Modeling of the Spread of Forest Fire

CSE 6730 Project 2 Final Report

Team ID: Team 2.35

Team Members: Huaidong Yang, Yingdan Wu

Git Repository Link: https://github.gatech.edu/ywu624/CSE6730_SP20_Proj2_ForestFires

The Spread of Forest Fires

This two-part tutorial illustrates different ways to model the spread of a fire among a certain forest. The parts, in brief, are as follows:

[Part 0]: We'll show how the spread of fire can be modeled over a geometric region using cellular automata (CA).

[Part 1]: This part explains how to use and implement a Markov chain model for forest fire spread.

Part 0: A cellular automata model

In the first part of this tutorial, we'll apply the concept of cellular automata to the modeling of fire spread.

The phenomenon to be modeled and simulated

We model the spread of fire in forest with geographically randomly distributed trees. To simplify the system, we assume that trees are distributed randomly with a given probability, p, among a square-shaped geographical area. For examply, p = 0 means there are no trees in the space, whereas p = 1 means trees are everywhere with no open space left in the area. If 0<p<1, then we have some space with trees and the other is open space. Then, we set fire randomly to one of the trees in this forest to see if the fire started by the model eventually destroys the entire forest. A tree will catch fire if there is at least one tree burning in its neighborhood, and the burning tree will be charred completely after one time step.

As a first cut, let's try using a cellular automaton (CA) as the conceptual model.

Quiz 1-Please list the possible states in the model.

In [ ]:

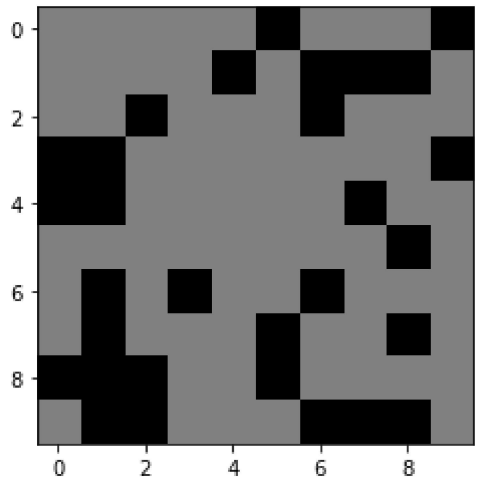
```
# Possible states:
EMPTY_SITE = 0,
ALIVE_TREE = 1,
BURNT_TREE = 2
```

Quiz 2-Please create a square-shaped domain with randomly distributed trees for a given probability p = 0.7.

Following the above states, we start with a 10 by 10 grid with randomly picked state (0 or 1) for each cell.

In [5]:

```
import numpy as np
import copy
import random
import matplotlib.pyplot as plt
mat = np.zeros ((5, 5))

m=10
n=10
Graph=np.zeros([m,n],int)
forest_ratio=0.7
fire_candidates=[]

for i in range(m):
    for j in range(n):
        rd=np.random.randint(0,100)/100
        #print(rd)
        if rd<forest_ratio:
            Graph[i][j]=1
            fire_candidates.append((i,j))
        else:
            Graph[i][j]=0
plt.imshow(Graph,cmap = 'gray',interpolation='nearest',vmin=0,vmax=2)
plt.show()
```



The above graph shows a 10 by 10 grid space with black cell denoting empty site and grey cell denoting alive tree.

Note that this model doesn't have cyclic local dynamics; possible state transitions are always one way from a tree (1) to a burnt tree (2), which is different from the dynamics models. So the whole system eventually falls into a static final configuration with no further changes possible.

Quiz 3-Find the transition status and final status of the system.

In [9]:

```
T=0
fire=random.choice(fire_candidates)
#print (fire)

in_fire=[fire]
directions=[(-1,0),(1,0),(0,1),(0,-1)]
while in_fire:
    for _ in range(len(in_fire)):
        x,y=in_fire.pop(0)
        Graph[x][y]=-1
        for dir in directions:
            nx,ny=x+dir[0],y+dir[1]
            if 0<=nx<m and 0<=ny<n and Graph[nx][ny]==1:
                in_fire.append((nx,ny))
        T+=1
    if not T%10==1:
        print("Time step", T)
        plt.imshow(Graph,cmap = 'gray',interpolation='nearest',vmin=0,vmax=2)
        plt.show()
```

The results as shown above tells that the system reaches the final (static) status after several hundred time steps. Addtionally, black cell denotes empty site; grey cell denotes alive tree; white cell denotes burnt tree.

But the total area burned in the final onfiguration greatly depends on the density of trees p. If you start with a sufficiently large value of p, you will see that a significant portion of the forest will be burned down eventually. This phenomenon is called percolation in statistical physics, which intuitively eans that something found a way to go through a large portion of material from one side to the other.

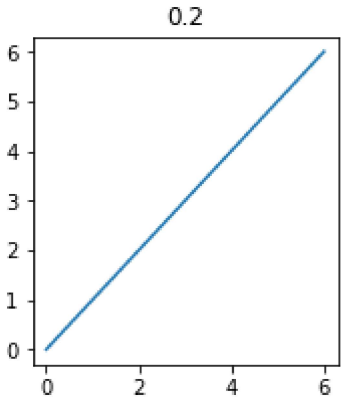
Quiz 4-Please create a 100 by 100 grid for the system and answer the following questions.

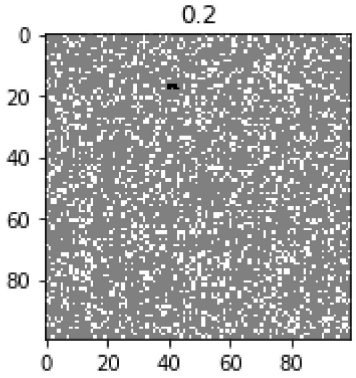
1. Compare the final results for different values of p.
2. Plot the total burned area as a function of p.
3. Plot the time until the fire stops spreading as a function of p.

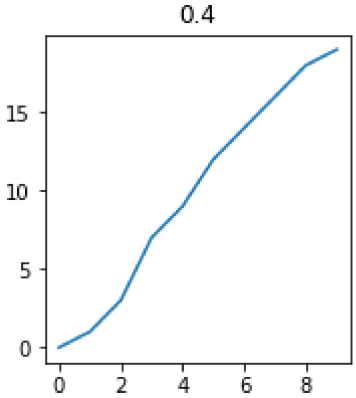
In [26]:

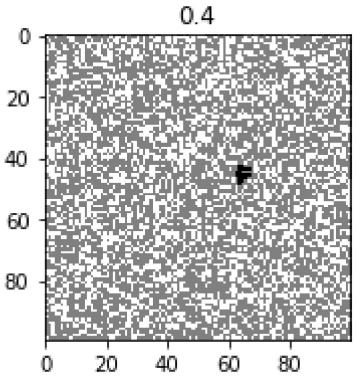
```
import numpy as np
import copy
import random
import matplotlib.pyplot as plt

class Solution:
    def __init__(self):
        self.burned_area_his=[]
        self.burned_area=[]
        self.duration=[]
        self.final=[]
        self.prob=[]

    def forest_fire(self,m,n,p):
        #0: no-fuel    1:fuel    2:burned
        Graph=np.zeros([m,n],int)
        forest_ratio=p
        fire_candidates=[]
        burned_area=[0]
        for i in range(m):
            for j in range(n):
                rd=np.random.randint(0,100)/100
                #print(rd)
                if rd<forest_ratio:
                    Graph[i][j]=1
                    fire_candidates.append((i,j))
                else:
                    Graph[i][j]=0
        # print(Graph)

        T=0
        fire=random.choice(fire_candidates)
        #print (fire)

        in_fire=[fire]
        directions=[(-1,0),(1,0),(0,1),(0,-1)]
        while in_fire:
            burned_area.append(burned_area[-1]+len(in_fire))
            for _ in range(len(in_fire)):
                x,y=in_fire.pop(0)
                Graph[x][y]=-1
                for dir in directions:
                    nx,ny=x+dir[0],y+dir[1]
                    if 0<=nx<m and 0<=ny<n and Graph[nx][ny]==1:
                        Graph[nx][ny]=2
                        in_fire.append((nx,ny))
            T+=1
            #print(T)
            # if T<10:
            #     # print(T,Graph)
            #     plt.figure()
            #     plt.imshow(Graph,cmap = 'gray',interpolation='nearest')

        self.burned_area_his.append(burned_area)
        self.burned_area.append(burned_area[-1]/m/n)
        self.duration.append(T)
        self.final.append(Graph)
    def repeat(self,num,m,n,p):
        self.prob.append(p)
```

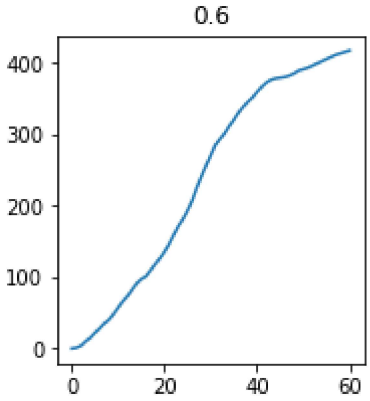
```
        burned_area=0
        duration=0
        for _ in range(num):
            self.forest_fire(m,n,p)
            burned_area+=self.burned_area.pop(-1)
            duration+=self.duration.pop(-1)
        self.burned_area.append(burned_area/n)
        self.duration.append(duration/n)
        if p in {0.2,0.4,0.6,0.8}:
            plt.figure()
            plt.subplot(1,2,1)
            x=np.arange(1,self.duration[-1],1)
            plt.plot(self.burned_area_his[-1])
            plt.title(p)
            plt.subplot(1,2,2)
            plt.imshow(self.final[-1],cmap = 'gray',interpolation='nearest')
            plt.title(p)
            fig=plt.gcf()
            fig.set_size_inches(6,3)

    def main(self):
        forest_ratios=[0.1+i*0.05 for i in range(17)]
        for p in forest_ratios:
            self.repeat(5,100,100,p)
        plt.figure()
        plt.subplot(1,2,1)
        plt.plot(test.prob,test.duration)
        plt.title("Duration vs Probability")
        plt.subplot(1,2,2)
        plt.plot(test.prob,test.burned_area)
        plt.title("Burned vs Probablity")
        fig=plt.gcf()
        fig.set_size_inches(15,5)



test=Solution()
test.main()

plt.show()
```

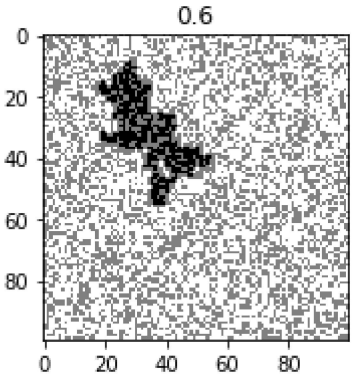

0.2
0
1
2
3
4
5
6
0
2
4
6

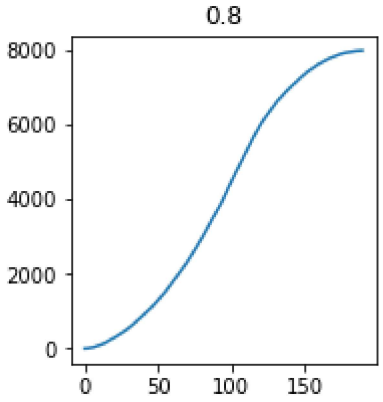

0.2
0
20
40
60
80
0
20
40
60
80

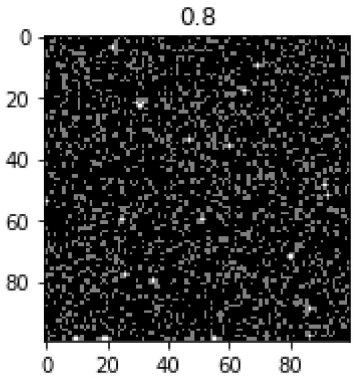

0.4
0
5
10
15
0
2
4
6
8

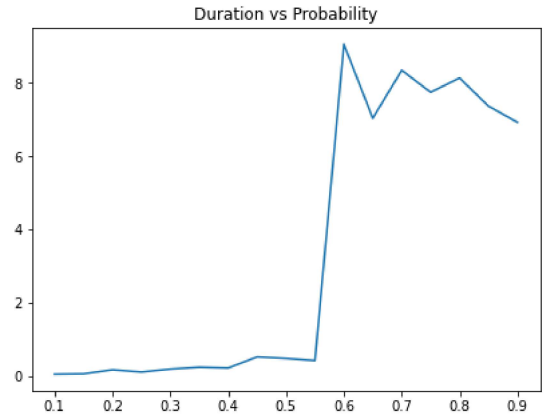

0.4
0
20
40
60
80
0
20
40
60
80

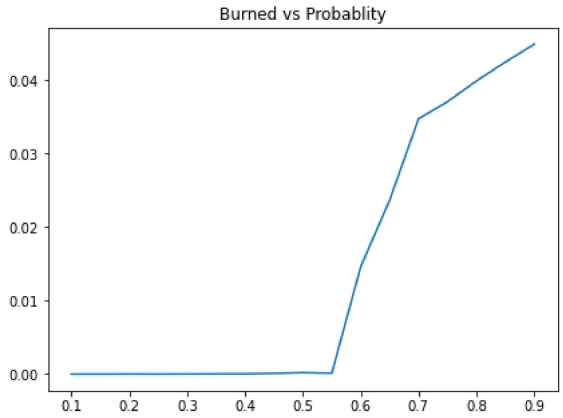

0.6
0
100
200
300
400
0
20
40
60


0.6
0
20
40
60
80
0
20
40
60
80


0.8
0
2000
4000
6000
8000
0
50
100
150


0.8
0
20
40
60
80
0
20
40
60
80





The next analytical method is for studying critical thresholds for forest fire CA model. The percolation threshold may be estimated analytically by a method called renormalization group analysis. This is a serious mathematical technique developed and used in quantum and statistical physics. Here, we specifically focus on the basic idea of the analysis and how it can be applied to specific CA models. The following is the detailed steps for this analysis.

1. Define a property of a “portion” of the forest fire system, which is defined as the probability for a portion to conduct a fire from one side to another side through it;
2. Calculate the property at the smallest scale, $q_1 = p$. This is usually at the single-cell level, which should be immediately obtainable from a model parameter;
3. Derive a mathematical relationship between the property at the smallest scale, q_1, and the same property at a one-step larger scale, $q_2 = \Phi(q_1)$. This derivation is done by using single cells as building blocks to describe the process at a larger scale (e.g., two-by-two blocks);
$$\Phi(q_1) = q_1^4 + 4q_1^3(1 - q_1) + 4q_1^2(1 - q_1^2)$$
4. Assume that the relationship derived above can be applied to predict the property at even larger scales, and then study the asymptotic behavior of the iterative map $q_{s+1} = \Phi(q_s)$ when scale s goes to infinity.

Quiz 5-Estimate the critical percolation threshold for the forest fire model. Hint: use a Cobweb plot.

In [2]:

```
import numpy as np
from scipy.optimize import fsolve

def myFunction(x):
   F = x**4 + 4*x**3*(1-x) + 4*x**2*(1-x)**2 - x
   return F


x = fsolve(myFunction,0.5)
print('pc=',x)
```

```
pc= [0.38196601]
```

So, the bottom line is, if the tree density in the forest is below 38%, the burned area will remain small, but if it is above 38%, almost the entire forest will be burned down.

Part 1: Stochastic Cellular Model with Markov Chain

The model in Part 0 is a deterministic cellur model, which means once the forest is formed and the fire location is determined, the results are fixed. However, in this model, stochastic model is added. It is modeled as a random ohenomenon on a regular spatial grid, specially, an interacting pacrticle system modeled as a continuous-time Markov chain on a latice.

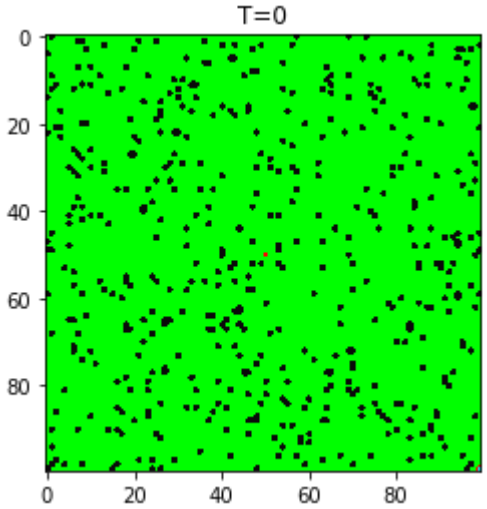
Assuming a $n * m$ grid with 1 to represent forest area and 0 to represent non-forest area. The cell (i,j) can past the fire to four directions (i+1,j), (i-1,j), (i,j+1),(i,j-1) in random amounts of time $T_{1,0}$,$T_{-1,0}$, $T_{0,1}$,$T_{0,-1}$. The forest itself can be burning in random ampunt of time $T_{0,0}$. Each duration, T, is assumed to be indepenedent and exponentially distributed with mean $1/\lambda$

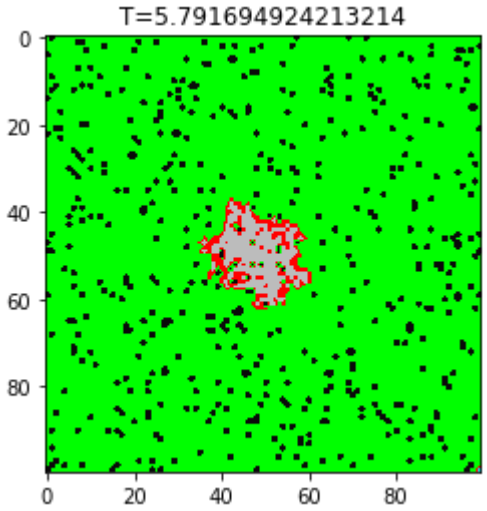
Each node can have four status, non-forest:0 in black, forest:1 in green, burning:2 in red, and burned:3 in grey.

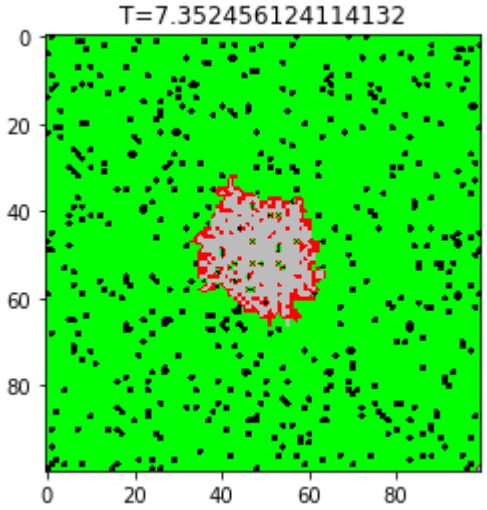
#Exercise 1: Fire without wind

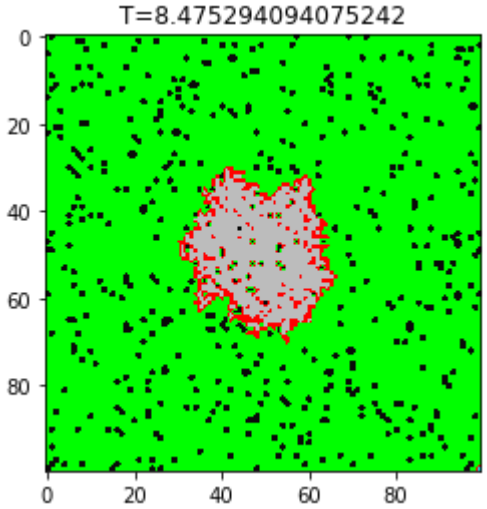
In [1]:
```
import numpy as np
import copy
import random
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import heapq

# Define the colors we want to use
blue = np.array([12/256, 238/256, 246/256, 1])
black = np.array([11/256, 11/256, 11/256, 1])
grey = np.array([189/256, 189/256, 189/256, 1])
red = np.array([1, 0, 0, 1])
green=np.array([0,1,0,1])

mapping = np.linspace(-1,4,256)
newcolors = np.empty((256, 4))
newcolors[mapping <=4] = blue
newcolors[mapping <= 3] = grey
newcolors[mapping <= 2] = red
newcolors[mapping <= 1] = green
newcolors[mapping <= 0] = black

# Make the colormap from the listed colors
my_colormap = ListedColormap(newcolors)

def gen_forest(m,n,p):
    Graph=np.zeros([m,n],int)
    forest_ratio=p
    for i in range(m):
        for j in range(n):
            rd=np.random.randint(0,100)/100
            if rd<forest_ratio:
                Graph[i][j]=1
            else:
                Graph[i][j]=0
    Graph[-1][-1]=4
    return Graph

def set_fire(x,y,Graph):
    Graph[x][y]=1
    fire=(x,y)
    fire_list=[(0,2,fire[0],fire[1])]  #timestamp, status, x, y
    return fire_list

def get_dur():
    dur0=np.random.exponential(1)  #spread time
    dur1=np.random.exponential(1)
    dur2=np.random.exponential(1)
    dur3=np.random.exponential(1)
    dur4=np.random.exponential(1/2)   #burning time
    return [dur0,dur1,dur2,dur3,dur4]

def forest_fire(m,n,p):
    Graph=gen_forest(m,n,p)  #generate the forest with the forest ratio=p, her
e we use p=0.95
    fire_list=set_fire(m//2,n//2,Graph)  #set fire at the center of the forest
```

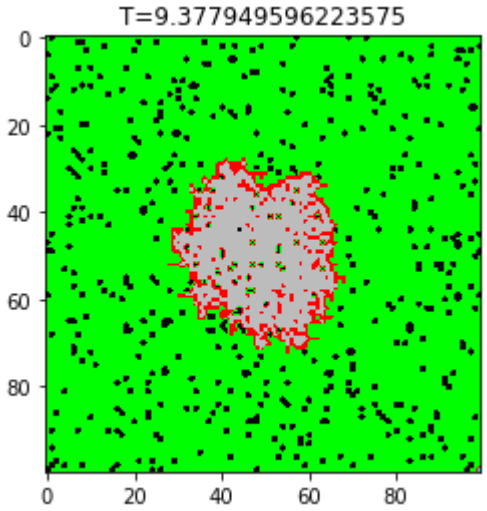
```
    directions=[(-1,0),(1,0),(0,1),(0,-1)]   #s,n,e,w
    iter,T=0,0
    #start the fire spread in the priority queue based on their time stamp
    while fire_list:
        iter+=1
        T,status,x,y=heapq.heappop(fire_list)
        dur=get_dur()
        #for the burning area
        if status==2:
            Graph[x][y]=2
            #next step is burned out in dur[0]
            heapq.heappush(fire_list,(T+dur[0],3,x,y))
            #spread the fire to its fore neighbors
            for (i,dir) in enumerate(directions):
                nx,ny=x+dir[0],y+dir[1]
                if 0<=nx<m and 0<=ny<n and Graph[nx][ny]==1:
                    heapq.heappush(fire_list,(T+dur[i],2,nx,ny))
        #for the burned area
        elif status==3:
            Graph[x][y]=3
        if iter%1000==1 and iter<5000:
            plt.figure()
            plt.imshow(Graph,cmap= my_colormap)
            plt.title("T="+str(T))
    plt.figure()
    plt.imshow(Graph,cmap= my_colormap)
    plt.title("Final")

forest_fire(100,100,0.95)
plt.show()
```

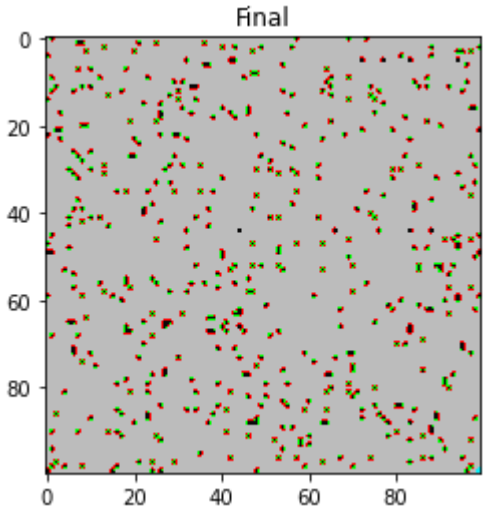

T=0
0
20
40
60
80
0
20
40
60
80


T=5.791694924213214
0
20
40
60
80
0
20
40
60
80


T=7.352456124114132
0
20
40
60
80
0
20
40
60
80







The results show that the fire spreads to its neighour randomly but remains radiul. The node in black is the non-forest area. The node in green is the forest area. The node in red is the burning area. The node in grey is the burned area. According to the results, we can see that while the fire spreading outside, there are still regions that are burning, which is what the fire should be in the real world. Therefore, this model is more closed to the real forest fire than the pure cellular model.

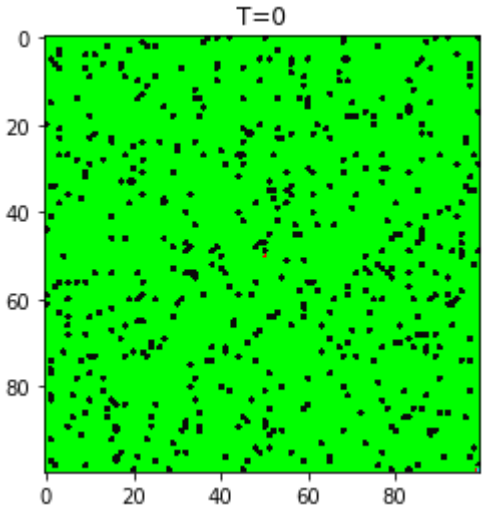
#Exercise 2: Fire with Wind

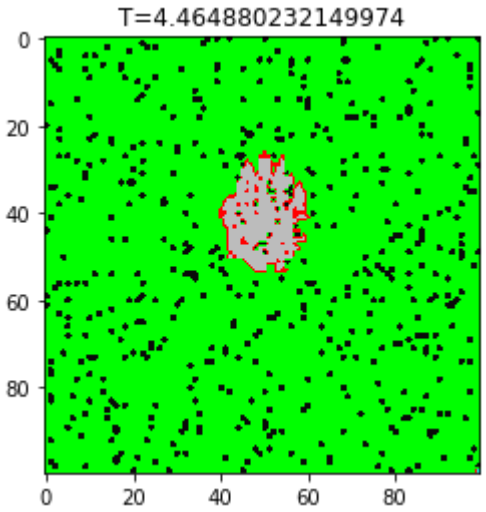
Now, we will introdue the wind to the model. The wind has a magnitude ,W, and a direction, θ. The wind will influece the time that the fire pass from one site to its neighors. Basically, the magnitude of the wind in the North-South, $W * sin\theta$ will scale the mean value of the $T_{1,0}, T_{-1,0}$'s exponential distribution. Meanwhile, the magnitude of the wind in the West-East, $W * cos\theta$ will scale the mean value of the $T_{0,1}, T_{0,-1}$'s exponential distribution.

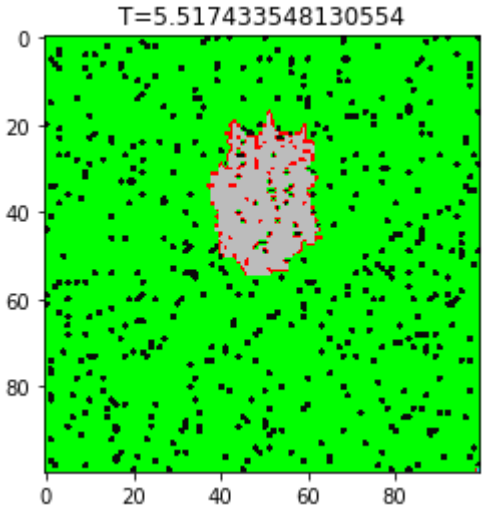
Here, we simulate a wind in North-East direction. It's shown in the results that the fire spreads to the North-East direction much faster than the other directions.

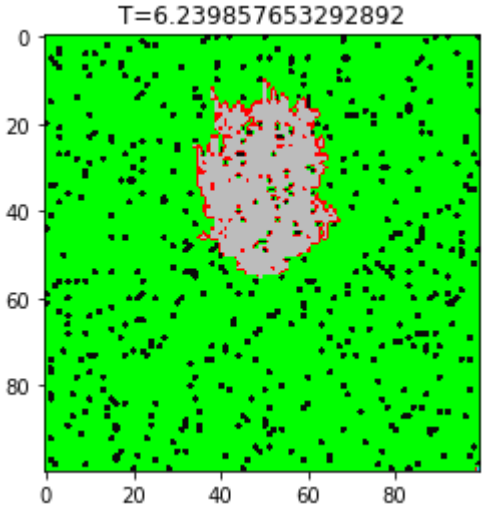
In [14]:
```
def get_wind(wind_M,wind_theta):
    theta=wind_theta/180*np.pi
    ns=wind_M*np.sin(theta)
    if ns<0:ns=1/(0-ns)
    ew=wind_M*np.cos(theta)
    if ew<0:ew=1/(0-ew)
    if wind_theta in {0,180}:ns=1
    elif wind_theta in {90,270}:ew=1
    if wind_M==0:
        ns=1
        ew=1
    wind=[1/ns,ns,1/ew,ew]
    dur0=np.random.exponential(1)*wind[0]  #spread time
    dur1=np.random.exponential(1)*wind[1]
    dur2=np.random.exponential(1)*wind[2]
    dur3=np.random.exponential(1)*wind[3]
    dur4=np.random.exponential(1/2)   #burning time
    return [dur0,dur1,dur2,dur3,dur4]

def forest_fire_wind(m,n,p,wind_M,wind_theta):
    Graph=gen_forest(m,n,p)
    fire_list=set_fire(m//2,n//2,Graph)
    directions=[(-1,0),(1,0),(0,1),(0,-1)]   #s,n,e,w
    iter,T=0,0
    while fire_list:
        iter+=1
        T,status,x,y=heapq.heappop(fire_list)
        dur=get_wind(wind_M,wind_theta)
        if status==2:
            Graph[x][y]=2
            heapq.heappush(fire_list,(T+dur[0],3,x,y))
            for (i,dir) in enumerate(directions):
                nx,ny=x+dir[0],y+dir[1]
                if 0<=nx<m and 0<=ny<n and Graph[nx][ny]==1:
                    heapq.heappush(fire_list,(T+dur[i],2,nx,ny))
        elif status==3:
            Graph[x][y]=3
        if iter%1000==1 and iter<5000:
            plt.figure()
            plt.imshow(Graph,cmap= my_colormap)
            plt.title("T="+str(T))
    plt.figure()
    plt.imshow(Graph,cmap= my_colormap)
    plt.title("Final")

print("Noth Wind in level 2 intensity")
forest_fire_wind(100,100,0.95,5,90)
```

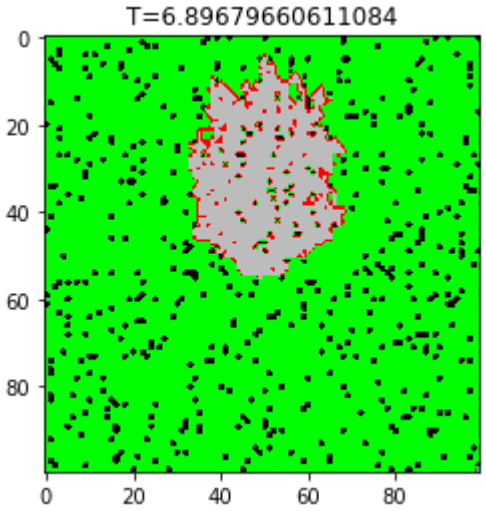
```
Noth Wind in level 2 intensity
```

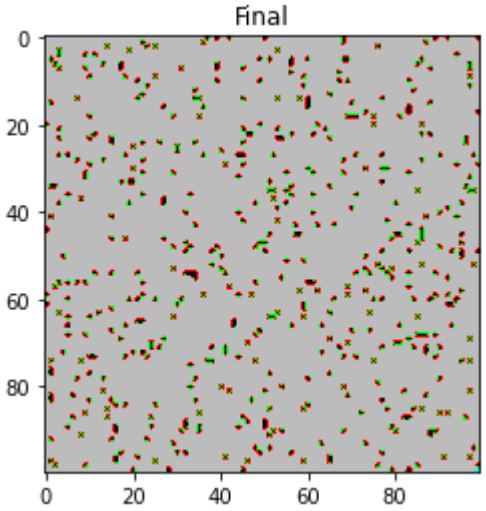






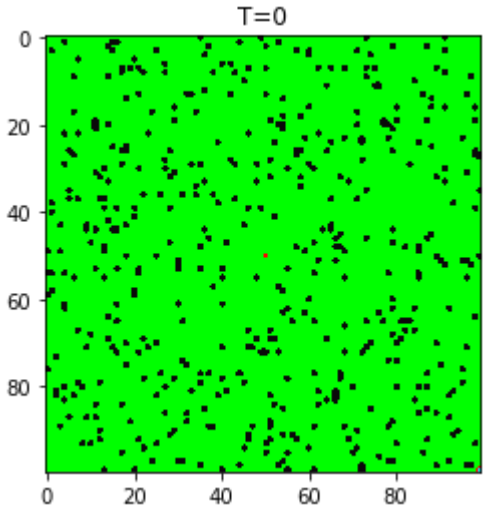

T=6.239857653292892
0
20
40
60
80
0
20
40
60
80

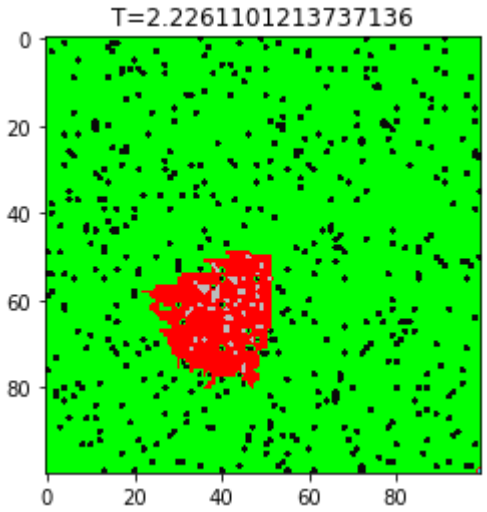

T=6.89679660611084
0
20
40
60
80
0
20
40
60
80

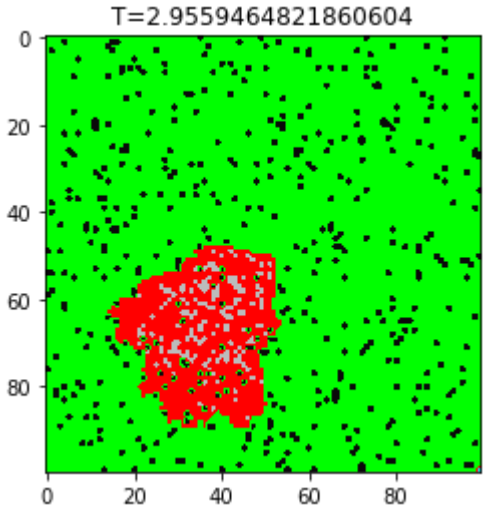

Final
0
20
40
60
80
0
20
40
60
80

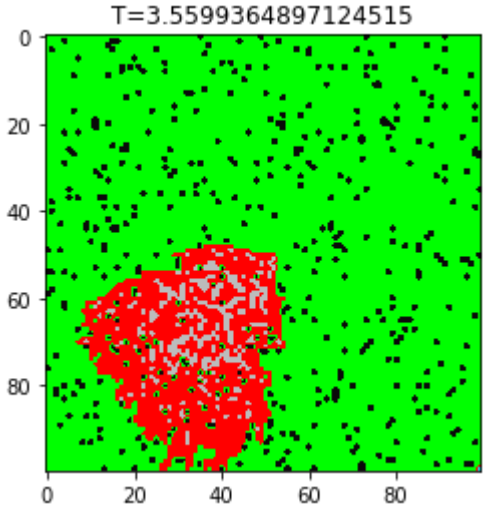
In [8]:
```
print("Southwest Wind in level 5 intensity")
forest_fire_wind(100,100,0.95,10,235)
```

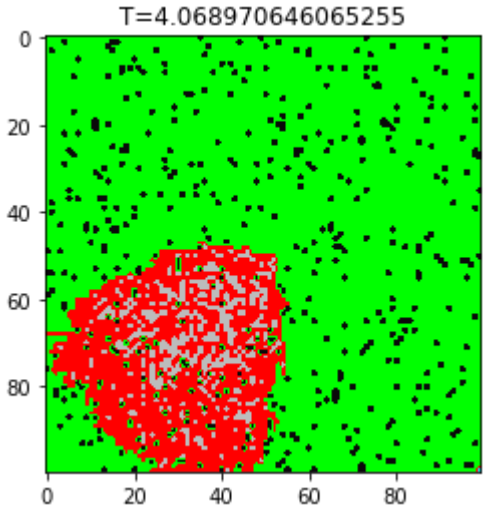
```
Southwest Wind in level 5 intensity
```

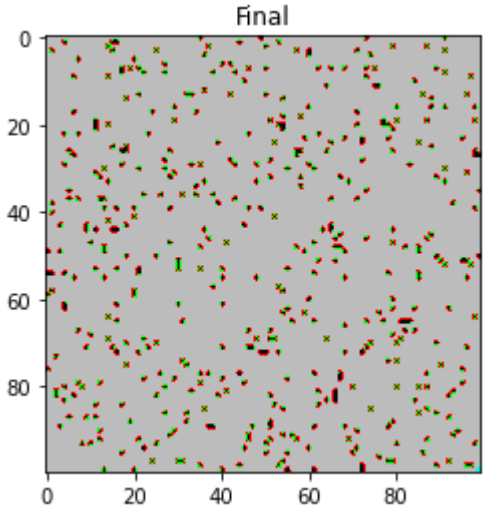







T=3.5599364897124515
0
20
40
60
80
0
20
40
60
80


T=4.068970646065255
0
20
40
60
80
0
20
40
60
80


Final
0
20
40
60
80
0
20
40
60
80

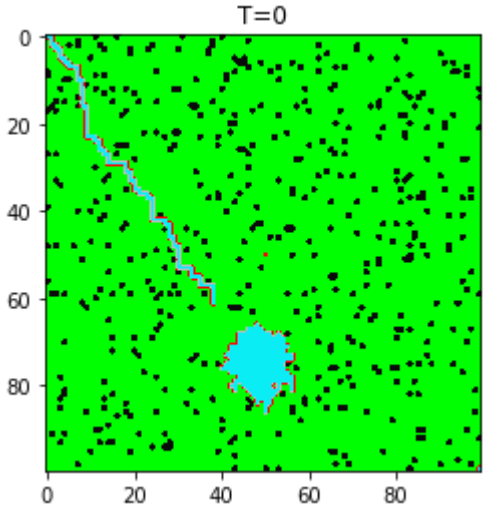
Here, we generate two forest fires, one with level 2 wind in north and the other one with level 10 wind in southeast. From the figures showing the procedure, we can see that the wind direction does influence the spreading direction of the fire. Additionally, the magnitude of the wind will influence the speed of the fire. With small magnitude, level=2, in the region that is involved in the fire, only the edges of the forest is in fire. However, with large magnitude, level=10, most of the area involved in the fire is in red, which means the fire spread so fast that the fire center doesn't have time to burn out.

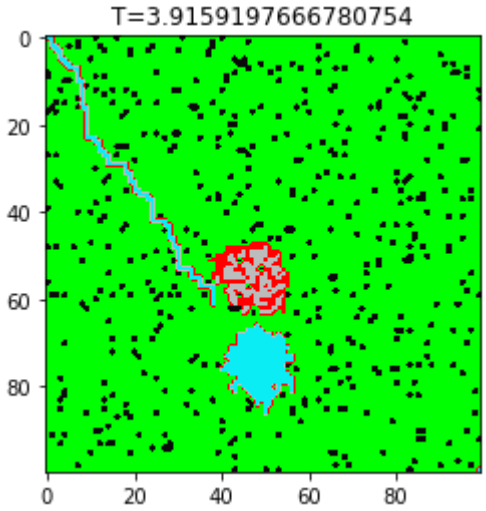
#Exercise 3: Fire with Wind and Water

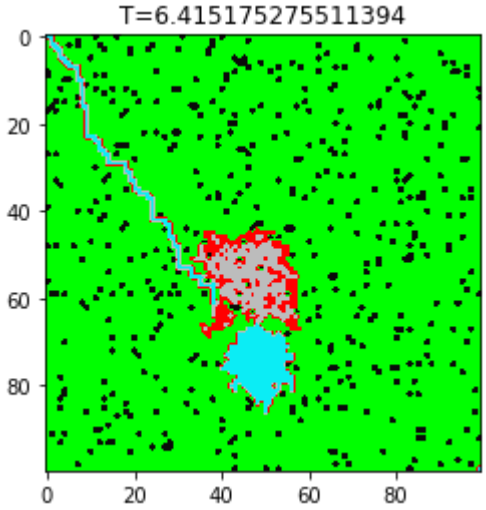
Next, we add a pool and a river into the forest. Without loss of generalosity, we generate a pool to the south of the fire center, and genrate a river flowing from the northwest to the southeast in the forest. In reality, the humidity near the water is much higher than the dry area, which makes it hard to burn. Even if it is burned, the fire lasting time is much shorter in these areas. We add this effect into the code by checking if an area is near the pool or the river. If it is near, we increase the spreading period from its neighbor and its burning time for itself.

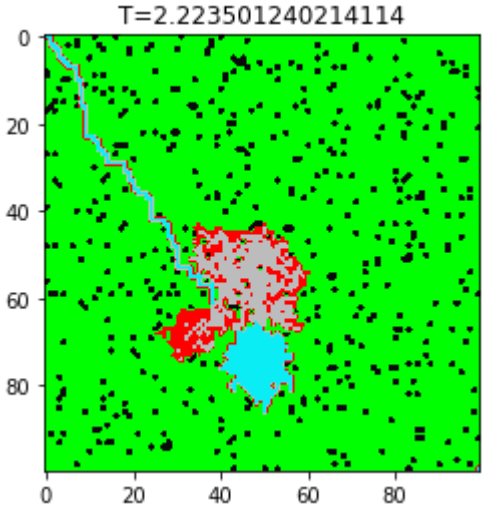
In [12]:
```
def pool_gen(Graph,area):
    m,n=len(Graph[0]),len(Graph)
    candidates={(3*n//4,m//2)}
    while area>0:
        i,j=candidates.pop()
        Graph[i][j]=4
        if (0<=i+1<n and 0<=j<m and Graph[i+1][j]!=4):candidates.add((i+1,j))
        if (0<=i-1<n and 0<=j<m and Graph[i-1][j]!=4):candidates.add((i-1,j))
        if (0<=i<n and 0<=j+1<m and Graph[i][j+1]!=4):candidates.add((i,j+1))
        if (0<=i<n and 0<=j-1<m and Graph[i][j-1]!=4):candidates.add((i,j-1))
        area-=1

def river_gen(Graph,area):
    m,n=len(Graph[0]),len(Graph)
    head=(0,0)
    while area>0:
        i,j=head[0],head[1]
        Graph[i][j]=4
        rd=np.random.randint(0,10)
        if rd<=5:
            head=(i+1,j)
        else:
            head=(i,j+1)
        area-=1

def near_water(Graph,x,y):
    n,m=len(Graph),len(Graph[0])
    distances={1,2,3,4,5}
    directions={(1,0),(-1,0),(0,1),(0,-1)}
    for dis in distances:
        for dir in directions:
            nx,ny=x+dis*dir[0],y+dis*dir[1]
            if 0<=nx<n and 0<=ny<m and Graph[nx][ny]==4:return True
    return False

def forest_fire_wind_water(m,n,p,wind_M,wind_theta,pool_ratio,river_ratio):
    Graph=gen_forest(m,n,p)
    #geneate a pool whose area is some ratio of the forest
    pool_gen(Graph,int(pool_ratio*m*n))
    #geneate a river whose area is some ratio of the forest
    river_gen(Graph,int(river_ratio*m*n))
    fire_list=set_fire(m//2,n//2,Graph)
    directions=[(-1,0),(1,0),(0,1),(0,-1)]   #s,n,e,w
    iter,T,Tmax=0,0,0
    while fire_list:
        iter+=1
        T,status,x,y=heapq.heappop(fire_list)
        dur=get_wind(wind_M,wind_theta)
        if status==2:
            Graph[x][y]=2
            #check if the point is near the water, if it is near, the fire las
ting locally is shorten
            if near_water(Graph,x,y):
                heapq.heappush(fire_list,(0.1*T+dur[0],3,x,y))
            else:
                heapq.heappush(fire_list,(T+dur[0],3,x,y))
```

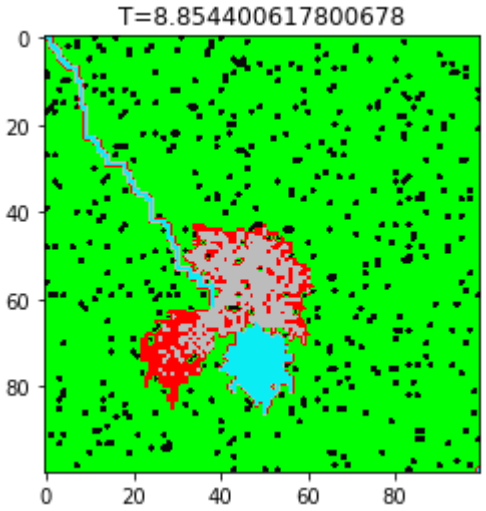
```
            for (i,dir) in enumerate(directions):
                nx,ny=x+dir[0],y+dir[1]
                if 0<=nx<m and 0<=ny<n and Graph[nx][ny]==1:
                    #check if the point is near the water, it it is near, the
 time spread to this point is increased
                    if near_water(Graph,nx,ny):
                        heapq.heappush(fire_list,(T+10*dur[i],2,nx,ny))
                    else:
                        heapq.heappush(fire_list,(T+dur[i],2,nx,ny))
        elif status==3:
            Graph[x][y]=3
        if iter%500==1 and iter<2500:
            plt.figure()
            plt.imshow(Graph,cmap= my_colormap)
            plt.title("T="+str(T))
    plt.figure()
    plt.imshow(Graph,cmap= my_colormap)
    plt.title("Final")

forest_fire_wind_water(100,100,0.95,5,235,0.02,0.01)
```

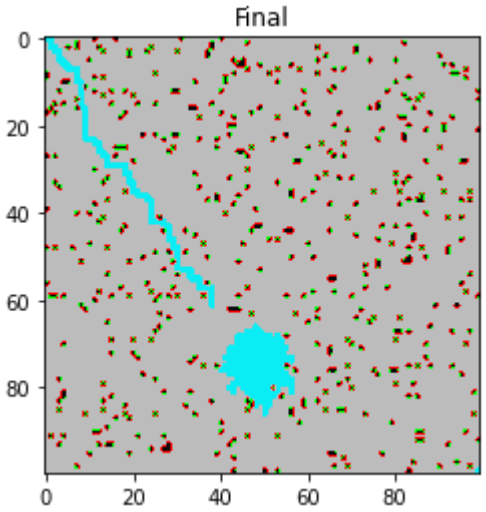

T=0
0
20
40
60
80
0
20
40
60
80


T=3.9159197666780754
0
20
40
60
80
0
20
40
60
80


T=6.415175275511394
0
20
40
60
80
0
20
40
60
80







Here, we genrate a pool with 2% of the forest and a river with 1% of the forest. The fire is set with a southeast wind in level 5. From the firing procedure figures, we can see that the spread of the fire becomes slow when it is near the water, blue area. In other words, the pool and the river influence the spread of the fire, which makes the simulated fire closer to the reality.

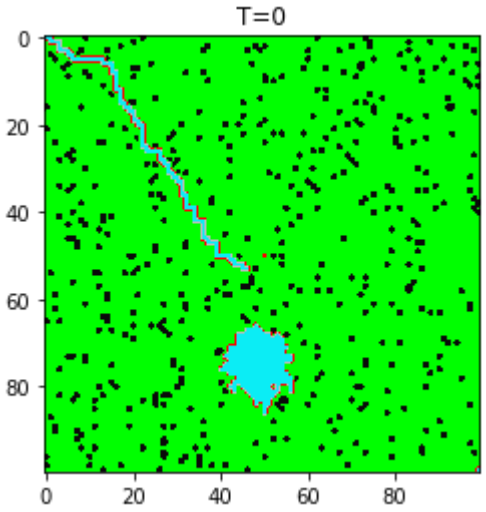
#Exercise 4: Fire with Wind and Water and Extinguishment

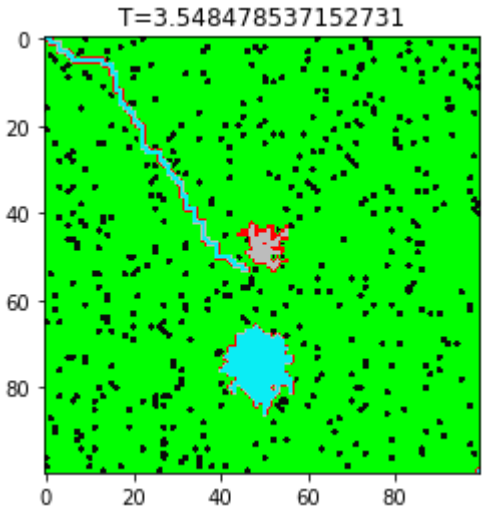
In the real world, the most concern for the forest fire is about the extinguishment. Here, we simulate the forest fire as the extinguishment is done by the scanning of helicopters. The fire is first found by people in the time stamp: "startT". The helicopters cruise in the west-east direction. The cruise period for the helicopter is: "periodT", which means every "periodT" time, the helicopter appears above a certain area. For each helicopter, it pours the water into a certain area, which lasts for time: "lastingT".

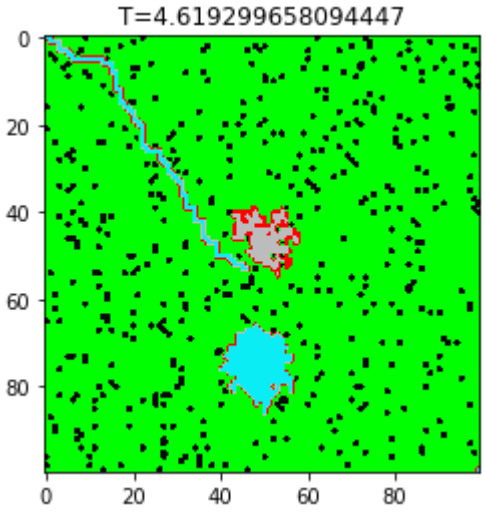
In [16]:
```
def extinguish(x,y,T,startT,lastingT,periodT):
    if T>startT and T%periodT<lastingT:return True
    else:return False

def get_burned_ratio(Graph,n,m):
    area=0
    for i in range(n):
        for j in range(m):
            if Graph[i][j]==3:
                area+=1
    return area/(m*n)

def forest_fire_wind_water_extinguish(m,n,p,wind_M,wind_theta,pool_ratio,river
_ratio,startT,lastingT,periodT,printPro):
    Graph=gen_forest(m,n,p)
    pool_gen(Graph,int(0.02*m*n))
    river_gen(Graph,int(0.01*m*n))
    fire_list=set_fire(m//2,n//2,Graph)
    directions=[(-1,0),(1,0),(0,1),(0,-1)]   #s,n,e,w
    iter,T,Tmax=0,0,0
    while fire_list:
        iter+=1
        T,status,x,y=heapq.heappop(fire_list)
        Tmax=max(T,Tmax)
        dur=get_wind(wind_M,wind_theta)
        if status==2:
            if extinguish(x,y,T,startT,lastingT,periodT):Graph[x][y]=3
            else:
                Graph[x][y]=2
                if near_water(Graph,x,y):
                    heapq.heappush(fire_list,(0.1*T+dur[0],3,x,y))
                else:
                    heapq.heappush(fire_list,(T+dur[0],3,x,y))
                for (i,dir) in enumerate(directions):
                    nx,ny=x+dir[0],y+dir[1]
                    if 0<=nx<m and 0<=ny<n and Graph[nx][ny]==1:
                        if near_water(Graph,nx,ny):
                            heapq.heappush(fire_list,(T+10*dur[i],2,nx,ny))
                        else:
                            heapq.heappush(fire_list,(T+dur[i],2,nx,ny))
        elif status==3:
            Graph[x][y]=3
        if printPro:
            if iter%200==1 and iter<1000:
                plt.figure()
                plt.imshow(Graph, cmap=my_colormap)
                plt.title("T="+str(T))
    if printPro:
        plt.figure()
        plt.imshow(Graph, cmap=my_colormap)
        plt.title("startT="+str(startT)+"   lastingT="+str(lastingT)+"   perio
dT="+str(periodT))
    burned_ratio=get_burned_ratio(Graph,n,m)
    return burned_ratio
```

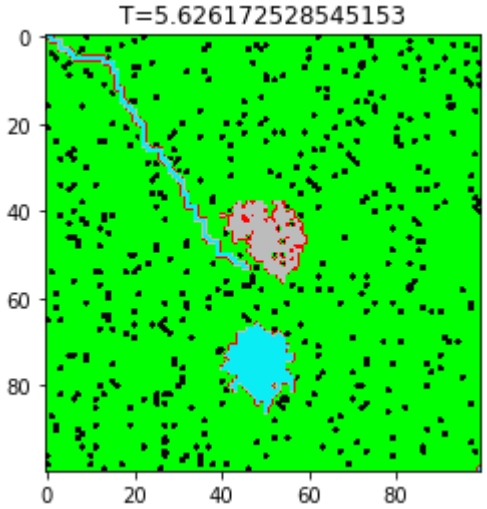
```
forest_fire_wind_water_extinguish(100,100,0.95,0,0,0.02,0.01,startT=2,lastingT
=2,periodT=5,printPro=True)
```

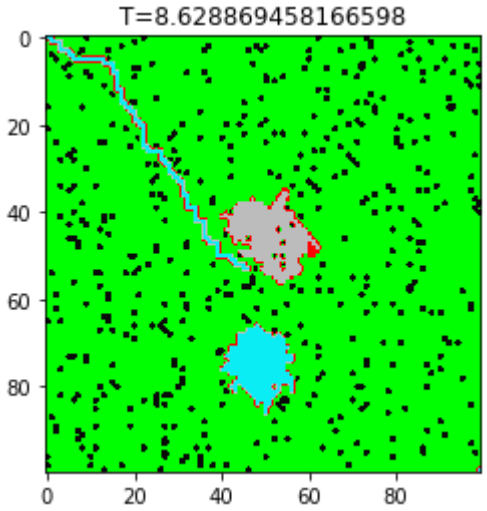
Out[16]: 0.2629

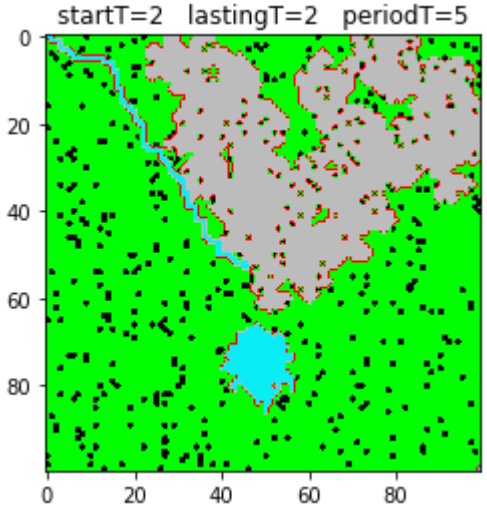

T=0
0
20
40
60
80
0
20
40
60
80


T=3.548478537152731
0
20
40
60
80
0
20
40
60
80


T=4.619299658094447
0
20
40
60
80
0
20
40
60
80



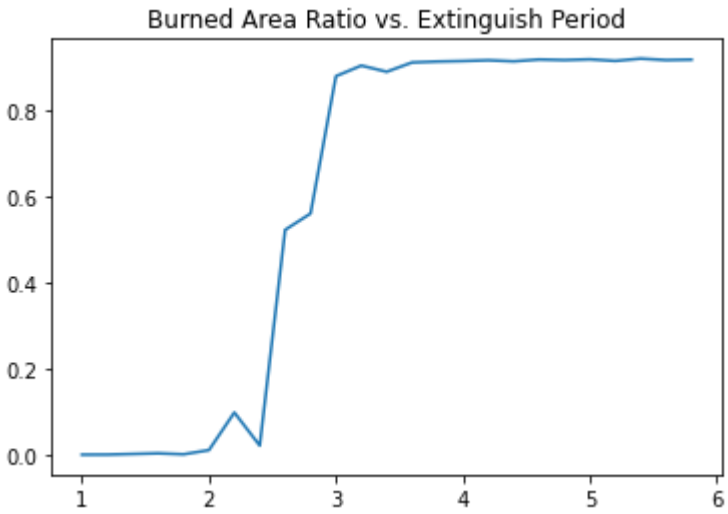




Here, we genrates the fire with no wind. Due to the existence of the pool and the water, the fire is stuck in the west and south directions, but spread to the north and east directions. With the extingshment starts at 2 timeunit, appears every 5 timeunit, lasting for 2 timeunit for each one, the fire is finally put out. The total burned area is 26.29% of the forest.

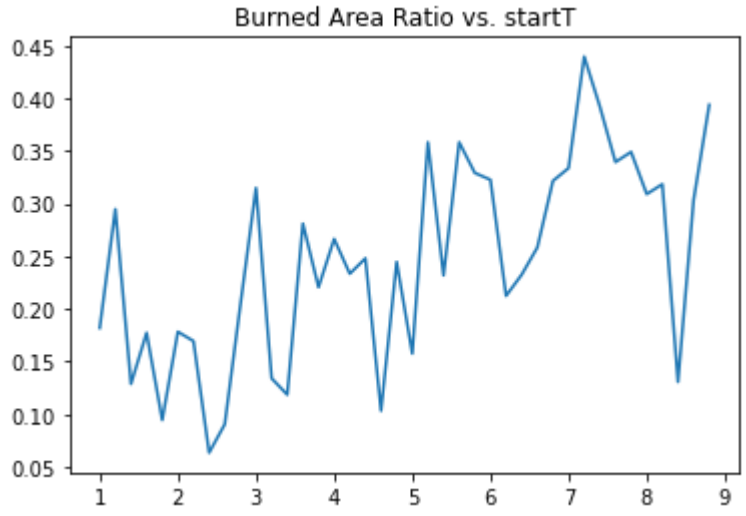
The influnce of the staring time, period, lasting time of the extinguishment on the ratio of the burned area is tested above.

In [23]:
```
def burned_area_exting_period():
    burned_areas=[]
    periods=np.arange(1,6,0.2)
    for period in periods:
        burned_areas.append(forest_fire_wind_water_extinguish(100,100,0.95,0,0
,0.02,0.01,1,1,period,False))
    plt.figure()
    plt.plot(periods,burned_areas)
    plt.title("Burned Area Ratio vs. Extinguish Period")

burned_area_exting_period()
```



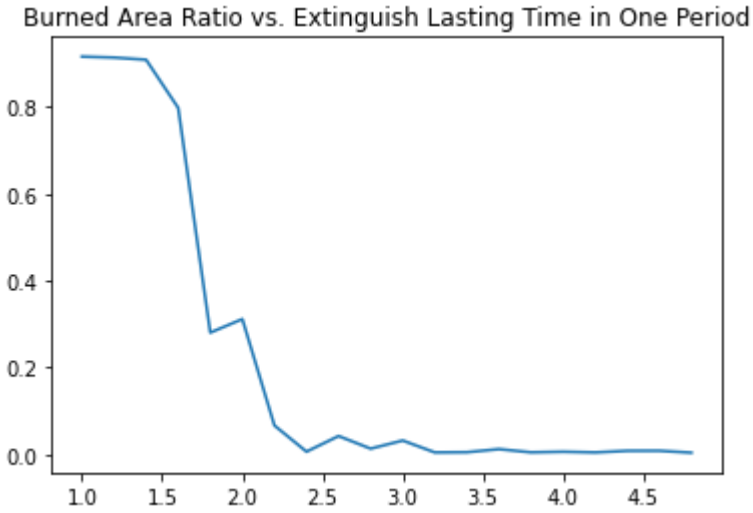
With the increase of the extinguishment period, the frequency of the extinguishment drops, which makes the fire hard to put out.There is a critcal point of the extinguishment period about 2 to 3. Below this extinguishment period, the fire can be put out with burned area ratio closed to ratio. Above it, the fire can't be put out with burned area ratio closed to original forest ratio 0.95.

In [20]:
```
def burned_area_startT():
    burned_areas=[]
    startTs=np.arange(1,9,0.2)
    for startT in startTs:
        burned_areas.append(forest_fire_wind_water_extinguish(100,100,0.95,0,0
,0.02,0.01,startT,2,5,False))
    plt.figure()
    plt.plot(startTs,burned_areas)
    plt.title("Burned Area Ratio vs. startT")
burned_area_startT()
```



With the change of the starting time of the extinguishment, the final burned area is not influnced much, which actually fluctates with respect to the starting time. It is reasonable, since we don't take the total area of extinguishment into concern. In other words, no matter how much region is involved in the fire, we always apply the extinguishment to the total graph, the whole forest, which is a waste of resources.

In [22]:
```
def burned_area_lastingT():
    burned_areas=[]
    lastingTs=np.arange(1,5,0.2)
    for lastingT in lastingTs:
        burned_areas.append(forest_fire_wind_water_extinguish(100,100,0.95,0,0
,0.02,0.01,2,lastingT,5,False))
    plt.figure()
    plt.plot(lastingTs,burned_areas)
    plt.title("Burned Area Ratio vs. Extinguish Lasting Time in One Period")
burned_area_lastingT()
```



With the change of the lasting time, the final burned area decreases with lasting time. As long as the lasting time reaches the period, the extinguishment takes place for all time for all area, which is not possible, which is 5 in the figure. However, the lasting time doesn't need to reach that large. The fire can be put out with the lasting time equals to 2 timeunit, as shown in the figure, which is very helpful information when determine the number of helicopters that are needed during the extinguishment.

In [ ]:
```
```

Part 2: A Reaction-Diffusion Model

In the 2^{nd} part of this tutorial, we'll apply the reation diffusion equation to the modeling of fire spread. The dynamics aspects of the fire forest propogation can be analyzed using the hyperbolic reation diffusion equations. Since the reation process is involved here, the tree catching on fire does not have to become char for sure now. Instead, the fire can burn the tree down or be set off due to artificial extinguishment or the climate change. Hence, the next status for a burning tree can be a burnt tree or a green tree. Also, the process take some time instead of being conducted within only one time step. The governing equation for the reaction can be found in some literature [1].

The derivation of the model

We have the nomenclature of parameters in the model as following table

Symbols	Parameters
D	Diffusion coefficient
N	The density of the burnt trees
τ	Relaxation time in the presence of the fire flux
F	Reaction term between green and burning trees
r	Reaction constant
Δt	Time increment of time discretization
Δx	Distance increment of spatial discretization
i, j	Grid indice

Here, we use N to denote the density of burnt trees among a certain region (a node in an n by n grid). $N = 1$ means that all the trees within that region are burnt, whereas $N = 0$ means that all the trees within that region are green. For $0 < N < 1$, a portion of N of the trees within the region is burning. Since the reation process is involved here, the tree catching on fire does not have to become char for sure now. Instead, the fire can burn the tree down or be set off due to artificial extinguishment or the climate change. Hence, the next status for a burning tree can be a burnt tree or a green tree.

If we want to study the spatial-temporal evolution of N, we can adopt the hyperbolic reaction-diffusion equation as following

$$\frac{\partial N}{\partial t} = D\frac{\partial^2 N}{\partial x^2} + F(N)$$

Notably, the reaction function can be expressed as following,

$$F(N) = r(1-N)^{\beta}N$$

where $\beta(>1)$ denotes the number of burning trees needed in order to set fire to a near green tree and we have the density of green trees as $1 - N$.

Quiz 1-Please provide the discretized governing equation for solving the forest fire model and the corresponding codes.

Applying the finite element method to solve the above PDE, we need to discretize the spatial grids and time increment first and have the following iterative relation,

$$N_{i,j}^{n+1} = N_{i,j}^{n} + \frac{D\Delta t}{\Delta x^2}(N_{i+1,j}^{n} + N_{i,j+1}^{n} - 4N_{i,j}^{n} + N_{i-1,j}^{n} + N_{i,j-1}^{n}) + r\Delta t(1 - N_{i,j}^{n})^{\beta} N_{i,j}^{n}$$

Following these states, we create a 100 by 100 grid with randomly picked state for each cell. The global status of the grid can be updated using FEM by each time step.

In [1]:

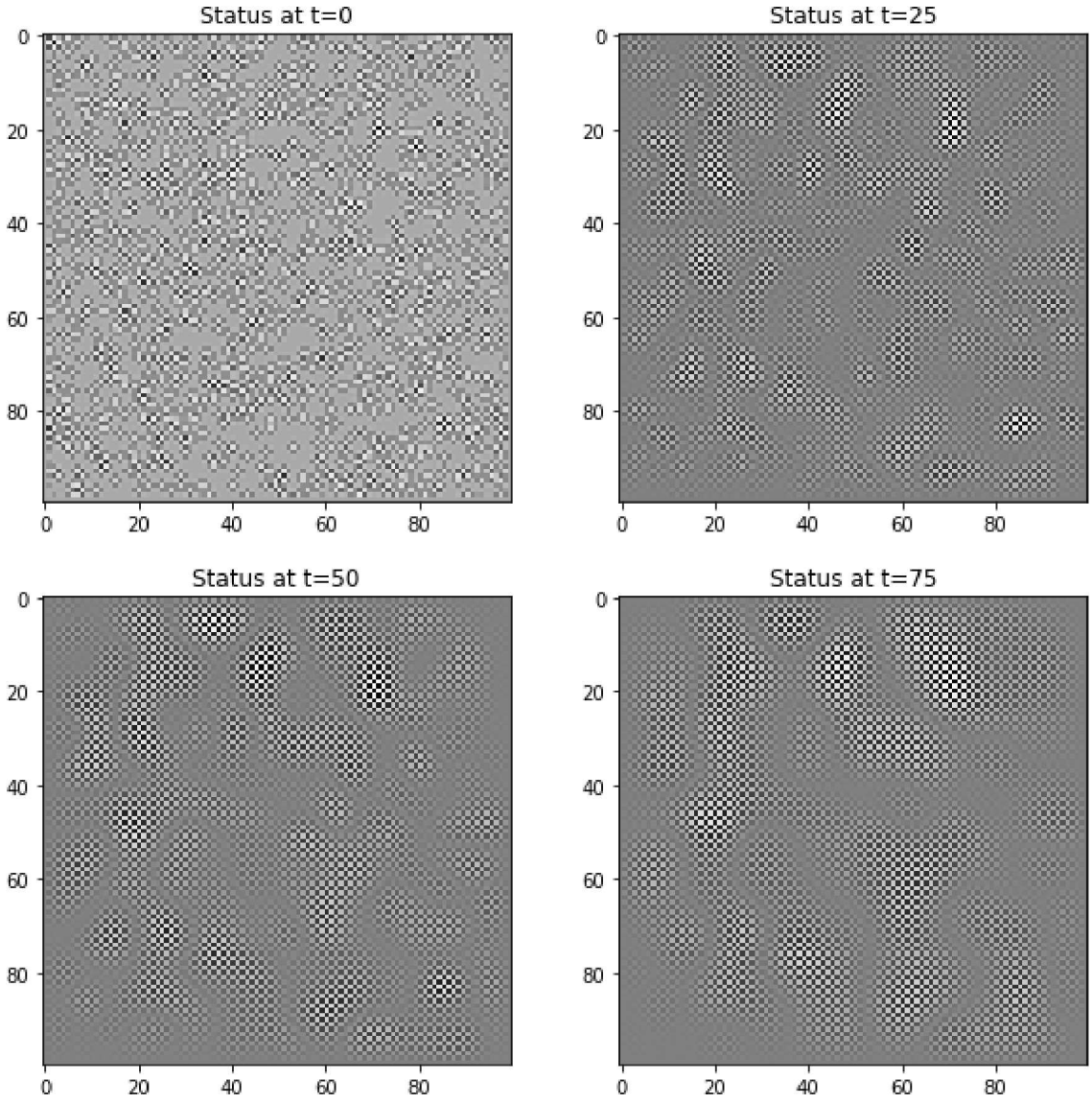
```
import numpy as np
import matplotlib.pyplot as plt
def EDM(r, beta, D, dt, dx, N, M, Soln0):

    Soln=np.zeros((N, M, M))
    Soln[0,:,:]=Soln0
    Coeff=D*dt/dx**2

    if(Coeff>=0.5): print("Hard to converge")
    for n in range(1,N-1):
        temp=(Soln[n-1,0:-2,1:-1]+Soln[n-1,1:-1,0:-2]-4*Soln[n-1,1:-1,1:-1]+Soln[n-1,1:
-1,2:]+Soln[n-1,2:,1:-1])
        Soln[n,1:-1,1:-1]+=(Coeff*temp+dt*r*(1-Soln[n-1,1:-1,1:-1])**beta*Soln[n-1,1:-1
,1:-1])

    return Soln
```

Quiz 2-Please solve the system given that $r = 10$, $\beta = 10$, $D = 10$.

Plot the results at transition time.

In [3]:

```
import matplotlib.pyplot as plt
import numpy as np

r=10
beta=10
D=100
dt=0.001
dx=1
N=100 #time disc
M=100 #space disc

Init=np.random.normal(0, 0.1, (M,M))
for i in range(M):
    for j in range(M):
        if (Init[i,j]<-0.1): Init[i,j]=1
        else: Init[i,j]=0
Soln=EDM(r, beta, D, dt, dx, N, M, Init)

plt.subplot(2,2,1)
plt.imshow(Soln[2,:,:],cmap="gray",interpolation="nearest")
plt.title("Status at t=0")
plt.subplot(2,2,2)
plt.imshow(Soln[24,:,:],cmap="gray",interpolation="nearest")
plt.title("Status at t=25")
plt.subplot(2,2,3)
plt.imshow(Soln[49,:,:],cmap="gray",interpolation="nearest")
plt.title("Status at t=50")
plt.subplot(2,2,4)
plt.imshow(Soln[98,:,:],cmap="gray",interpolation="nearest")
plt.title("Status at t=75")
fig=plt.gcf()
fig.set_size_inches(10,10)
plt.show()
```



After completing the above simulation, we consider about the reaction process further. Now we introduce another parameter into the forest fire system-τ as the relaxation time or the delay time in the appearance of the fire flux. This makes the process of the spread of fire flux more realistic. Accordingly, the PDE equation describing such diffusion-reaction model can be expresses as following. [1]

$$\frac{\partial N}{\partial t} = D\frac{\partial^2 N}{\partial x^2} + F(N) + \tau\frac{\partial F(N)}{\partial t}$$

Quiz 3-Please provide the discretized governing equation for solving the updated forest fire model and the corresponding codes.

Applying the finite element method to solve the above PDE, we need to discretize the spatial grids and time increment first and have the following iterative relation,

$$N_{i,j}^{n+1} = N_{i,j}^{n} + \frac{\frac{D\Delta t}{\Delta x^2}(N_{i+1,j}^{n} + N_{i,j+1}^{n} - 4N_{i,j}^{n} + N_{i-1,j}^{n} + N_{i,j-1}^{n}) + r\Delta t(1-N_{i,j}^{n})^{\beta}N_{i,j}^{n}}{1 - r(1-N_{i,j}^{n})^{\beta} + \beta r N_{i,j}^{n}(1-N_{i,j}^{n})^{\beta-1}}$$

Following these states, we create a 100 by 100 grid with randomly picked state for each cell. The global status of the grid can be updated using FEM by each time step.

In [19]:

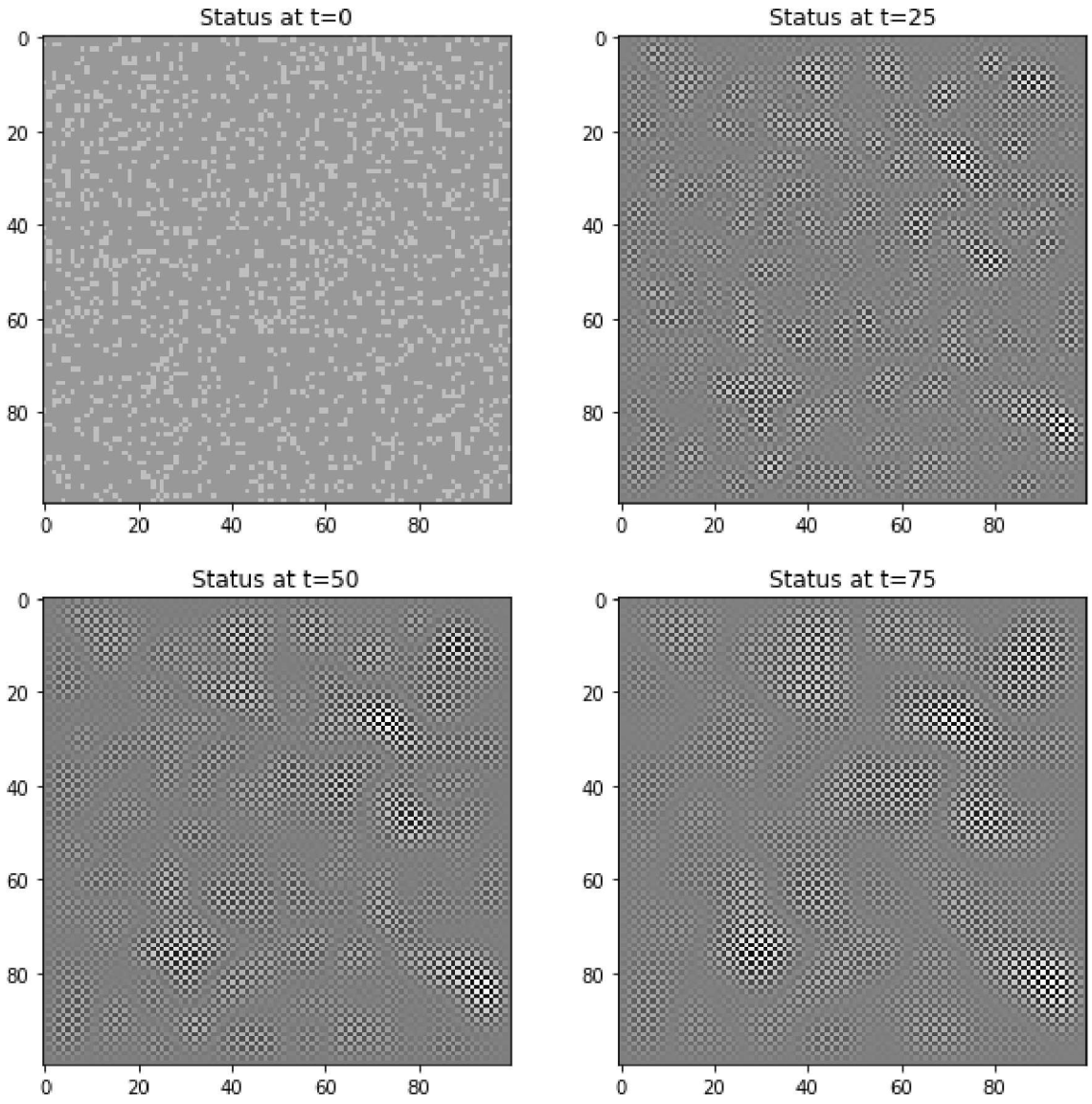
```
import numpy as np
import matplotlib.pyplot as plt
def EDM(tau, r, beta, D, dt, dx, N, M, Soln0):

    Soln=np.zeros((N, M, M))
    Soln[0,:,:]=Soln0
    Coeff=D*dt/dx**2

    if(Coeff>=0.5): print("Hard to converge")
    for n in range(1,N-1):
        temp=(Soln[n-1,0:-2,1:-1]+Soln[n-1,1:-1,0:-2]-4*Soln[n-1,1:-1,1:-1]+Soln[n-1,1:
-1,2:]+Soln[n-1,2:,1:-1])
        Soln[n,1:-1,1:-1]+=(Coeff*temp+dt*r*(1-Soln[n-1,1:-1,1:-1])**beta*Soln[n-1,1:-1
,1:-1])/(1-tau*r*(1-Soln[n-1,1:-1,1:-1])**beta+tau*beta*r*(1-Soln[n-1,1:-1,1:-1])**(bet
a-1)*Soln[n-1,1:-1,1:-1])

    return Soln
```

Quiz 4-Please solve the system given that $r = 10$, $\beta = 10$, $D = 10$, $\tau = 0.5$.

Plot the results at transition time.

In [31]:

```
import matplotlib.pyplot as plt
import numpy as np
tau = 10
r=10
beta=10
D=100
dt=0.001
dx=1
N=100 #time disc
M=100 #space disc

Init=np.random.normal(0, 0.1, (M,M))
for i in range(M):
    for j in range(M):
        if (Init[i,j]<-0.1): Init[i,j]=1
        else: Init[i,j]=0
Soln=EDM(tau, r, beta, D, dt, dx, N, M, Init)

plt.subplot(2,2,1)
plt.imshow(Soln[0,:,:],cmap="gray",interpolation="nearest")
plt.title("Status at t=0")
plt.subplot(2,2,2)
plt.imshow(Soln[24,:,:],cmap="gray",interpolation="nearest")
plt.title("Status at t=25")
plt.subplot(2,2,3)
plt.imshow(Soln[49,:,:],cmap="gray",interpolation="nearest")
plt.title("Status at t=50")
plt.subplot(2,2,4)
plt.imshow(Soln[98,:,:],cmap="gray",interpolation="nearest")
plt.title("Status at t=75")
fig=plt.gcf()
fig.set_size_inches(10,10)
plt.show()
```



Reference

[1] Méndez, V. and Llebot, J.E., 1997. Hyperbolic reaction-diffusion equations for a forest fire model. Physical Review E, 56(6), p.6557.

Contribution

Workload	Expected Contributor
Brainstorm the topic of the project and shape the scope of the simulation and then design the tutorial w.r.t. the experiments to conduct	Huaidong Yang & Yingdan Wu
Basic Cellular Model (Part 0) and Diffusion Reaction Model (Part 2)	Yingdan Wu
Advanced CA Model (Part 1) to include the effects of wind, extinguishment, water systems (river and lake), etc. using Markov Chain	Huaidong Yang