Numerical Methods : Rate equations or MC approaches for modelling growth - Task 2 Report -

Léo BECHET, M2 CompuPhys 2024-2025

Task 2

Disclaimer

Due to rerunning the whole notebook multiple times, images used in the interpretations will differ from the simulation.

Please check the first code block below if you do not have numba installed. Use `task2and3` and not `task2_parallel`

In [1]:
```
# from task2and3 import Simulation, Monomer ; _IS_PARALLEL = False# Witho
from task2_parallel import Simulation ; _IS_PARALLEL = True# With numba


from matplotlib.colors import ListedColormap
import os
import time
import numpy as np
import matplotlib.pyplot as plt
```

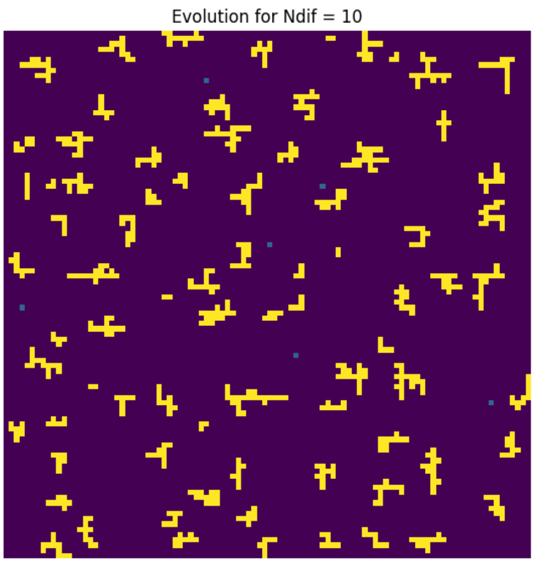
In [2]:
```
# from task2and3 import Simulation, Monomer ; _IS_PARALLEL = False# Witho
from task2_parallel import Simulation ; _IS_PARALLEL = True# With numba






from matplotlib.colors import ListedColormap
import os
import time
import numpy as np
import matplotlib.pyplot as plt

def RunForNdif(Ndif_A, steps=40000, size=(100,100), coverage_limit=0.08):
    island_cellEvo = []
    island_numEvo = []
    monomer_numEvo = []

    # size = (100, 100)
    sim = Simulation(size, 0.5)

    # steps = 40000
    # Ndif_A = 50
```

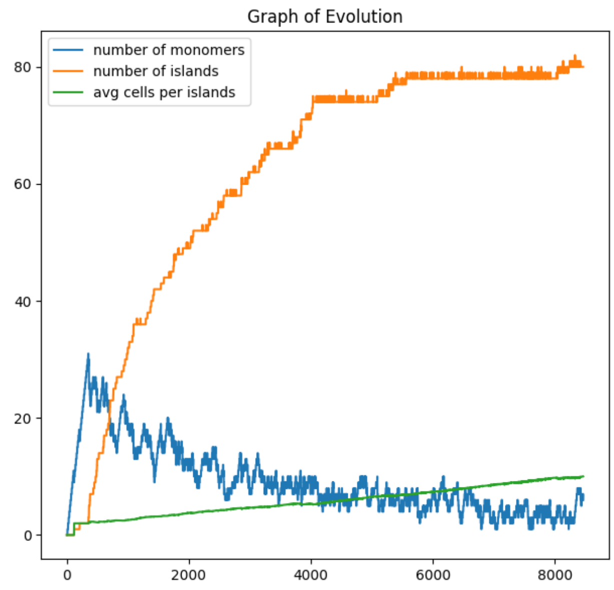
```python
    # coverage_limit = 0.2 # stop limiter


    sim_cells = size[0]*size[1]
    i=0
    while True:
        i+=1
        # deposit a monomer every n steps
        if i%Ndif_A == 0:
            sim.Deposit("A")

        sim.Step() # step

        # Compute average number of cells per island
        isl, cells = sim.NumIslands()
        island_numEvo.append(isl)
        monomer_numEvo.append(sim.NumMonomers())


        try:
            island_cellEvo.append( sum(cells)/len(cells) )
        except ZeroDivisionError:
            island_cellEvo.append( 0 )

        # print("========================================================
        fill_ratio = sum(cells)/sim_cells # < Replaced below when not usi
        # =========== DEBUG ARRAY FILL RATIO =================
        if not _IS_PARALLEL:
            new_array = np.array([[1 if isinstance(cell, Monomer) else 0
            fill_ratio = np.sum(new_array.flatten())/sim_cells


        # ====================================================

        if i%1000 == 0:os.system("clear");print(f'SIM {i}\t{fill_ratio*10

        #Stop condition due to coverage limit, here we take aggregated co
        if fill_ratio >= coverage_limit:
            print('Reached 20% fill')
            break




    # Create a figure with two subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

    # Plot the image on the left
    ax1.imshow([
        [0 if isinstance(x, str) or x == 0 or not getattr(x, 'aggregated'
        for row in sim.grid
    ])



    # Grid for print
    # Initialize a new array with the same shape
    new_array = np.zeros(sim.grid.shape, dtype=int)

    # Set values based on conditions
    new_array[sim.grid == 0] = 0
    new_array[sim.grid == 1] = 1
    new_array[sim.grid == 2] = 2
    new_array[sim.grid == 11] = 3
    new_array[sim.grid == 12] = 4

    if _IS_PARALLEL: cbar = ax1.imshow(new_array, cmap='viridis')


    ax1.set_title(f'Evolution for Ndif = {Ndif_A}')
    ax1.axis('off')  # Hide the axis

    # Plot the graphs on the right
    ax2.plot(monomer_numEvo, label="number of monomers")
    ax2.plot(island_numEvo, label="number of islands")
    ax2.plot(island_cellEvo, label="avg cells per islands")
    ax2.set_title('Graph of Evolution')
    ax2.legend()

    # Adjust layout to prevent overlap
    plt.tight_layout()

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

Let's use the above function to run multiple simulations with different N_dif

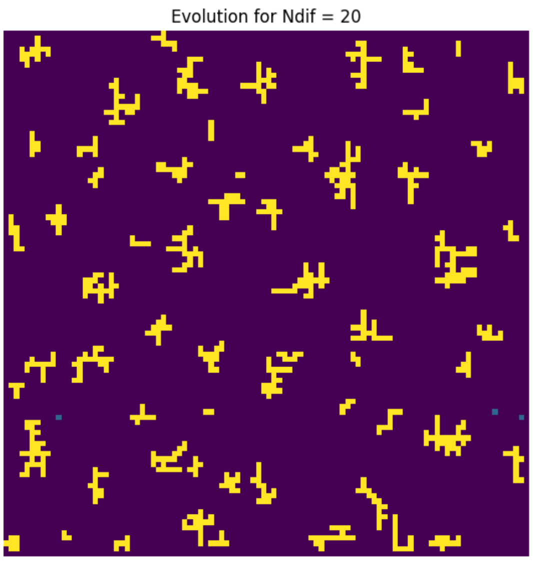
In [21]:
```
RunForNdif(10)
RunForNdif(20)
RunForNdif(50)
```

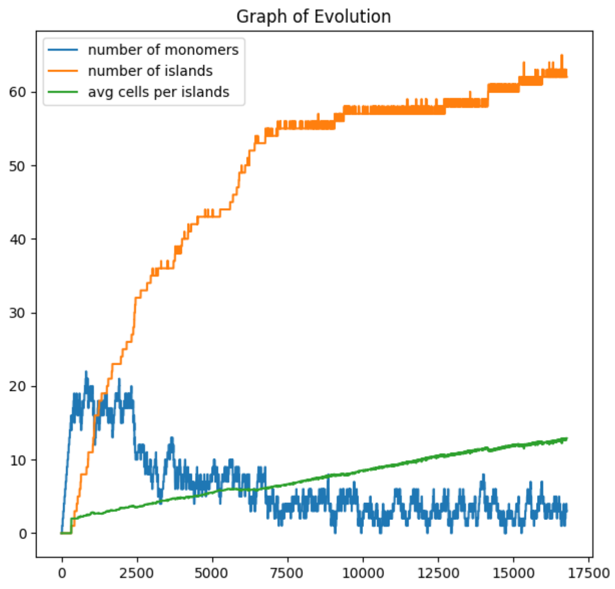
```
Sim v 0.2 (Numba Parallelized)
SIM 1000        0.8099999999999999%
SIM 2000        1.8599999999999999%
SIM 3000        2.88%
SIM 4000        3.84%
SIM 5000        4.83%
SIM 6000        5.75%
SIM 7000        6.710000000000001%
SIM 8000        7.630000000000001%
Reached 20% fill
```



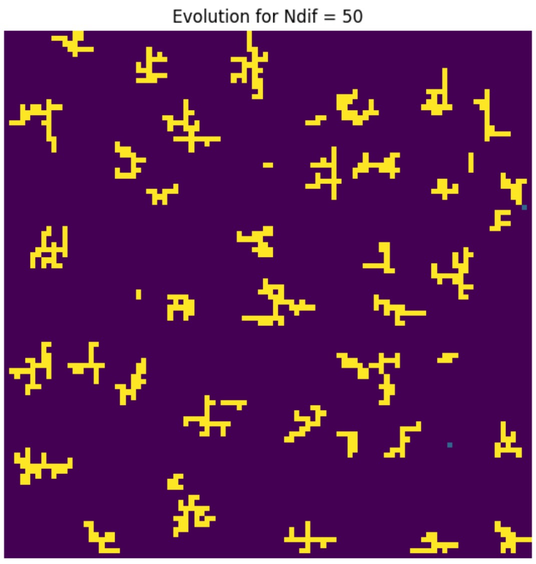


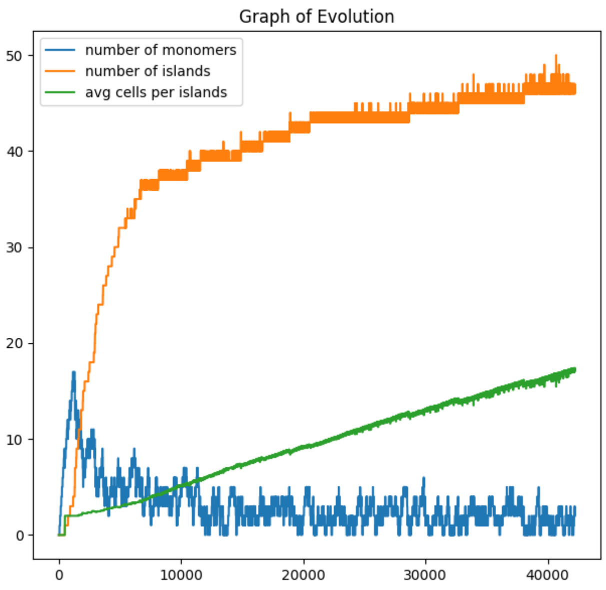
```
Sim v 0.2 (Numba Parallelized)
SIM 1000        0.31%
SIM 2000        0.83%
SIM 3000        1.4200000000000002%
SIM 4000        1.8900000000000001%
SIM 5000        2.3800000000000003%
SIM 6000        2.8899999999999997%
SIM 7000        3.39%
SIM 8000        3.8899999999999997%
SIM 9000        4.37%
SIM 10000       4.87%
SIM 11000       5.28%
SIM 12000       5.7700000000000005%
SIM 13000       6.260000000000001%
SIM 14000       6.65%
SIM 15000       7.21%
SIM 16000       7.630000000000001%
Reached 20% fill
```





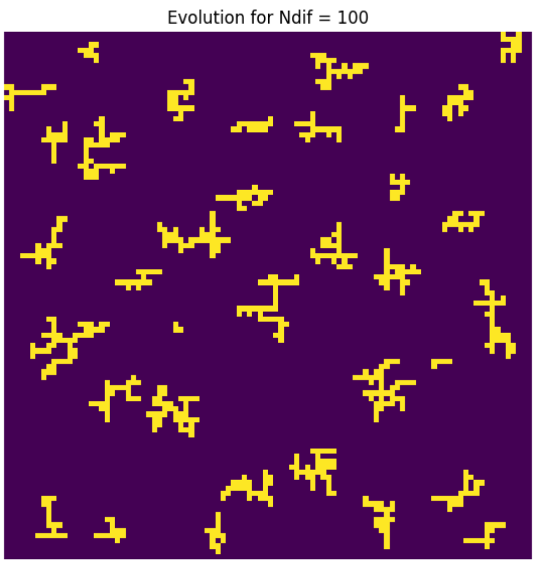
```
Sim v 0.2 (Numba Parallelized)
SIM 1000        0.06%
SIM 2000        0.33999999999999997%
SIM 3000        0.53%
SIM 4000        0.75%
SIM 5000        0.9199999999999999%
SIM 6000        1.11%
SIM 7000        1.34%
SIM 8000        1.51%
SIM 9000        1.73%
SIM 10000       1.92%
SIM 11000       2.15%
SIM 12000       2.36%
SIM 13000       2.53%
SIM 14000       2.73%
SIM 15000       2.92%
SIM 16000       3.1199999999999997%
SIM 17000       3.3099999999999996%
SIM 18000       3.49%
SIM 19000       3.6799999999999997%
SIM 20000       3.8899999999999997%
SIM 21000       4.09%
SIM 22000       4.26%
SIM 23000       4.47%
SIM 24000       4.66%
SIM 25000       4.8500000000000005%
SIM 26000       5.050000000000001%
SIM 27000       5.19%
SIM 28000       5.4%
SIM 29000       5.58%
SIM 30000       5.79%
SIM 31000       5.93%
SIM 32000       6.16%
SIM 33000       6.34%
SIM 34000       6.510000000000001%
SIM 35000       6.710000000000001%
SIM 36000       6.9%
SIM 37000       7.07%
SIM 38000       7.249999999999999%
SIM 39000       7.42%
SIM 40000       7.6%
SIM 41000       7.779999999999999%
SIM 42000       7.9799999999999995%
Reached 20% fill
```

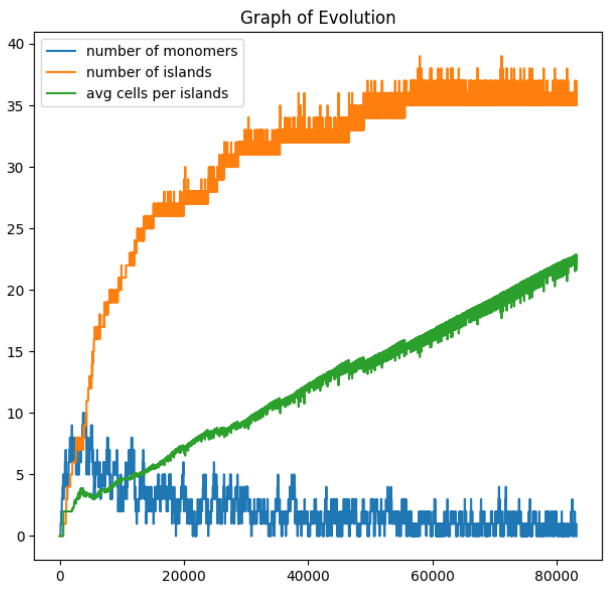




```
In [22]: RunForNdif(100)
```

```
Sim v 0.2 (Numba Parallelized)
SIM 1000        0.04%
SIM 2000        0.13%
SIM 3000        0.24%
SIM 4000        0.31%
SIM 5000        0.41000000000000003%
SIM 6000        0.53%
SIM 7000        0.64%
SIM 8000        0.73%
SIM 9000        0.83%
SIM 10000       0.95%
SIM 11000       1.03%
SIM 12000       1.13%
SIM 13000       1.23%

...

SIM 77000       7.470000000000001%
SIM 78000       7.5600000000000005%
SIM 79000       7.64%
SIM 80000       7.739999999999999%
SIM 81000       7.82%
SIM 82000       7.9%
SIM 83000       7.9799999999999995%
Reached 20% fill
```



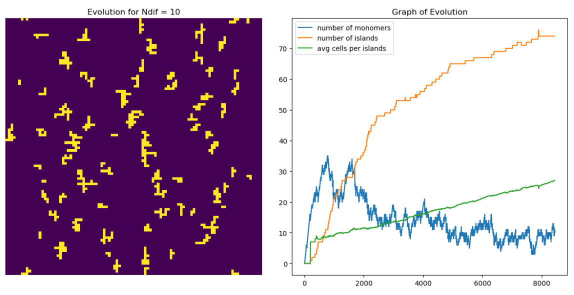


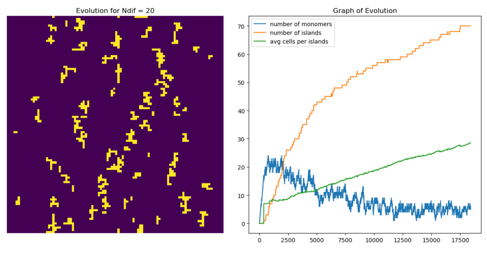
N_{dif} and $\frac{F}{D}$ relation

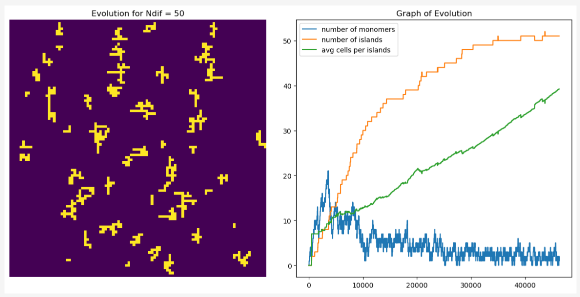
N_{dif} represents the number of simulation steps inbetween 2 deposition of a monomer on the grid. F is the deposition rate, and D is the diffusion rate. Thus, N_{dif} can be expressed as a ratio of the diffusion rate to the deposition rate :

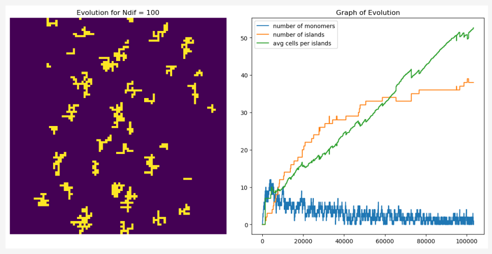
$$N_{dif} = \frac{D}{F} \Leftrightarrow \frac{F}{D} = \frac{1}{N_{dif}}$$

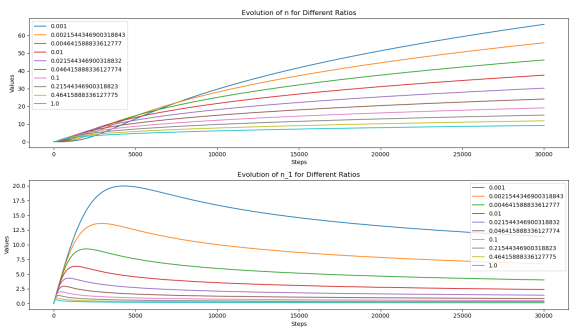
Monomers and islands evolution









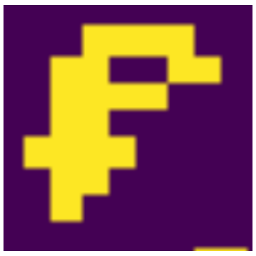
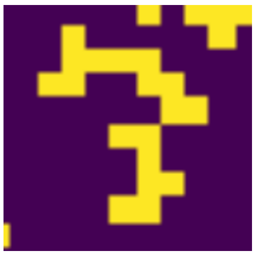
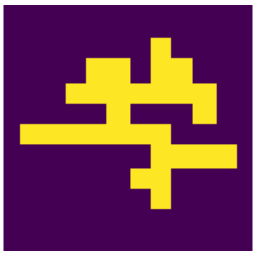
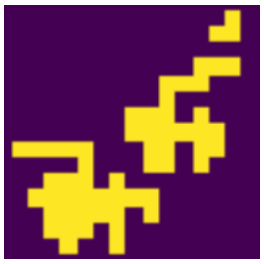
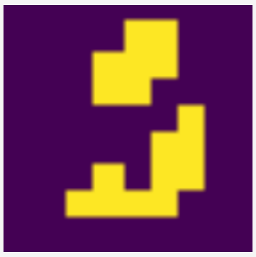
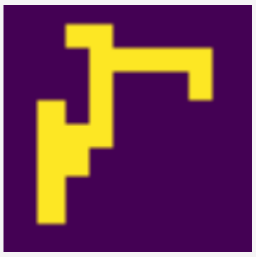
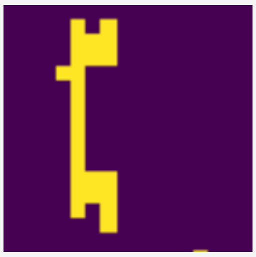
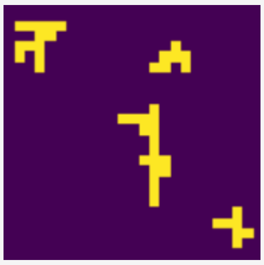


Please note that for readability we stopped the simulation around 8% fill, not 20% as specified in the subject.

The 4 images above seems to display the start of the curves we have obtained in Task 1, displayed at the bottom. We can recognise the shape fo n_1, the evolution of density of monomers, and n the evolution of number of islands.

We would need to run more simulations with a larger grid size in order to approximate an infinite grid to confirm that it does indeed fit well to the Task 1 theory. It would allow a smoother evolution and reduce the amounts of bumps in the curves. Furthermore, averaging the results of multiple simulations would also better or results. Unfortunately we do not have access to such computing power. One could however try to fit the theoretical parameters of a simulation using the evolution of n and n_1. Here due to the relatively small scale of the gird and the computation time already required for very shaky curves, we will not do that. We'd like to mention that we may develop a parallelization technique in the technical report of the simulation.

In our case, we can see the number of island stabilizing, meaning we are still in the phase of creation of nucleation sites. The limit on coverage does not allow us to go past. We could try with bigger simulations, but we do our code is currently not optimized enough and doesn't support parallelization.

$N_{dif} = 10$	$N_{dif} = 20$	$N_{dif} = 50$	$N_{dif} = 100$
			
			

$N_{dif} = 10$	$N_{dif} = 20$	$N_{dif} = 50$	$N_{dif} = 100$

In the table above, each column has 4 screenshots of different islands inside its simulation. We can see that the average size of the islands increase with the value of N_{dif}. This is due to the fact that monomers have a higher amount of steps inbetween the addition of a monomer. This give them a higher chance of aggregating with an already existing island.

The average minimum size of islands seems to also be increasing. From 2 to 5 cells for $N_{dif} = 10$ to 7+ for $N_{dif} = 100$.

We can also spot the emergence of more complex structures. $N_{dif} = 10$ has a tendency to have smaller blobs, where $N_{dif} = 100$ leads to more "thready" islands. We also have empty spots directly inside islands.

Deposition in a limited area

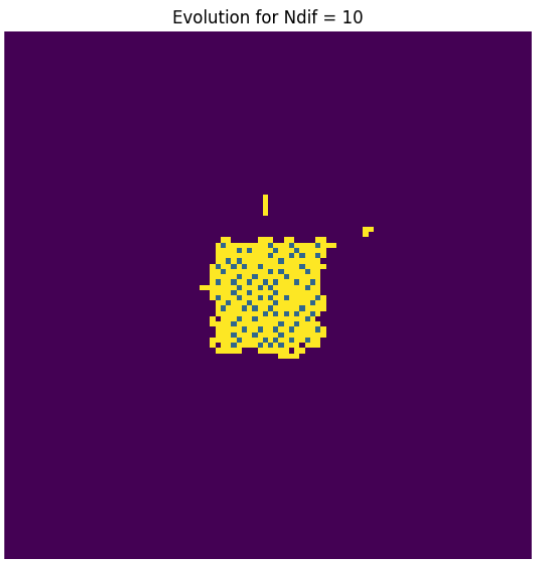
In [29]:

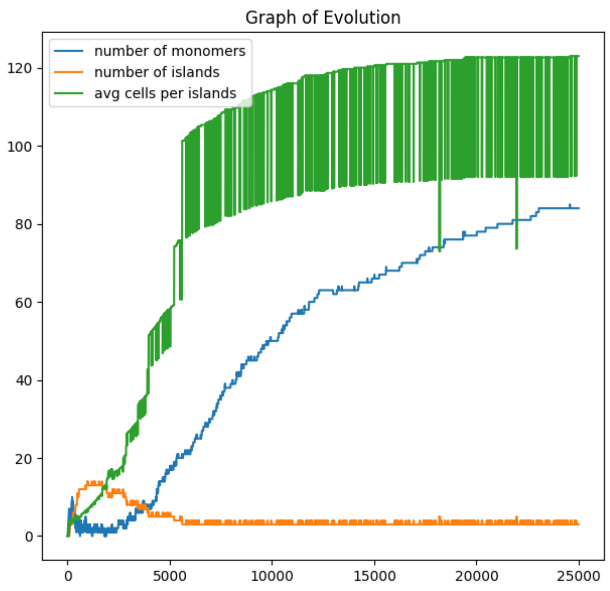
```
def RunForNdif_centered(Ndif_A, steps=40000, size=(100,100), coverage_lim
    island_cellEvo = []
    island_numEvo = []
    monomer_numEvo = []

    # size = (100, 100)
    sim = Simulation(size, 0.5)

    # steps = 40000
    # Ndif_A = 50
    # coverage_limit = 0.2 # stop limiter


    sim_cells = size[0]*size[1]
    i=0
    while True:
        i+=1
        # deposit a monomer every n steps
        if i%Ndif_A == 0:
            sim.Deposit("A", 50, 50, 20, 20) # < Only modified line here,
    sim.Step() # step

    # Compute average number of cells per island
    isl, cells = sim.NumIslands()
    island_numEvo.append(isl)
    monomer_numEvo.append(sim.NumMonomers())


    try:
        island_cellEvo.append( sum(cells)/len(cells) )
    except ZeroDivisionError:
        island_cellEvo.append( 0 )

    # print("=====================================================
    fill_ratio = sum(cells)/sim_cells # < Replaced below when not usi
    # =========== DEBUG ARRAY FILL RATIO ==================
    if not _IS_PARALLEL:
        new_array = np.array([[1 if isinstance(cell, Monomer) else 0
        fill_ratio = np.sum(new_array.flatten())/sim_cells
    # ====================================================

    if i%1000 == 0:os.system("clear");print(f'SIM {i}\t{fill_ratio*10

    #Stop condition due to coverage limit, here we take aggregated co
    if fill_ratio >= coverage_limit or i>steps:
        print('Reached 20% fill')
        break




# Create a figure with two subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Plot the image on the left
ax1.imshow([
    [0 if isinstance(x, str) or x == 0 or not getattr(x, 'aggregated'
    for row in sim.grid
])



# Grid for print
# Initialize a new array with the same shape
new_array = np.zeros(sim.grid.shape, dtype=int)

# Set values based on conditions
new_array[sim.grid == 0] = 0
new_array[sim.grid == 1] = 1
new_array[sim.grid == 2] = 2
new_array[sim.grid == 11] = 3
new_array[sim.grid == 12] = 4

if _IS_PARALLEL: cbar = ax1.imshow(new_array, cmap='viridis')


ax1.set_title(f'Evolution for Ndif = {Ndif_A}')
ax1.axis('off')  # Hide the axis
```

```python
    # Plot the graphs on the right
    ax2.plot(monomer_numEvo, label="number of monomers")
    ax2.plot(island_numEvo, label="number of islands")
    ax2.plot(island_cellEvo, label="avg cells per islands")
    ax2.set_title('Graph of Evolution')
    ax2.legend()

    # Adjust layout to prevent overlap
    plt.tight_layout()

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

In [30]:
```python
RunForNdif_centered(10, steps=25000) #< maxx 25k steps
```

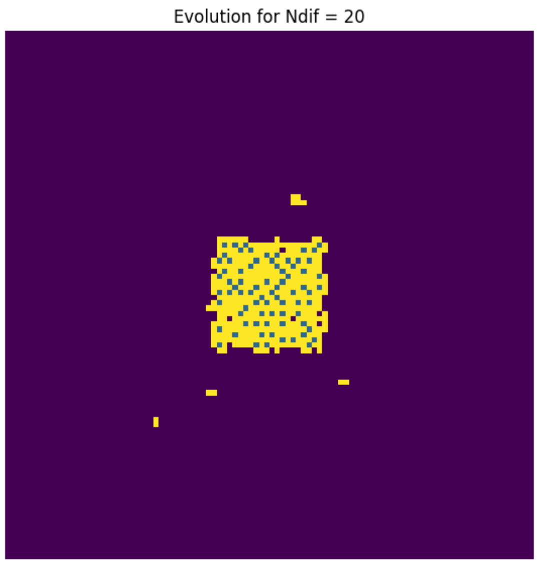
```
Sim v 0.2 (Numba Parallelized)
SIM 1000        0.8999999999999999%
SIM 2000        1.6400000000000001%
SIM 3000        2.1399999999999997%
SIM 4000        2.58%
SIM 5000        2.9000000000000004%
SIM 6000        3.1%
SIM 7000        3.19%
SIM 8000        3.2800000000000002%
SIM 9000        3.38%
SIM 10000       3.4299999999999997%
SIM 11000       3.4799999999999995%
SIM 12000       3.54%
SIM 13000       3.5700000000000003%
SIM 14000       3.5900000000000003%
SIM 15000       3.61%
SIM 16000       3.63%
SIM 17000       3.63%
SIM 18000       3.65%
SIM 19000       3.66%
SIM 20000       3.6799999999999997%
SIM 21000       3.6799999999999997%
SIM 22000       3.6799999999999997%
SIM 23000       3.6799999999999997%
SIM 24000       3.6799999999999997%
SIM 25000       3.6900000000000004%
Reached 20% fill
```

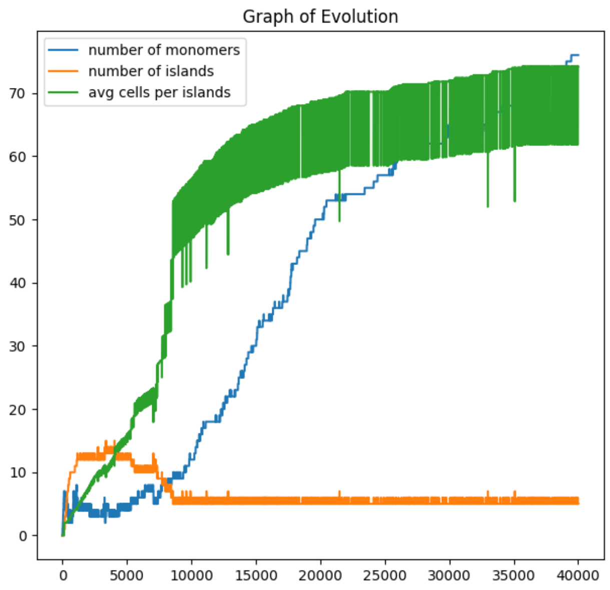




```
In [32]: RunForNdif_centered(20, steps=40000)
```

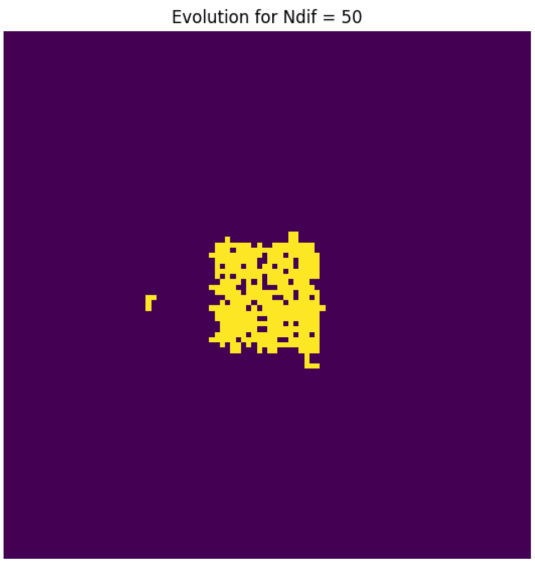
```
Sim v 0.2 (Numba Parallelized)
SIM 1000        0.44%
SIM 2000        0.89%
SIM 3000        1.23%
SIM 4000        1.63%
SIM 5000        1.91%
SIM 6000        2.17%
SIM 7000        2.32%
SIM 8000        2.55%
SIM 9000        2.7%
SIM 10000       2.82%
SIM 11000       2.96%
SIM 12000       3.0300000000000002%
SIM 13000       3.1199999999999997%
SIM 14000       3.2099999999999995%
SIM 15000       3.25%
SIM 16000       3.29%
SIM 17000       3.34%
SIM 18000       3.39%
SIM 19000       3.4099999999999997%
SIM 20000       3.44%
SIM 21000       3.46%
SIM 22000       3.51%
SIM 23000       3.51%
SIM 24000       3.51%
SIM 25000       3.51%
SIM 26000       3.55%
SIM 27000       3.5700000000000003%
SIM 28000       3.5700000000000003%
SIM 29000       3.5900000000000003%
SIM 30000       3.5999999999999996%
SIM 31000       3.62%
SIM 32000       3.64%
SIM 33000       3.64%
SIM 34000       3.6700000000000004%
SIM 35000       3.6999999999999997%
SIM 36000       3.71%
SIM 37000       3.71%
SIM 38000       3.71%
SIM 39000       3.71%
SIM 40000       3.71%
Reached 20% fill
```

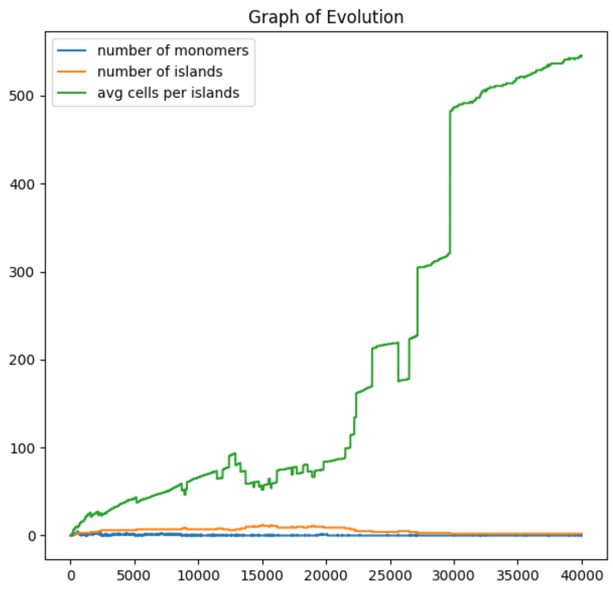




In [7]:
```
RunForNdif_centered(50)
```

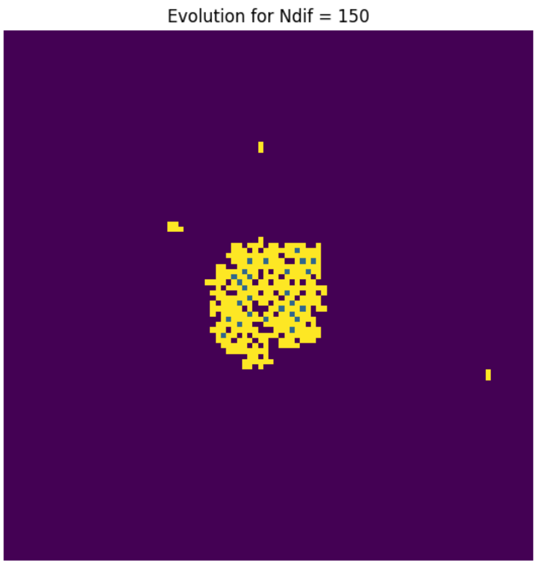
```
Sim v 0.2
SIM 1000        0.48%
SIM 2000        1.06%
SIM 3000        1.71%
SIM 4000        2.19%
SIM 5000        2.52%
SIM 6000        2.98%
SIM 7000        3.34%
SIM 8000        3.73%
SIM 9000        4.2%
SIM 10000       4.63%
SIM 11000       4.99%
SIM 12000       5.26%
SIM 13000       5.62%
SIM 14000       5.92%
SIM 15000       6.239999999999999%
SIM 16000       6.59%
SIM 17000       6.87%
SIM 18000       7.12%
SIM 19000       7.31%
SIM 20000       7.5600000000000005%
SIM 21000       7.829999999999999%
SIM 22000       8.02%
SIM 23000       8.3%
SIM 24000       8.62%
SIM 25000       8.709999999999999%
SIM 26000       8.84%
SIM 27000       9.07%
SIM 28000       9.21%
SIM 29000       9.45%
SIM 30000       9.71%
SIM 31000       9.83%
SIM 32000       9.98%
SIM 33000       10.190000000000001%
SIM 34000       10.25%
SIM 35000       10.4%
SIM 36000       10.489999999999998%
SIM 37000       10.61%
SIM 38000       10.73%
SIM 39000       10.82%
SIM 40000       10.91%
```

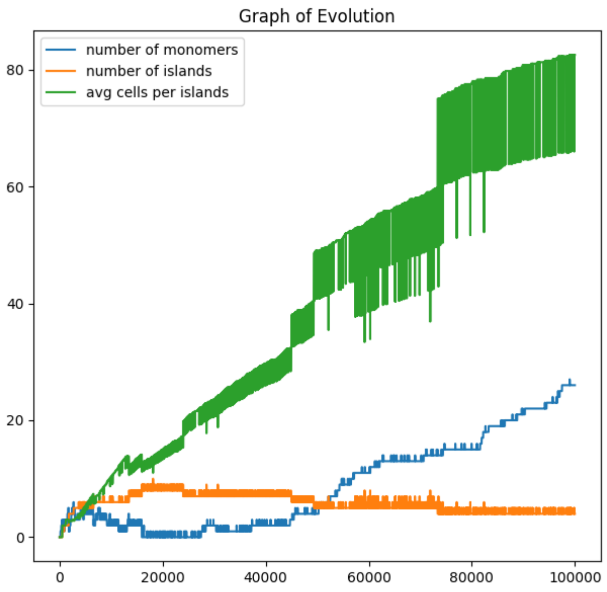




```
In [34]: RunForNdif_centered(150, steps=100000)
```

```
Sim v 0.2 (Numba Parallelized)
SIM 1000        0.04%
SIM 2000        0.11%
SIM 3000        0.16%
SIM 4000        0.2%
SIM 5000        0.27%
SIM 6000        0.33%
SIM 7000        0.41000000000000003%
SIM 8000        0.48%
SIM 9000        0.54%
SIM 10000       0.62%

...

SIM 94000       3.25%
SIM 95000       3.26%
SIM 96000       3.27%
SIM 97000       3.29%
SIM 98000       3.29%
SIM 99000       3.29%
SIM 100000      3.3000000000000003%
Reached 20% fill
```





We are unfortunately limited by the implementation of our simulation. When limiting the deposition zone, we would need to have a small deposition area compared to the total size, aswell as a very high N_{dif}. This is not the case here.

We would expect that for higher values of diffusion, the particles will move further inside the simulation. the density would look like a 2D gaussian with a plateau of the size of, and centered on, the deposition area. We would also see the same thing for the mean size of islands.

Looking at the evolution of the average cells per islands, we can spot the time when islands are joined together by a monomer.

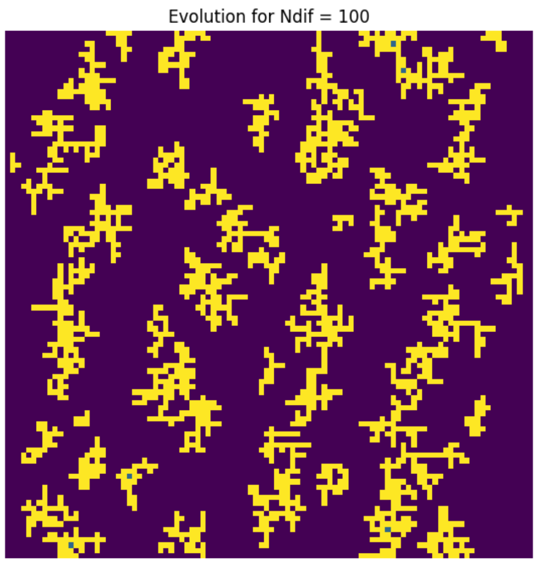
Different diffusion rate in function of the

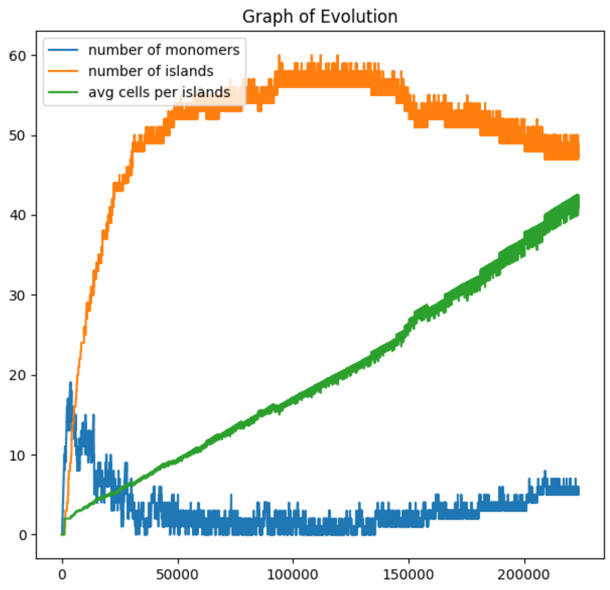
direction

```
In [38]: def RunForNdif_rate(Ndif_A, rate, steps=40000, size=(100,100), coverage_l
             island_cellEvo = []
             island_numEvo = []
             monomer_numEvo = []

             # size = (100, 100)
             sim = Simulation(size, rate)

             # steps = 40000
             # Ndif_A = 50
             # coverage_limit = 0.2 # stop limiter


             sim_cells = size[0]*size[1]
             i=0
             while True:
                 i+=1
                 # deposit a monomer every n steps
                 if i%Ndif_A == 0:
                     sim.Deposit("A") # < Only modified line here, limits the depo

                 sim.Step() # step

                 # Compute average number of cells per island
                 isl, cells = sim.NumIslands()
                 island_numEvo.append(isl)
                 monomer_numEvo.append(sim.NumMonomers())


                 try:
                     island_cellEvo.append( sum(cells)/len(cells) )
                 except ZeroDivisionError:
                     island_cellEvo.append( 0 )

                 # print("=========================================================
                 fill_ratio = sum(cells)/sim_cells # < Replaced below when not usi
                 # =========== DEBUG ARRAY FILL RATIO =================
                 if not _IS_PARALLEL:
                     new_array = np.array([[1 if isinstance(cell, Monomer) else 0
                     fill_ratio = np.sum(new_array.flatten())/sim_cells
                 # ====================================================

                 if i%1000 == 0:os.system("clear");print(f'SIM {i}\t{fill_ratio*10

                 #Stop condition due to coverage limit, here we take aggregated co
                 if fill_ratio >= coverage_limit :
                     print('Reached 20% fill')
                     break




             # Create a figure with two subplots
             fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

             # Plot the image on the left
             ax1.imshow([
        [0 if isinstance(x, str) or x == 0 or not getattr(x, 'aggregated'
        for row in sim.grid
    ])



    # Grid for print
    # Initialize a new array with the same shape
    new_array = np.zeros(sim.grid.shape, dtype=int)

    # Set values based on conditions
    new_array[sim.grid == 0] = 0
    new_array[sim.grid == 1] = 1
    new_array[sim.grid == 2] = 2
    new_array[sim.grid == 11] = 3
    new_array[sim.grid == 12] = 4

    if _IS_PARALLEL: cbar = ax1.imshow(new_array, cmap='viridis')


    ax1.set_title(f'Evolution for Ndif = {Ndif_A}')
    ax1.axis('off')  # Hide the axis

    # Plot the graphs on the right
    ax2.plot(monomer_numEvo, label="number of monomers")
    ax2.plot(island_numEvo, label="number of islands")
    ax2.plot(island_cellEvo, label="avg cells per islands")
    ax2.set_title('Graph of Evolution')
    ax2.legend()

    # Adjust layout to prevent overlap
    plt.tight_layout()

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

In [39]:
```
RunForNdif_rate(100,0.01)
```

```
Sim v 0.2 (Numba Parallelized)
SIM 1000        0.0%
SIM 2000        0.06%
SIM 3000        0.16%

...

SIM 217000      19.470000000000002%
SIM 218000      19.54%
SIM 219000      19.63%
SIM 220000      19.71%
SIM 221000      19.8%
SIM 222000      19.88%
SIM 223000      19.950000000000003%
Reached 20% fill
```





We implement the difference of diffusion using the ration $rate = \frac{N_{dif}(x)}{N_{dif}(y)}$. The current $rate$ is set to 0.01, which would be equivalent to $N_{dif}(x) = 10$ and $N_{dif}(y) = 10^3$.

Looking at the finale grid, we can spot that the islands seems to "trail" in the direction of the highest rate. This is due to the fact that, by changing the rate of diffusion on X compared to Y, we effectively, "stretch" the simulation.