Simulació d'un incendi forestal

Grau en Intel·ligència Artificial

Optimització

Autors:

Eduard Barnadas Conangla, **Pau Hidalgo Pujol**

6 de maig de 2024

Índex

1 Primera sessió **2**

1.1 Introducció i objectius 2

1.2 Codi 2

1.3 Conclusions 4

2 Segona sessió **5**

2.1 Introducció i objectius 5

2.2 Generació dels fitxers 5

2.3 Expansió del foc sobre el terreny 7

2.4 Implementacions addicionals 8

2.5 Conclusions 9

1 Primera sessió

1.1 Introducció i objectius

L'objectiu d'aquesta primera sessió era implementar un autòmat cel·lular amb les regles de Wolfram Alpha, per poder analitzar el seu comportament i entendre millor com funciona. A més, es buscarà també generalitzar aquest autòmat per tal que sigui capaç de combinar diverses regles

1.2 Codi

El codi s'ha separat en tres funcions principal, més una addicional per tal de poder visualitzar els resultats utilitzant pygame i algunes auxiliars que comentarem més endavant.

La primera, i la més bàsica, és la fució rule, que donat un nombre genera la regla associada a aquell. Per fer-ho, utilitza la funció format, per transformar el nombre enter a binari. A més, la opció de format és 08b per tal d'incloure també els 8 incials i assegurar-se que té 8 bits de llargada. El seguent pas, després d'haver determinar l'índex d'aquest, és aconseguir la regla per cada cel·la específica. En aquest cas, utilitza un diccionari, que té com a claus les 8 possibles combinacions de 3 bits (en el mateix ordre que a Wolfram, és a dir: 111, 110, 101, 100, 011, 010, 001 i 000), que bàsicament són els bits de 7, 6, 5, 4, 3, 2, 1 i 0; i guarda cada element de la llista de 8 bits que haviem generat abans.

Amb això, tenim el conjunt total de regles que defineixen l'autòmata. La funció cellular_automaton aprofita aquesta, i rep el número de regla corresponent, així com durant quants passos s'ha d'executar. A continuació, genera un primer estat inicial que consisteix en tot 0 excepte un 1 al mig. Llavors, per cada pas genera l'estat següent, aplicant les regles necessàries (utilitza un join de 3 cel·les, transformades a string per funcionar com a clau del diccionari), i els va guardant en una llista.

Aquesta llista es pot visualitzar amb la funció draw_automaton, que obre una finestra de pygame i dibuixa quadrats negres on hi ha un 1 i blancs on hi ha 0 (mateix estil que a la web de Wolfram Alpha). També incorpora un *slider* que permet veure l'evolució de l'estat en cada pas (com es va generant). Per exemple, podem observar l'autòmat 30 en 100 iteracions:

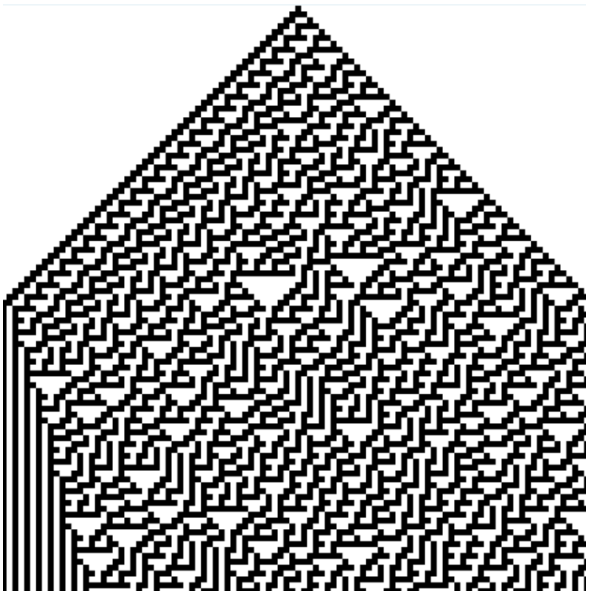

Figura 1: 100 iteracions de l'autòmat 30

Després d'obtenir aquest autòmat, se'ns demanava també poder generalitzar per combinar diverses regles. Per fer-ho, utilitzem la funció combine automatons.

Aquesta, rep 2 autòmats, ja definits, com a paràmetre. Llavors, itera per cada un dels seus estats, i els combina aplicant la regla escollida. Per tant, aquesta combinació es realitza a posteriori, és a dir: realment el que es fa és generar dues capes d'autòmates (o més, ja que es poden fer combinacions de combinacions), i s'ajunten aquestes capes amb la funció escollida.

Com a funcions implementades, hi ha or, and, nand i xor, tot i que se'n podrien crear més. A més, com hem comentat, es poden realitzar combinacions de combinacions, creant així encara més possibilitats. Aquestes combinacions creen patrons molt complexos. Podem observar-ne un exemple a continuació:

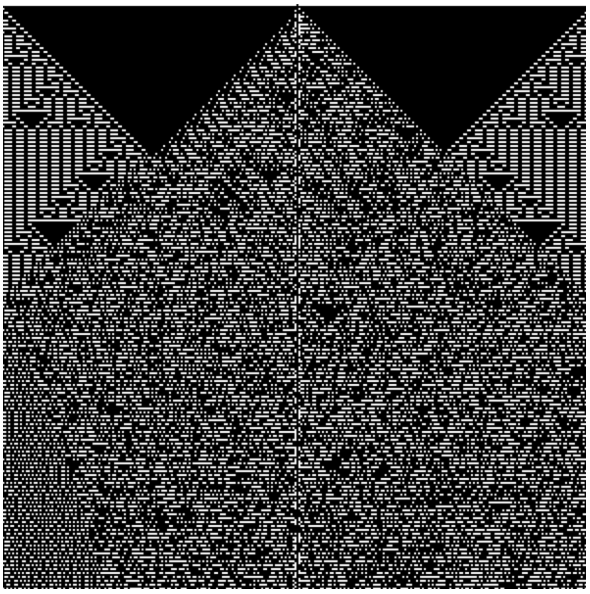

Figura 2: Estat generat per la combinació usant xor dels autòmats 129 i 30, posteriorment combinat amb l'autòmat 5 usant or, en 300 passos

El nombre de combinacions possibles, per tant, és infinit.

Addicionalment, i per tal de tenir un codi més ben estructurat, s'han ajuntat aquestes funcions en una sola classe de Python.

1.3 Conclusions

L'autòmat i les funcions implementades compleixen amb l'objectiu a la perfecció, incorporant també una forma fàcil i senzilla de poder veure aquests estats. A més, és molt eficient, i encara que li demanis autòmats amb molts de passos és capaç de generar-los sense cap problema.

Mencionar també, que la base d'aquest codi prové del ChatGPT. Per aquest motiu, hem decidit no presentar tan sols la versió en una classe (que és la que hem creat nosaltres, basant-nos en l'altra), sinó el codi que utilitza diverses funcions. Concretament està generat amb el ChatGPT4, però la versió gratuïta (3.5) també va ser capaç d'arribar a la mateixa sol·lucció. A més, la sol·lució proposada pel Chat ja d'entrada era bona, i no contenia errors de codi. Els canvis que se li han realitzat han sigut simplement de formatar una mica al nostre gust, retocar alguns paràmetres i en definitiva acabar de polir el codi. Tot i això, hem pogut veure també com aquestes eines de IA són capaçes de generar un codi correcte i que es pot utilitzar. Addicionalment, també hem comptat amb l'ajuda del Copilot, que degut a la seva integració al Visual Studio Code permet canviar algunes coses tinguent en compte el context de tot l'script.

2 Segona sessió

2.1 Introducció i objectius

En aquesta segona sessió, haurem de simular la propagació del foc en un incendi forestal. Per a això haurem de treballar amb dades en format IDRISI32 i amb dues o més capes que ens indiquin característiques sobre el terreny. En el nostre cas, aquestes seran l'humitat, el combustible (o quantitat de vegetació que hi ha en una àrea), la temperatura, l'elevació del terreny i el bioma. Cal mencionar, però, que algunes d'aquestes capes no afecten a la simulació de l'incendi (com és el cas de bioma), sinó que afecten a les altres capes. A més, durant la generació de l'incendi es crea també la capa vent, el foc i el temps de foc, i la capa hidroavió. Més endavant, comentarem què representa cada una d'aquestes.

2.2 Generació dels fitxers

Com ja hem dit prèviament, els nostres fitxers de terreny han de tenir un format en concret, aquest consisteix en tenir dos fitxers per a cada capa de terreny de la nostra simulació: un que contindrà tots els valors de les caselles en forma de columna, i un altre que ens indicarà les característiques de com és el nostre primer fitxer, és a dir, quantes files i columnes té la matriu de terreny, quin és el mínim i el màxim valor etc.

Per a constrir aquests fitxers hem definit una classe que serà l'encarregada de generar el terreny (i els seus corresponents fitxers).

En la creació, simplement guardarà la seed a utilitzar. Quan es crida el seu mètode generate data, que rep com a input un nombre natural n, que serà el nombre de fileres i columnes de les nostres matrius, i crida els mètodes de generació de cada una de les capes. La primera és l'altitud del terreny, que es genera utilitzant perlin noise ja que s'assembla molt més a la distribució real de les muntanyes (amb 8 octaves, i alguns altres paràmetres per acabar d'ajustar-ho). A continuació, s'incialitza una humitat. A partir d'aquestes dues capes, es creen la capa temperatura i bioma, que simplement tallen en funció de certs valors d'altitud i humitat i defineixen les característiques de cada tall. A més, també s'encarrega de crear els llacs (o el mar).

Els 5 biomes (contant els llacs) creats són:

- 0: mar o llacs
- 1: desert
- 2: bosc normal
- 3: jungla
- 4: alta muntanya

Per tenir una mica més de coherència, després de crear els llacs es modifica l'humitat per tal que el seu voltant en tingui més.

Finalment, es genera i guarda la vegetació. Aquesta utilitza una distribució normal, els paràmetres de la qual canvien en funció del bioma del punt corresponent. Per exemple, les muntanyes i els deserts tenen molta menys vegetació que una jungla. A més, dins els boscos també es creen petites zones de clarianes, si es vol. Per fer-ho, escull un punt aleatori del mapa, i a continuació es va expandint fins a arribar a un cert radi. Tot i això, per evitar que aquests llacs siguin rodons, té també un factor de deformació, que en cada iteració es va modificant i permet que alguns punts s'allunyin més del centre i altres menys. A vegetació també li apliquem un *filtre gaussià* per a tenir una superfície contínua, ja que així és com seria en la majoria de casos de la vida real (és a dir la vegetació i la humitat acostumen a augmentar i reduir-se gradualment).

Totes aquestes representacions són en matrius, ja que és més fàcil de generar i d'interpretar, per exemple, com a coordenades. Tot i això, als fitxers .img es guarden com una columna (que després es torna a passar a matriu).

Podem observar una generació de nombres aleatòris amb una normal:

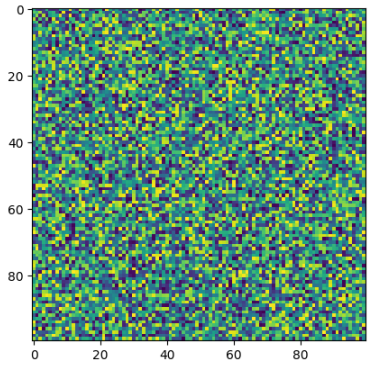

Figura 3: Plot de la matriu generada amb nombres aleatòris de numpy.

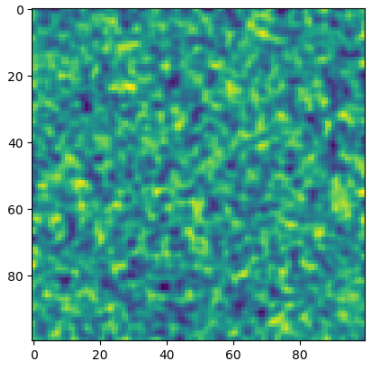

Figura 4: Plot resultant d'aplicar un filtre gaussià a la matriu de nombres aleatòris.

O bé la generació de l'altitud del terreny usant soroll de perlin:

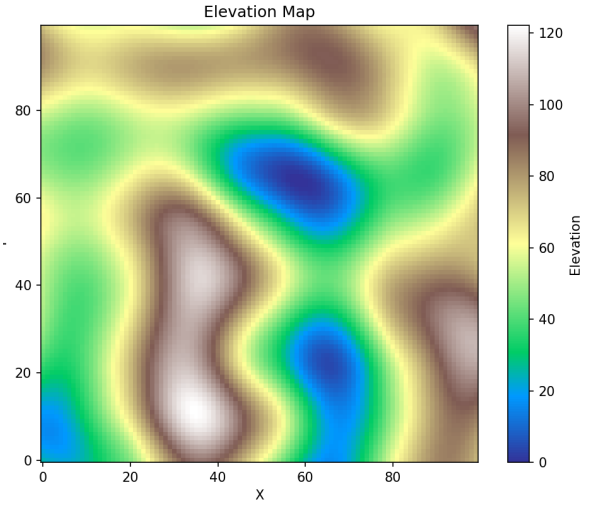

Figura 5: Generació de la seed 123

Com podem veure, la generació de nombres aleatoris escollida s'adequa a les dades, ja que cada una és més similar a com seria en la realitat la vegetació o l'altura.

2.3 Expansió del foc sobre el terreny

El codi per tal de simular l'incendi es troba al fitxer fire_simulation, que conté una classe del mateix nom. Aquesta té mètodes per generar les dades, llegir-les, simular l'incendi i visualitzar-lo. Els dos primers utilitzen la classe de l'apartat anterior.

El mètode *simulate fire*, és l'encarregat d'actualitzar totes les matrius que contenen les dades d'**humitat**, **vegetació**, *temperatura* i **foc**. Rep simplement el nombre de passos que es volen generar, la direcció del vent i el nombre d'hidroavions. Prepara les dades necessàries i crida al mètode propagate fire.

Aquest s'encarrega, com indica el seu nom, de donat un estat, propagar l'incendi. La lògica de propagació és la següent: Si una casella està encesa, li incrementa la temperatura (no és massa important). Si fa més de 10 passos que està encesa, passa a estar molt encesa (expandirà el foc més fàcilment). Llavors, per cada pas una casella encesa crema una unitat de vegetació. Si arriba a zero, la casella passa a l'estat cremat.

A continuació, mira les direccions en les que es pot extendre el vent (a partir del donat, i l'altitud, s'han calculat abans) i es mou més o menys de forma aleatòria en aquelles direccions. Per cada un dels moviments, comprova que estigui dins el rang de la matriu, i que no tingui ja foc.

Llavors, realitza un càlcul: si la umitat és més petita que 15 vegades el foc de la cel·la veïna més la temperatura entre 6 (el 15 i el 6 són valors més o menys arbitraris, escollits segons el nostre criteri, per tal que sigui aproximadament quan l'humitat és del 20%, que és quan la fusta crema bé), comença un foc allà. Sinó, disminueix la humitat un 10% i agumenta la temperatura.

D'aquesta manera, aconseguim simular l'expansió d'un foc d'una manera bastant realista, ja que

té en compte diversos factors que a més es van modificant amb el temps. Per tant, és un autòmat que té en compte diverses capes (dimensions) en la seva funció d'evolució.

Addiconalment, hi ha els hidroavions, que s'activen a partir del pas 20. Aquests tenen funcions per intentar apagar el foc, i s'explicaran més endavant.

El model SDL d'aquesta propagació seria més o menys el següent:

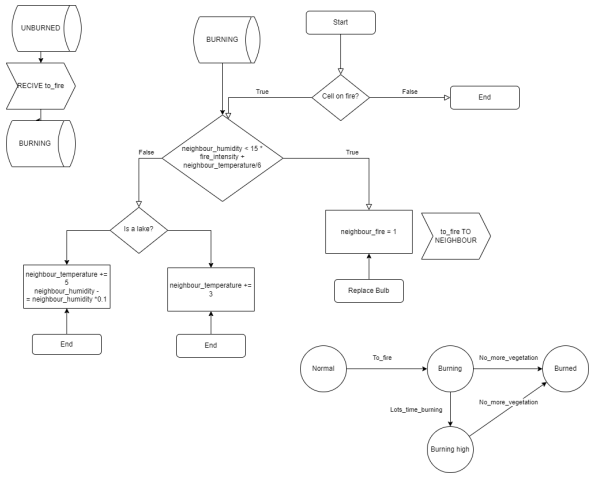

Figura 6: Pseudo SDL per representar les regles d'explansió del foc. Combina un simple arbre de decisió, que és el que fa realment el programa, amb un tros d'un diagrama d'estats i processos (que en SDL té més sentit, però no és el que fa el programa).

2.4 Implementacions addicionals

Per tal de fer la simulació lleugerament més entretinguda i simular d'una forma més realista el món real, vam optar per crear una classe hidroavio. Aquesta, com el seu nom indica, intentarà apagar o col·laborar en apagar el foc. Per fer-ho, incorpora l'habilitat de calcular les distàncies a l'existència d'aigua més propera i al foc més proper, així com la de moure's cap a aquests punts i la de descarregar aigua.

Aquesta descàrrega té un efecte d'un radi de 15, en el que si hi ha foc, aquest s'apaga en aquella cel·la, i sinó simplement simular el fet de mullar la vegetació seca, és a dir sumar-li 40 a la humiditat. Sabem que en cap cas la simulació amb hidroavions és extrapolable al mon real, ja que no ens hem basat en cap dada d'aquest (com ara la velocitat dels hidroavions o la quantitat d'aigua que despleguen), però hem cregut que seria una bona eina d'experimentació per veure si en algun cas es pot apagar el foc, contenir-lo etc.

Altres consideracions, que creiem que podrien ser una bona implementació al model, ja que també afecten als incendis, seria la implementació de camins als boscos (molts cops actuen de tallafoc), de cicles meteorològics o de dia i nit (ja que les humitats canvien, així com el vent... i aquests fets poden afectar a la simulació).

2.5 Conclusions

Després de provar amb diverses seeds, hem vist com es comportava aquest incendi d'una forma bastant entenedora. Per començar, veiem com s'expandeix evidentment de forma més ràpida per aquells llocs on hi ha menys humitat (per exemple, li costa expandir-se al voltant d'un llac). A més, els llocs amb menys vegetació li provoquen que passi a estat cremat més ràpidament, en algusn casos ni tan sols pintant-se de color vermell (no arriba a haver-hi un foc elevat allà). Aquest fet a vegades provoca que l'incendi no es continuï d'expandir, com per exemple en el cas dels deserts.

Veiem també com en general, l'expansió de l'incendi comença lentament però a mesura que Pel que fa a l'hidroavió, en alguns casos és capaç fins i tot d'aturar l'incendi del tot. Tot i això, en d'altres no ho aconsegueix, tot i que permet relantitzar una mica l'evolució d'aquest.

Podem veure unes imatges del model a continuació:

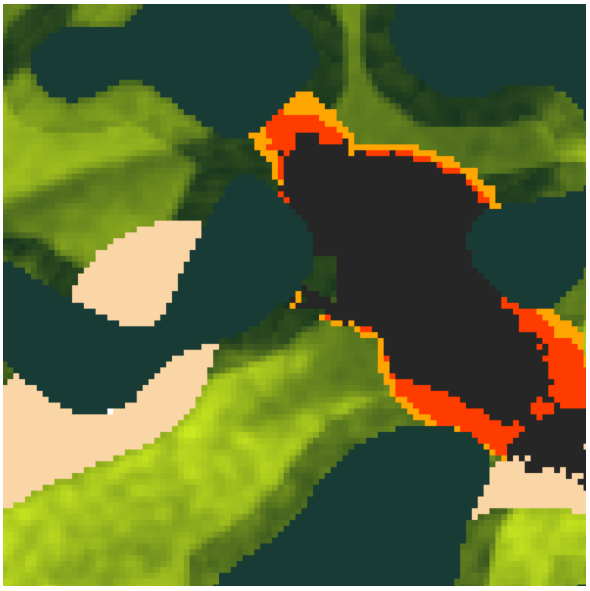

Figura 7: Imatge de mitja simulació

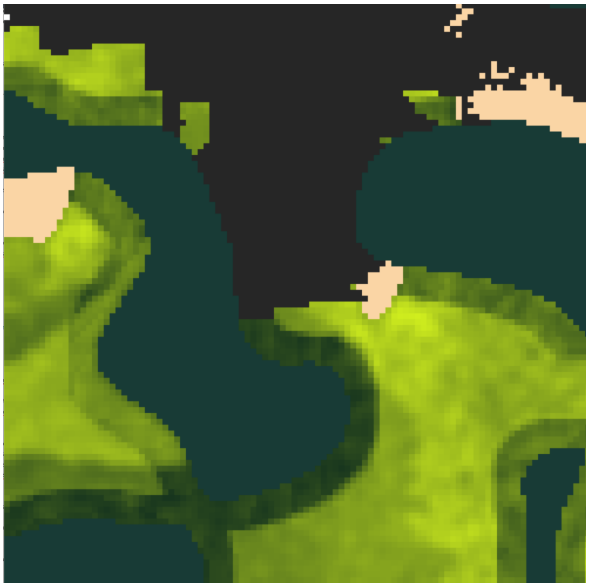

Figura 8: L'hidroavió és capaç d'apagar l'incendi

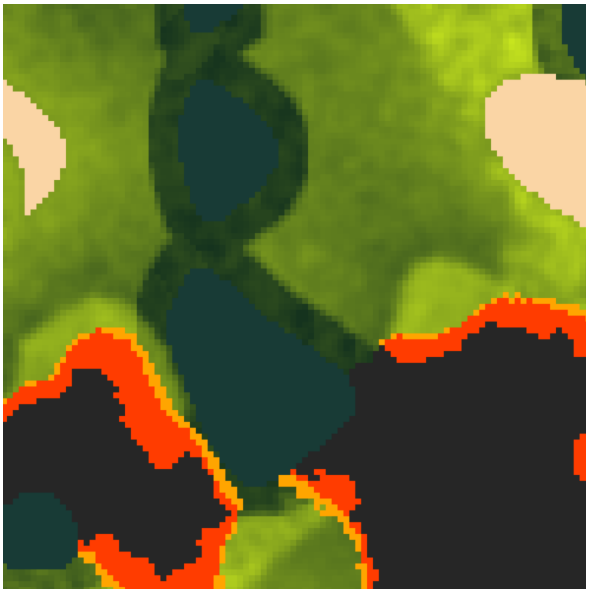

Figura 9: Es poden començar incendis des des dos punts diferents

Un fet positiu del model implmentat és que no és gaire lent, tardant tan sols uns 5 segons per una graella de 100 per 100. A més, els resultats de la generació es podrien guardar de manera relativament senzilla, tot i que no s'ha fet perquè creiem que la interpretació d'aquests resultats ja es fa amb la visualització.

A partir d'aquesta segona pràctica, hem pogut observar com un autòmat amb unes regles específiques de propagació és capaç de simular, de forma bastant realista, la propagació d'un foc. S'ha vist com es poden incorporar més d'una capa a aquest autòmat i les seves interaccions, i com afecten aquestes al resultat final.

Evidentment, no és un model perfecte, ja que en la vida real hi ha moltes variables que afecten

a aquesta propagació, però tot i això mostra una estimació de l'evolució d'aquest que pot servir per guiar una mica quines son les zones de més alt risc, creiem que els autòmats son una eina molt potent i que, emprada amb les dades reals d'incendis forestals, podria servir per predir amb molta precisió l'expansió d'aquests.

Pel que fa l'ús del Chat, en aquesta segona s'ha utilitzat menys, i en general de forma puntual tan sols per preguntar en cas de dubte sobre alguna funció (o per intentar arreglar errors). Tot i això, segurament aquesta eina seria capaç de crear codi similar, ja que hem vist que era capaç de generar codi molt complex.