igualmente, las opciones preconstruidas en dicho entorno de programación facilitan la construcción de la interfaz. Si bien su potencialidad es inferior a la de Java, para desarrollos menores constituye una opción, a nuestro entender, preferente.

9.3 TRABAJOS FUTUROS

Considerando el propósito final que podría tener una aplicación como la aquí desarrollada, consideramos dos mejoras principales:

1. Mecanizar la ejecución de diferentes escenarios o paisajes: el usuario simplemente cargaría un paisaje de su elección, y la aplicación realizaría todo el procesado conforme a el autómata de Alexandridis et Alter sobre dicho paisaje elegido a discreción. Una salvedad importante en este punto es que el modelo de Alexandridis et Alter utiliza unas constantes deducidas de forma empírica, con lo cual los paisajes de elección deberían ser similares a las características de la isla de Spetses, y aún así habría que hacer una adaptación de dichos parámetros a las nuevas

condiciones. Tal adaptación se podría realizar según explicamos en el siguiente párrafo.

2. Cálculo empírico de los parámetros anteriormente mencionados, para otros escenarios distintos al original empleado por Alexandridis et Alter. Para ello, sería necesario presentar un procesamiento por lotes. El usuario cargaría un paisaje en el que se hubieran producido uno o varios incendios reales y de los cuales se tuviesen datos acerca del resultado de los mismos, y de las condiciones atmosféricas y propias del terreno en que se produjeron. Entonces, la aplicación se ejecutaría repetidamente sobre dicho nuevo paisaje, a fin de sacar una correlación estadística que permitiera refinar los parámetros que en el modelo presentado por Alexandridis et Alter son semiempíricos y que, por tanto, sería necesario refinar para estos nuevos paisajes.

En definitiva, estas dos mejoras permitirían modelar incendios futuros con los parámetros semiempíricos afinados, en zonas en las que ya se hubiesen producido previamente incendios de los cuales se conociesen sus características. Considerando que en el clima Mediterráneo y en los bosques de pinares los incendios son repetitivos, una vez se tuvieran refinados los parámetros se podrían modelar distintas posibilidades de incendios futuros, para así poder realizar políticas preventivas con mejor conocimiento del dominio en cuestión (el paisaje cuyo incendio queremos prevenir y su forma de responder ante incendios en función de sus parámetros propios y de los atmosféricos).

También queremos hacer mención de los llamados sistemas de información geográfica o SIG. Al igual que un procesador de textos facilita y amplía en gran medida el desarrollo de trabajos escritos (como, por ejemplo, una memoria de un PFC) resulta evidente que los SIG constituyen una herramienta de automatización y mejora extraordinaria en relación a todo tipo de trabajo informático que tenga relación con el procesamiento de imágenes que contengan o de las que haya que extraer datos cuantitativos y cualitativos.

BIBLIOGRAFIA

[Albini, 1976]: Albini, Frank, *"Estimating Wildfire Behavior and Effects",* USDA Forest Service, General Technical Report INT-30, 1976.

[Alexandridis, 2008]: A. Alexandridis, D. Vakalis, C.I. Siettos, G.V. Bafas, "*A cellular automata model for forest fire spread prediction: The case of the wildfire that swept through Spetses Island in 1990*", Applied Mathematics and Computation 204 (2008), pp 191-201.

[Anderson, 1982]: Anderson: D.H. Anderson, E.A. Catchpole, N.J. De Mestre and T. Parkes, "*Modelling the spread of grass fires",* Journal of the Australian Mathematical Society 23 (1982),pp. 451–466.

[Berjak, 2002]: Stephen G Berjak, John W Hearne, "*An improved cellular automaton model for simulating fire in a spatially heterogeneous Savanna system*", Ecological Modelling 148 (2002), pp 133-151.

[Chow, 1994]: Alex C. Chow, Bernard P. Zeigler, "*Parallel DEVS: a Parallel, Hierarchical, Modular Modelling Formalism*", Proceedings of the 1994 Winter Simulation Conference, ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila.

[Clarke, 1994]: Keith C. Clarke, James A. Brass, and Philip J. Riggan, "*A Cellular Automaton Model of Wildfire Propagation and Extinction*", Photogrammetric Engineering & Remote Sensing, Vol. 60, No. 11, November 1994, pp. 1355-1367.

[Corine L.C.]: http://www.eea.europa.eu/data-and-maps/data/corine-land-cover-2000-clc2000-seamless-vector-database

[D'Ambrosio, 2006]: D. D'Ambrosio, S. Di Gregorio, W. Spataro and G.A. Trunfio, "*A Model for the Simulation of Forest Fire Dynamics using Cellular Automata*", Proc. of the iEMSs Third Biennial Meeting: "Summit on Environmental Modelling and Software", Burlington, USA, July 2006.

[Finney, 1994]: Mark A. Finney, "*Modeling the spread and behavior of prescribed natural fires*", Proc. 12th Conf. Fire and Forest Meteorology, pp 138-143, 1994.

[Fons, 1946]: Wallace L. Fons, "*Analysis of fire spread in light forest fuels*", Journal of Agricultural Research, Vol. 72, No. 3, (1946), pp. 93-122. http://naldc.nal.usda.gov/catalog/IND43970058

[Giambiasi, 2002]: Wainer, G., and N. Giambiasi.. *"N-dimensional cell-DEVS. Discrete Events Systems: Theory and Applications", Kluwer, Vol.* 12, No 1 (2002), pp. 135–157.

[GIS]; http://en.wikipedia.org/wiki/Geographic_information_system

[Grass G.I.S.]: http://grass.osgeo.org/

[Guariso, 2002]: Giorgio Guariso & Matteo Baracani, "*A simulation software of forest fires based on two-level cellular automata*", Proceedings of the IV International Conference on Forest Fire Research, 2002 Wildland Fire Safety Summit, 18–23 November 2002, Luso, Portugal. (Ed. DX Viegas) (CD-ROM) (Millpress Science Publishers: Rotterdam, the Netherlands).

[Hargrove, 2000]: W.W. Hargrove , R.H. Gardner, M.G. Turner, W.H. Romme, D.G. Despain, "*Simulating fire patterns in heterogeneous landscapes*", Ecological Modelling 135 (2000), pp. 243 – 263.

[Innocenti, 2004]: Eric Innocenti, Antoine Aiello, Jean-Francois Santucci, David R.C.Hill, "*Active-DEVS: a computational model for the simulation of forest fire propagation*", 2004 IEEE International Conference on Systems, Man and Cybernetics, Vol. 2, pp. 1857-1863.

[Karafyllidis, 1997]: Ioannis Karafyllidis, Adonios Thanailakis, "*A model for predicting forest fire spreading using cellular automata*", Ecological Modelling 99 (1997), pp. 87-97.

[Ljiljana, 2006]: Ljiljana Bodrozic, Darko Stipanicev, Marijo Seric, "*Forest fires spread modeling using cellular automata approach*", Modern trends in control, Vratislav Hladky, Jan Paralič, Jan Vaščak, editor(s), Košice, Slovakia : equilibria (2006), pp. 23-33.

[Lopes, 2002]: A.M.G. Lopes, M.G. Cruz and D.X.Viegas, "*FireStation - An integrated software system for the numerical simulation of fire spread on complex topography*", Environmental Modelling and Software, Volume 17, Number 3, 2002, pp. 269-285.

[LUA]: http://www.lua.org

[MacLeod, 2006]: Matthew MacLeod, Rachid Chreyh, Gabriel A. Wainer, "*Improved Cell-DEVS Models for Fire Spreading Analysis",* Proceedings of ACRI. Lecture Notes in Computer Science, Volume 4173, page 472-481, Perpignan, France - September 2006.

[Muzy, 2002]: Alexandre Muzy, Gabriel A. Wainer, Eric Innocenti, Antoine Aiello, Jean-François Santucci, "*Cell-DEVS quantization techniques in a fire spreading application*", Proceedings of Winter Simulation Conference, San Diego, CA. U.S.A, Vol. 1 (2002), pp. 542-549.

[Muzy, 2005]: Alexandre Muzy, Eric Innocenti, Antoine Aiello, Jean-François Santucci, Gabriel A. Wainer, "*Specification of Discrete Event Models for Fire*

Spreading", SIMULATION: Transactions of the Society for Modeling and Simulation, Volume 81, Number 2, page 103--117 - 2005.

[Muzy, 2006]: Alexandre Muzy, Eric Innocenti, Antoine Aiello, Jean-Francois Santucci, Paul-Antoine Santoni, David R.C.Hill, "*Dynamic structure cellular automata in a fire spreading application*". Informatics in Control, Automation and Robotics I, 2006, pp 247-254.

[Netlogo]: https://ccl.northwestern.edu/netlogo/

[Pastor, 2003]: E. Pastor, L. Zárate, E. Planas, J. Arnaldos, "*Mathematical models and calculation systems for the study of wildland fire behaviour*", Progress in Energy and Combustion Science 29 (2003), pp. 139-153.

[Perry, 1999]: George L.W. Perry, Ashley D. Sparrow and Ian F. Owens, "*A GIS-supported model for the simulation of the spatial structure of wildland fire, Cass Basin, New Zealand*", Journal of Applied Ecology 36 (1999), pp. 502-518.

[Quartieri, 2010]: Joseph Quartieri, Nikos E. Mastorakis, Gerardo Iannone, Claudio Guarnaccia, "*A Cellular Automata Model for Fire Spreading Prediction*", Latest Trends on Urban Planning and Transportation (2010), pp. 173-179.

[REPAST]: http://repast.sourceforge.net

[Rothermel, 1972]: Richard C. Rothermel, "*A mathematical model for predicting fire spread in wildland fuels*", USDA Forest Service Research Paper INT-115 January 1972.

[Rothermel, 1983]: Richard C. Rothermel, "*How to Predict the Spread and Intensity of Forest and Range Fires*", Intermountain Forest and Range Experiment Station Ogden, UT 84401 General Technical Report INT-143 June 1983. http://www.fs.fed.us/rm/pubs_int/int_gtr143.pdf

[Russo, 2014]: Lucia Russo, Paola Russo, Dimitris Vakalis, Constantinos Siettos, "*Detecting Weak Points of Wildland Fire Spread: A Cellular Automata Model Risk Assessment Simulation Approach*", Chemical Engineering Transactions, Vol. 36 (2004), pp. 253-258.

[Santos]: Rafael Santos, "*Java Image Processing Cookbook*". http://www.lac.inpe.br/JIPCookbook/, link "*A Brief Tutorial on Supervised Image Classification*". INPE (Instituto Nacional de Pesquisas Espaciais, Brasil).

[TerraMe]: http://www.terrame.org/doku.php

[Ulam, 2001]: Stanislaw M. Ulam, "*Aventuras de un matemático. Memorias de Stanislaw M. Ulam*", Nivola Libros y Ediciones, S.L., ISBN: 8495599430 ISBN-13: 9788495599438, 1ª edición, 1 de octubre de 2001.

[Urquía, 2011]: Alfonso Urquía, "*Modelado de Sistemas Mediante DEVS. Teoría y práctica*". http://www.euclides.dia.uned.es/aurquia/Files/MSD_textoBase.pdf, (2011)

[Valette, 1990]: Valette J. C.. "*Inflammabilité des espèces forestières meditérranéennes, conséquences sur la combustibilité des formation forestières*", Paris : Revue forestière française (1990), 76-91.

[Van Wagner, 1969]: C. E. Van Wagner, "*A simple fire-growth model*", C.E. Forestry Chronicle (1969), pp. 103-104.

[Wainer, 1998]: Wainer, G., "*Discrete-event cellular models with explicit delays*", PhD thesis, Universite d'Aix-Marseille III, France, 1998.

[Wainer, 2001]: Gabriel A. Wainer, Norbert Giambiasi, "*Application of the Cell-DEVS paradigm for cell spaces modeling and simulation*" , Simulation, Volume 71, Number 1, page 22--39 - January 2001.

[Wainer, 2006]: *"Applying Cell-DEVS Methodology for Modeling the Environment, Gabriel A. Wainer*", SIMULATION: Transactions of the Society for Modeling and Simulation International, Volume 82, Number 10, page 635-660 - October 2006.

[Wainer, 2009]: Gabriel A. Wainer "*Discrete-Event Modeling and Simulation: A Practitioner's Approach* (first ed.)", CRC Press, (2009), ISBN 978-1-4200-5336-4.

[Yassemi, 2008]: S. Yassemi, S. Dragicevic, M. Schmidt, "*Design and implementation of an integrated GIS-based cellular automata model to characterize forest fire behaviour*", Ecological Modelling 210 (2008), pp. 71-84.

[Zeigler, 1976]: Zeigler, B. P., "*Theory of modeling and simulation"* (first ed.), 1976, Wiley Interscience, New York, ISBN 0-12-778455-1.