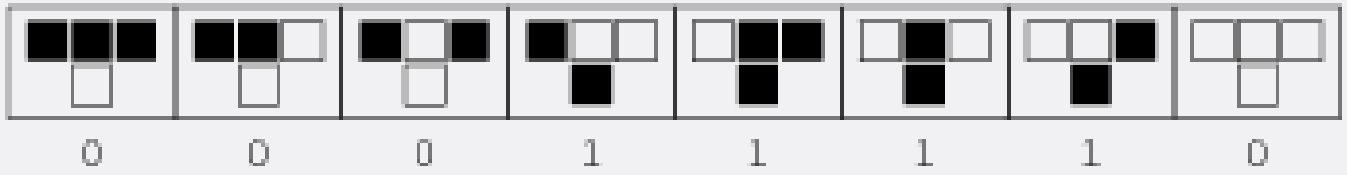
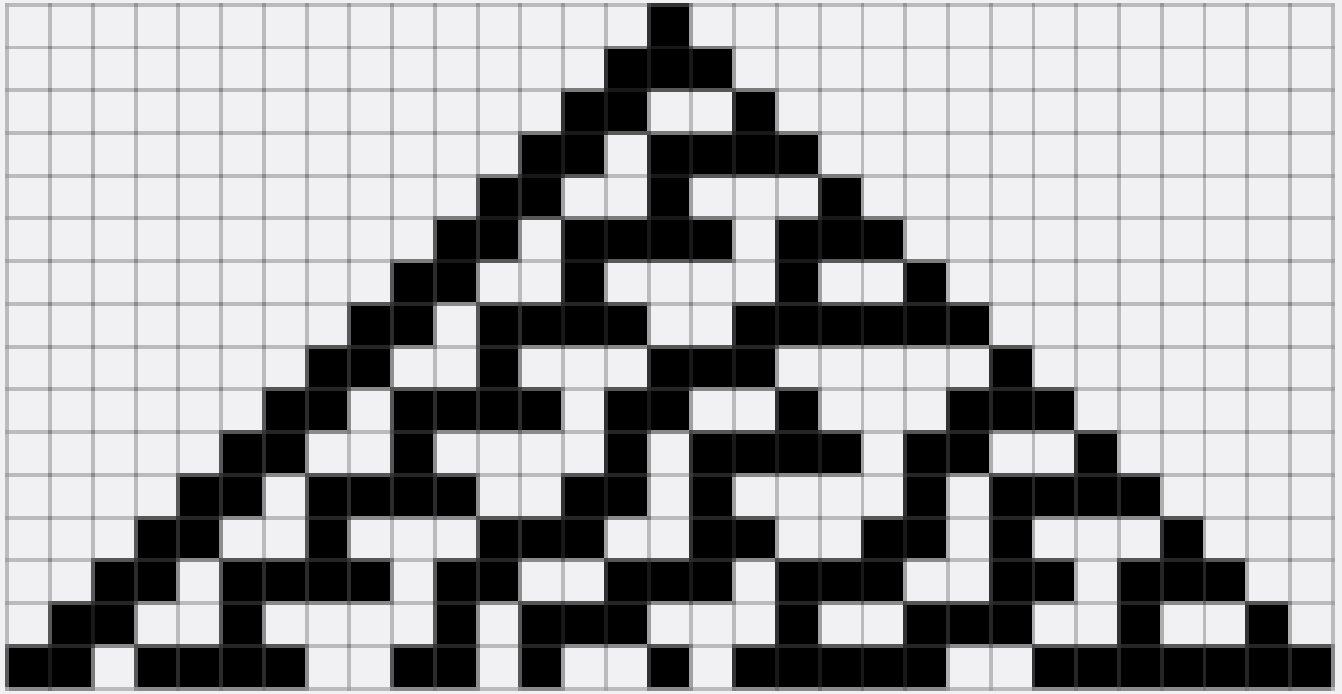
Présentation projet QGoL

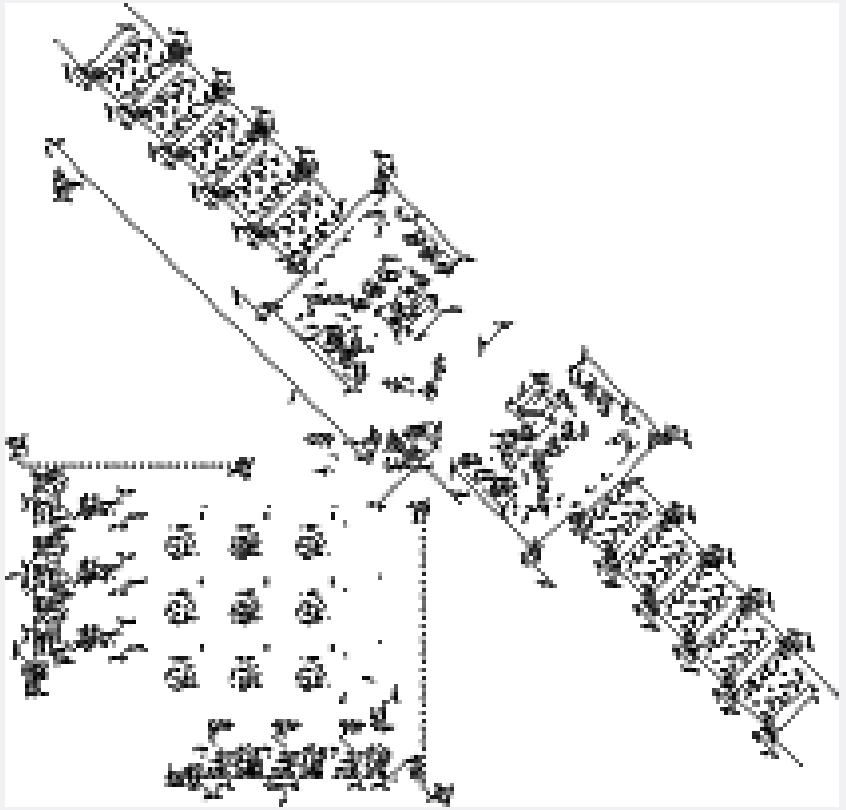
Automates cellulaires

rule 30





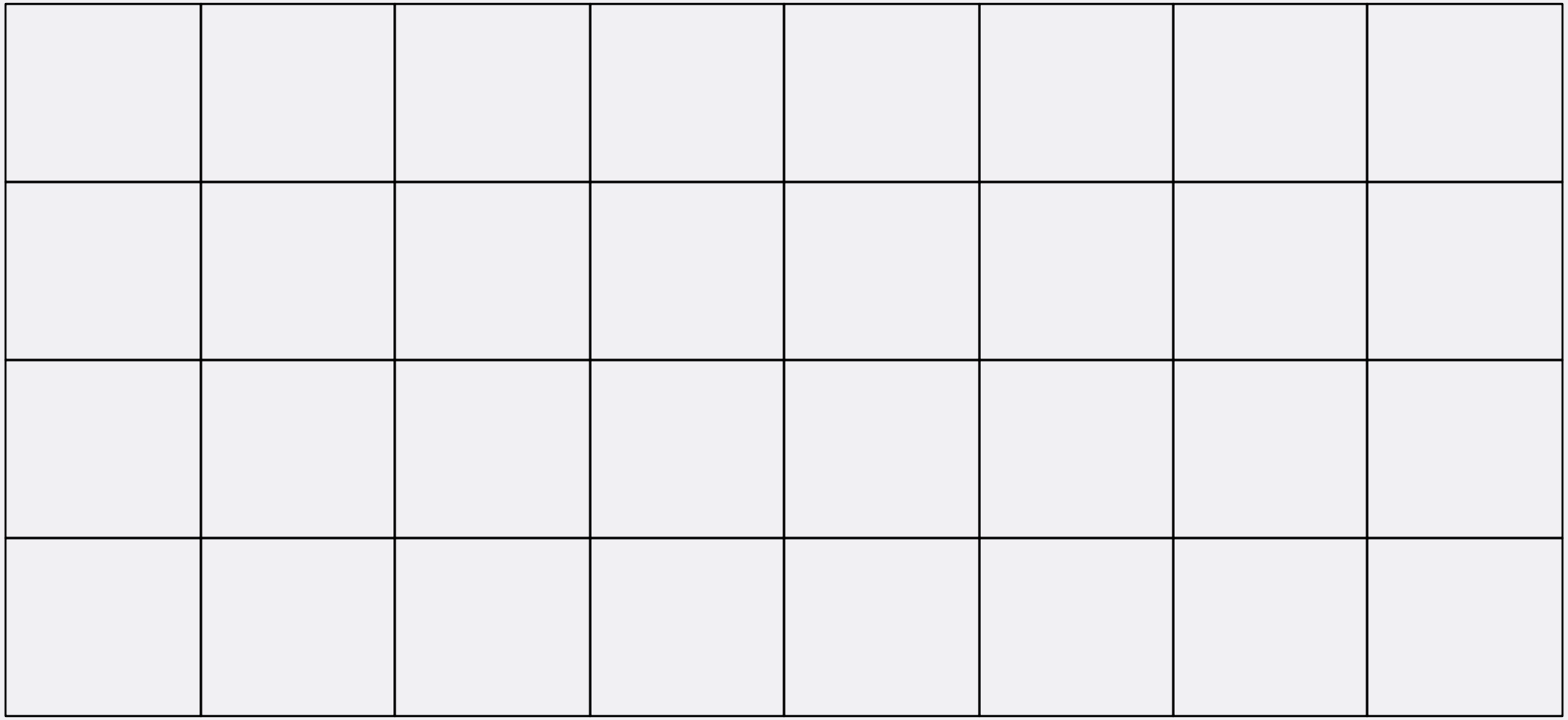
Le Game of Life de Conway



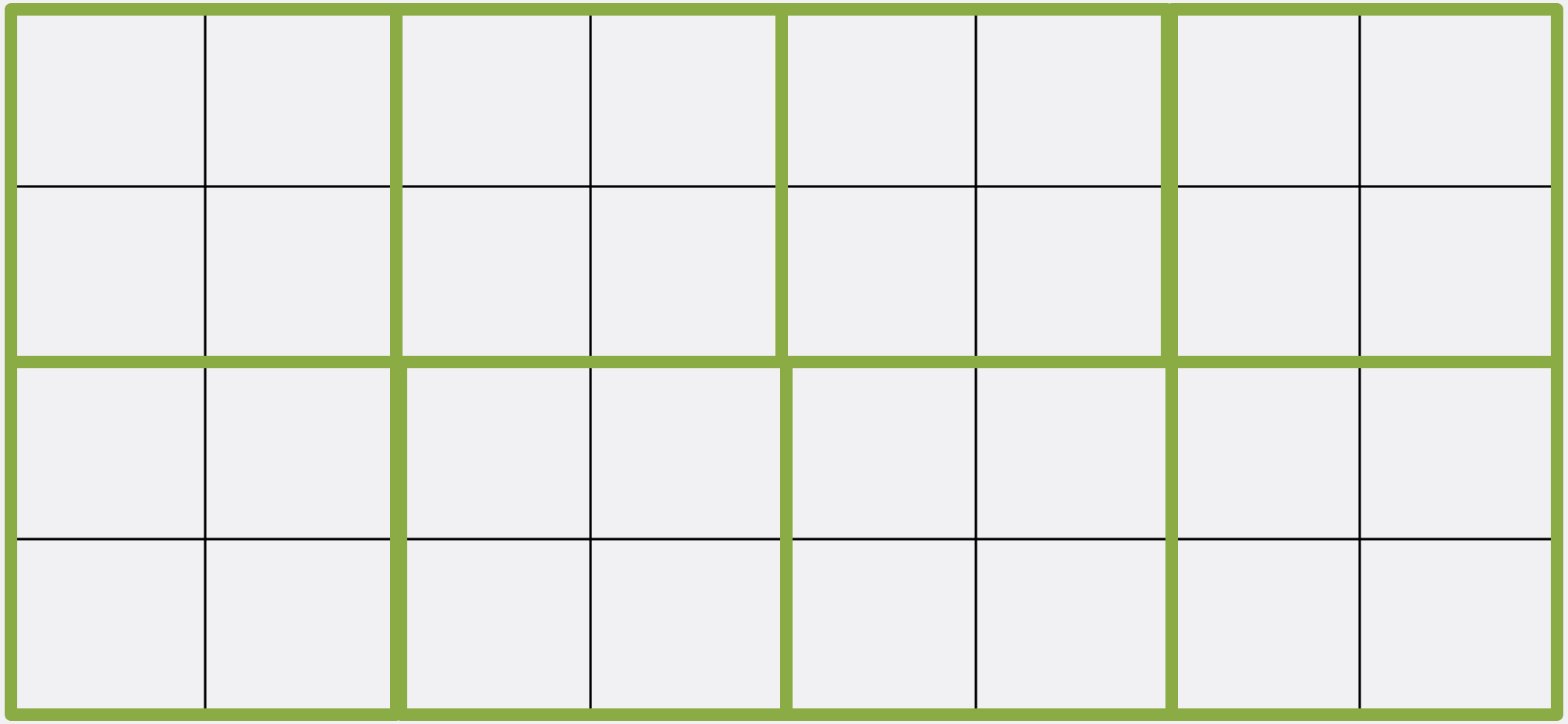
Un QGoL par Arrighi et Grattage

https://arxiv.org/abs/1010.3120

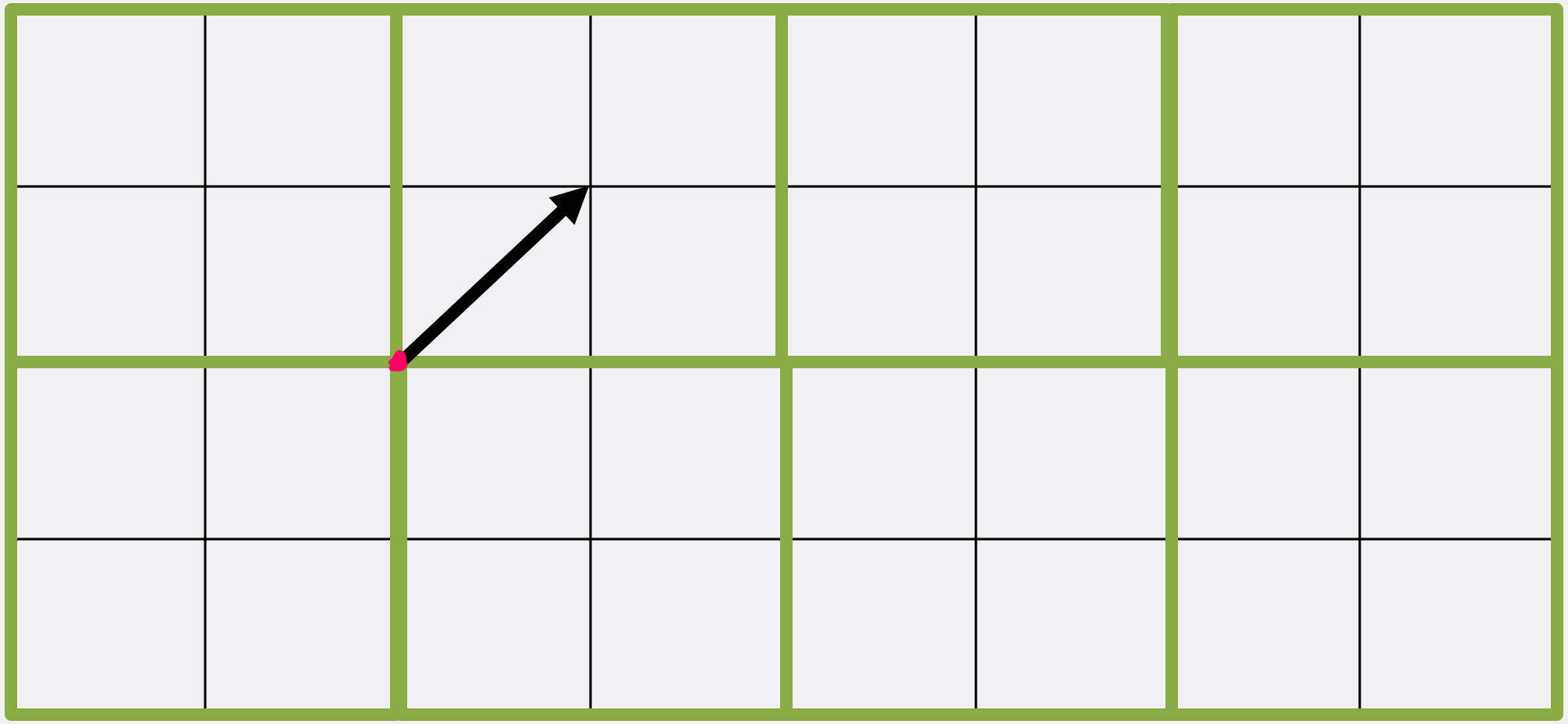
Principe de l’automate



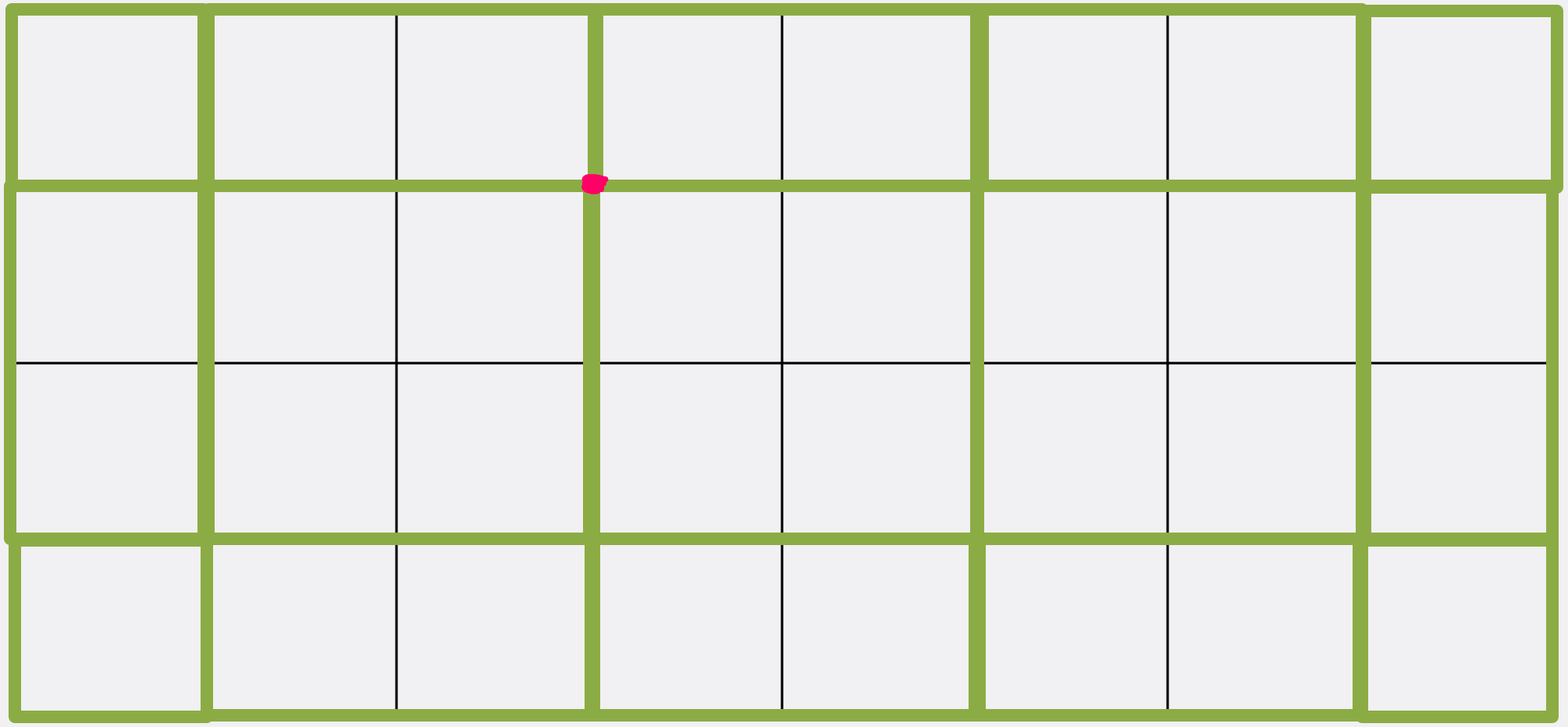
Principe de l’automate



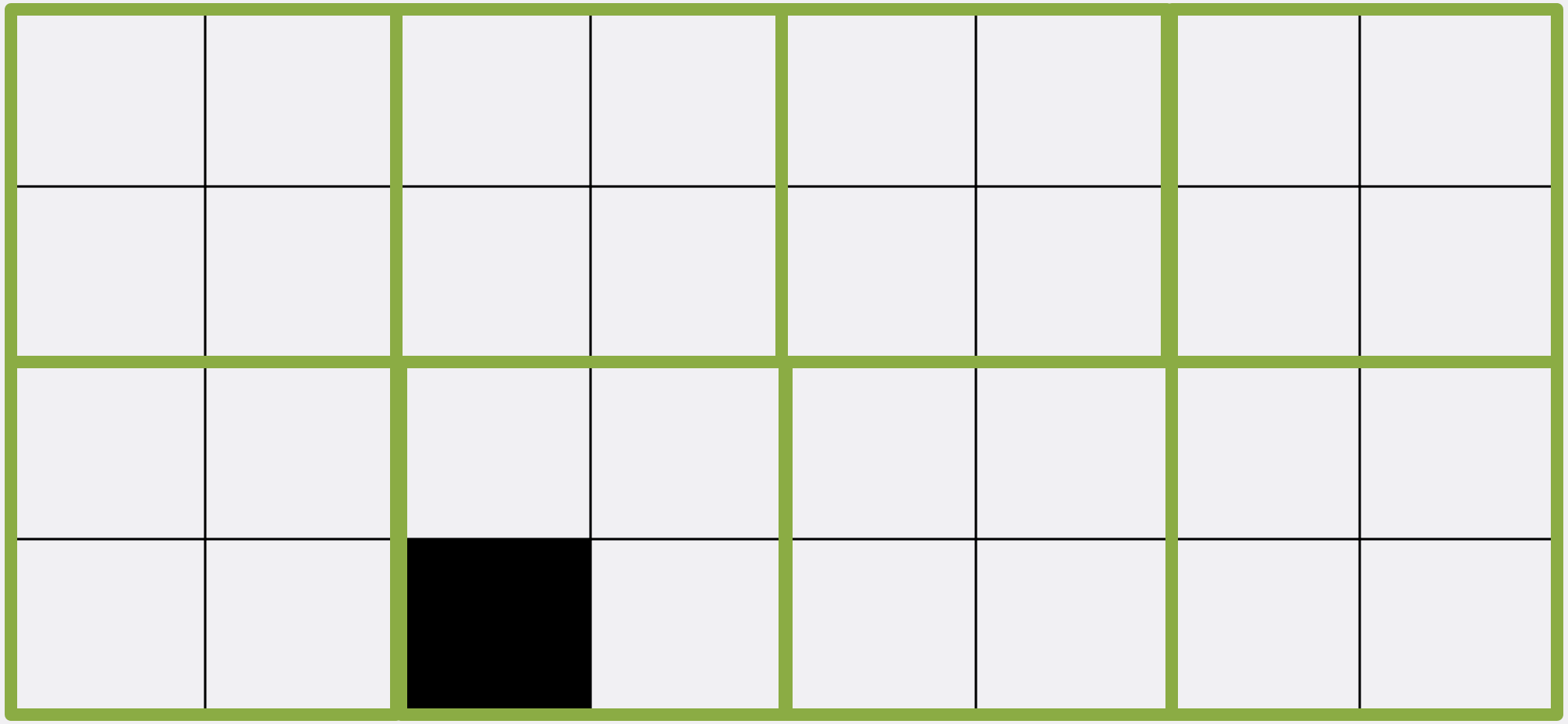
Principe de l’automate



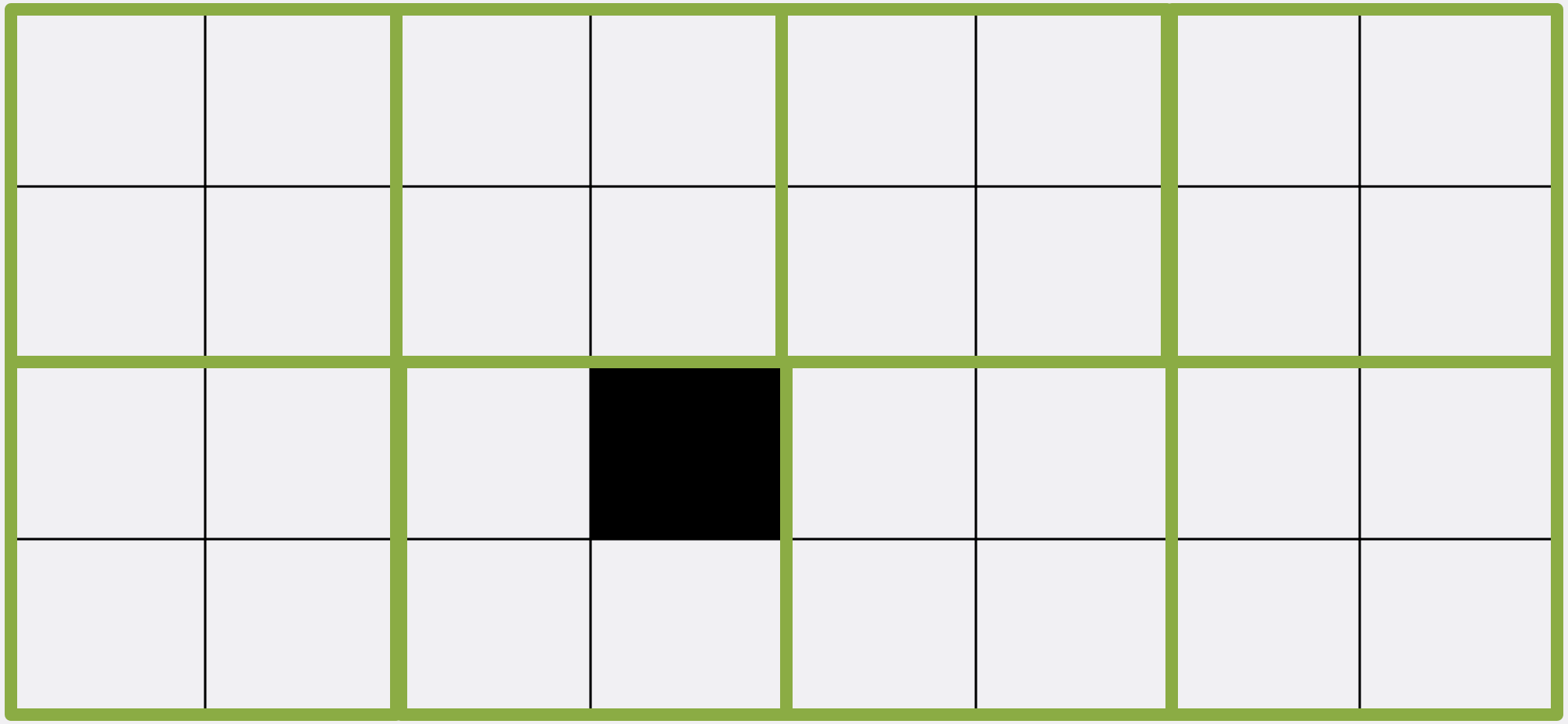
Principe de l’automate



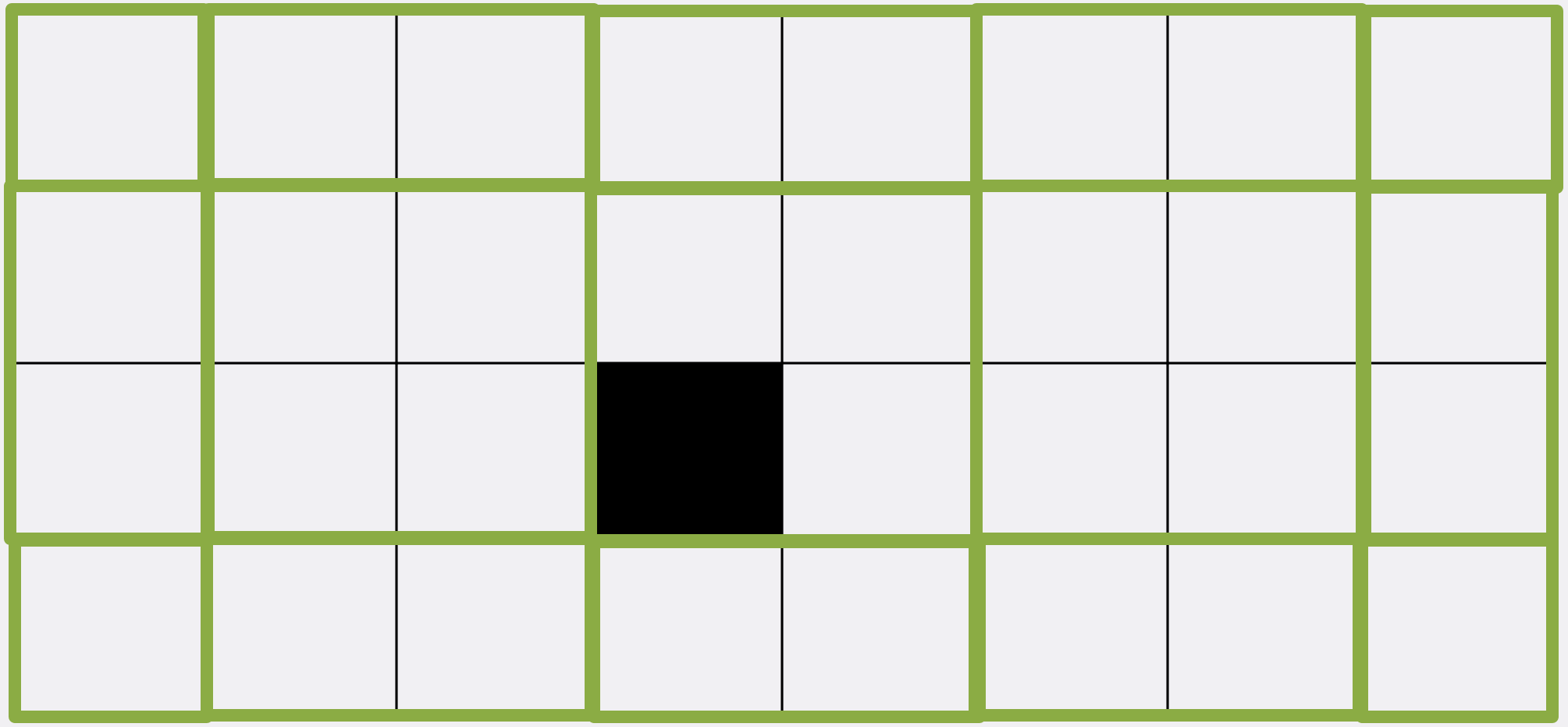
Principe de l’automate



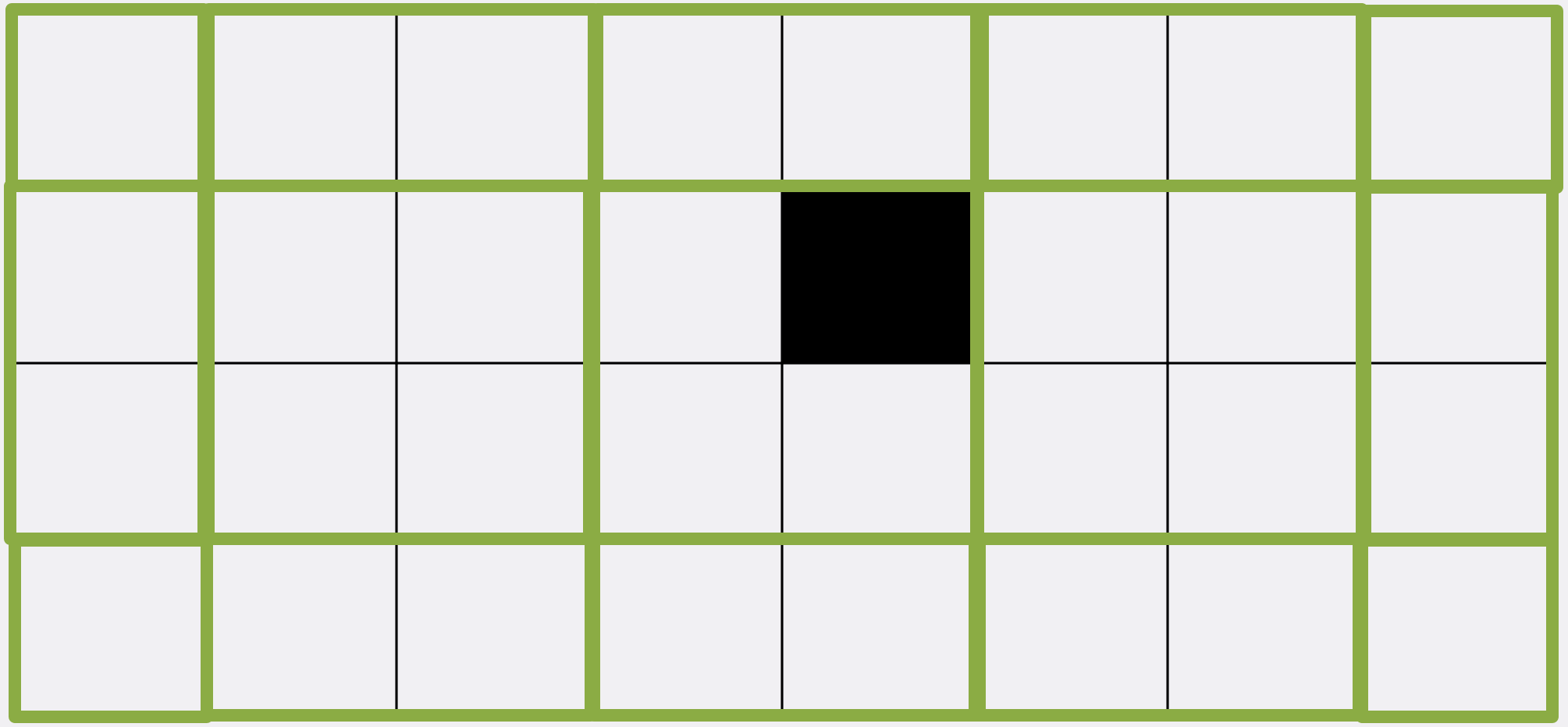
Principe de l’automate



Principe de l’automate

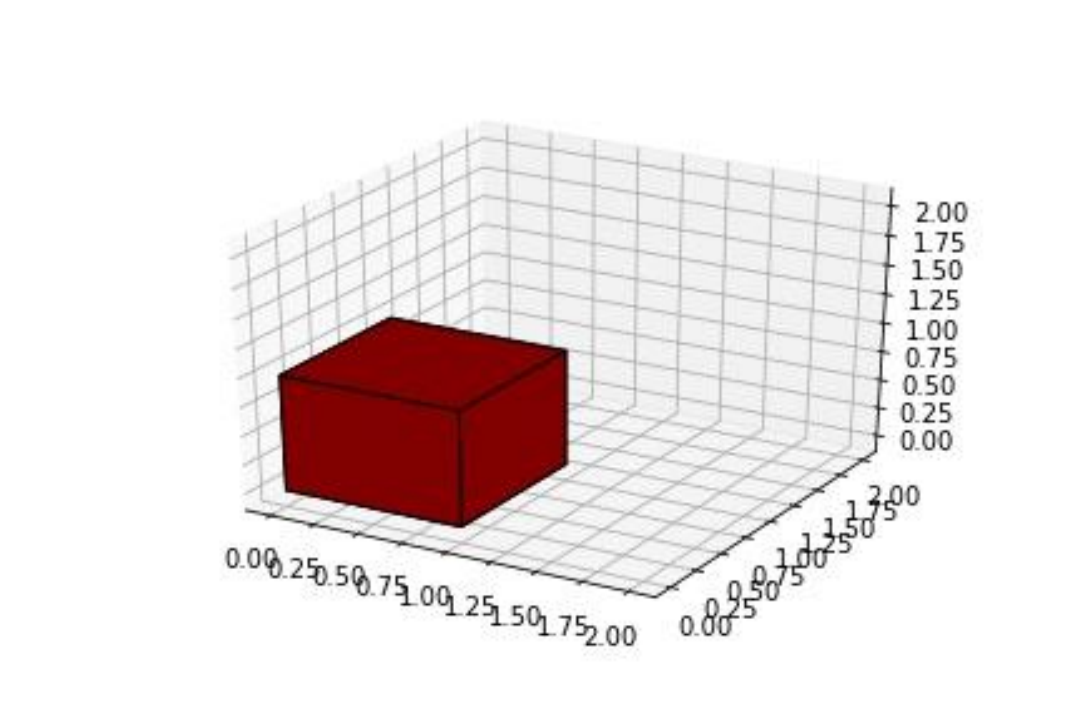


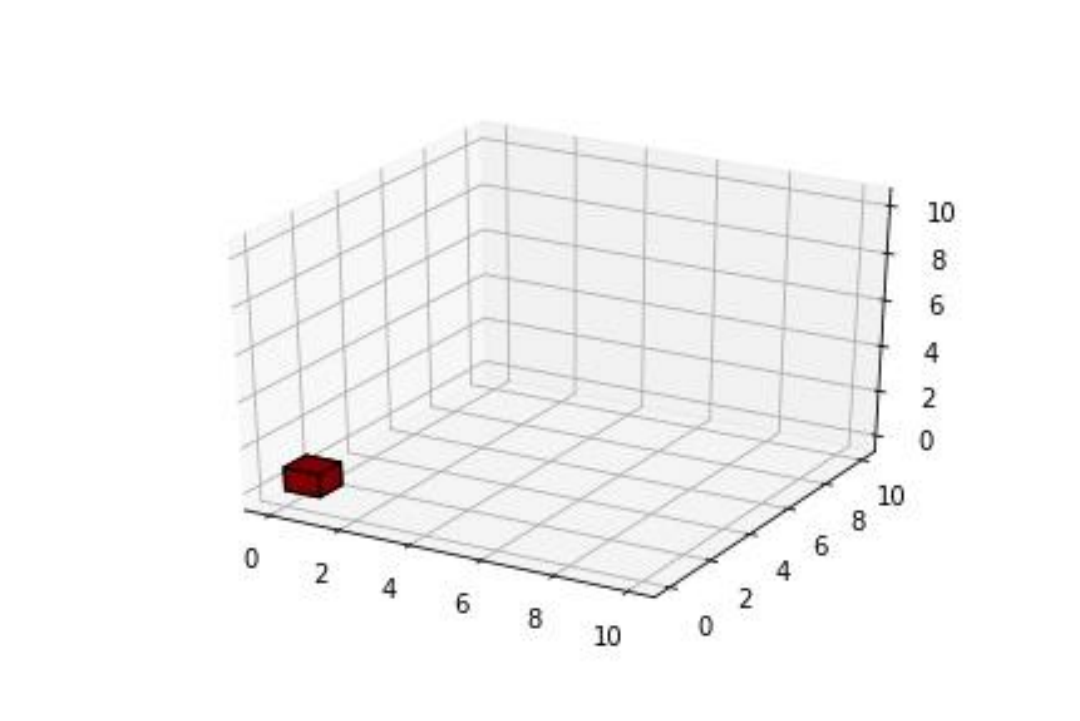
Principe de l’automate

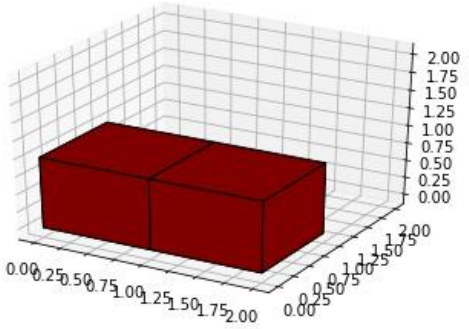


Matplotlib
Toolkit

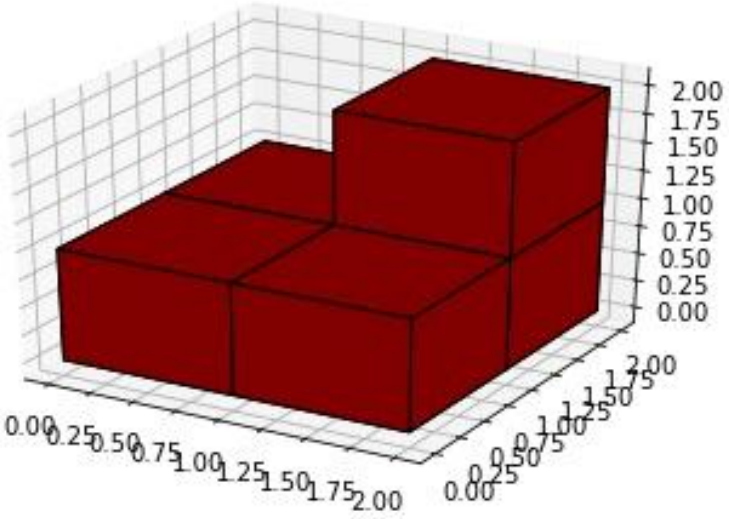
Mouvement du signal

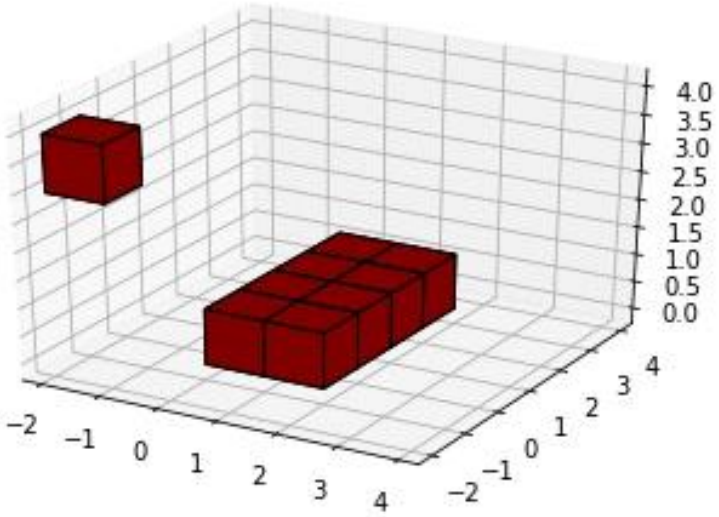




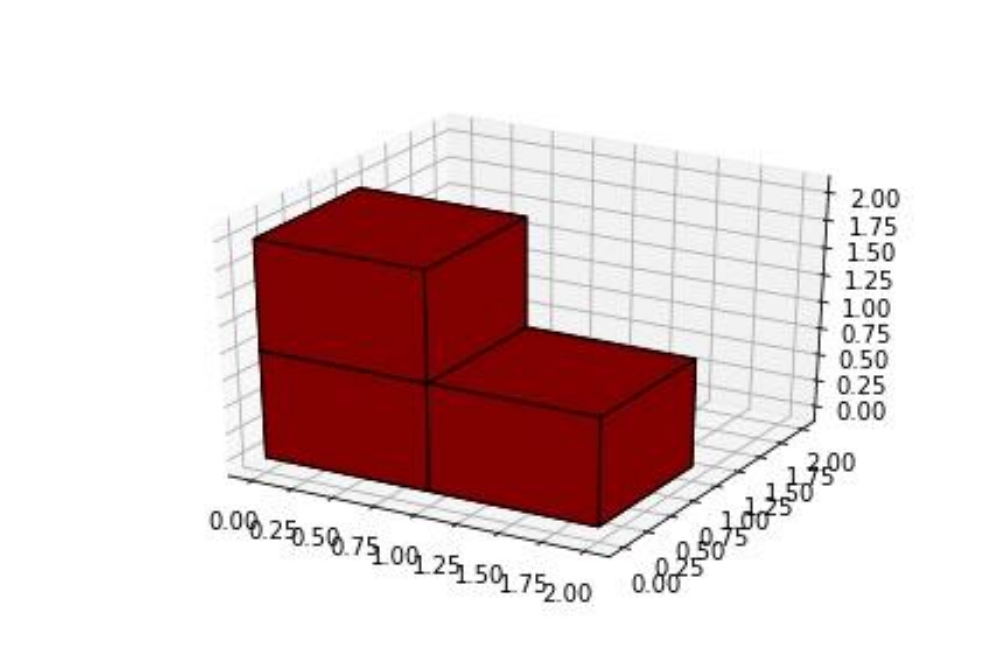


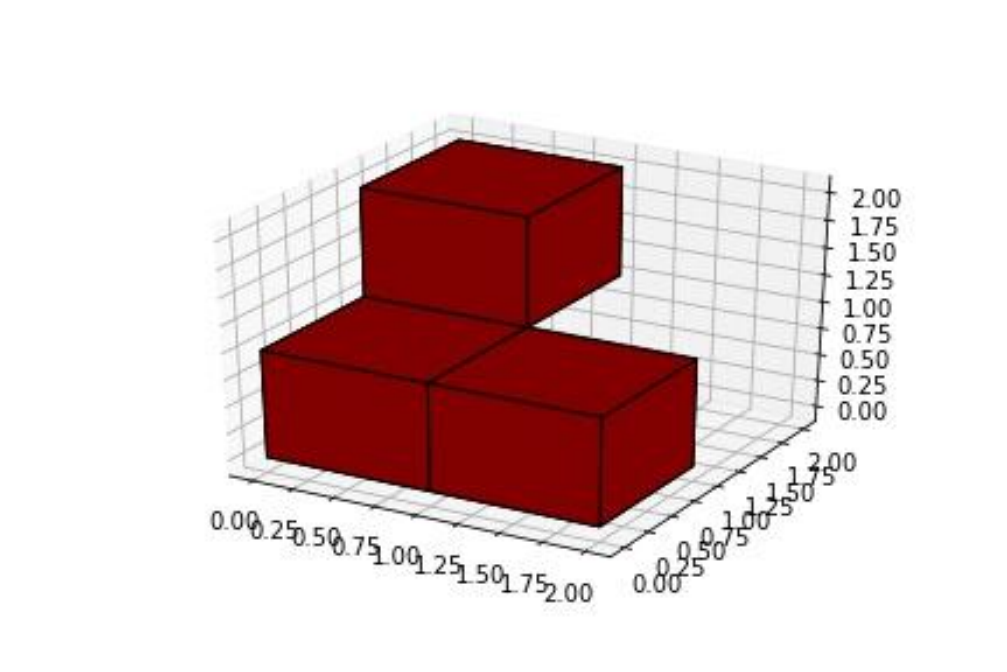
La barrière et les murs





La porte Hadamard





Code

- Programmation Orientée Objet

```python
""" Defines the Cube class """
from obj.base import Position, Cell
from obj import partition
from log.log import debg

class Cube:

    def __init__(self,op=None,celltype=Cell):
        """ A cube of eight cells.
        Cells are of the provided celltype """
        if celltype is None:
            self.cube = None
        else:
            self.cube = [[[celltype() for _ in range(2)] for _ in range(2)] for _ in range(2)]
        self.op = op
        self.p = None # position (unused)

    def activation(self,*args):
        """ Activates the specified Cell. """
        if len(args) == 1:
            if type(args) == type(Cell()):
                for c in self.flatcube():
                    if c == args[0]:
                        c.activate()
        elif len(args) >= 3:
            self.cube[args[0]][args[1]][args[2]].activate()
        elif len(args) == 2:
            raise NotImplementedError
        else:
            print("WARNING : You should use the .activate method instead.")
            self.activate()

    def reverse(self):
        """ Reverses the cube. Assumes that the celltype is Cell. """
        nc = [[[Cell() for _ in range(2)] for _ in range(2)] for _ in range(2)]
        for i in range(len(self.cube)):
            for j in range(len(self.cube[i])):
                for k in range(len(self.cube[i][j])) :
                    nc[1-i][1-j][1-k].v = self[i,j,k].v
        for i in range(len(self.cube)):
            for j in range(len(self.cube[i])):
                for k in range(len(self.cube[i][j])) :
                    self.cube[i][j][k].v = nc[i][j][k].v

    def wall(self):
```

Index

Sub-modules

Package **obj**

▶ EXPAND SOURCE CODE

Sub-modules

obj.base

obj.config

Defines the Config class

obj.cube

Defines the Cube class

obj.cubes

Defines the Cubes class. This class handles several cubes.

obj.partition

Defines the Partition class

obj.qcubes

Defines the SCubes class. This class handles several QCubes.

obj.super

Defines the Super class

obj.unit

Defines a usable Unitary class

Index

Super-module

obj

Classes

Cube

- activate
- activation
- adj
- bump
- cclass
- copy
- cross
- f
- flatcube
- from_pos
- len
- positions
- reverse
- reversed
- wall
- walled

Module **obj.cube**

Defines the Cube class

▸ EXPAND SOURCE CODE

Classes

```
class Cube (op=None, celltype=obj.base.cell.Cell)
```

A cube of eight cells. Cells are of the provided celltype

▸ EXPAND SOURCE CODE

Methods

```
def activate(self)
```

activates all the Cells of a given Cube

▸ EXPAND SOURCE CODE

```
def activation(self, *args)
```

Activates the specified Cell.

▸ EXPAND SOURCE CODE

```
def adj(self, i, j, k)
```

Adjacent cells of a given cell.

▸ EXPAND SOURCE CODE

```
def bump(self)
```

Number of crossing particles

▸ EXPAND SOURCE CODE

```
def cclass(self)
```

Index

Super-module

obj

Classes

Cube

activate
activation
adj
bump
cclass
copy
cross
f
flatcube
from_pos
len
positions
reverse
reversed
wall
walled

Module **obj.cube**

Defines the Cube class

▶ EXPAND SOURCE CODE

Classes

```
class Cube (op=None, celltype=obj.base.cell.Cell)
```

A cube of eight cells. Cells are of the provided celltype

▶ EXPAND SOURCE CODE

Methods

```
def activate(self)
```

activates all the Cells of a given Cube

▼ EXPAND SOURCE CODE

```
def activate(self):
        """ activates all the Cells of a given Cube """
        for c in self.flatcube():
                c.activate()
```

```
def activation(self, *args)
```

Activates the specified Cell.

▶ EXPAND SOURCE CODE

```
def adj(self, i, j, k)
```

Adjacent cells of a given cell.

▶ EXPAND SOURCE CODE

```
def bump(self)
```

Code

- Programmation Orientée Objet
- Gestion des versions avec GitHub

Code

- Programmation Orientée Objet
- Gestion des versions avec GitHub
- Tests automatisés avec Pytest

Objets

- QGOL

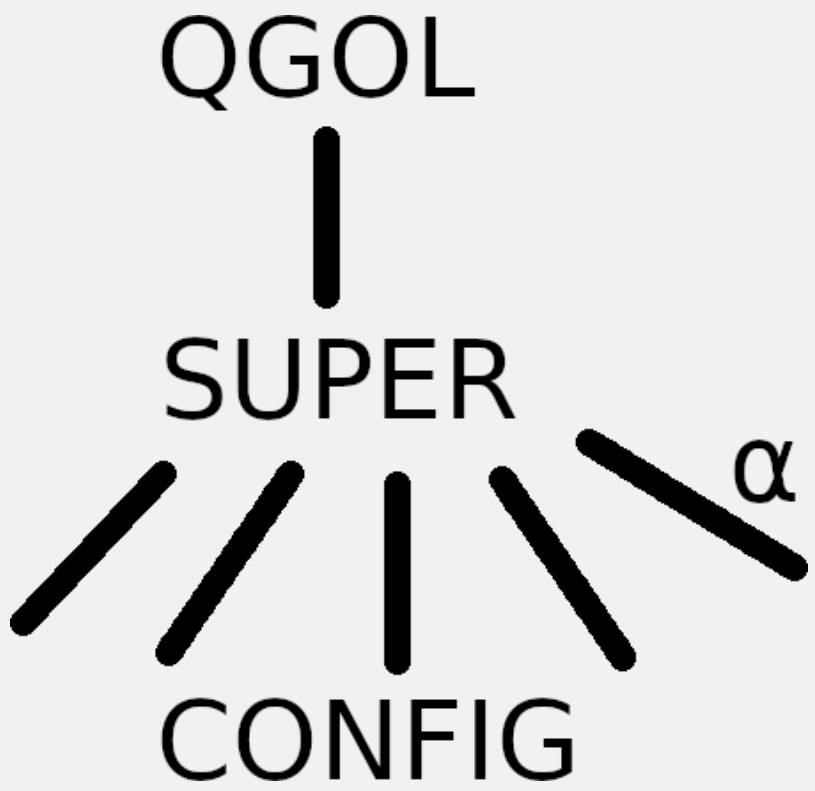
Objets

- QGOL

- Config

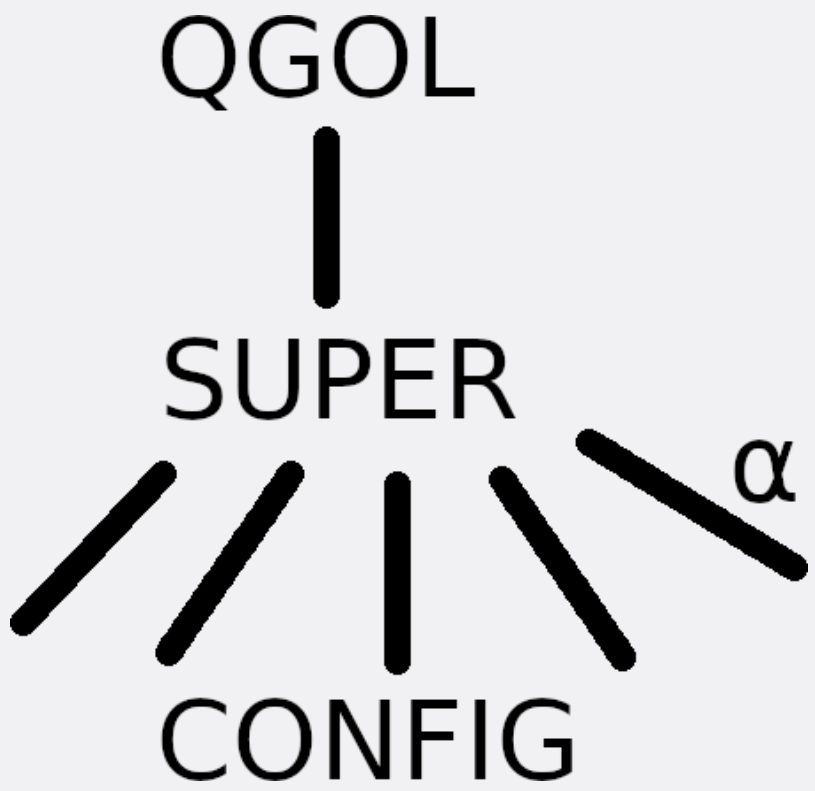
Objets

- QGOL
- Super
- Config



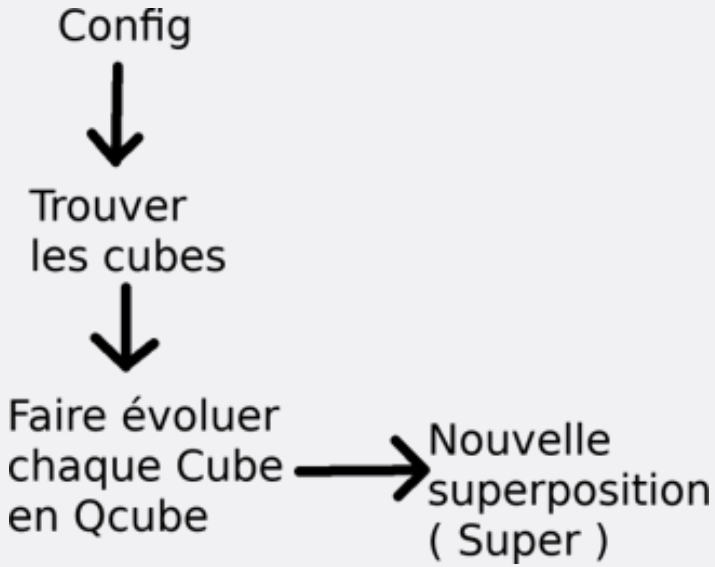
Objets

- QGOL
- Super
- Config
- Cube
- QCube



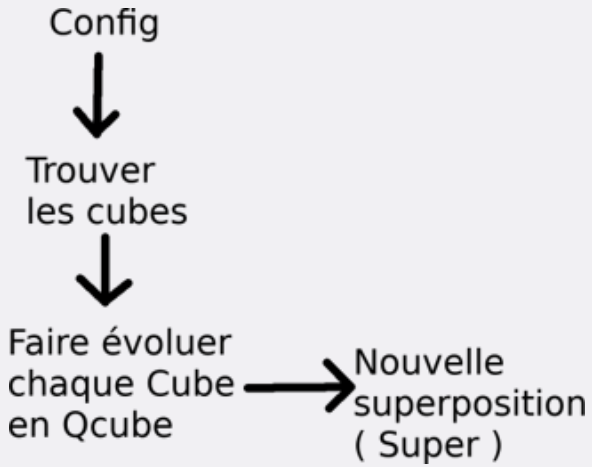
Evolution

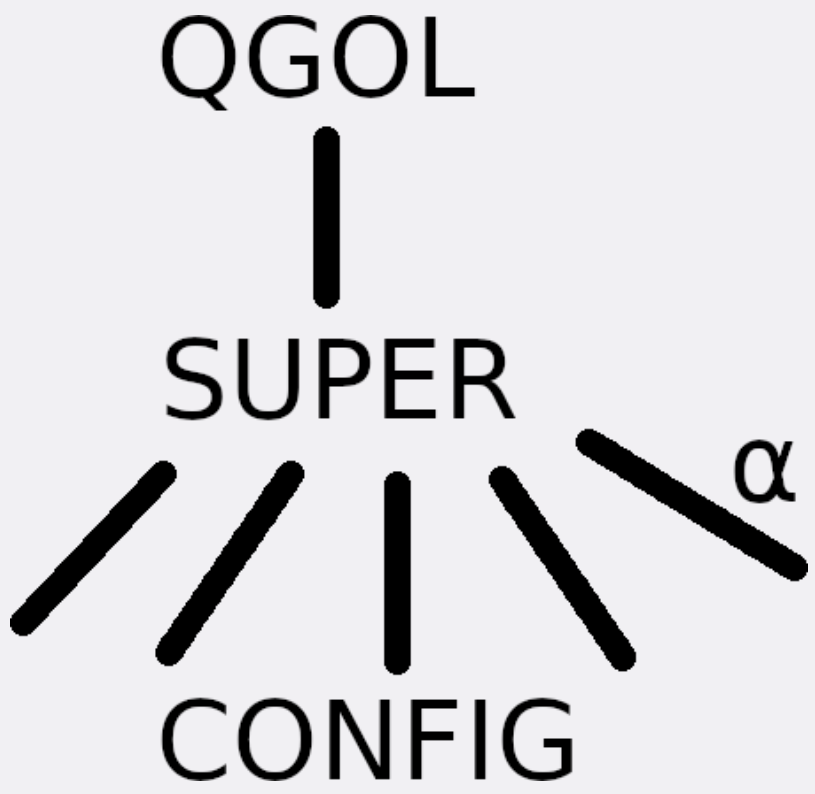
- QGOL
- Super
- Config
- Cube
- QCube



Implémentation

- QGOL
- Super
- Config
- Cube
- QCube





Structure *defaultdict*

dict[clé] = valeur (accès très rapide)

Structure *defaultdict*

dict[clé] = valeur

dict[Config] = amplitude (assignation)

Structure *defaultdict*

dict[clé] = valeur

dict[Config] = amplitude

dict[Config] += amplitude (interférences)

Structure *defaultdict*

Aussi utilisée pour gérer les Qcubes.

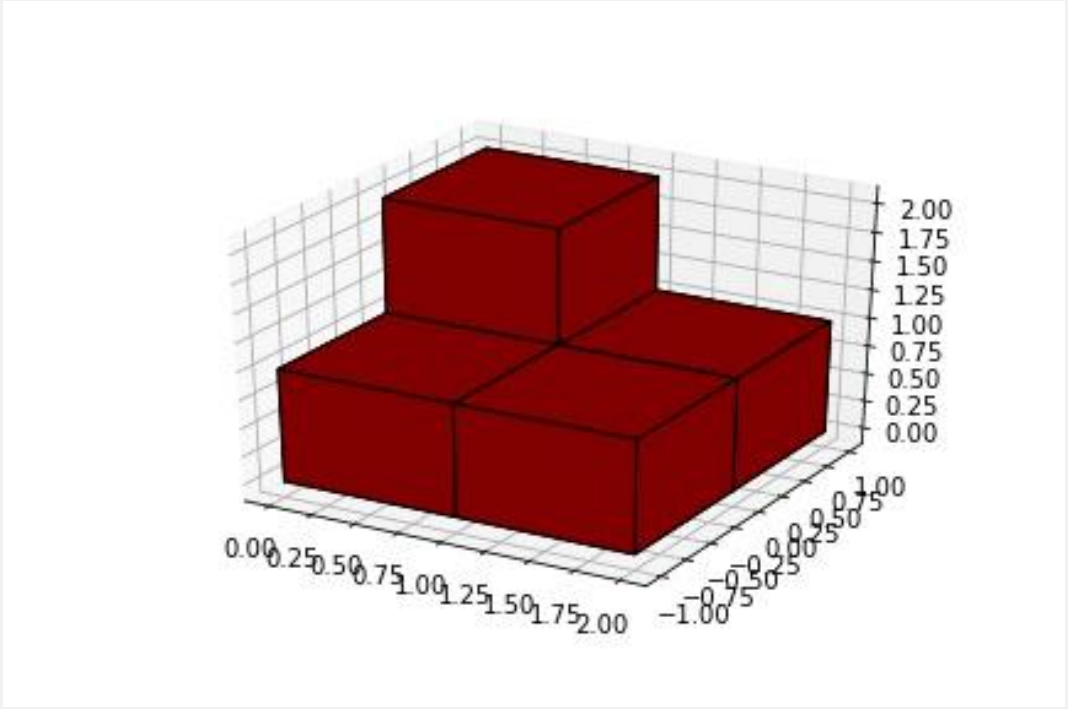
Universalité

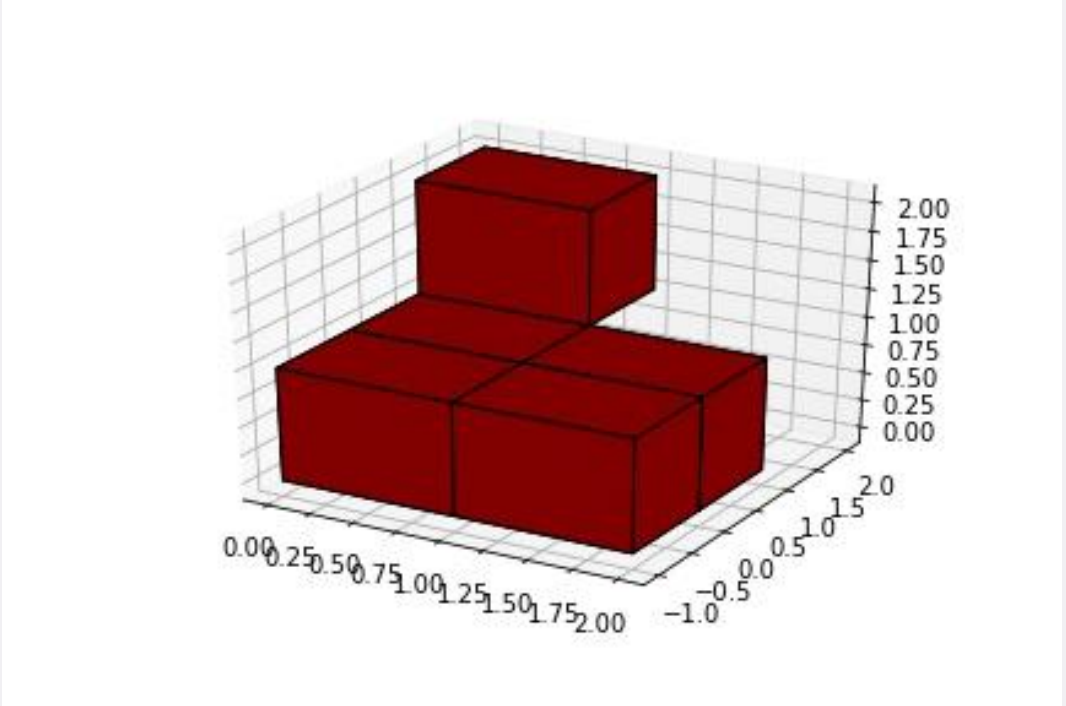
- Possibilité de générer des portes quelconques
- exemple de la porte hadamard :

Structure *defaultdict*

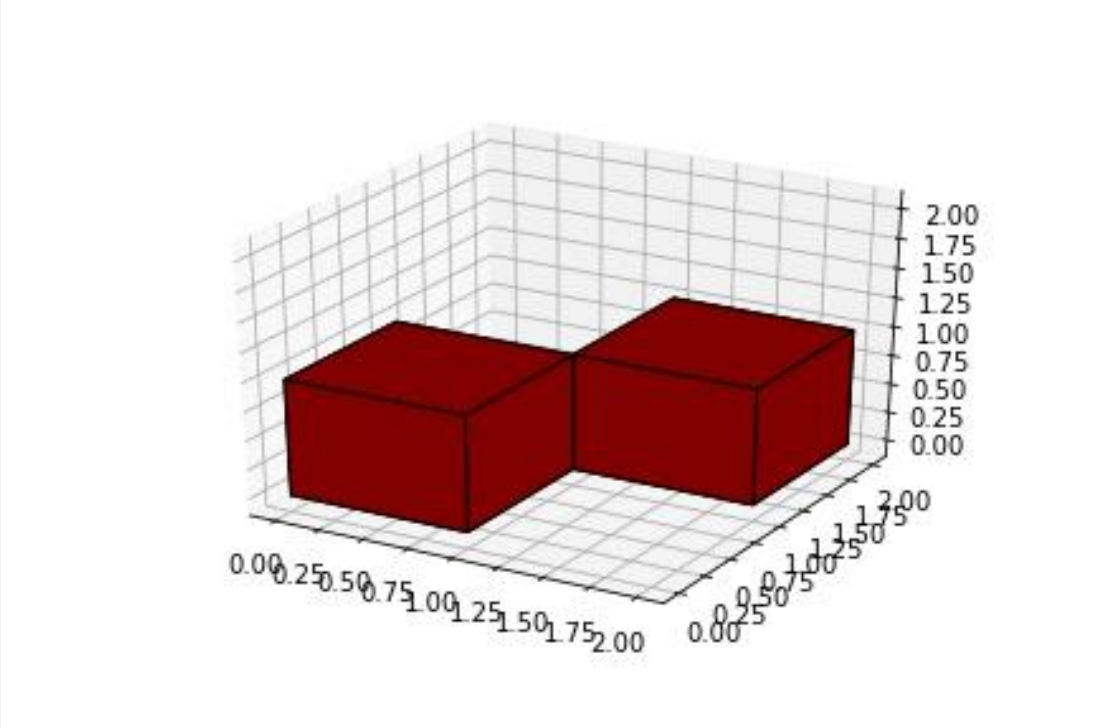
Aussi utilisée pour gérer les Qcubes.

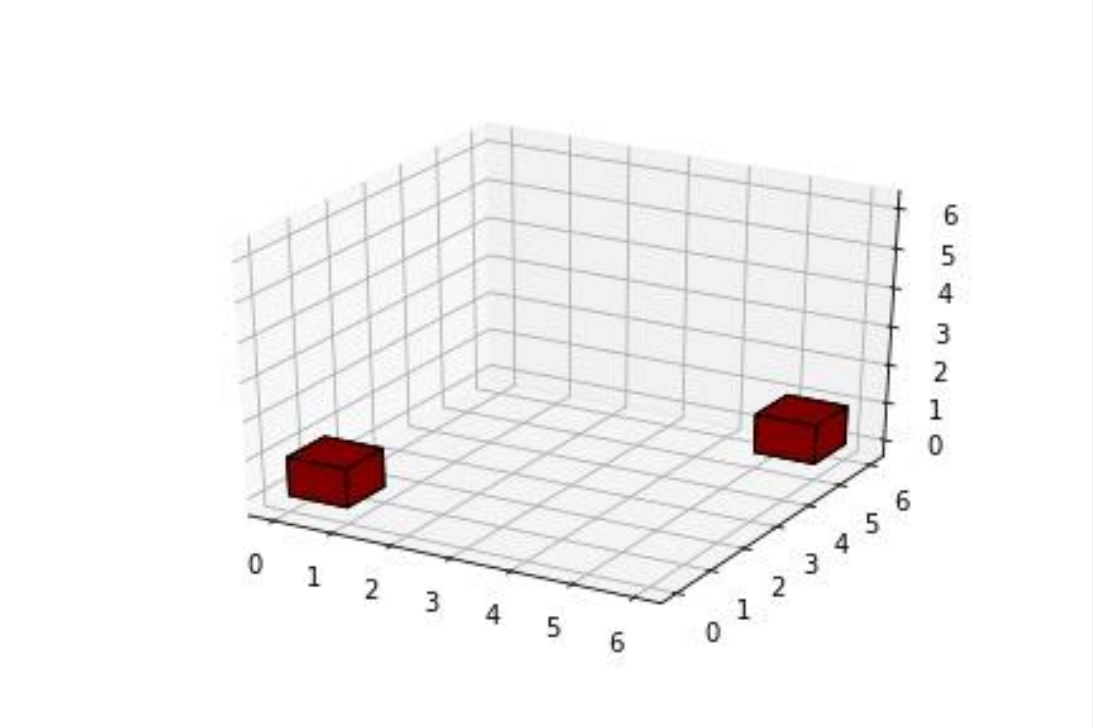
La porte Hadamard



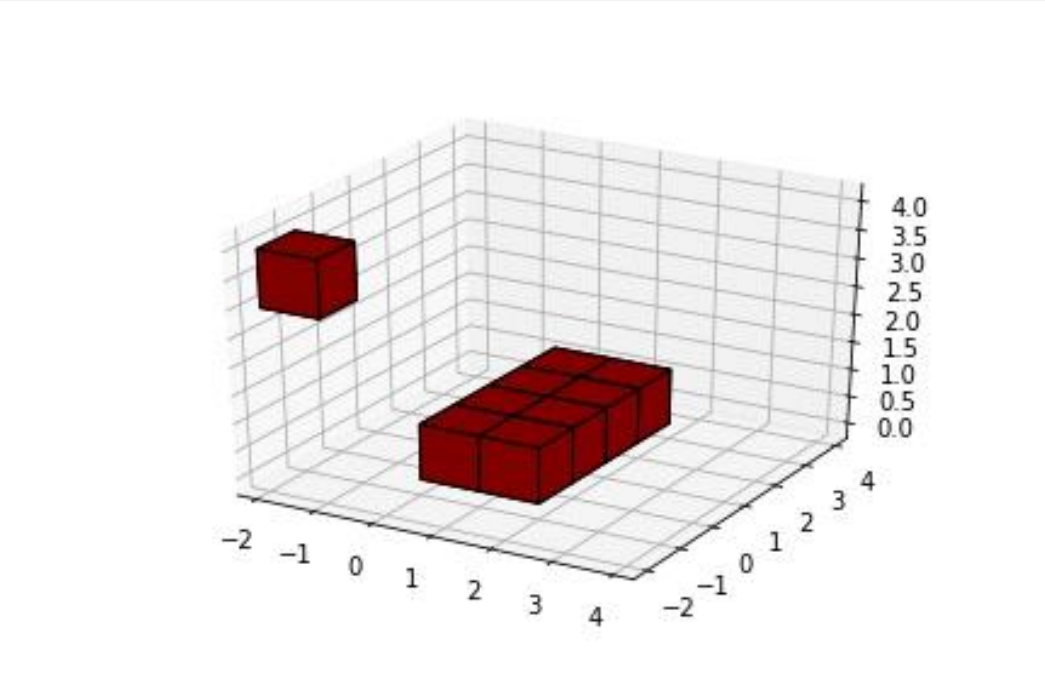


La rotation pi/4





La porte NOT



La porte cNOT

$$cNot|\psi\rangle = (I \otimes H)(cR(\pi/4))^4(I \otimes H)|\psi\rangle$$

Évolutions futures

- Ajouter des "routines" de création de portes

Évolutions futures

- Ajouter des "routines" de création de portes
- Revoir le partage des différentes configurations