Modelling of Forest Fire Expansion Report

Introduction and Background

This report explains the steps involved in simulating the expansion of fire in a forest environment with trees that are burning or not burning. For any given area, a number of factors, including the proximity of at least one tree that is burning at its Moore neighbourhoods determine if trees in that area would burn. Another factor that could cause an expansion of fire at any location is the probability of lightning striking the forest environment. If any of this happens, and such trees are not resistant to fire, they get burned, otherwise, they don't get burnt if there are no external influences to cause a fire. Consequently, in the following parts, I'll explain how this simulation was completed in accordance with the coursework requirements.

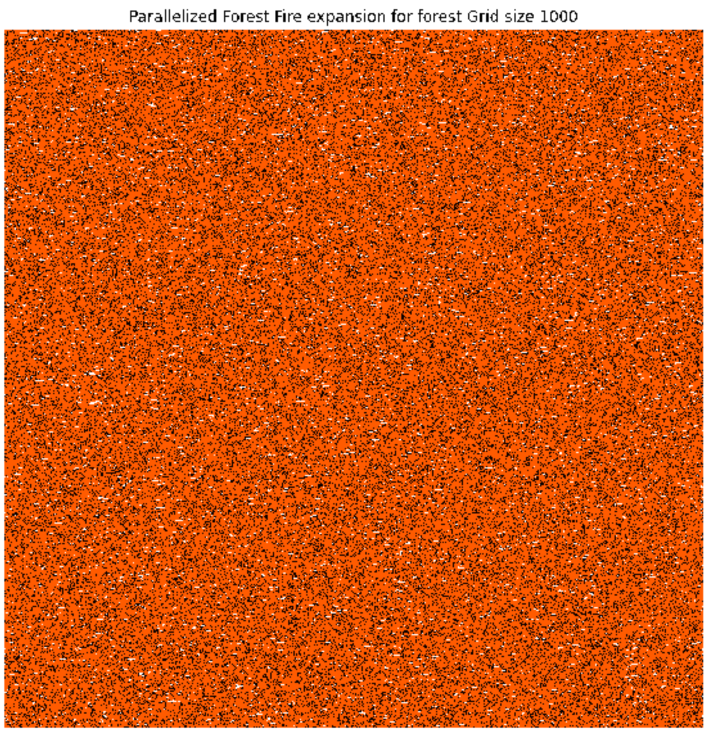
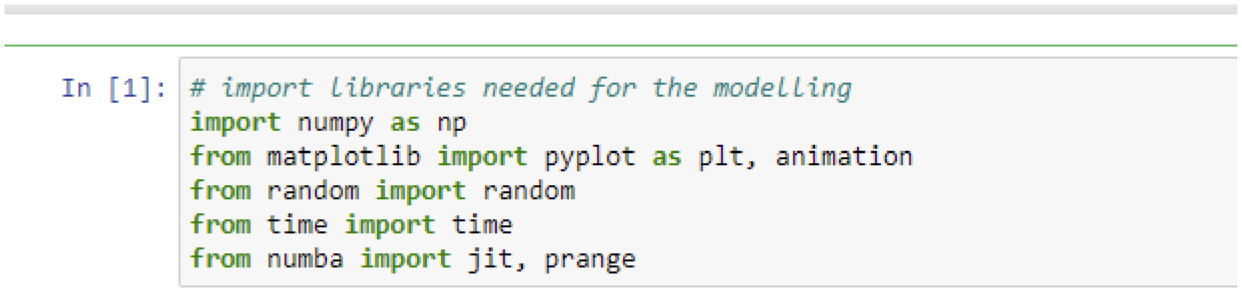


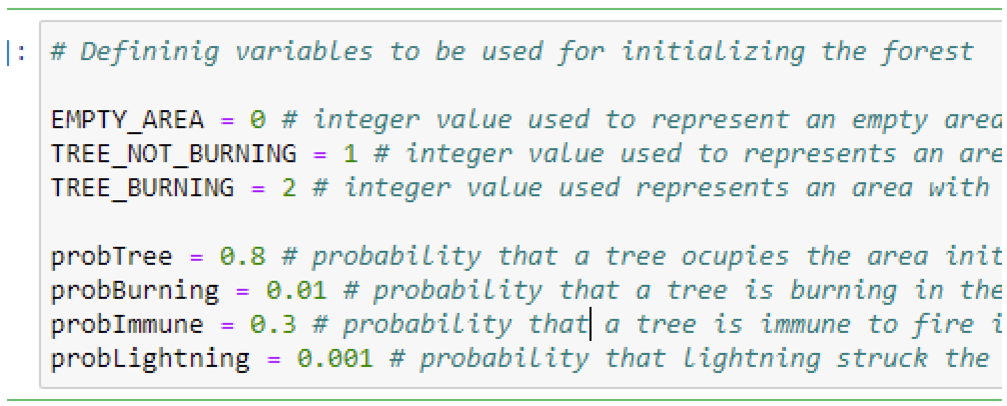
Figure 1- more images of the forest in the simulation images folder

Modelling

To begin the simulation, I defined certain variables required to set up the forest site as provided in the coursework description. Some of such variables are the **probTree, probBurning, Empty Area, TREE NOT BURNING** and **TREE BURNING** variables. This was crucial prior to the forest fire expansion to help create the initial forest grid. The whole forest fire expansion modelling was done using python on one of the school lab computers with the Numpy, Numba, Time, Random and Matplotlib libraries imported.



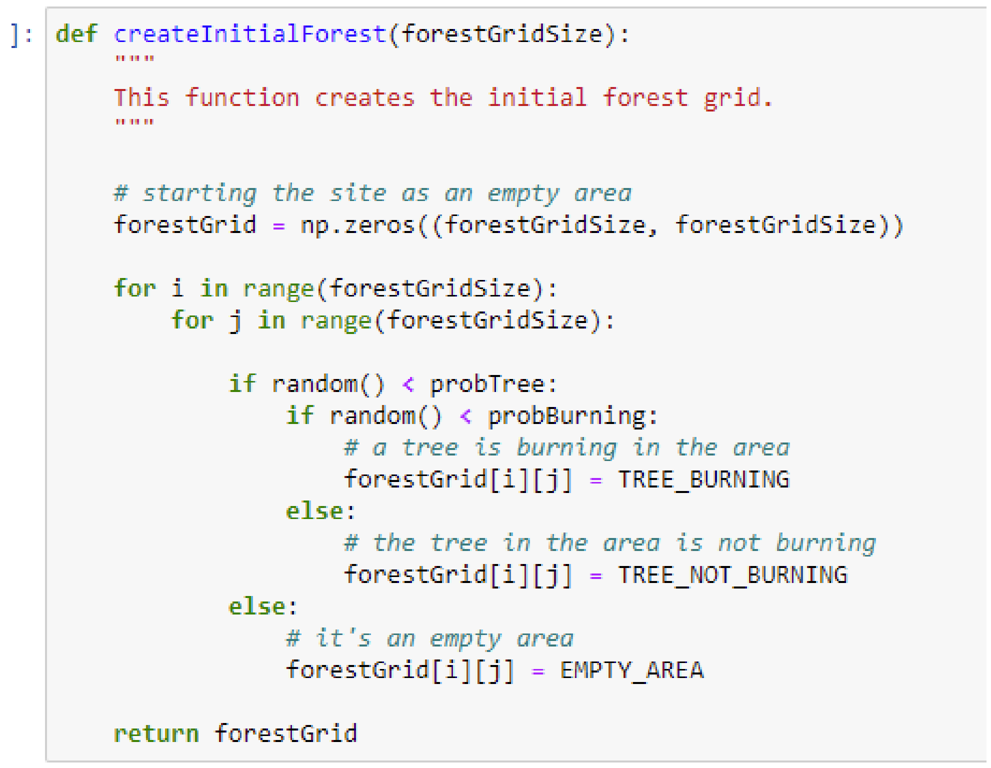
```
In [1]: # import libraries needed for the modelling
        import numpy as np
        from matplotlib import pyplot as plt, animation
        from random import random
        from time import time
        from numba import jit, prange
```



```python
# 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 created the initial forest grid based on each size of the forest indicated in the coursework description. The initial forest grid was created with a two-dimensional Numpy array filled with EMPTY_AREA value which is zero. Following that, the grid dimensions at location n x n are looped through. At any given location, if the system provides a random number that is less than the **probTree**, another if condition is made to determine the existence of a tree that is burning. If the system provides a random number less than **probBurning**, the burning tree value 1 is assigned to that location, otherwise, the value of a tree that is not burning is added to that location of the forest grid.

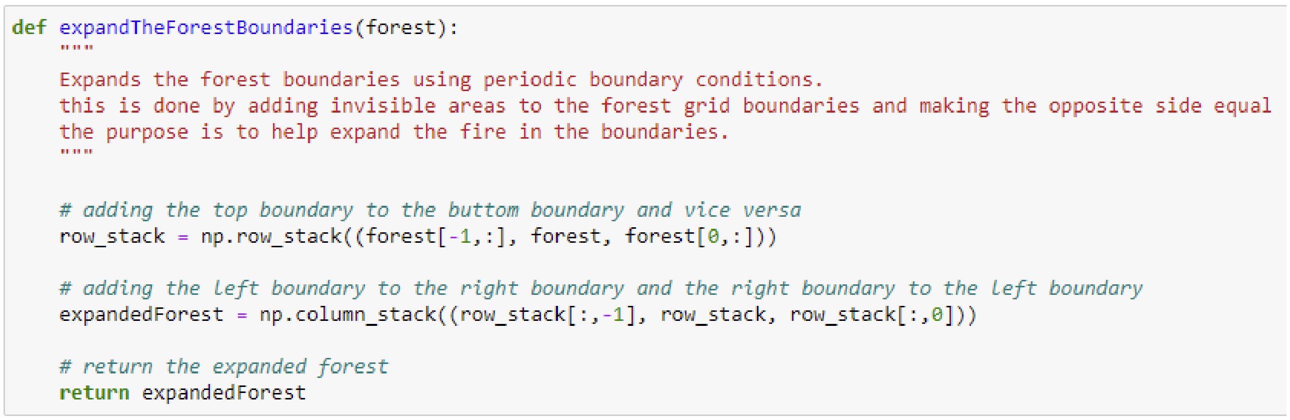
If otherwise, if the system does not provide a random number less than the **probTree** value, the location is initialized with the value of an **Empty Area**. The **createInitialForest** function was created to achieve this purpose.



```python
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
```

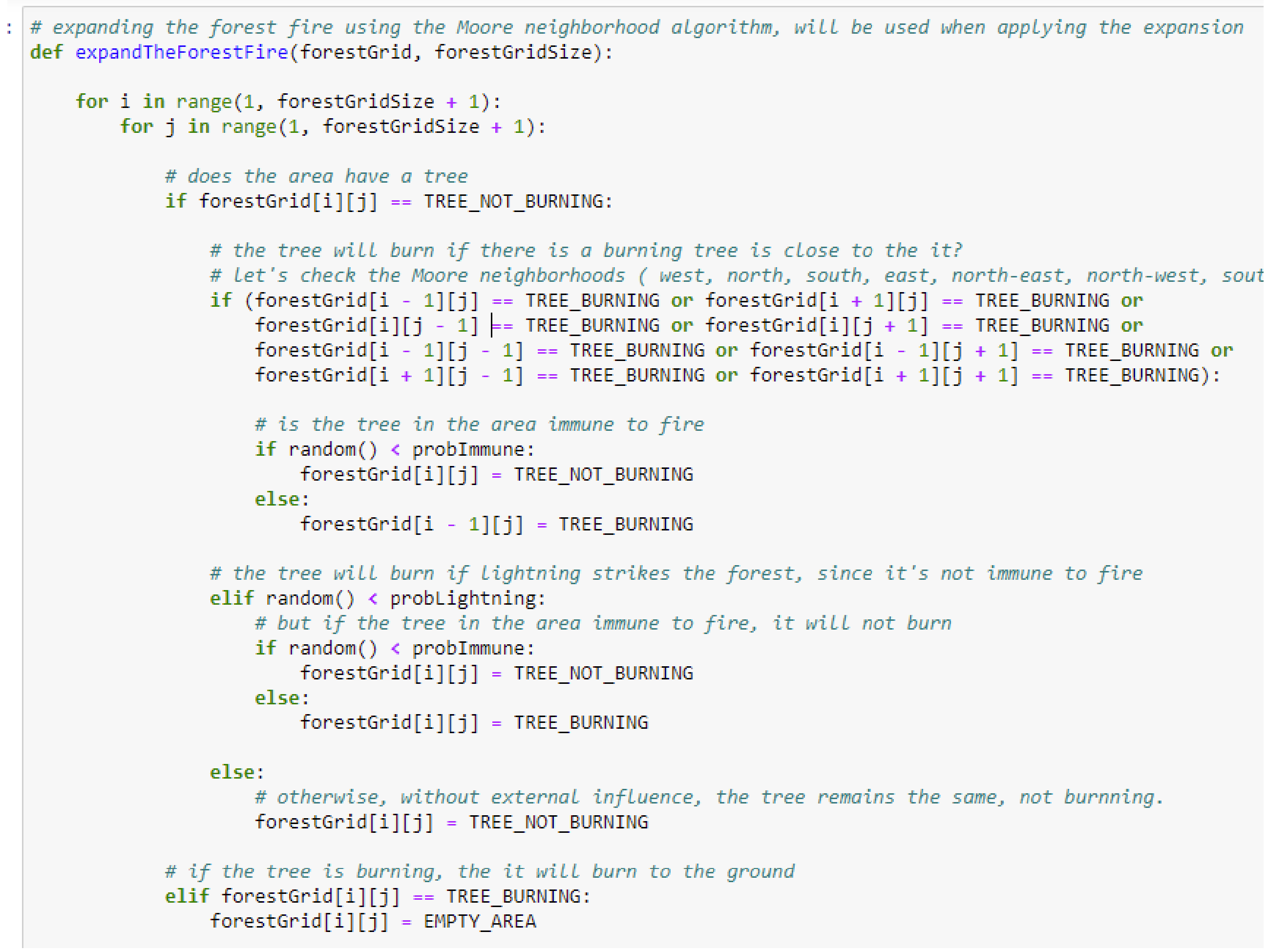
Secondly, the fire expansion rules are applied by calling the **expandTheForestFire** function. The function implements the conditions under which the forest fire expands. However, before doing that, because the grids at the forest boundaries may not have complete Moore neighbours, that is, having North, South, West, East, North-East, South-East, North-West and South-West grids. This limitation is because they are at the boundaries. Hence, to successfully apply the forest fire expansion rules in those areas, we apply the

expandTheForestBoundaries function that creates a layer of “invisible area” around the forest. This method of expanding the forest boundaries is known as Periodic Boundary Conditions (Shiflet & Shiflet, 2006).



```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
```

Following that, the expansion rules are applied, skipping the invisible areas created earlier, as they are not part of the forest. This means that for every iteration of the forest fire expansion, the forest boundaries are first expanded, fire expansion rules applied to the forest, and then invisible areas are removed.

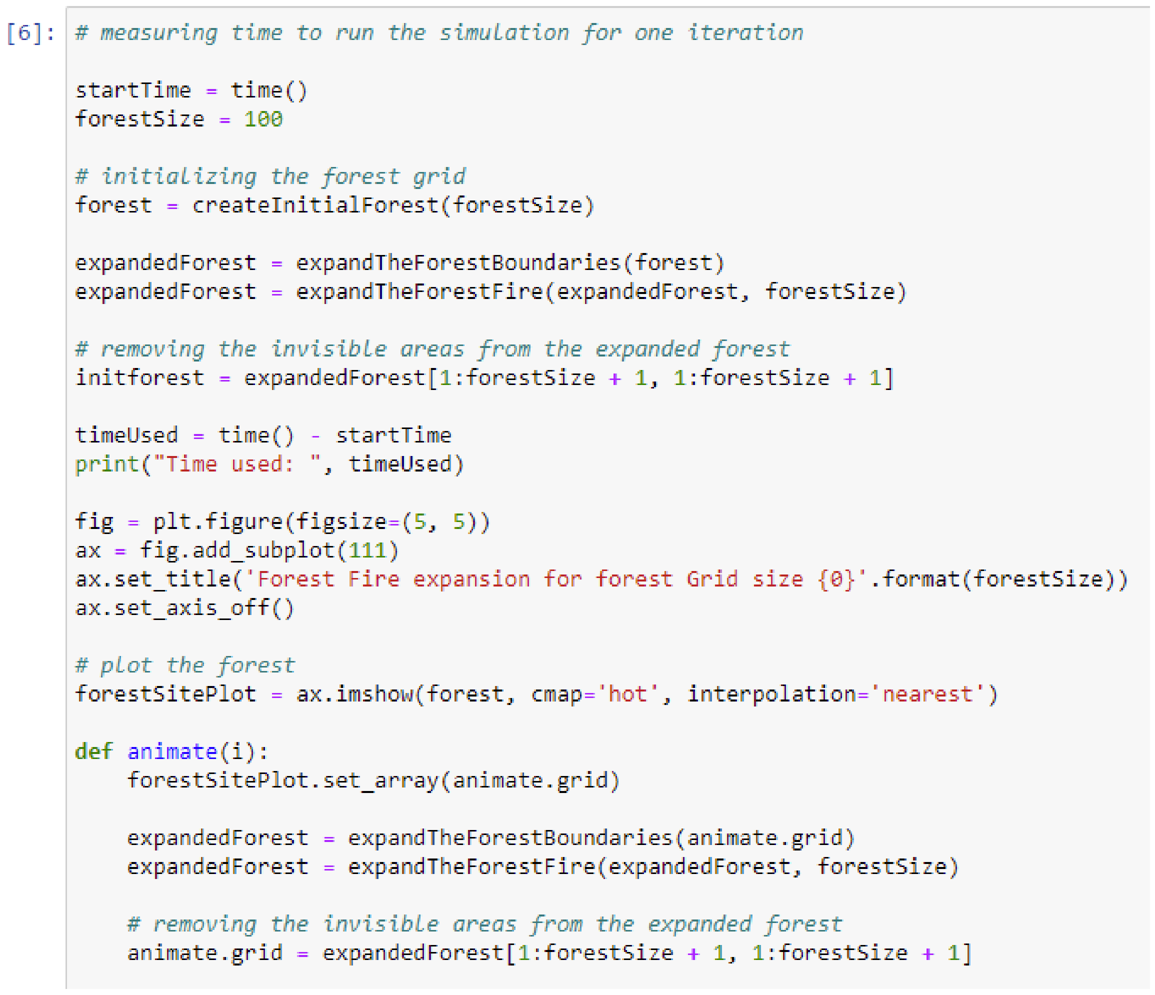


```python
: # expanding the forest fire using the Moore neighborhood algorithm, will be used when applying the expansion
  def expandTheForestFire(forestGrid, forestGridSize):

      for i in range(1, forestGridSize + 1):
          for j in range(1, forestGridSize + 1):

              # 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, sout
                  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
```

The expansion of fire at any specific area on the forest site is influenced by external influences and the nature of the trees at that location. As a result, if an area contains a tree that is not burning, but one of its neighbours is on fire, the tree in that area will burn if it is not immune to the fire. The immunity of a tree is determined by checking if the random number generated by the computer at that region is less than **probImmune**. If it is, according to the coursework requirement, the tree is immune to fire and does not get burnt. If otherwise, that condition is not true, then the tree gets burnt because it was influenced by one of its neighbours. The same is true for lightning strikes which are also an external influence on the forest. If an area contains a tree that is not burning, and the random number generated by the computer is less than the **probLightning,** which means lightning struck the site. Then that tree burns if it's not immune to fire. Where there is no external influence, the tree does not burn.

Similarly, if an area already has a tree that is burning, then the tree burns to the ground, we change the value in the area to the value of an Empty Area, zero. And if an area is an empty area, we leave it empty and move to the next cell or area on the forest grid.

To visualize the forest grid, after the forest has been initialized, I utilised the **animation.FunctionAnimation** in the **Matplotlib** library to animate the result of each iteration of the forest grid. First to display the initial forest, and animate the result of expanding the forest boundaries and fire for every iteration. This is done 40 times as shown in the frames= 40 value on the diagram below. The images below show how a forest of size 100 was initialized, animated, for every iteration and saved to a GIF file.

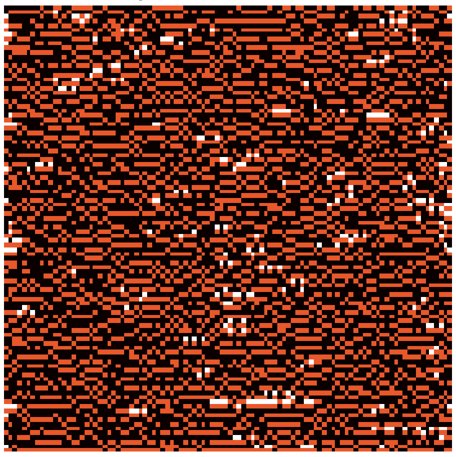


```
[6]: # measuring time to run the simulation for one iteration

     startTime = time()
     forestSize = 100

     # initializing the forest grid
     forest = createInitialForest(forestSize)

     expandedForest = expandTheForestBoundaries(forest)
     expandedForest = expandTheForestFire(expandedForest, forestSize)

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

     timeUsed = time() - startTime
     print("Time used: ", timeUsed)

     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]
```

```
    # 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 expansion for forest Grid size 100



Methods and Implementation

The forest fire expansion model was implemented sequentially and using parallelization. Parallelization helped to make the best use of the computer's processing resources in order to speed it up. This was achieved using the **Numba** variable that provides just in time compilation of functions and can parallelize it.

```
jit(nopython=True, parallel=True)
def createInitialForestWithParallelization(forestGridSize):
    """
    This function creates the initial forest grid.
    """

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

    for i in prange(forestGridSize):
        for j in prange(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
```

With the addition of **jit(nopython=True, parallel=True)** from Numba to the top of the functions and changing the range function used in the sequential implementation to **numba.prange**, they get compiled before the parallelized execution. Additionally, because it was difficult to detect when all the 40 iterations have been completely executed. I, therefore, measured time for a single iteration of the forest grid, which included initializing the forest, and expanding the boundaries and the forest fire. The time was measured for every forest grid size given or created. The table below shows the time difference between the two implementations for all 6 forest sizes.

Forest Grid Sizes	Parallel	Sequential
100 X 100	0.12	0.07
400 X 400	1.32	1.35
800 X 800	5.3	4.89
1000 X 1000	7.92	7.66
1200 X 1200	10.92	13.14
2000 X 2000	31.76	36.20

The images of the animation of all forest sizes were saved in the simulation images folder submitted with this report.

Conclusion and Future Work.

For future work, I would recommend using selected colours to represent the fire. I realized the computer took a lot of time to complete the execution when I used the "hot" colourmap provided in Matplotlib. I would also try out the multiprocessing library, so I can use a more manual approach to improve my parallelization implementation.

References

Shiflet, A. B. & Shiflet, G. W. (2006) *Introduction to computational science: Modeling and simulation for the sciences*. Princeton; Oxford: Princeton University Press.