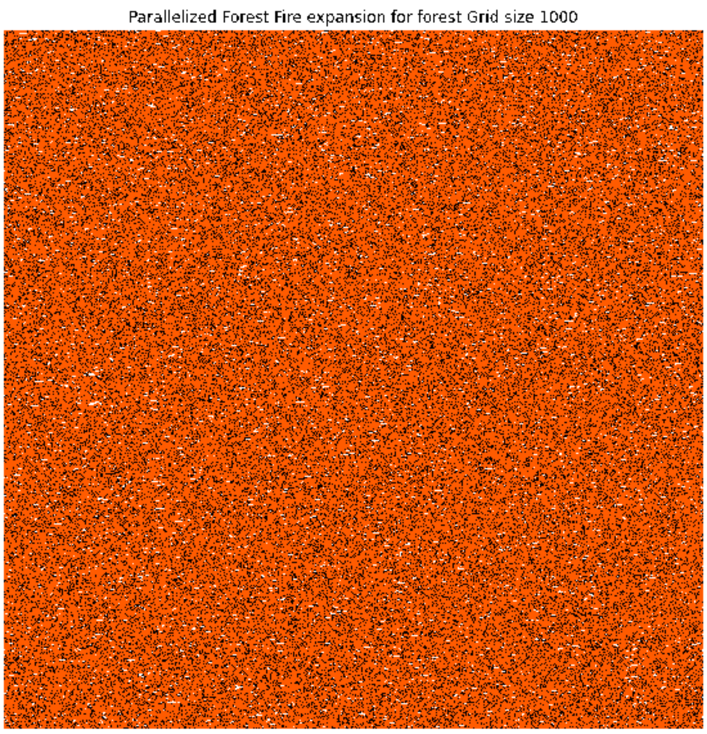
Modelling of Forest Fire Expansion Report

Introduction and Background

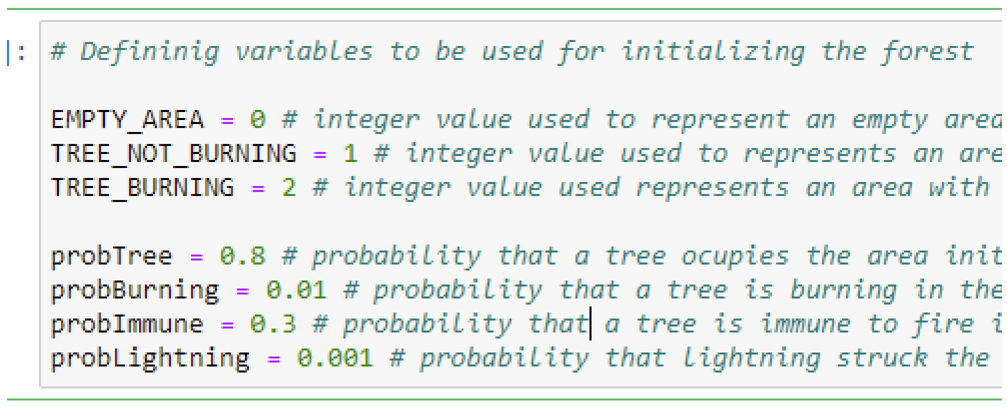
This study explains the steps involved in simulating the spread of fire in a forest environment with both burning and non-burning trees. A number of factors, including as the proximity of one of the forest's trees to a burning tree in its Moore neighbourhood or the potential of lightning strikes on the forest site, could influence the spread of fire at any position of the forest. Furthermore, some tree species are fire resistant, meaning that even if there is a risk of fire, they will not be easily burned. As a result, in the following parts, I'll explain how this simulation was completed in accordance with the coursework requirements.



To begin the simulation, I needed to acquire certain information to set up the forest site. This was crucial prior to the spread of the fire on the site. As a result, this was done in Python using the Numpy package.

I also built variables for the various probabilities presented in the coursework and entered values into the Numpy grid to represent burning and non-burning trees, empty ground, and the boundary. I chose close colours to depict trees and fire in the simulation based on the notion that trees are green and fire is red.

Finally, the animation delay represents a 300 millisecond burn time for the fire flames. The section is seen in the image below.



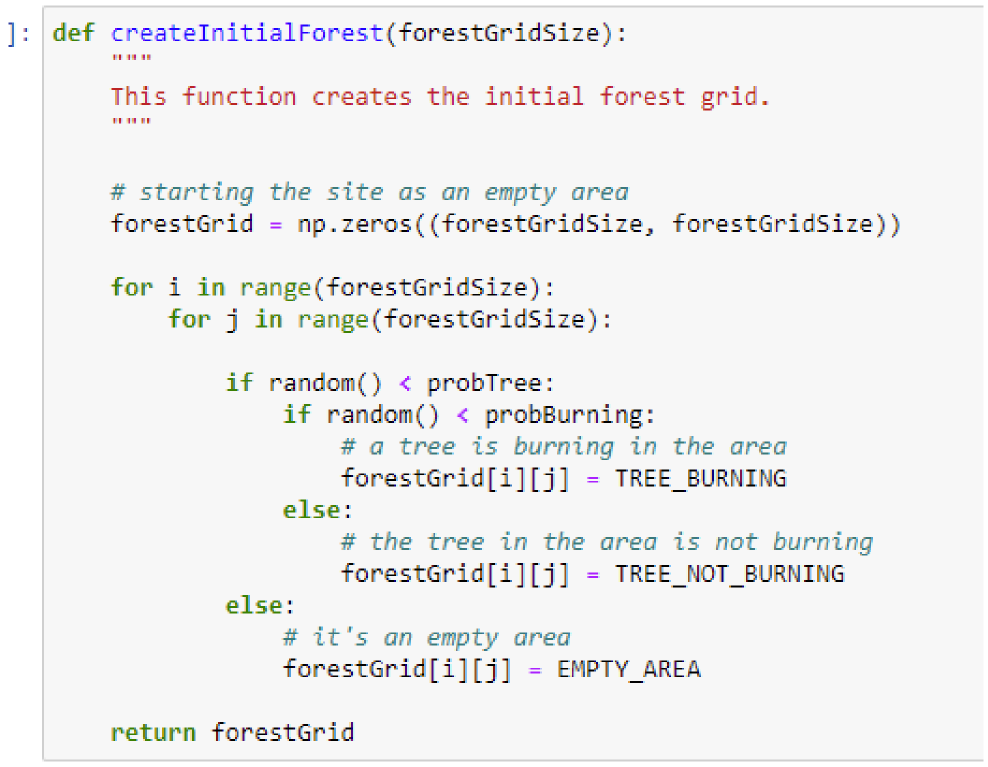
```
# Defininig variables to be used for initializing the forest

EMPTY_AREA = 0 # integer value used to represent an empty area
TREE_NOT_BURNING = 1 # integer value used to represents an are
TREE_BURNING = 2 # integer value used represents an area with

probTree = 0.8 # probability that a tree ocupies the area init
probBurning = 0.01 # probability that a tree is burning in the
probImmune = 0.3 # probability that a tree is immune to fire i
probLightning = 0.001 # probability that lightning struck the
```

To begin, I constructed an empty forest site grid based on the size of the forest indicated in the specification, then filled it with vacant lands, burning and non-burning trees, using two probabilities, probTree and probBurning.

As a result, if the chance of there being a tree at any particular point on the forest is true, the probability of that tree burning is checked. We only set the tree value if the tree isn't burning; otherwise, we mark the location as having a burning tree.



```
def createInitialForest(forestGridSize):
    """
    This function creates the initial forest grid.
    """

    # starting the site as an empty area
    forestGrid = np.zeros((forestGridSize, forestGridSize))

    for i in range(forestGridSize):
        for j in range(forestGridSize):

            if random() < probTree:
                if random() < probBurning:
                    # a tree is burning in the area
                    forestGrid[i][j] = TREE_BURNING
                else:
                    # the tree in the area is not burning
                    forestGrid[i][j] = TREE_NOT_BURNING
            else:
                # it's an empty area
                forestGrid[i][j] = EMPTY_AREA

    return forestGrid
```

The spread rules are then applied to the site. But first, I extended the site's limits using period boundary conditions to avoid boundary effects that would prohibit us from correctly applying the spread at the borders. This is accomplished by producing ghost zones on opposing sides to make it appear as if the two boundaries are linked. The spread is then applied to the site after the ghost zones at the top, left, right, and bottom have been produced.

```python
def expandTheForestBoundaries(forest):
    """
    Expands the forest boundaries using periodic boundary conditions.
    this is done by adding invisible areas to the forest grid boundaries and making the opposite side equal
    the purpose is to help expand the fire in the boundaries.
    """

    # adding the top boundary to the buttom boundary and vice versa
    row_stack = np.row_stack((forest[-1,:], forest, forest[0,:]))

    # adding the left boundary to the right boundary and the right boundary to the left boundary
    expandedForest = np.column_stack((row_stack[:,-1], row_stack, row_stack[:,0]))

    # return the expanded forest
    return expandedForest
```

The spread of fire at a specific location on the forest site is influenced by nearby burning influences and the nature of the trees. As a result, if an incident occurs, the fire must be distributed over the site in a loop, according to the specifications.

Because the region has trees, we look to see if any of its Moore neighbours (trees to the north, north-east, east, southeast, south, south-west, west, and north-west) have a burning tree. If one of them has a fire in it, If a tree is not immune, it will burn; otherwise, it will not burn. In a similar vein, if the likelihood of

True, lightning strikes, and the tree is not exempt; it burns, but it would not burn otherwise.

Otherwise, if an area contains a burning tree, we set the grid position as vacant land since the tree burns to the ground; otherwise, if there is nothing there initially, the remaining area on the site remains empty. Take a look at the diagram below.

```python
            # does the area have a tree
            if forestGrid[i][j] == TREE_NOT_BURNING:

                # the tree will burn if there is a burning tree is close to the it?
                # let's check the Moore neighborhoods ( west, north, south, east, north-east, north-west, so
                if (forestGrid[i - 1][j] == TREE_BURNING or forestGrid[i + 1][j] == TREE_BURNING or
                    forestGrid[i][j - 1] == TREE_BURNING or forestGrid[i][j + 1] == TREE_BURNING or
                    forestGrid[i - 1][j - 1] == TREE_BURNING or forestGrid[i - 1][j + 1] == TREE_BURNING or
                    forestGrid[i + 1][j - 1] == TREE_BURNING or forestGrid[i + 1][j + 1] == TREE_BURNING):

                    # is the tree in the area immune to fire
                    if random() < probImmune:
                        forestGrid[i][j] = TREE_NOT_BURNING
                    else:
                        forestGrid[i - 1][j] = TREE_BURNING

                # the tree will burn if lightning strikes the forest, since it's not immune to fire
                elif random() < probLightning:
                    # but if the tree in the area immune to fire, it will not burn
                    if random() < probImmune:
                        forestGrid[i][j] = TREE_NOT_BURNING
                    else:
                        forestGrid[i][j] = TREE_BURNING

                else:
                    # otherwise, without external influence, the tree remains the same, not burnning.
                    forestGrid[i][j] = TREE_NOT_BURNING

            # if the tree is burning, the it will burn to the ground
            elif forestGrid[i][j] == TREE_BURNING:
                forestGrid[i][j] = EMPTY_AREA

            # otherwise since the area is empty, it will remain empty
            else:
                forestGrid[i][j] = EMPTY_AREA

    return forestGrid
```

I utilised the animation to depict each iteration of the spread of fire. Matplotlib's FunctionAnimation function was used to animate the forest fires. It shows the initialised site first, then launches the VisualizeForestFireSpread function, which expands the forest boundaries and spreads the fire with each iteration. For each of the forest sizes, the spread was applied 60 times. The code can be found in the diagram below.

```
fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(111)
ax.set_title('Forest Fire expansion for forest Grid size {0}'.format(forestSize))
ax.set_axis_off()

# plot the forest
forestSitePlot = ax.imshow(forest, cmap='hot', interpolation='nearest')

def animate(i):
    forestSitePlot.set_array(animate.grid)

    expandedForest = expandTheForestBoundaries(animate.grid)
    expandedForest = expandTheForestFire(expandedForest, forestSize)

    # removing the invisible areas from the expanded forest
    animate.grid = expandedForest[1:forestSize + 1, 1:forestSize + 1]

animate.grid = forest

anim = animation.FuncAnimation(fig, animate, interval=100, frames=40)

# saving the animation as gif
anim.save('forest_fire_expansion_modelling_{0}.gif'.format(forestSize))

# show the plot
plt.show()
```

```
Time used:  0.07217001914978027
```

Forest fire simulations were created in two ways: sequentially and through parallelization. The goal of parallelization is to make the best use of the computer's processing resources in order to speed up simulations. Significant gains were achieved by parallelizing the functions. The time required for a single iteration of forest for each forest size is listed in the table below. That is, the time it takes to start the forest, expand the site's limits, and spread the fire in seconds. This information was gathered for all forest sizes, as shown in the table below.