Computational Physics I

Name: Sudip KC

Physics | Class of 2022 | Jacobs University Bremen

One-Dimensional cellular automata

In the first part, the rule 150 was executed. Using this rule and starting with a single seed site the boundary conditions were tested.

Similarly, the different other rules like rule 30, rule 90, rule 18, rule 73 and rule 136 were also tested with a single seed at the center of the road as well as the random initial position on the road and it was made sure that the probability for each site to have the value 1 was 0.5, otherwise the probability was zero. The patterns formed can be seen below for different rules, and for use and non use of the periodic boundary conditions it was noticed that there was a slightly change in the nature of the pattern. One peculiar pattern was seen with rule 136. Since it results 1 at time t+1 only when we get pattern 111 or 011 at time t. So, with an initial seed at the centre, which represented 010 we got 0 for all the latteice points at time t+1. Similarly results were seen in case of random seeds as well after few time steps.

Finally, the traffic rule 184 was also implemented with periodic boundary conditions. A traffic jam with 6 of 18 cars, 12 of 36 cars as well as 18 of 54 cars were simulated in a ring shaped street of 100 cells and the resulting pattern was plotted below.

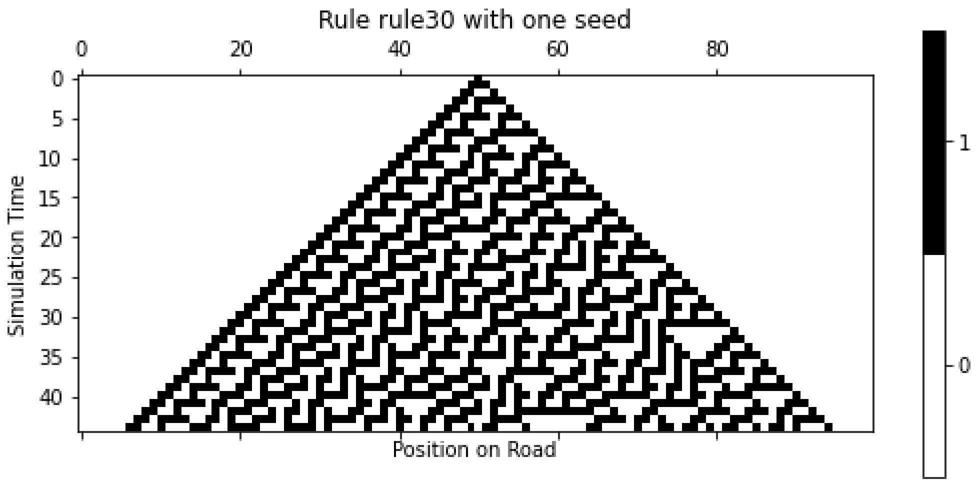
Validation of the program

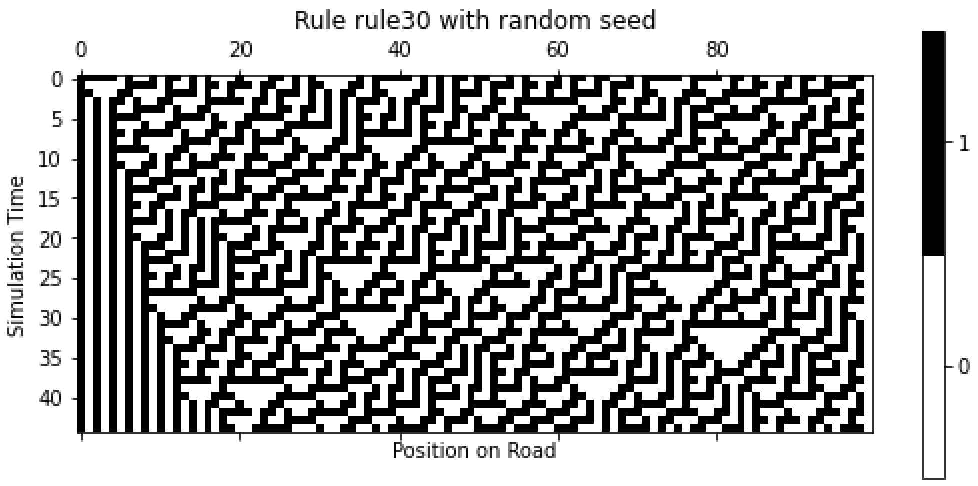
The nature of the pattern which was obtained from the computed program for different rules were compared to that of the theoritical plots given in the lecture slides. The two patterns were found to be identical, which gives us the concrete proof that our computed program was correct. Later on, for the part with implementation of the traffic rule 184, doubling and tripling the cars on the street gave us more bulky (dense) traffic which is in accordance with the real world behaviour.

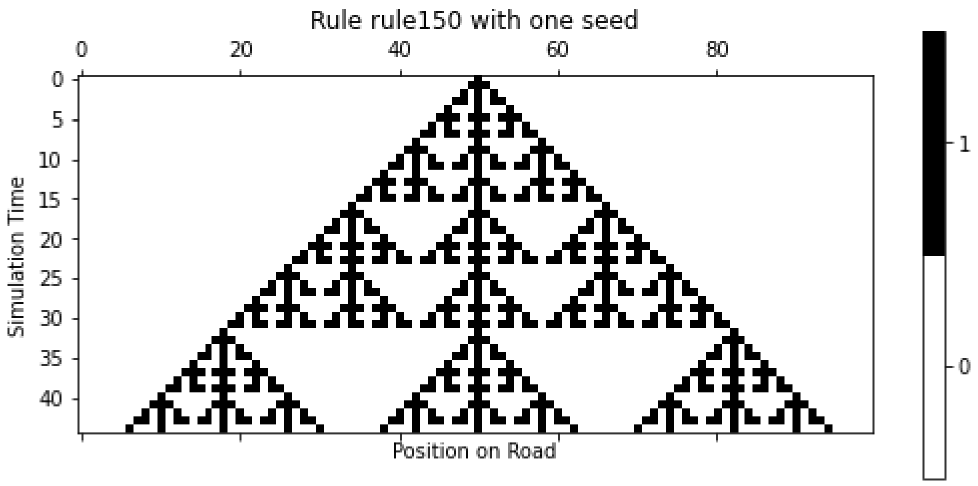
In [3]:
```python
from pylab import *
import numpy as np
import matplotlib.pyplot as plt
from random import random, randint

N = 100     #number of sites
no_iterration = 45  #number of iterations
boundrycondition = False

shape = (no_iterration, N)       #Lattice Matrix
matrix = np.zeros(shape)
matrix[0][int(N/2)] = 1

trafficjam= False
no_cars, traffic_cars = 18, 6
```

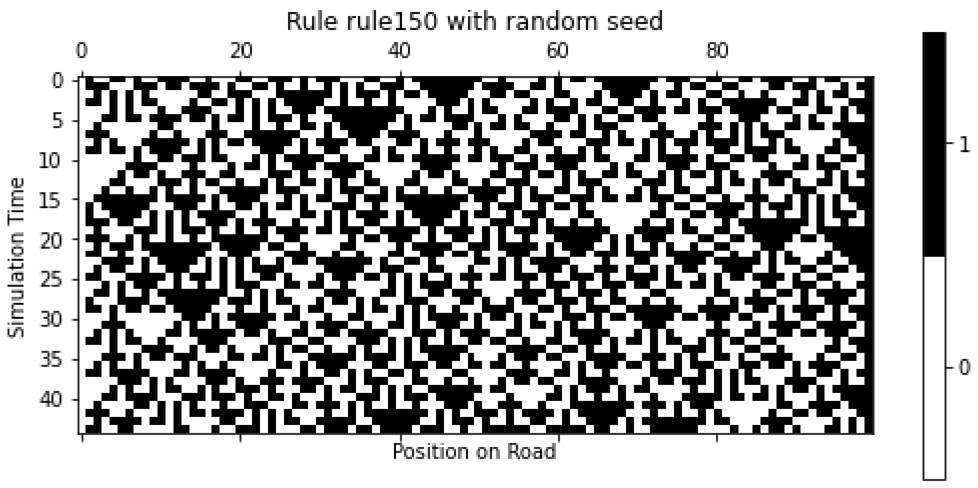
```
#Converting binary strings to decimal for rule defination.
def bintodec(left, center, right):
    binary = str(int(left)) + str(int(center)) + str(int(right))
    return int(binary, 2)


#Initial feed of the Lattice matrix based on traffic flow.
def populate():
    matrix = np.zeros(shape)
    if(trafficjam != True):
        for i in range(N):
            r = random()
            if(r <= 0.5):
                matrix[0][i]= 1
            else:
                 matrix[0][i]= 0
        return matrix
    else:
        pos = randint(0, N - traffic_cars)     #Position of the traffic jam
        print(pos)
        for i in range(traffic_cars):         #Populate cars at position of t
            matrix[0][pos+i] = 1
        for j in range (no_cars - traffic_cars): #Populate remaining cars  ra
            r = randint (0, 99)
            while (r>= pos and r<=pos+traffic_cars):
                 r = randint (0, 99)
            matrix[0][r] = 1
        return matrix

def Plot(result, title):
#get discrete colormap for results plotting
    cmap = plt.get_cmap('Greys', np.max(result)-np.min(result)+1)
    mat = plt.matshow(result,cmap=cmap,vmin = np.min(result)-.5, vmax = np.ma
    cax = plt.colorbar(mat, ticks=np.arange(np.min(result),np.max(result)+1))
    plt.xlabel('Position on Road')
    plt.ylabel('Simulation Time')
    plt.title(title, loc='center')
    plt.show()


 #Implementaion of different rules
def rule150(left, center, right):
    return (left+center+right)%2

def rule30(left, center, right):
    x = bintodec(left, center, right)
    if (x == 1 or x == 2 or x == 3 or x == 4):
        return 1
    else:
        return 0

def rule90(left, center, right):
    x = bintodec(left, center, right)
    if (x == 1 or x == 3 or x == 4 or x == 6):
        return 1
    else:
        return 0


def rule18(left, center, right):
    x = bintodec(left, center, right)
    if (x == 1 or x == 4):
        return 1
    else:
        return 0


def rule73(left, center, right):
    x = bintodec(left, center, right)
    if (x == 0 or x == 3 or x == 6 ):
        return 1
    else:
        return 0


def rule136(left, center, right):
    x = bintodec(left, center, right)
    if ( x == 3 or x == 7):
        return 1
    else:
        return 0


def rule184(left, center, right):
    x = bintodec(left, center, right)
    if ( x == 3 or x == 4 or x==5 or x==7):
        return 1
    else:
        return 0


    #Iteration at all lattice points for time evolution of the traffic.
def iterate(matrix, rule):
    for i in range(1, no_iterration):
        for j in range (N):
            if (j==0 or j==N-1):
                if(boundrycondition):   #Creating ring structure of the stree
                    left, right = matrix[i-1, N-1], matrix[i-1, 0]
                else:
                    left, right = 0, 0
            else:
                left, right = matrix[i-1][j-1], matrix[i-1][j+1]
            center = matrix[i-1][j]
            matrix[i][j] = rule(left, center, right)
    return matrix


#Plotting position on the road as a function of simulation time for different
rules = [rule30, rule150, rule90, rule18, rule73, rule136]
for rule in rules:
    result1 = iterate(matrix, rule)
    Plot(result1, 'Rule {} with one seed'.format(rule.__name__))
    p_matrix = populate()
    result2 = iterate(p_matrix, rule)
    Plot(result2, 'Rule {} with random seed'.format(rule.__name__))

#Simulating traffic jam by implementing rule 184 for different car numbers.
trafficjam = True

p_matrix = populate()
result2 = iterate(p_matrix, rule184)
Plot(result2, 'Rule 184 traffic with 6 out of 18 cars')

no_cars, traffic_cars = 36, 12
p_matrix = populate()
result2 = iterate(p_matrix, rule184)
Plot(result2, 'Rule 184 traffic with 12 out of 36 cars')
```

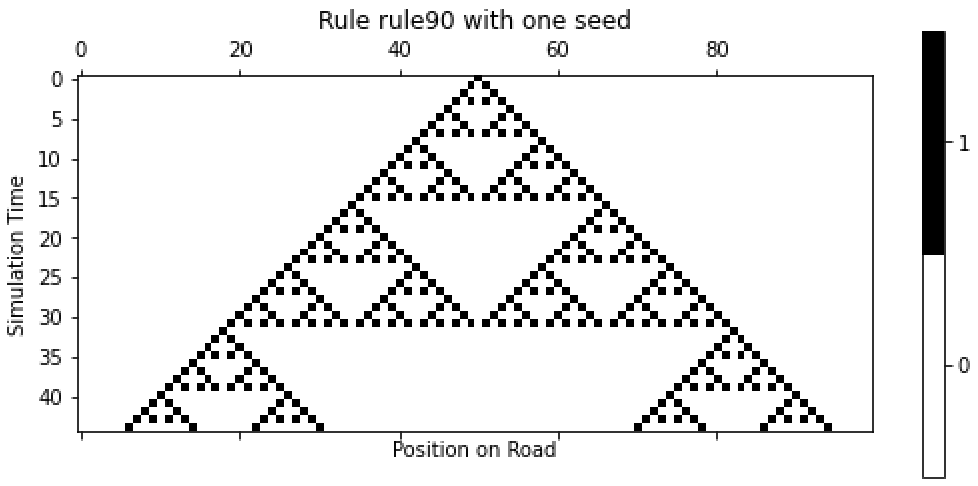
```
no_cars, traffic_cars = 54, 18
p_matrix = populate()
result2 = iterate(p_matrix, rule184)
Plot(result2, 'Rule 184 with 18 out of 54 cars')
```

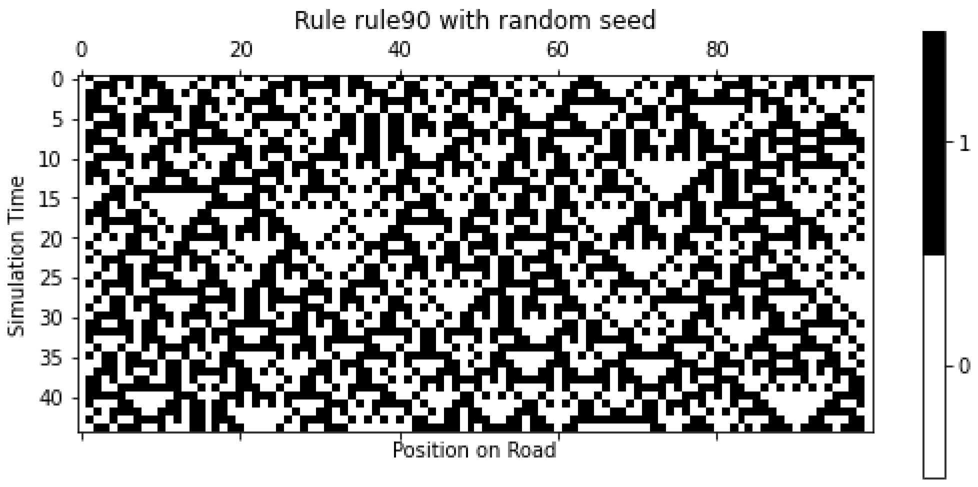


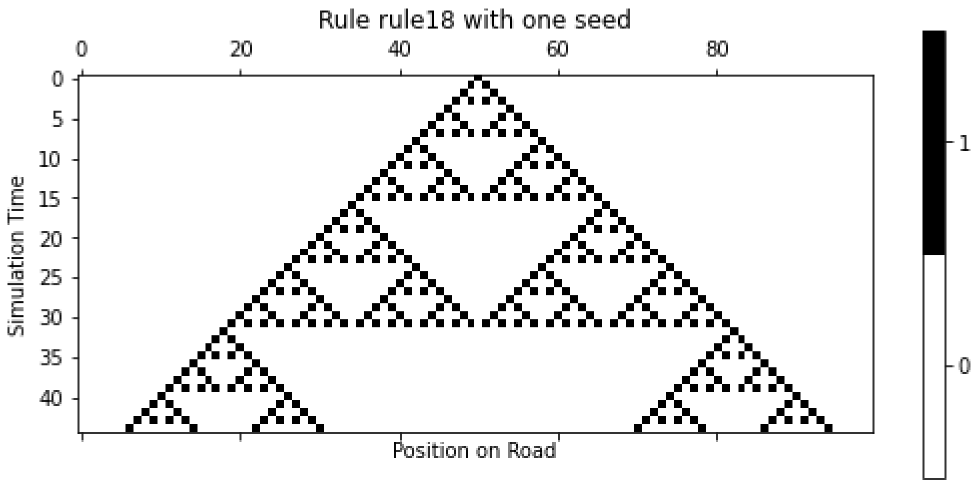


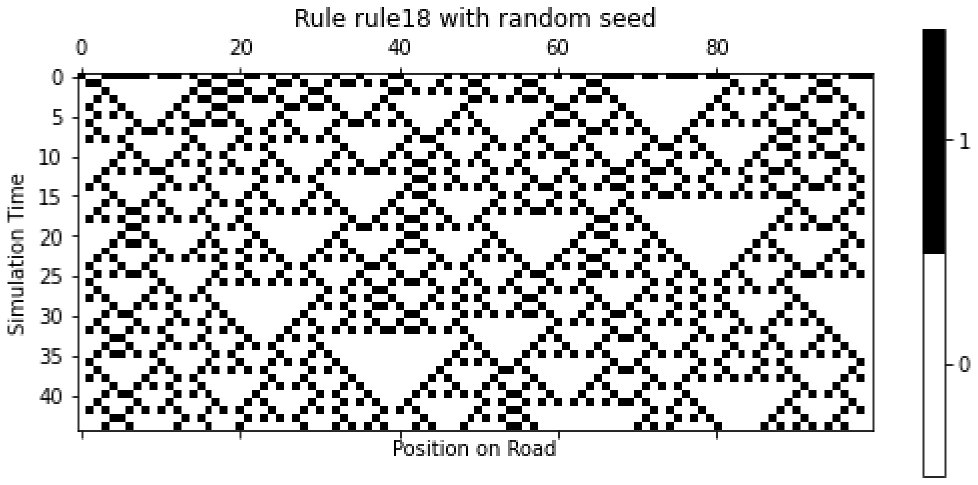


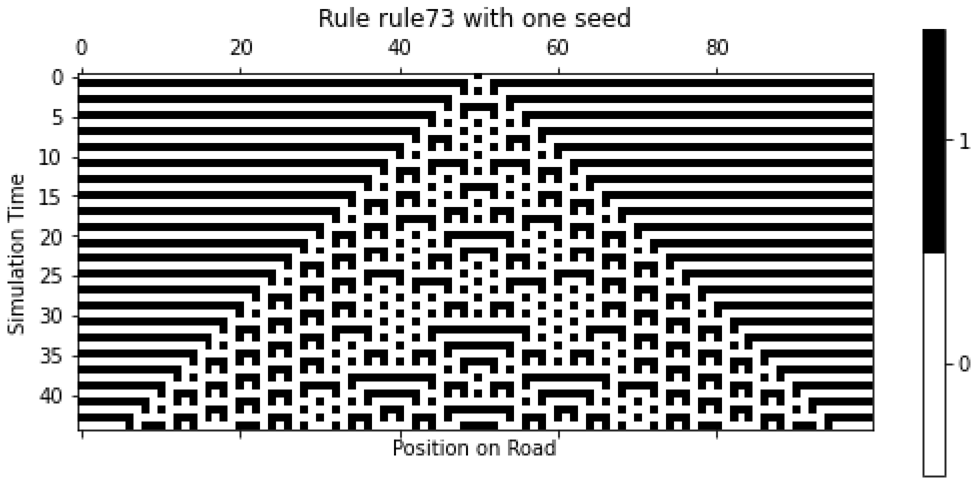

Rule rule150 with random seed
Position on Road
Simulation Time
1
0

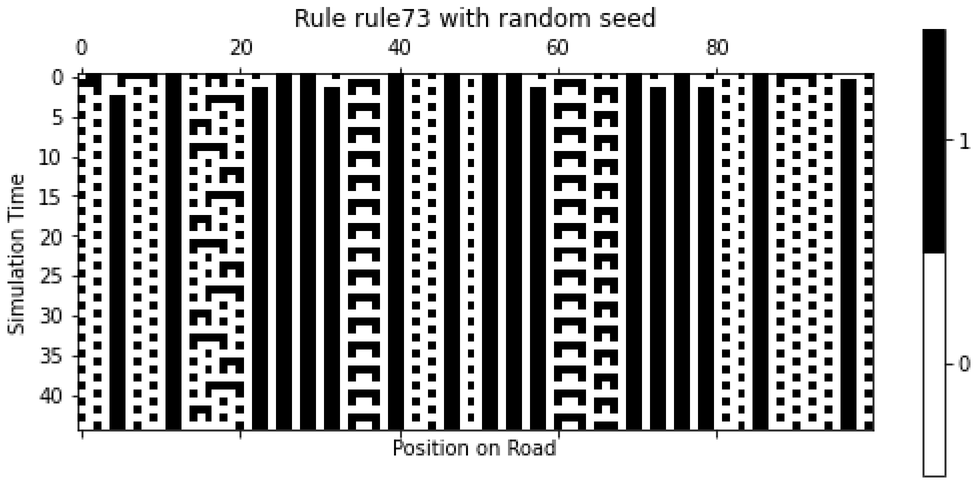

Rule rule90 with one seed
Position on Road
Simulation Time
1
0

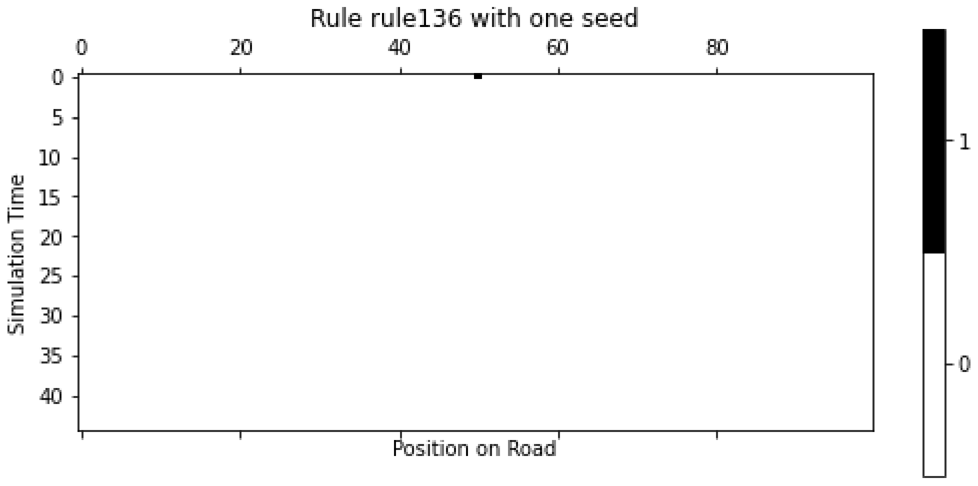

Rule rule90 with random seed
Position on Road
Simulation Time
1
0

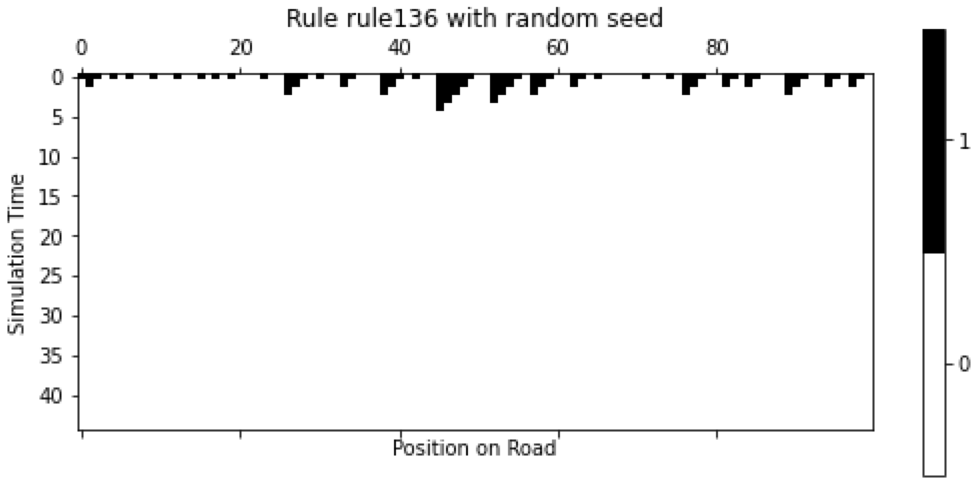

Rule rule18 with one seed
Position on Road
Simulation Time
1
0


Rule rule18 with random seed
Simulation Time
Position on Road
1
0

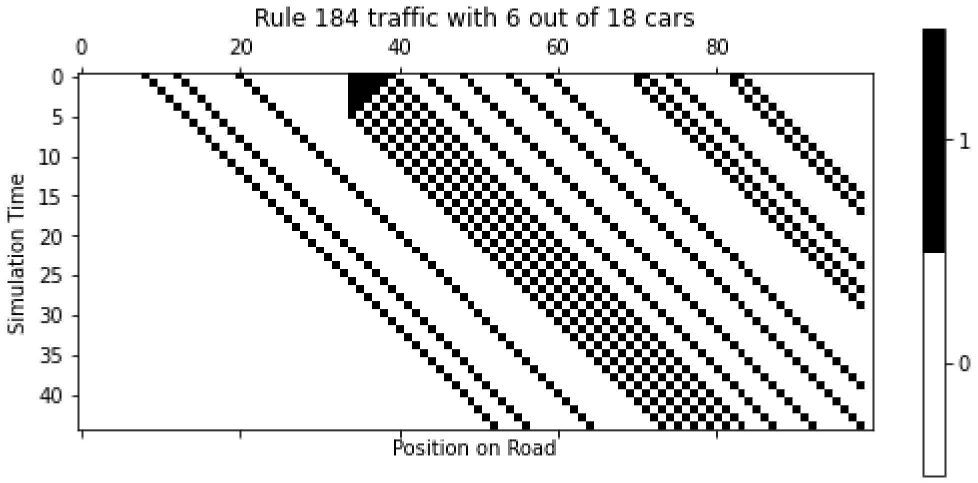

Rule rule73 with one seed
Simulation Time
Position on Road
1
0


Rule rule73 with random seed
Simulation Time
Position on Road
1
0

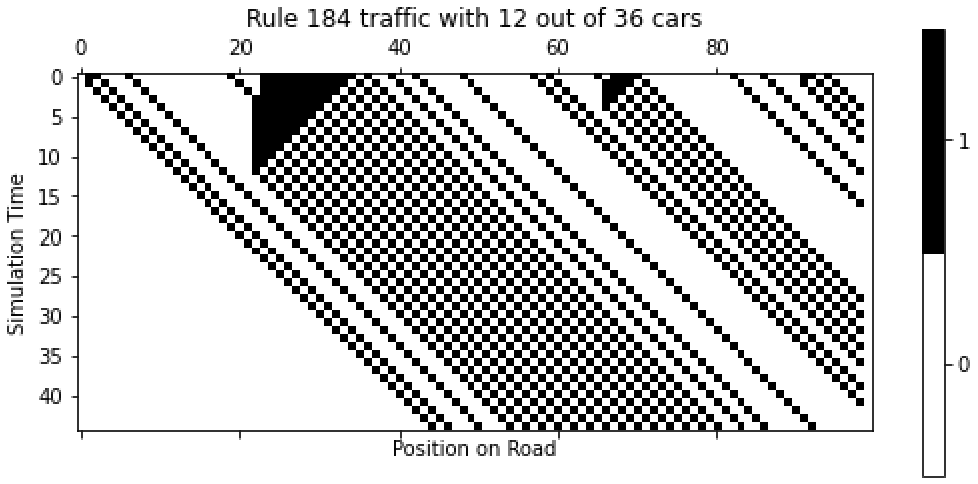

Rule rule136 with one seed
Simulation Time
Position on Road
1
0


Rule rule136 with random seed
Simulation Time
Position on Road

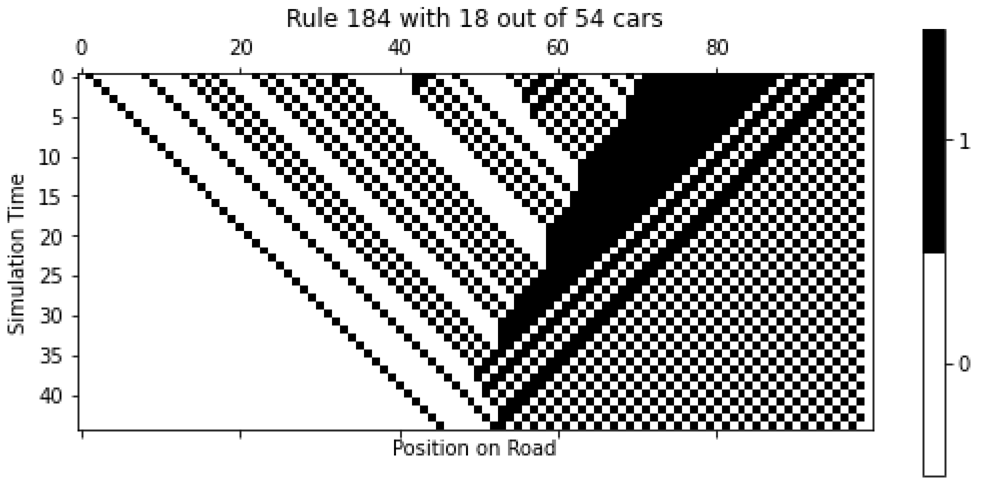
```
34
```


Rule 184 traffic with 6 out of 18 cars
Simulation Time
Position on Road

```
23
```


Rule 184 traffic with 12 out of 36 cars
Simulation Time
Position on Road

```
71
```


Rule 184 with 18 out of 54 cars
0
20
40
60
80
Simulation Time
0
5
10
15
20
25
30
35
40
Position on Road
1
0

```
In [ ]:
```

```
In [ ]:
```