Quintavalle Pietro - Forest Fire Simulation

1. Design the pseudocode for the forest fire model

Initialize Grid Function

1. **Define the Function**: Specify the input parameters (e.g., grid dimensions) and the expected output (the initialized grid).
2. **Create the Grid**: Initialize an $m \times n$ grid.
3. **Assign Cell Types**: Assign different types to the cells in the grid to represent water bodies, dry grass, dense trees, burning areas, and normal forest.
4. **Return the Grid**: Return the initialized grid.

Algorithm Initialize Grid

function InitializeGrid(m, n, prob_tree, prob_burning, prob_water, prob_dry_grass, prob_dense_trees)
  Create an empty $m \times n$ grid
  for each cell (i, j) in the grid **do**
    Generate a random number $rand \in [0, 1)$
    if $rand <$ prob_tree **then**
      if $rand <$ prob_burning **then**
        Set grid$[i][j]$ to 'Burning'
      else
        Set grid$[i][j]$ to 'NormalForest'
      end if
    else if $rand <$ prob_tree + prob_water **then**
      Set grid$[i][j]$ to 'Water'
    else if $rand <$ prob_tree + prob_water + prob_dry_grass **then**
      Set grid$[i][j]$ to 'DryGrass'
    else if $rand <$ prob_tree + prob_water + prob_dry_grass + prob_dense_trees **then**
      Set grid$[i][j]$ to 'DenseTrees'
    else
      Set grid$[i][j]$ to 'Empty'
    end if
  end for
  return grid
end function

Plot Grid Function

The `plot_grid` function is designed to visualize the current state of the forest grid, using different colors to represent different types of areas.

Algorithm Plot Grid Function

function PlotGrid(grid)
  for each cell (i, j) in grid **do**
    Determine the color based on grid$[i][j]$
    Plot the cell at position (i, j) with the determined color
  end for
  Display the plot
end function

In this pseudocode:

- We define a color mapping to associate each type of cell with a specific color.
- We loop through each cell in the grid, determine its type, and get the corresponding color from the color mapping.
- We then plot each cell at its respective position using the specified color.
- Finally, we show the plot.
  This function provides a visual representation of the forest grid, making it easier to understand the current state of the forest and observe how it changes over time during the simulation.

Neighbours Function

The `neighbours` function is designed to identify and return the neighboring cells of a given point in the grid. Neighboring cells can be defined in various ways, but for this example, we'll consider the cells surrounding a given cell (horizontal, vertical, and diagonal neighbors) as its neighbors.

Algorithm Neighbours Function

function Neighbours(grid, i, j)
  Initialize a list neighbors to store neighboring cells
  for each neighboring position (x, y) of (i, j) **do**
    if (x, y) is inside the boundaries of grid **then**
      Add grid$[x][y]$ to neighbors
    end if
  end for
  return neighbors
end function

In this pseudocode:

- We first determine the dimensions of the grid.
- We initialize an empty list to store the coordinates of the neighboring cells.
- We define the relative positions of the neighboring cells with respect to a given cell.
- We loop through each relative position to find the neighboring cells, ensuring they are within the bounds of the grid.
- Finally, we return the list of neighboring cells.
  This function will be crucial for determining how the fire spreads throughout the forest during the simulation.

Propagate Function

The `propagate` function is designed to manage fire propagation based on the type of area, its current state, and the states of its neighboring cells.

Algorithm Propagate Function

function PROPAGATE(grid, $i, j, p, pstart$)
  Calculate the probability of cell (i, j) catching fire based on neighboring burning cells and p
  Add $pstart$ to the calculated probability
  Generate a random number $rand$
  if $rand <$ calculated probability **then**
    Set grid$[i][j]$ to 'Burning'
  end if
end function

In this pseudocode:

- We start by getting the current state of the cell.
- If the cell is currently `Burning`, it will transition to `Burned` in the next state.
- If the cell is `Burned` or `Water`, it retains its state.
- We then get the neighboring cells and check if any of them are 'Burning'. If a burning neighbor is found, we apply the probability p to determine if the current cell will start burning.
- We also apply the probability of spontaneous ignition p_{start}.
- If none of the above conditions are met, the cell retains its current state.

Update Grid Function

The `update_grid` function is designed to manage fire dynamics and update the grid's state based on the fire propagation rules.

Algorithm Update Grid Function

function UPDATEGRID(grid, $p, pstart$)
  Create a copy new_grid of grid
  for each cell (i, j) in grid **do**
    if grid$[i][j]$ is 'NormalForest' **then**
      Call Propagate(new_grid, $i, j, p, pstart$)
    end if
  end for
  return new_grid
end function

In this pseudocode:

- We start by getting the dimensions of the grid.
- We create a new grid to store the updated state. This is important to ensure that

updates to one cell do not immediately affect the calculations for neighboring cells.

- We loop through each cell in the grid and update its state based on the fire propagation rules defined in the `propagate` function.
- Finally, we return the updated grid.

Simulate Function

The `simulate` function orchestrates the entire simulation over multiple iterations, updating the forest grid at each time step using the previously developed functions.

Algorithm Simulate Function

function SIMULATE(m, n, iterations, $p, pstart$)
  grid $\leftarrow$ InitializeGrid(m, n)
  for $i \leftarrow 1$ to iterations **do**
    Call PlotGrid(grid)
    grid $\leftarrow$ UpdateGrid(grid, $p, pstart$)
  end for
  Call PlotGrid(grid)
end function

In this pseudocode:

- We start by setting the current grid to the initial grid.
- We then loop through each iteration of the simulation.
- In each iteration, we update the grid's state using the `update_grid` function.
- We also plot the grid's state at specified intervals to visualize the progression of the forest fire.
- Finally, we return the final state of the grid after the simulation.

2. Regarding the components of the update grid function

Random variables involved in the update grid function

Random Variable

1. **Spontaneous Ignition**: The probability p_{start} that a tree will catch fire spontaneously, regardless of its neighboring cells. This is a random variable as it introduces uncertainty and variability in the simulation.
2. **Fire Propagation**: The probability p that a tree will catch fire from a neighboring burning tree. This probability can be influenced by various factors such as wind, humidity, and vegetation type, introducing randomness to the simulation.
3. **Wind Speed and Direction**: While these might be constants in the provided code, they can be considered as random variables if they are allowed to vary over time or space, introducing randomness in the fire's propagation direction and speed.

4. **Terrain Elevation**: The elevation of the terrain at each grid cell. Variations in terrain elevation can influence fire spread, introducing randomness if the terrain is not uniform.

Utilization in the Simulation

- **Spontaneous Ignition**: Used to determine if a tree will catch fire spontaneously during each iteration.

```
if current_state == 'NormalForest' and random.random() < pstart:
    return 'Burning'
```

In the `propagate_with_humidity` function, this snippet represents the probability p_{start} of a tree catching fire spontaneously.

- **Fire Propagation**: Used in combination with other factors (e.g., wind, terrain) to calculate the probability of a tree catching fire from its burning neighbors.

```
if grid[ni][nj] == 'Burning':
    ...
    if random.random() < adjusted_p:
        return 'Burning'
```

In the same function, this snippet shows how the fire propagation probability p is used, potentially adjusted based on other factors like wind and terrain.

- **Wind Speed and Direction**: Influence the fire propagation probabilities, making certain directions more likely for fire spread.

```
# From the simulate function
current_grid = update_grid(current_grid, p, pstart, w_speed, w_direction,
terrain, humidity_grid)
```

The wind speed (`w_speed`) and direction (`w_direction`) are parameters in the `update_grid` function, influencing fire propagation.

- **Terrain Elevation**: Can influence fire spread, possibly making uphill or downhill spreads more likely.

```
# From the update_grid function
for i in range(m):
        for j in range(n):
                new_grid[i][j] = propagate_with_humidity(grid, i, j, p,
pstart, w_speed, w_direction, terrain, humidity_grid[i][j])
```

Role of Random Number Generators in the Update Grid Function

Random number generators are utilized in various places in the code to introduce randomness and simulate probabilistic behavior. Below is a code snippet from the `propagate_with_humidity` function where a random number is used to determine if a tree catches fire due to propagation:

```
if grid[ni][nj] == 'Burning':
    ...
    if random.random() < adjusted_p:
        return 'Burning'
```

Here, `random.random()` generates a random float between 0.0 and 1.0, and if this number is less than the adjusted probability `adjusted_p` (which accounts for factors like wind and humidity), the tree catches fire.

Stochastic Processes in the Update Grid Function

The `update_grid` function embodies a stochastic process, where the future state of the forest grid depends on its current state, influenced by random variables and probabilities. The randomness in fire propagation and spontaneous ignition makes the forest fire spread a stochastic process.

```
def update_grid(grid, p, pstart, w_speed, w_direction, terrain,
humidity_grid):
    m = len(grid)
    n = len(grid[0])
    new_grid = [['NormalForest' for _ in range(n)] for _ in range(m)]
    for i in range(m):
        for j in range(n):
            new_grid[i][j] = propagate_with_humidity(grid, i, j, p,
pstart, w_speed, w_direction, terrain, humidity_grid[i][j])
    return new_grid
```

Markov Chain Characteristics and the Number of Markov Chains Involved

Markov Chain Characteristics

The `update_grid` function exhibits characteristics of a Markov Chain because the future state of each cell (or each small part of the forest) depends solely on its current state and the current state of its neighboring cells, not on the sequence of states that preceded it.

Number of Markov Chains

There are as many Markov Chains as there are cells in the grid. Each cell can be considered as a separate Markov Chain, with its state transitioning based on the probabilities of fire ignition and spread. The state transitions of each cell are independent of its past states, adhering to the Markov property.
Given that the grid is initialized with dimensions m (rows) and n (columns), each cell in the grid can be considered as a separate Markov Chain, as its future state depends only on its current state and the current states of its neighboring cells.

Therefore, the total number of Markov Chains (N) in the grid can be described by the formula:

$$N = m \times n$$

Here, m is the number of rows and n is the number of columns in the grid. Each cell in the $m \times n$ grid acts as an independent Markov Chain.

3. Explore alternative modeling approaches: Investigate the impact of additional factors

Wind Speed and Direction

- **Influence**: Wind influences the direction and speed of fire spread.
- **Implementation in the Code**: The `calculate_directional_influence` function in the code takes wind speed and direction into account to adjust the fire spread probabilities. The function calculates how much the wind influences the spread of the fire based on the difference between the wind direction and the angle between two cells.

```
# In the update_grid function
influence = calculate_directional_influence(wind_direction, angle)
adjusted_p = p + influence
```

The `calculate_directional_influence` function is used to calculate how much the wind influences the spread of the fire based on the wind direction and the angle between two cells. The fire spread probability `p` is then adjusted based on this influence.

Also in the `propagate_with_humidity` function multiplies the base spread probability with a factor that includes the wind speed and directional influence, which increases the likelihood of fire spreading in the wind's direction.

```
# In the propagate_with_humidity function
directional_influence = calculate_directional_influence(w_direction,
angle)
terrain_influence = 1 + elevation_diff / 100
vegetation_factor =
VEGETATION_FIRE_SPREAD_PROBABILITIES.get(current_state, 1.0)
```

```
humidity_factor = 1 - humidity
wind_influence_coefficient = 2.0  # Coefficient to amplify wind effect

# The probability is adjusted by multiplying with the wind influence
adjusted_p = p * vegetation_factor * terrain_influence * humidity_factor *
(1 + wind_influence_coefficient * w_speed * directional_influence)
adjusted_p = max(0, min(1, adjusted_p))

if random.random() < adjusted_p:
    return 'Burning'
```

In this snippet the code ensures that the wind's direction has a more pronounced effect on fire propagation, consistent with the observed behavior of wildfires in strong wind condition.

Different Vegetation Types

- **Influence**: Various vegetation types have different levels of flammability.
- **Implementation in the Code**: The `VEGETATION_FIRE_SPREAD_PROBABILITIES` dictionary in the code maps different vegetation types to their respective fire spread probabilities. These probabilities are then used to adjust the fire spread probabilities in the simulation.

```
# At the beginning of the code
VEGETATION_FIRE_SPREAD_PROBABILITIES = {
    'NormalForest': 1.0,
    'DenseForest': 1.5,
    # Other vegetation types can be added here
}
```

Different vegetation types are mapped to their respective fire spread probabilities in the `VEGETATION_FIRE_SPREAD_PROBABILITIES` dictionary. These probabilities can then be used to adjust the fire spread probabilities in the simulation.

Terrain Elevation

- **Influence**: Terrain elevation can affect fire spread, as fire tends to spread more rapidly uphill.
- **Implementation in the Code**: The `terrain` parameter in the `update_grid` function is used to adjust the fire spread probabilities based on the elevation of the terrain. However, the specific implementation details would need to be reviewed in the code to provide a precise description.

```
# In the update_grid function
# (Assuming that terrain elevation is considered in the calculation of
```

```
adjusted_p)
adjusted_p = ...  # Adjusted based on terrain elevation
```

The `terrain` parameter in the `update_grid` function is intended to be used to adjust the fire spread probabilities based on the elevation of the terrain. The specific implementation would depend on how terrain elevation is meant to influence fire spread.

Humidity

- **Influence**: Higher humidity can reduce fire spread, while lower humidity can increase it.
- **Implementation in the Code**: The `humidity_grid` parameter in the `update_grid` function and the `calculate_humidity` function are used to calculate and consider humidity levels across the grid, influencing fire spread probabilities.

```
# In the update_grid function
humidity = humidity_grid[i][j]
adjusted_p *= (1 - humidity)
```

The `humidity_grid` parameter in the `update_grid` function provides humidity levels across the grid, and the fire spread probability `p` is adjusted based on these humidity levels.