UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

PRÁCTICA 3 LAB: DISEÑO DE AUTÓMATAS CELLULARES

Grado de Inteligencia Artificial

OPTIMIZACIÓN

Autores:

Luis-Kazuto Gutiérrez Kitajima (49243695M)
Adrià Flores Albó (41603787F)

Profesores:
Pau Fonseca Casas

6 de mayo de 2024

Índice

1. Introducción

Este trabajo aborda el desarrollo de modelos de autómatas celulares. Este proyecto se divide en dos sesiones distintas, cada una enfocada en aspectos específicos del proceso.

En la primera sesión, nos centraremos en la implementación de un autómata celular basado en las reglas de Wolfram, que nos permitirá simular la evolución temporal de un sistema discreto a partir de reglas simples de interacción entre células.

En la segunda sesión, ampliaremos nuestro enfoque hacia la modelización de la propagación de incendios forestales. Utilizaremos datos reales sobre humedad, vegetación y otros factores relevantes, los cuales serán procesados y utilizados como entradas para nuestro modelo de autómata celular. Exploraremos la interacción entre diferentes capas de información, representadas por matrices bidimensionales, para simular la propagación del fuego en un entorno forestal.

En ambos casos, nuestro objetivo es desarrollar modelos que nos permiten entender mejor el funcionamiento de Autómatas. Además, aprovecharemos herramientas de inteligencia artificial proporcionadas por ChatGPT, para agilizar el proceso de desarrollo y mejorar la calidad del código generado.

2. Parte 1: Diseño de un Autómata celular de Wolfram

Para este apartado, con ayuda de ChatGPT, hemos desarrollado una clase de Python que representa un autómata celular siguiendo las reglas de Wolfram, usando la librería de *PyGame* para representar el proceso de evolución de la capa.

El código es el siguiente:

```
import pygame
import numpy as np

class WolframCellularAutomata:
    def __init__(self, rule, width=800, height=400, cell_size=4):
        self.rule = rule

        self.width = width
        self.height = height
        self.cell_size = cell_size

        self.rows = self.height // self.cell_size
        self.cols = self.width // self.cell_size

        self.grid = np.zeros((self.rows, self.cols), dtype=int)
        self.grid[0, self.cols // 2] = 1
        self.current_row = 0

    def apply_rule(self, left, center, right):
        pattern = 4 * left + 2 * center + right
        return (self.rule >> pattern) & 1

    def update(self):
        if self.current_row < self.rows - 1:
            for i in range(1, self.cols - 1):
                left = self.grid[self.current_row, (i - 1) % self.cols]
                center = self.grid[self.current_row, i]
                right = self.grid[self.current_row, (i + 1) % self.cols]
                self.grid[self.current_row + 1, i] = self.apply_rule(left, center,
                    right)
            self.current_row += 1

    def draw(self, screen):
        for i in range(self.cols):
            for j in range(self.rows):
                if self.grid[j, i]:
                    pygame.draw.rect(screen, (255, 255, 255), (i * self.cell_size, j
                        * self.cell_size, self.cell_size, self.cell_size))

        font = pygame.font.Font(None, 36)
        rule_text = font.render("Rule " + str(self.rule), True, (255, 255, 255))
        screen.blit(rule_text, (10, 10))
```

```
41
42      def run(self):
43          pygame.init()
44          screen = pygame.display.set_mode((self.width, self.height))
45          clock = pygame.time.Clock()
46
47          running = True
48          while running:
49              for event in pygame.event.get():
50                  if event.type == pygame.QUIT:
51                      running = False
52
53              self.update()
54
55              screen.fill((0, 0, 0))
56              self.draw(screen)
57              pygame.display.flip()
58              clock.tick(50)
59
60          pygame.quit()
61
62
63
64  rule = 30
65  width, height = 800, 400
66  cell_size = 5
67
68  ca = WolframCellularAutomata(rule, width, height, cell_size)
69  ca.run()
```

- **__init__():** Este método toma como parámetros el 'rule' que determina el comportamiento del autómata celular, así como el 'width', 'height' y 'cell_size' para definir las dimensiones de la cuadrícula. Inicializa la cuadrícula con ceros y establece el estado inicial del autómata colocando un 1 en el centro de la primera fila.

- **apply_rule():** Este método toma tres valores booleanos, 'left', 'center' y 'right', que representan el estado de las celdas vecinas. Utiliza estos valores para calcular el estado de la celda central en la siguiente fila, de acuerdo con la regla especificada. La regla se aplica utilizando un patrón de bits que representa el estado de las celdas vecinas.

- **update():** Este método actualiza el estado de todas las celdas en la siguiente fila de la cuadrícula. Itera sobre cada celda en la fila actual y utiliza el método 'apply_rule()' para determinar su estado en la siguiente fila.

- **draw():** Este método se encarga de representar gráficamente el autómata celular en una ventana de pygame. Itera sobre todas las celdas en la cuadrícula y dibuja un rectángulo blanco en las posiciones correspondientes a las celdas activas.

- **run():** Este método ejecuta el autómata celular en un bucle de pygame. Inicializa la ventana de pygame y el reloj, y luego ejecuta un bucle principal en el que se actualiza y dibuja el autómata en cada iteración del bucle. El bucle se ejecuta hasta que se cierra la ventana de pygame.

Podemos ver un ejemplo de nuestra ejecución para la regla número 90 a continuación:

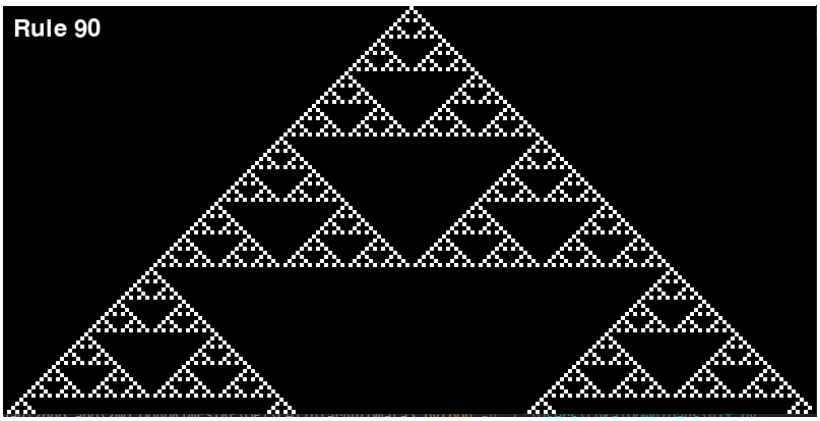

Figura 1: representación de la regla 90 con nuestro problema

El siguiente paso en esta práctica consistía en la capacidad de combinar reglas de Wolfram, lo que resulta en la creación de una nueva capa donde se utilizan ya sea una intersección o una unión de las reglas. Para poder implementar esto en nuestra clase, cambiamos lo siguiente:

- **__init__():** Añadimos los parámetros 'rule1', 'rule2' y 'mode', que indican las dos reglas que queremos juntar, i el metodo de fusión (iterseccion o unión). Ajustado el cálculo de 'self.cols' para que quepan las tres capas en la representación de *PyGame*. Creamos las matrices 'self.grid1', 'self.grid2' y 'self.combined_grid' para poder actualizar las capas de cada regla.
- **apply_rule():** como ahora mas reglas usaran este método, convertimos la regla en un parámetro 'rule'.
- **update():** Ahora este método actualizará las primeras dos capas usando el método apply_rule(). La capa que combina las dos actualizará su capa según el método de fusión que se le ha proporcionado.
- **draw():** Modificamos la lógica de dibujo para representar las tres capas del autómata celular. Ajustamos las coordenadas de dibujo para cada capa según el ancho extendido de la pantalla.

El codigo final queda de la siguiente manera:

```
1 import pygame
2 import numpy as np
3
4
5
6
7
```

```
class WolframCellularAutomataCombo:
    def __init__(self, rule1, rule2, mode='union', width=1200, height=400, cell_size=4):
        self.rule1 = rule1
        self.rule2 = rule2

        self.mode = mode
        self.width = width
        self.height = height

        self.cell_size = cell_size
        self.rows = self.height // self.cell_size
        self.cols = self.width // (3 * self.cell_size)

        self.grid1 = np.zeros((self.rows, self.cols), dtype=int)
        self.grid2 = np.zeros((self.rows, self.cols), dtype=int)
        self.combined_grid = np.zeros((self.rows, self.cols), dtype=int)

        self.grid1[0, self.cols // 2] = 1
        self.grid2[0, self.cols // 2] = 1
        self.combined_grid[0, self.cols // 2] = 1
        self.current_row = 0

    def apply_rule(self, rule, left, center, right):
        pattern = 4 * left + 2 * center + right
        return (rule >> pattern) & 1

    def update(self):
        if self.current_row < self.rows - 1:
            for i in range(1, self.cols - 1):
                left1 = self.grid1[self.current_row, (i - 1) % self.cols]
                center1 = self.grid1[self.current_row, i]
                right1 = self.grid1[self.current_row, (i + 1) % self.cols]
                self.grid1[self.current_row + 1, i] = self.apply_rule(self.rule1, left1, center1, right1)

                left2 = self.grid2[self.current_row, (i - 1) % self.cols]
                center2 = self.grid2[self.current_row, i]
                right2 = self.grid2[self.current_row, (i + 1) % self.cols]
                self.grid2[self.current_row + 1, i] = self.apply_rule(self.rule2, left2, center2, right2)

                if self.mode == 'union':
                    self.combined_grid[self.current_row + 1, i] = max(self.grid1[self.current_row + 1, i], self.grid2[self.current_row + 1, i])
                elif self.mode == 'intersection':
                    self.combined_grid[self.current_row + 1, i] = min(self.grid1[self.current_row + 1, i], self.grid2[self.current_row + 1, i])

            self.current_row += 1

```

```
    def draw(self, screen):
        for i in range(self.cols):
            for j in range(self.rows):
                # Dibuja la primera capa (rule1)
                if self.grid1[j, i]:
                    pygame.draw.rect(screen, (255, 0, 0), (i * self.cell_size, j *
                        self.cell_size, self.cell_size, self.cell_size))
                # Dibuja la segunda capa (rule2)
                if self.grid2[j, i]:
                    pygame.draw.rect(screen, (0, 255, 0), ((i + self.cols) * self.
                        cell_size, j * self.cell_size, self.cell_size, self.cell_size
                        ))
                # Dibuja la tercera capa (combinada)
                if self.combined_grid[j, i]:
                    pygame.draw.rect(screen, (0, 0, 255), ((i + 2 * self.cols) * self
                        .cell_size, j * self.cell_size, self.cell_size, self.
                        cell_size))

        # Dibuja l neas de separaci n entre las capas
        pygame.draw.line(screen, (255, 255, 255), (self.cols * self.cell_size - 1, 0)
            , (self.cols * self.cell_size - 1, self.height))
        pygame.draw.line(screen, (255, 255, 255), ((2 * self.cols) * self.cell_size -
             1, 0), ((2 * self.cols) * self.cell_size - 1, self.height))

        #Texto que indica cada regla
        font = pygame.font.Font(None, 24)
        rule1_text = font.render("Rule " + str(self.rule1), True, (255, 255, 255))
        rule2_text = font.render("Rule " + str(self.rule2), True, (255, 255, 255))
        combined_text = font.render("Combined: "+ str(self.mode), True, (255, 255,
            255))
        screen.blit(rule1_text, (10, 10))
        screen.blit(rule2_text, (self.cols * self.cell_size + 10, 10))
        screen.blit(combined_text, ((2 * self.cols) * self.cell_size + 10, 10))

    def run(self):
        pygame.init()
        screen = pygame.display.set_mode((self.width, self.height))
        clock = pygame.time.Clock()

        running = True
        while running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False

            self.update()

            screen.fill((0, 0, 0))
            self.draw(screen)
            pygame.display.flip()
            clock.tick(60)
```

```
97
98          pygame.quit()
99
100
101
102 rule1 = 238
103 rule2 = 90
104 mode = 'union'
105
106 width, height = 1200, 400
107 cell_size = 2
108
109 ca = WolframCellularAutomataCombo(rule1, rule2, mode, width, height, cell_size)
110 ca.run()
```

También ofrecemos un ejemplo de nuestra ejecución a continuación:

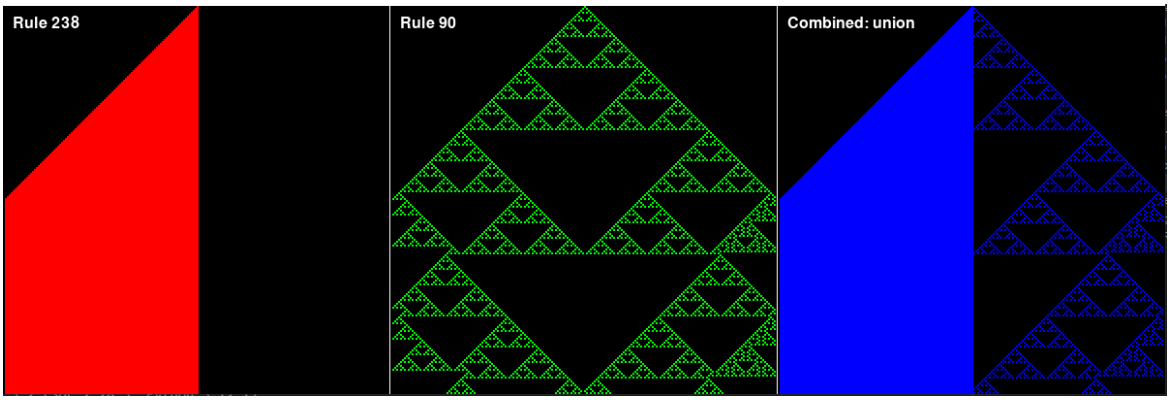

Figura 2: Ejemplo de ejecución de Union de nuestro Programa

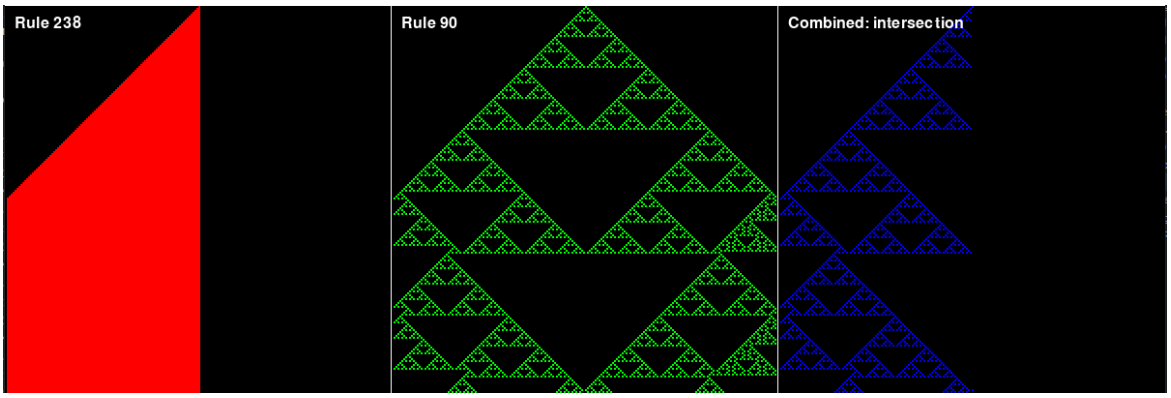

Figura 3: Ejemplo de ejecución de Intersección de nuestro Programa

3. Parte 2: Simular incendio forestal usando un Autómata Celular

En este apartado, explicamos el diseño de un autòmata celullar multicapa que nos permite simular un incendio forestal.

3.1. Introducción

Esta sección describe una simulación computarizada diseñada para modelar la propagación de incendios forestales utilizando datos reales, a través de un modelo dinámico basado en autómatas celulares. Esta simulación tiene como objetivo explorar la interacción entre diversos factores ambientales, como la humedad y el combustible, y su impacto en la dinámica y comportamiento del fuego en un entorno forestal.

Utilizando datos estructurados en el formato IDRISI32, el proyecto implementa un modelo que gestiona múltiples capas de información interactivas para visualizar de manera efectiva la expansión del fuego en un espacio bidimensional. Este enfoque permite no solo una representación visual detallada sino también un análisis más profundo de cómo los cambios en variables específicas pueden afectar la propagación del incendio.

Las simulaciones de incendios forestales son herramientas cruciales en la planificación y gestión de recursos para la prevención y control de incendios. Al proporcionar una plataforma para simular y entender el comportamiento del fuego bajo diferentes condiciones, estas simulaciones ayudan a mejorar las estrategias de respuesta y preparación ante emergencias reales. Además, ofrecen valiosas perspectivas para el diseño de políticas efectivas y la formación de personal en técnicas de manejo y extinción de incendios.

Este proyecto busca demostrar la utilidad de los autómatas celulares en la modelización de sistemas complejos y dinámicos, proporcionando una herramienta que facilita la comprensión de fenómenos naturales complejos como los incendios forestales. A lo largo de este informe, se detalla cómo se desarrolló la simulación, se describe su implementación técnica y se presentan las conclusiones derivadas de la experimentación con el modelo.

3.2. Objetivos

El objetivo de este proyecto es desarrollar una simulación de autómatas celulares que modele la propagación de incendios forestales en un entorno bidimensional. Esta simulación tiene como propósito explorar cómo las variables ambientales como la humedad y el combustible influyen en la dinámica del fuego. Al ajustar estos factores dentro de un modelo controlado, buscamos obtener una comprensión más clara de cómo afectan a la propagación del incendio.

Los objetivos específicos del proyecto son los siguientes:

- **Visualización Dinámica:** Proveer una representación visual que demuestre cómo el fuego se expande bajo diferentes condiciones ambientales. Esto permitirá a los usuarios ajustar parámetros y observar directamente cómo influyen en la propagación del fuego.
- **Análisis de Interacciones:** Investigar cómo la interacción entre múltiples capas de datos, como la humedad y el combustible, determina la velocidad y el patrón de la propagación del fuego.
- **Evaluación de Parámetros:** Usar el modelo para examinar cómo distintas configuraciones de variables pueden modificar la propagación del fuego, identificando configuraciones críticas que afectan el comportamiento del incendio.
- **Fomento del Entendimiento:** Mediante la simulación, mejorar la comprensión de cómo las condiciones ambientales afectan eventos complejos como los incendios forestales.

Este proyecto se enmarca en el desarrollo de una herramienta de simulación que permite visualizar y analizar el comportamiento del fuego bajo distintas condiciones, proporcionando una plataforma para explorar y entender mejor los efectos de diversas variables ambientales en la propagación del incendio.

3.3. SDL

Inicialmente empezamos el proyecto elaborando un modelo SDL ligeramente simplificado manualmente para ayudarnos a organizar el esqueleto de la implementación y comprender detenidamente el problema. En esta pequeña sección explicaremos brevemente las partes principales del esquema que, por tanto, también nos ayudarán a comprender la estructura principal de la simulación utilizando una autómata celular multicapa, olvidando inicialmente el tema de los generadores o cómo representar la simulación siquiera.

El esquema inicia con un estado inicial y se dirige al estado 'avanzar' que representará cada iteración temporal discreta de la simulación, situados en este estado pasamos por un condicional que comprueba si la simulación debe terminar, ya sea porque no quedan bloques quemándose o bien se ha adquirido un número de iteraciones exigido por el usuario. En caso de que la simulación deba proseguir, se itera por todas las celdas que se están quemando, las cuales perderán una cierta parte de vegetación, y además sus celdas vecinas también se verán afectadas por el fuego perdiendo humedad o bien empezando a arden y añadiéndose a la cola de celdas quemándose. También será necesario comprobar si la celda ha perdido toda la vegetación para configurarla en estado quemado. Una vez hayamos hecho una iteración de este tipo, comprobando lo que hemos comentado, volveremos al estado 'avanzar' y así sucesivamente.

3.4. Implementación

En esta sección del informe, nos enfocaremos en detallar y analizar la implementación del código en Python que constituye el núcleo de nuestra simulación de incendios forestales. El objetivo es proporcionar una visión clara de cómo las distintas partes del código interactúan para modelar la dinámica del fuego en un entorno bidimensional y cómo las variables ambientales, como la humedad y el combustible, son integradas y gestionadas dentro del sistema.

El código se ha estructurado de manera que refleja la lógica subyacente del modelo de autómatas celulares y su capacidad para simular la propagación del fuego bajo condiciones controladas. A través de una serie de funciones y clases, describiremos cómo se inicializan los datos, se procesan las interacciones y se visualizan los resultados de la simulación. Este análisis nos permitirá entender los mecanismos técnicos que soportan las observaciones y conclusiones obtenidas del modelo.

Cada segmento de código será examinado para explicar su propósito y funcionamiento dentro del contexto más amplio del proyecto. Esta descomposición no solo aclarará cómo se lleva a cabo la simulación, sino que también demostrará cómo se pueden ajustar y modificar los parámetros para explorar diferentes escenarios y resultados.

El archivo de constantes establece una base crucial para nuestra simulación de incendios forestales, definiendo una serie de parámetros que influyen directamente en el comportamiento y las condiciones del modelo. Al especificar variables como los estados del fuego y las dimensiones de la simulación, este archivo proporciona los elementos fundamentales que dictan cómo se inicializa y se ejecuta el proceso de simulación.

En la simulación, utilizamos diferentes estados para representar la condición de cada celda en el área de simulación: pendiente de quemar, en proceso de quemar y quemado. Estos estados son esenciales para controlar la progresión del fuego a través del modelo. La configuración del espacio de simulación se define mediante las dimensiones especificadas, lo que permite determinar el tamaño del área en la que el fuego puede propagarse, y el tamaño de cada celda afecta la granularidad de la visualización y la simulación.

La cantidad de inicios de fuego y la humedad máxima son parámetros que permiten variar las condiciones iniciales de la simulación, afectando cómo y dónde puede iniciar y propagarse el fuego. Estos aspectos son vitales para explorar diferentes escenarios de incendios y comprender la dinámica del fuego bajo diversas condiciones ambientales.

La incorporación de biomas en el modelo añade una capa adicional de complejidad y realismo. Cada bioma, definido con características específicas de propagación, vegetación, humedad y color, permite simular de manera más precisa cómo diferentes entornos naturales pueden influir en la propagación del fuego. Los biomas como el desierto, la vegetación y la selva se diferencian por su capacidad de combustión y resistencia al fuego, reflejando la diversidad de respuestas al fuego que se observan en la naturaleza. Como se puede observar los valores y la configuración de los biomas no está pensada inicialmente para ser realista, ya que no es realista que desierto y selva se encuentren en espacios cercanos, pero la idea de esta implementación es poder representar diferentes tipos de vegetación o terreno con diferentes factores de propagación, vegetación y humedad. Lo representamos como desierto y selva con sus respectivos colores para hacerlo más visual, pero básicamente mostramos que fácilmente se pueden computar todos los deferentes biomas que se desee en la simulación

Una paleta de colores específica se utiliza para visualizar el estado de las celdas durante la simulación, proporcionando una representación visual intuitiva de la intensidad y el calor del fuego. Esta visualización es clave para interpretar rápidamente el progreso y los efectos del incendio en diferentes áreas del modelo.

El archivo de hiperparámetros, por lo tanto, no solo define los valores utilizados a lo largo de la simulación sino que también establece las suposiciones bajo las cuales se realiza el modelo. Las configuraciones de los biomas, la humedad y los inicios de fuego se basan en suposiciones sobre las condiciones naturales, lo que puede influir en la generalización de los resultados. Ajustar estos

parámetros permite a los usuarios explorar cómo diferentes configuraciones afectan la dinámica del fuego, proporcionando una herramienta valiosa para estudiar y entender mejor los incendios forestales.

Con estas bases establecidas, el siguiente paso es proceder a la implementación del generador que utiliza estas constantes para configurar el entorno inicial en el que el fuego será simulado, permitiendo estudiar la interacción entre el fuego y el entorno de manera efectiva y educativa.

El generador de la simulación es un componente crucial que prepara el entorno para la simulación de incendios forestales, estableciendo las condiciones iniciales que reflejan tanto las características naturales como las configuraciones específicas deseadas para el estudio. Este proceso comienza con la creación de un mapa de biomas, donde el espacio de simulación se divide en secciones, y a cada sección se le asigna un bioma de acuerdo a probabilidades predefinidas. Cada bioma afecta de manera significativa las propiedades de las celdas, como la capacidad de combustión y la humedad, lo que tiene un impacto directo en cómo el fuego se propaga y se comporta dentro de esas áreas.

Para iniciar el fuego, el generador establece varios puntos de inicio aleatorios dentro del modelo, marcando ciertas celdas como activamente quemándose. La cantidad y ubicación de estos puntos de fuego son configurables, permitiendo la exploración de cómo diferentes concentraciones y distribuciones del fuego afectan la dinámica general de la propagación del incendio. Este aspecto es vital para entender las respuestas potenciales del entorno ante un incendio bajo diversas condiciones de inicio.

Simultáneamente, se configuran la vegetación y la humedad en el terreno, asignando valores específicos a cada celda basados en el bioma al que pertenecen. Esta distribución detallada asegura que la simulación refleje diferencias reales en la inflamabilidad y la resistencia al fuego entre distintos tipos de terrenos. Además, se introducen variaciones locales, como áreas con alta humedad representadas por lagos, utilizando una función que dibuja círculos de humedad en el mapa, lo cual afecta la capacidad del fuego para propagarse a través de estas zonas más húmedas.

Un paso final en la configuración es la aplicación de un filtro gaussiano sobre la humedad, lo que suaviza las transiciones entre áreas con diferentes niveles de humedad y crea un efecto más natural y realista. Este enfoque no solo mejora la visualización de la simulación, sino que también proporciona una representación más precisa de cómo la humedad podría distribuirse en un entorno natural y afectar la propagación del fuego.

Este generador es fundamental porque permite una considerable flexibilidad en la configuración del modelo y facilita la experimentación con variados escenarios de incendios. Las suposiciones inherentes en la distribución de biomas y características ambientales, aunque necesarias para la simulación, deben ser consideradas al evaluar la aplicabilidad de los resultados. La capacidad de ajustar estos parámetros permite a los investigadores y estudiantes explorar en profundidad la dinámica del fuego

y desarrollar una comprensión más rica de los factores que influyen en su comportamiento.

Con el entorno debidamente configurado por el generador, el modelo está ahora listo para ejecutarse, permitiendo observar la propagación del fuego y evaluar cómo las interacciones entre diferentes variables ambientales afectan este proceso. Esta fase de la simulación proporciona una herramienta valiosa para experimentar con estrategias de gestión de incendios y para educar sobre la prevención y el control de incendios de manera segura y controlada.

Avanzando en la implementación de nuestro proyecto de simulación de incendios forestales, el archivo principal desempeña un papel esencial al conectar la configuración inicial del entorno con la ejecución dinámica de la simulación. Este archivo comienza su trabajo cargando los datos generados por el generador, que incluyen biomas, vegetación, humedad y el estado inicial del fuego. Estos datos son fundamentales para establecer las condiciones adecuadas para iniciar la simulación, reflejando las variaciones ambientales que afectarán el comportamiento del fuego.

Una vez cargados, estos datos se organizan en estructuras adecuadas para ser utilizadas por la clase de simulación. Esta clase toma los datos y configura el entorno del modelo, estableciendo dónde ha comenzado el fuego, qué áreas presentan mayor humedad y dónde la vegetación es más densa. Cada uno de estos elementos influirá significativamente en cómo el fuego se propaga, impactando la velocidad a la que se mueve el fuego, su intensidad y la probabilidad de extensión a áreas adyacentes.

Por otro lado, se asume que los datos ingresados son representativos y precisos, lo cual es crucial para la validez de la simulación. Se presupone una correcta distribución de biomas y una asignación fiel de valores de humedad y vegetación que correspondan con las características de cada bioma. Estas suposiciones son esenciales ya que errores en los datos de entrada podrían llevar a resultados de simulación que no serían fiables o representativos del comportamiento real del fuego.

Este enfoque modular, donde un componente del sistema prepara los datos y otro los consume para ejecutar la simulación, mejora notablemente la organización del código y facilita tanto su prueba como su mantenimiento. Permite a los usuarios experimentar con variaciones en los datos de entrada sin alterar la lógica interna de la simulación, promoviendo así una exploración más profunda y un entendimiento mejorado del comportamiento del fuego bajo diversas condiciones.

Con los datos ya cargados y el entorno de la simulación listo, se establecen las bases para investigaciones detalladas sobre la dinámica del fuego. Este proceso no solo permite evaluar cómo diferentes condiciones iniciales afectan la propagación del incendio, sino que también es invaluable para el desarrollo de estrategias de prevención y control de incendios forestales. Al final, este análisis detallado prepara el escenario para la ejecución de la simulación, donde los resultados podrán ser analizados para obtener insights críticos sobre la gestión efectiva de incendios forestales.

Con esta descripción continuada y conectada, hemos cubierto cómo el archivo principal integra la preparación del entorno con la ejecución de la simulación, proporcionando una visión completa de cómo los diferentes componentes trabajan conjuntamente para simular y estudiar incendios forestales de manera eficaz.

La clase Incendi_Forestal constituye el núcleo de nuestro proyecto de simulación de incendios forestales, integrando y manipulando los diversos elementos configurados como el estado del fuego, la vegetación, la humedad y los biomas para simular la propagación del fuego en un entorno controlado. Iniciando con la inicialización del entorno de simulación, la clase establece el área de trabajo y los parámetros iniciales utilizando la biblioteca pygame para crear una interfaz visual donde se mostrarán los cambios en el entorno a medida que avanza la simulación. Esta visualización es fundamental para entender cómo diferentes condiciones afectan la propagación del fuego.

A medida que la simulación se ejecuta, el método update_world se encarga de revisar y actualizar el estado de cada celda, aplicando factores como la vegetación existente y la humedad, que están directamente influenciados por las características del bioma asignado a cada celda. Esto permite que la simulación refleje cómo diferentes entornos naturales responderían al fuego, abordando la propagación a celdas adyacentes y ajustando su estado en función de las condiciones existentes.

Simultáneamente, el método draw_world actualiza la representación visual del estado de la simulación en la ventana de pygame, modificando los colores de las celdas para reflejar los estados actualizados de quemado, en proceso de quemar o no quemado. Esta funcionalidad no solo es visualmente informativa sino también crítica para evaluar la dinámica de propagación y la efectividad de posibles estrategias de contención del fuego.

Una característica distintiva de esta clase es su capacidad de interacción con el usuario, permitiendo pausar, reanudar y ajustar la velocidad de la simulación mediante teclas específicas. Esta interactividad hace que la simulación sea más accesible y proporciona una herramienta educativa valiosa, demostrando en tiempo real cómo los cambios en el entorno afectan la propagación del fuego.

Asumiendo que los datos de entrada son precisos y representativos de las condiciones reales, la clase Incendi_Forestal está diseñada para ser flexible y permitir ajustes en los parámetros de bioma y otras variables. Esta flexibilidad es crucial para explorar cómo las variaciones en estos parámetros pueden influir en los resultados de la simulación, proporcionando un medio para estudios detallados sobre la dinámica del fuego.

Con la finalización de esta clase, hemos establecido una estructura robusta para ejecutar simulaciones dinámicas y visualmente ricas de incendios forestales. Esta clase no solo maneja la lógica de

propagación del fuego y su visualización sino que también ofrece herramientas para modificar la simulación en tiempo real, enriqueciendo la experiencia del usuario y profundizando la comprensión del comportamiento del fuego. Este enfoque integrado proporciona no solo una plataforma efectiva para simular eventos complejos, sino también un recurso educativo y experimental para estudiar y gestionar incendios forestales.

3.5. Conclusiones

Con la culminación de este proyecto de simulación de incendios forestales, hemos logrado desarrollar una herramienta sofisticada y educativa que permite visualizar y analizar la dinámica de la propagación del fuego en un entorno controlado. Gracias a la integración de componentes como la generación de biomas, la manipulación de las condiciones ambientales y la visualización interactiva, este proyecto no solo demuestra cómo diferentes factores como la vegetación, la humedad y los tipos de bioma afectan la propagación del fuego, sino que también proporciona una plataforma valiosa para la exploración y el aprendizaje.

Logros del Proyecto

- **Comprensión Dinámica del Fuego:** Hemos implementado un modelo de simulación que refleja con precisión el comportamiento del fuego bajo diversas condiciones ambientales. Esto permite a los usuarios ver en tiempo real cómo el fuego se expande o se contiene en respuesta a las variables modificadas.
- **Herramienta Educativa:** El proyecto sirve como un recurso educativo excepcional para estudiantes y profesionales interesados en la gestión de incendios forestales, la ecología del fuego y la planificación del uso del suelo. La capacidad de interactuar con la simulación y ver los efectos inmediatos de los cambios en las condiciones ambientales mejora la comprensión de los principios críticos de la ciencia del fuego.
- **Plataforma de Experimentación:** Ofrece una plataforma segura y controlada para experimentar con estrategias de prevención y contención de incendios, lo cual es crucial para el desarrollo de políticas y técnicas más efectivas en la gestión de incendios forestales.

A partir de lo que hemos conseguido, hay varias direcciones prometedoras que este proyecto podría tomar:

- **Expansión de Datos y Modelos:** Integrar datos reales de incendios pasados y condiciones meteorológicas para hacer que la simulación sea aún más precisa y relevante para situaciones específicas.
- **Desarrollo de Estrategias de Contención:** Utilizar la simulación para probar y desarrollar nuevas estrategias de contención de incendios, lo que podría llevar a mejoras significativas en la respuesta a emergencias y la planificación de la gestión del terreno.
- **Interfaz de Usuario Mejorada:** Ampliar la interfaz de usuario para incluir más opciones de personalización y análisis, permitiendo a los usuarios ajustar más parámetros y extraer datos específicos de sus simulaciones para análisis más detallados.
- **Colaboración Multidisciplinaria:** Colaborar con expertos en ecología, meteorología y gestión de desastres para enriquecer el modelo con perspectivas interdisciplinarias y aumentar su

utilidad práctica.

En conclusión, este proyecto no solo ha demostrado ser una herramienta efectiva para entender y visualizar la propagación del fuego, sino que también establece una base sólida para futuras investigaciones y desarrollos en el campo de la gestión y estudio de incendios forestales. La flexibilidad y la capacidad educativa de esta simulación la hacen invaluable para cualquier esfuerzo dirigido a mejorar la forma en que comprendemos y respondemos a los incendios forestales.