Forest fires

We will implement a cellular automaton that models the propagation of a forest fire.

1 - Rules of propagation

We model the forest by a grid of **n** rows and **m** columns.

Each cell of this grid can be:

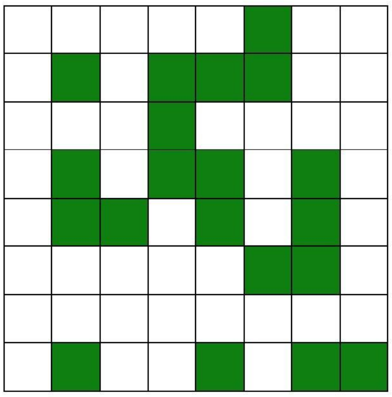
- empty
- a healthy tree
- a burning tree
- a burnt tree

Initially, each cell of this grid contains a healthy tree with probability p and is empty with probability $1-p$, where $0 \leq p \leq 1$. Randomly, we set fire to one of theses trees.

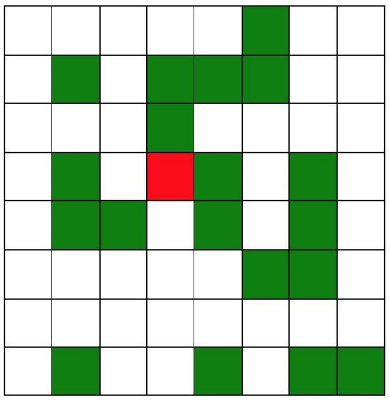
From an instant T to an instant $T+1$, the fire propagates according to these rules:

- an empty cell remains empty
- a healthy tree with at least one burning tree in its neighborhood catches fire (we use the Van Neumann neighborhood, i.e. the adjacent cells to the north, the south, the west and the east of the current cell).
- a healthy tree with no burning tree in its neighborhood remains healthy.
- a burning tree becomes a burnt tree
- a burnt tree remains burnt

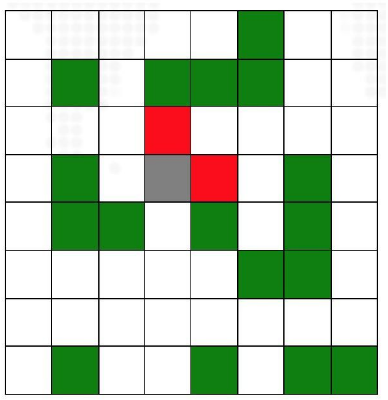
Here is an example of forest, where a healthy tree is represented by a green cell, a burning tree by a red cell, a burnt tree by a grey cell and where the empty cells are in white:



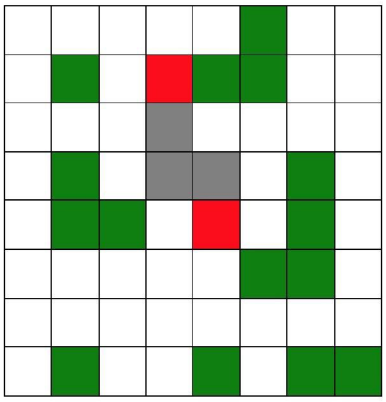
Randomly we set fire to one of those trees:



Next generation, the fire propagates according to the rules:



Next generation, the fire propagates according to the rules:



Etc.

In a first time we will assume that a cell on a side of the grid has no neighbor in this direction. For example, a cell on the right side has no neighbor at the east.

In a second time, we will assume that the grid is toric. For example, a cell on the right side will have a neighbor on the left side.

2 - Python Codes

Important note: in all the code, we will respect the principle of:

Inheritance: child classes inherit data and behaviors from parent class
Encapsulation: containing information in an object, exposing only selected information
Abstraction: only exposing high level public methods for accessing an object
Polymorphism: many methods can do the same task
Implementing an Interface
Implementing Generics with allow us to define a specification of a class or method that can be used with any data type
And a Model architecture

Divide and conquer paradigm

2.1 - A class "tree"

Implement a class "tree" containing the following attribute:

1. The state of the tree. It will be an integer equal to 0, 1, 2 or 3. A 0 for an empty cell, a 1 for a healthy tree, a 2 for a burning tree and a 3 for a burnt tree.

We will provide methods to this class:

1. A constructor with a default value equal to 0.
2. A method that returns a symbol for displaying a tree on the console. It returns a '.' for a state equal to 0, a 'T'' for a state equal to 1, a 'F' for a state equal to 2 and a 'B' for a state equal to3.
3. A method that takes as parameter a boolean indicating if there is a burning tree in the neighborhood. It will update the state of the tree according to the rules of propagation.

And more!!!!!

2.2 - A class "forest"

Implement a class "forest" containing the following attributes:

1. The number of rows of the grid.
2. The number of columns of the grid.
3. A two-dimensional list of objects of the class "tree" representing the grid.

We will provide methods to this class:

1. A constructor that takes as extra parameter a probability value in order to initialize the grid.
2. A method that displays the grid on the console.
3. A method that takes as parameter the coordinates of a cell of the grid and that returns **True** if that cell has a burning tree in its neighborhood and **False** otherwise.
4. A method that updates the grid according to the rules of propagation. Hint: you can use the function ""deepcopy" of the module "copy" to do that.
5. A method that returns the proportion of trees in the current state of the grid.
6. A method that takes as parameter a strictly positive integer and that generates this number of generations according to the rules of propagation.

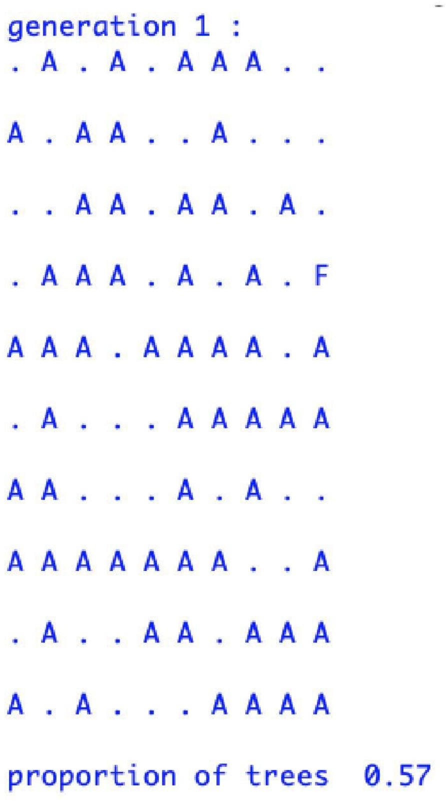
2.3 - A class "toricForest"

Implement a class "toricForest" that is a child class of the class "forest".

References

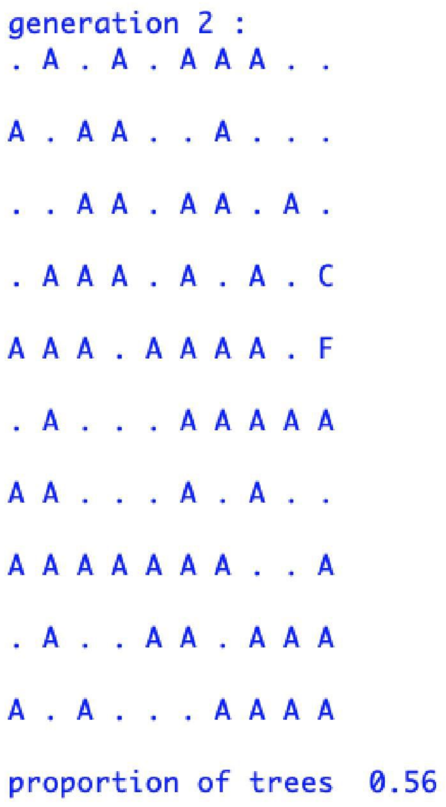
https://www.medisafe.fr/blog/mode-propagation-du-feu
https://www.nist.gov/video/fire-dynamics-simulator-fds-simulation-wildfire
https://tel.archives-ouvertes.fr/tel-00003730/document
http://ressources.univ-lemans.fr/AccesLibre/UM/Pedago/physique/02/recre/foret.html
http://cormas.cirad.fr/fr/applica/fireautomata.htm
http://perso-laris.univ-angers.fr/~hardouin/JalaliReport/JalaliProjectOnSIRModel.html
https://www.youtube.com/watch?app=desktop&v=oM7ROBuPc6U

Here is an example of propagation, first generation:



```
generation 1 :
. A . A . A A A . .

A . A A . . A . . .

. . A A . A A . A .

. A A A . A . A . F

A A A . A A A A . A

. A . . . A A A A A

A A . . . A . A . .

A A A A A A A . . A

. A . . A A . A A A

A . A . . . A A A A

proportion of trees  0.57
```

Second generation:



```
generation 2 :
. A . A . A A A . .

A . A A . . A . . .

. . A A . A A . A .

. A A A . A . A . C

A A A . A A A A . F

. A . . . A A A A A

A A . . . A . A . .

A A A A A A A . . A

. A . . A A . A A A

A . A . . . A A A A

proportion of trees  0.56
```

... tenth generation: