Digital Urban Visualization: Introduction to Programming II

06.10.2014

Chair of Information Architecture, ETH Zürich

- Recap of last week.
- How to read CSV files.
- More dimensional lists.
- Cellular Automata and other grid based simulations.
- Exercise.

Menu Items of Today

What is the difference?

```
for a in [1,3]:
    a = a * a
    print(a)
```

```
for a in [1,3]:
    a = a * a
print(a)
```

What is this?

```
def square(a):
    ret = a * a
    return ret
```

Recap of last Week

Introduction to Programming II

Last week we saw how to read a CSV file. But we also want to store data to a file. This can be done in a similar way as writing, but with just a few different commands.

Problem: We have two lists, one containing names and the other containing ages.

Names = ['Hans', 'John', 'Carl']

Ages = [23,41,32]

We want to store them in a CSV file to, for example, process them in Excel.

```
import csv
f = open('path/to/age.csv', 'r')
csvReader = csv.reader(f, delimiter = ';')
ageSum = 0

for row in csvReader:
        ageSum += int(row[1])
f.close()
print(ageSum)
```

```
import csv
f = open('path/to/age.csv', 'w')
csvWriter = csv.writer(f, delimiter = ';')
names = ['Hans','John','Carl']
ages = [23,41,32]
for i in range(0,len(names)):
        currRow = [names[i], ages[i]]
        csvWriter.writerow(currRow)
f.close()
```

Writing CSV Data Files

Introduction to Programming III

This code produces a CSV file with the name ages.csv.

The content of it looks the following:

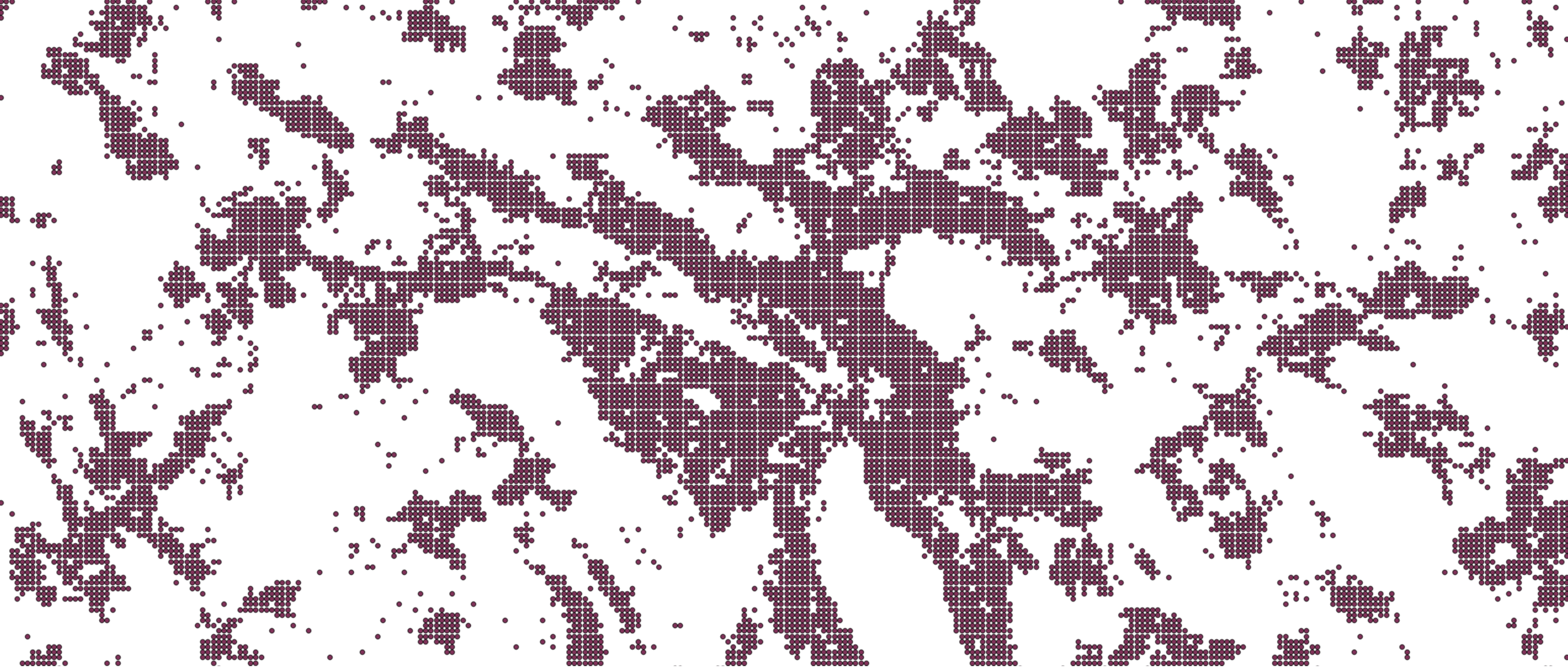
Hans, 23

John, 41

Carl, 32

It is possible to have lists within lists within lists within lists …

They can get very useful when we, for example, work with data wich is grid based and in two dimensions. An example for this is the data you can get from the Swiss Federal Statistics office.



More Dimensional Lists

Introduction to Programming III

A way to work with such data is to use two-dimensional lists. For example:

data = [[1,2,3],[4,5,6],[7,8,9]]

This can be understood as:

```
[[ 1, 2, 3]
 [ 4, 5, 6]
 [ 7, 8, 9]]
```

or

1	2	3
4	5	6
7	8	9

A way to work with such data is to use two-dimensional lists. For example:

data = [[1,2,3],[4,5,6],[7,8,9]]

To access a certain value in the two-dimensional list, we first access the corresponding element of the outer list and then the one of the inner one.
To set a variable *a* to the fourth element, we would do the following in the code:

a = data[1][0]

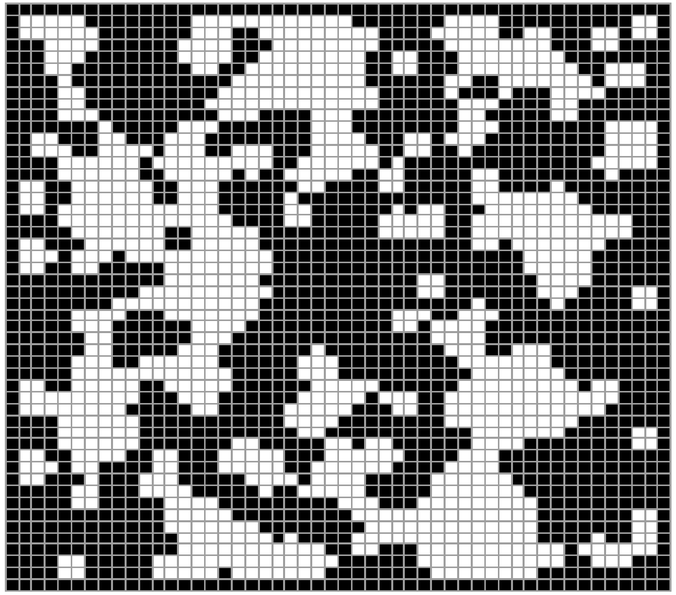
A way to work with such data is to use two-dimensional lists. For example:

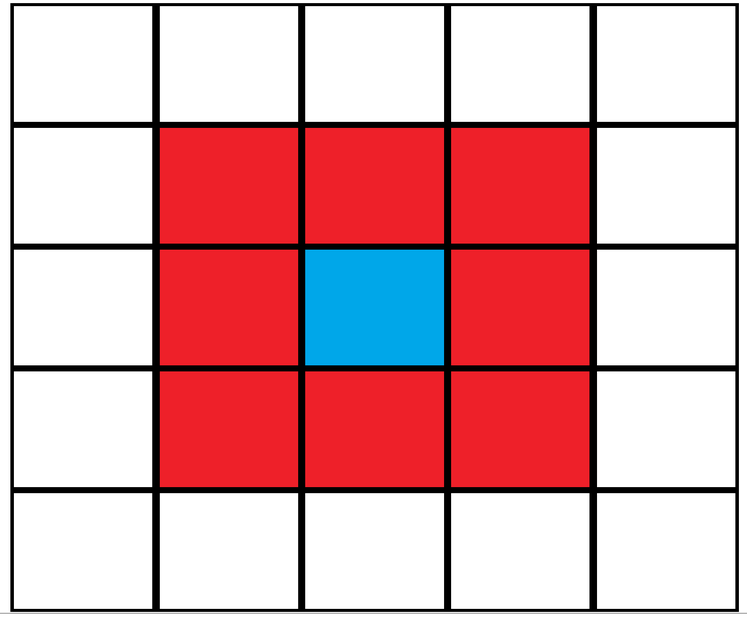
data = [[1,2,3],[4,5,6],[7,8,9]]

To set certain values in the *data* list, we would do it similar as accessing it, but put that element on the left-hand side of the assignment operator:

data[2][1] = 12

A cellular automaton consists of a regular grid of cells, each in one of a finite number of states, such as on and off. … For each cell, a set of cells called its neighborhood is defined relative to the specified cell. (wikipedia)

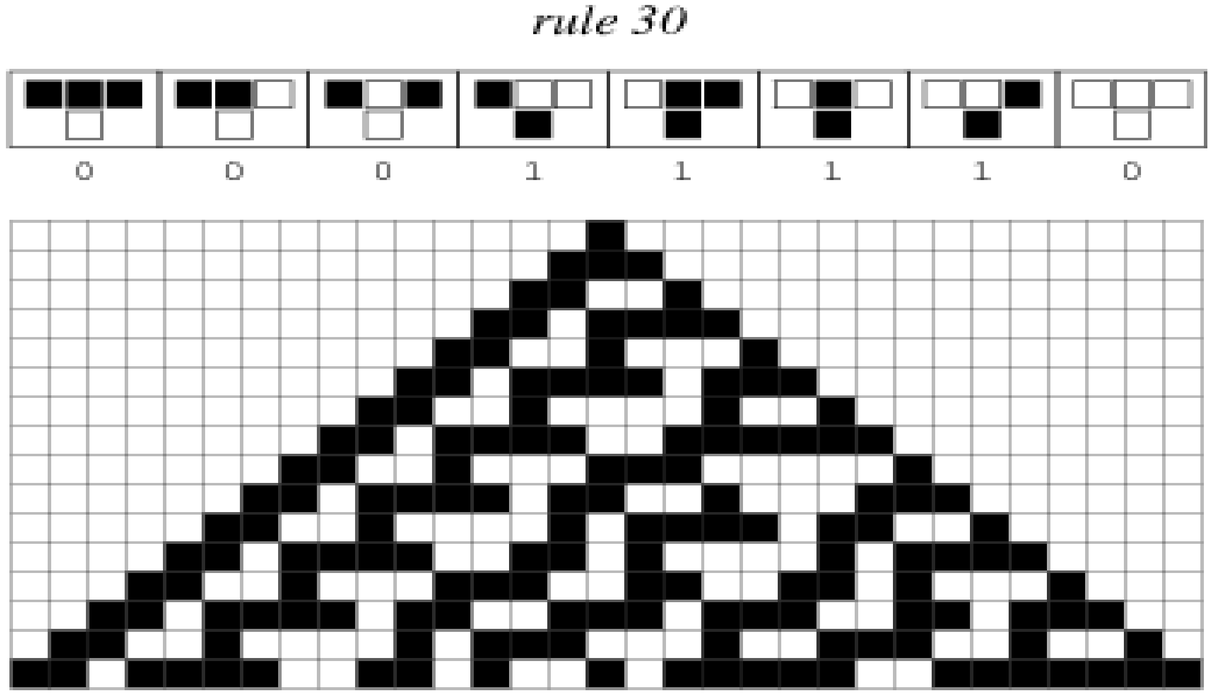




Cellular Automata

Introduction to Programming III

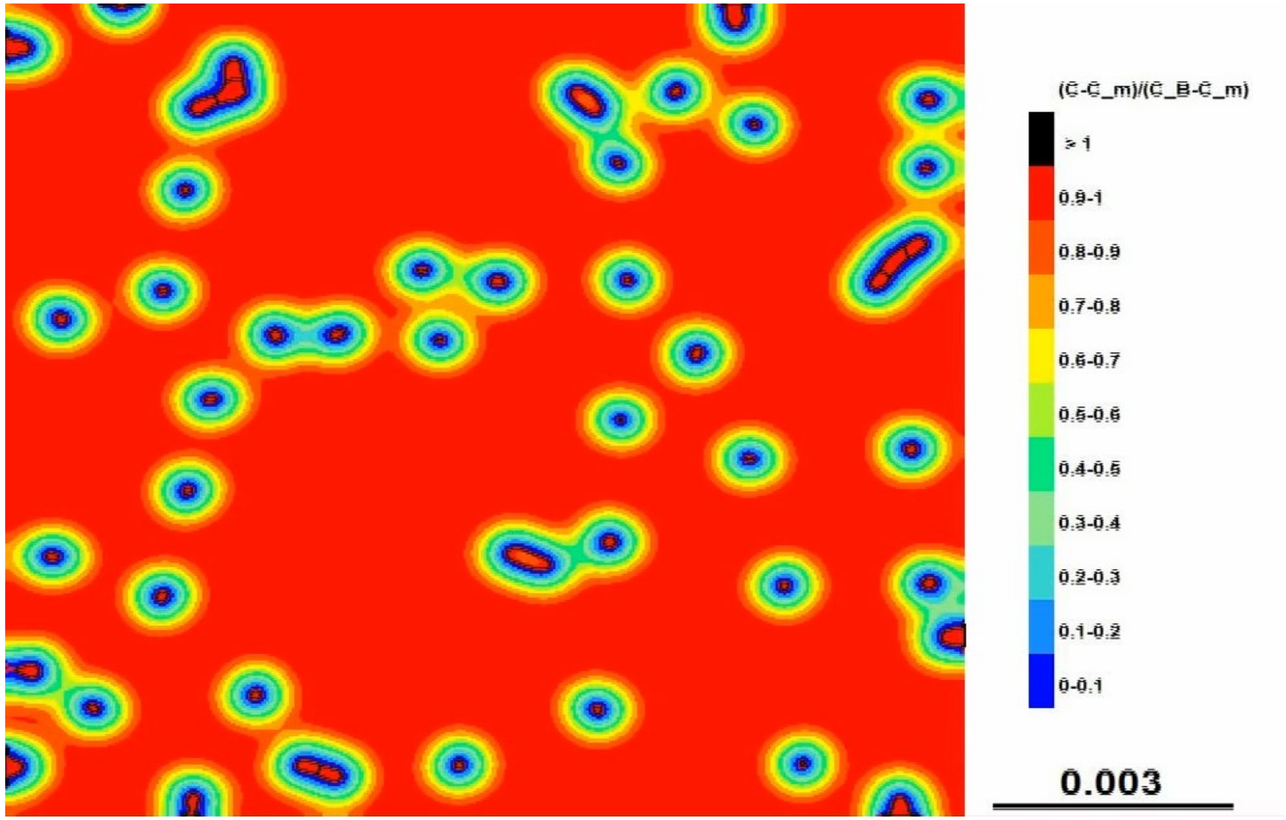
Cellular Automata have certain rules, how they change their status according to the current state and the state of the neighbors. In the one-dimensional case this could be:



Picture source: mathworld.wolfram.com

The simple rules of Cellular Automata can be extended and made more complicated, they can have bigger neighborhoods, states between zero and one, interact in more complicated ways, …

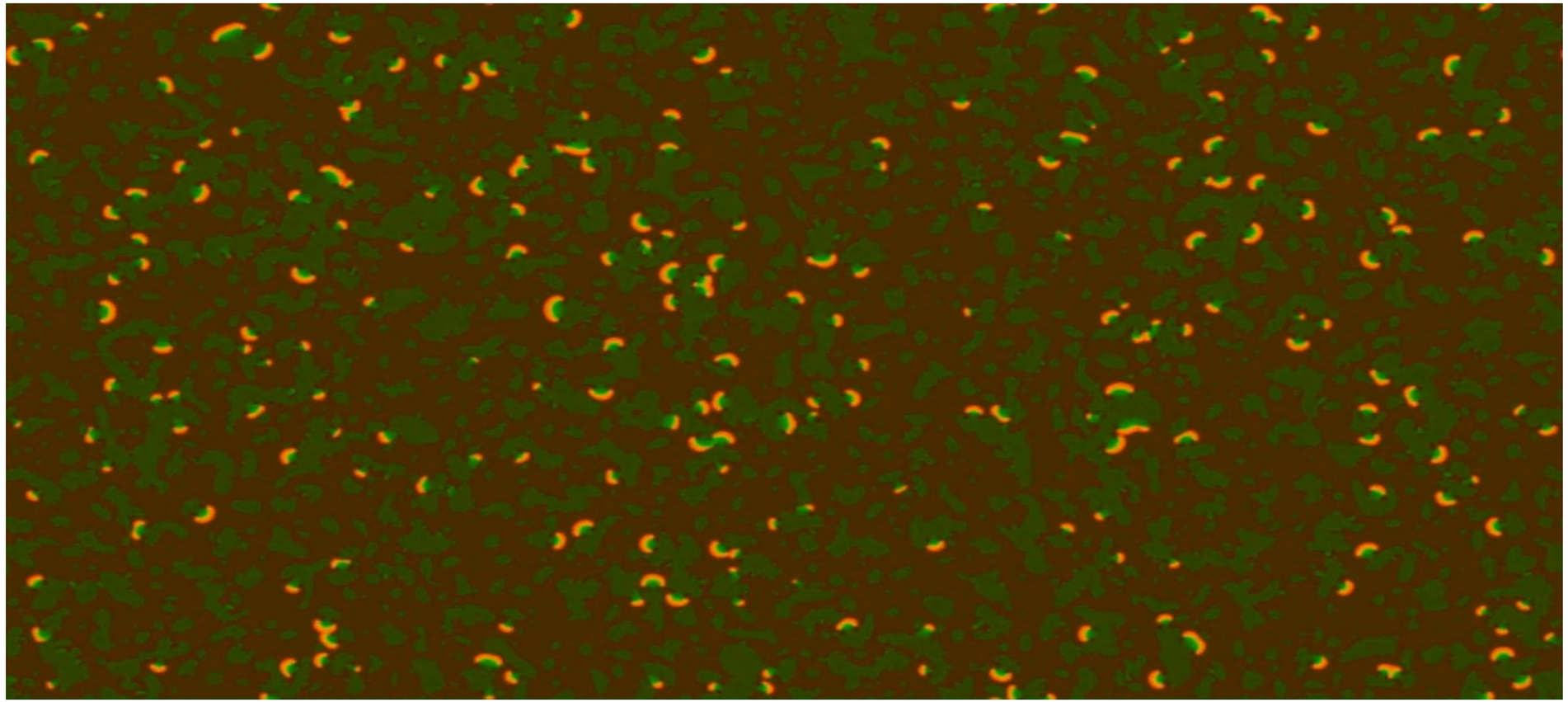
For example diffusion can be simulated, when we set the cells to contain some amount of a material and then all cell give 10% of their material to their neighbors.



Video source: youtube.com

Grid Based Interaction

Introduction to Programming III



Video source: youtube.com

Grid Based Interaction

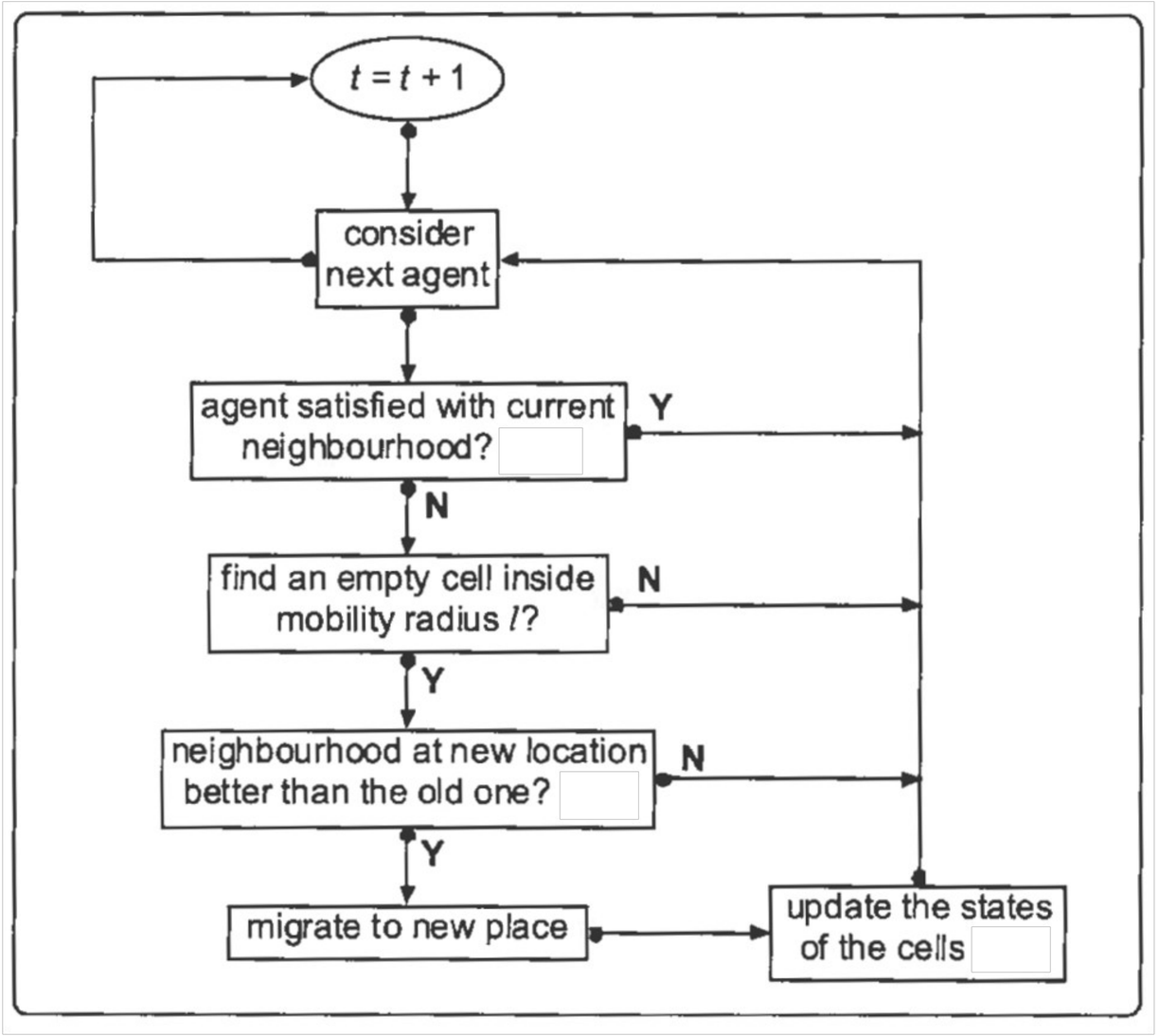
Introduction to Programming III

In todays exercise you will go through a visualization workflow.

1. We do a simulation of a behavior we are interested in.
2. We export the results to a file.
3. We load the data in a visualization software and animate it.

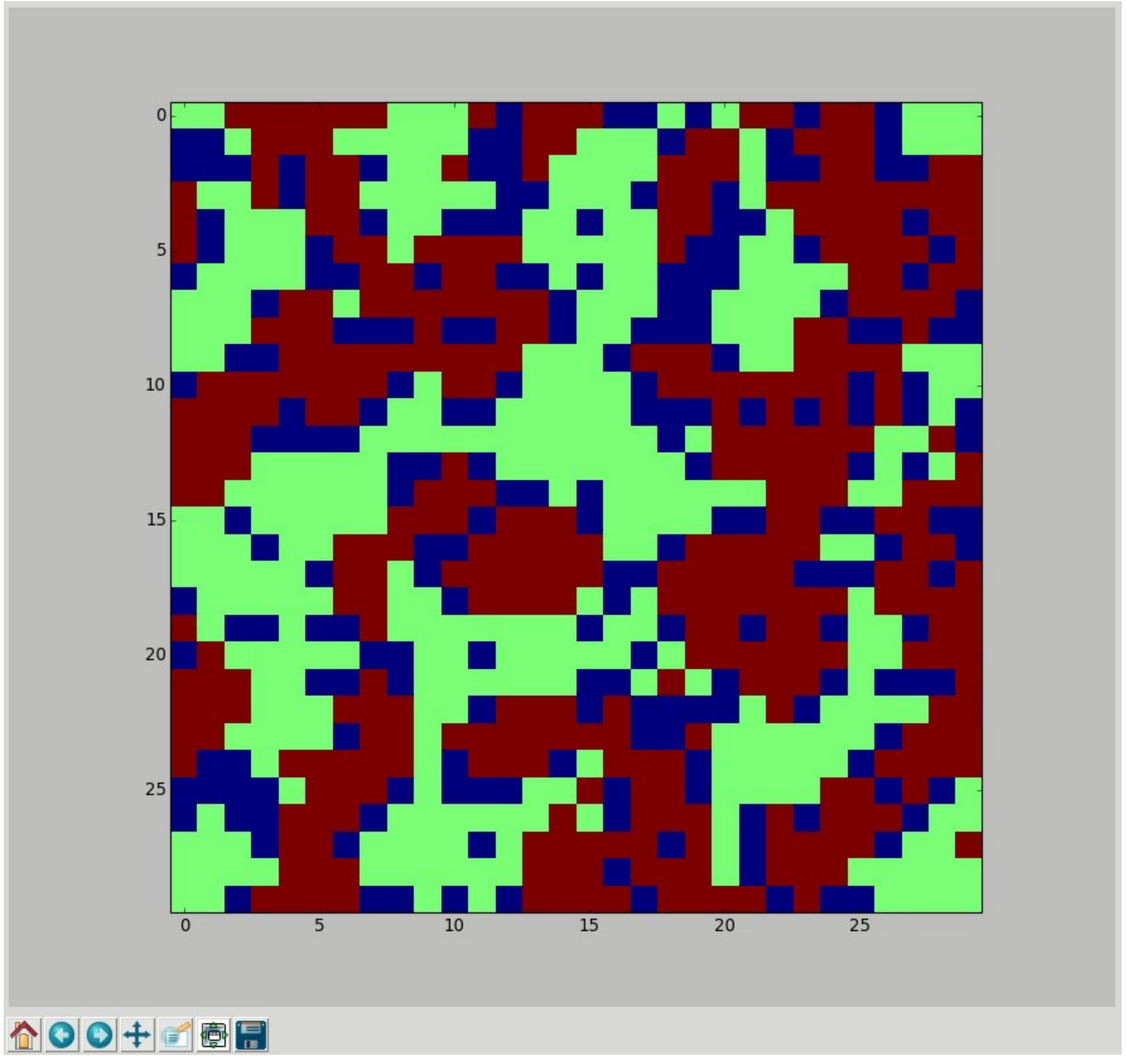
We will do a simulation of segregation dynamics.

- There are two types of agents in our simulation.
- Both types prefer to live within a neighborhood of their own type.
- If they are not happy within their current neighborhood, they look for a better location and when they find it, they move there.
- In our current version of the model, we have two agent types, indicated by numbers 1 and 2 in the *field* list. A zero indicates that the current field is not occupied.



Exercise

Introduction to Programming III



Exercise

Introduction to Programming III

First you will need to implement the function, which calculates for a given agent the number of neighboors of the same type as himself.

Maybe helpful: modulo works also for negative numbers, e.g. *-1%10 == 9*

Second you will need to implement the function which saves the current state of the simulation into a CSV file.

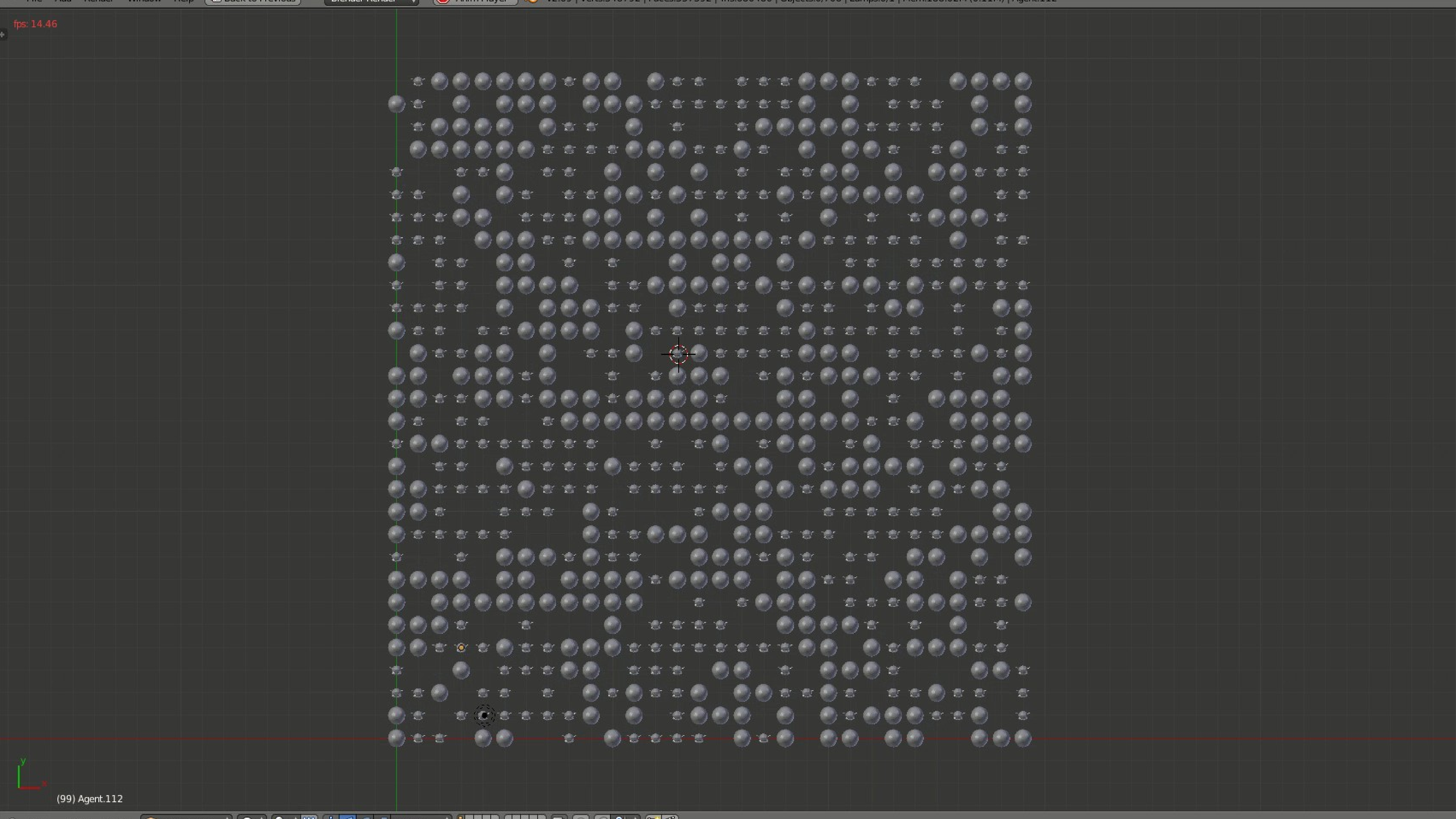
Hint: Loop through all the elements and call the *writer.writerow* function once for every field.

Careful: Do not save fields with a value not equal to 1 or 2.

You will open Blender and Produce you first visualization from Data and produced with only programming!

Exercise: 2nd Task

Introduction to Programming III


fps: 14.46
(99) Agent.112

- *http://www.tutorialspoint.com/python*
- *http://docs.python.org/3/*
- *http://greenteapress.com/thinkpython/html/index.html*
- *And many more...*

Helpful Links