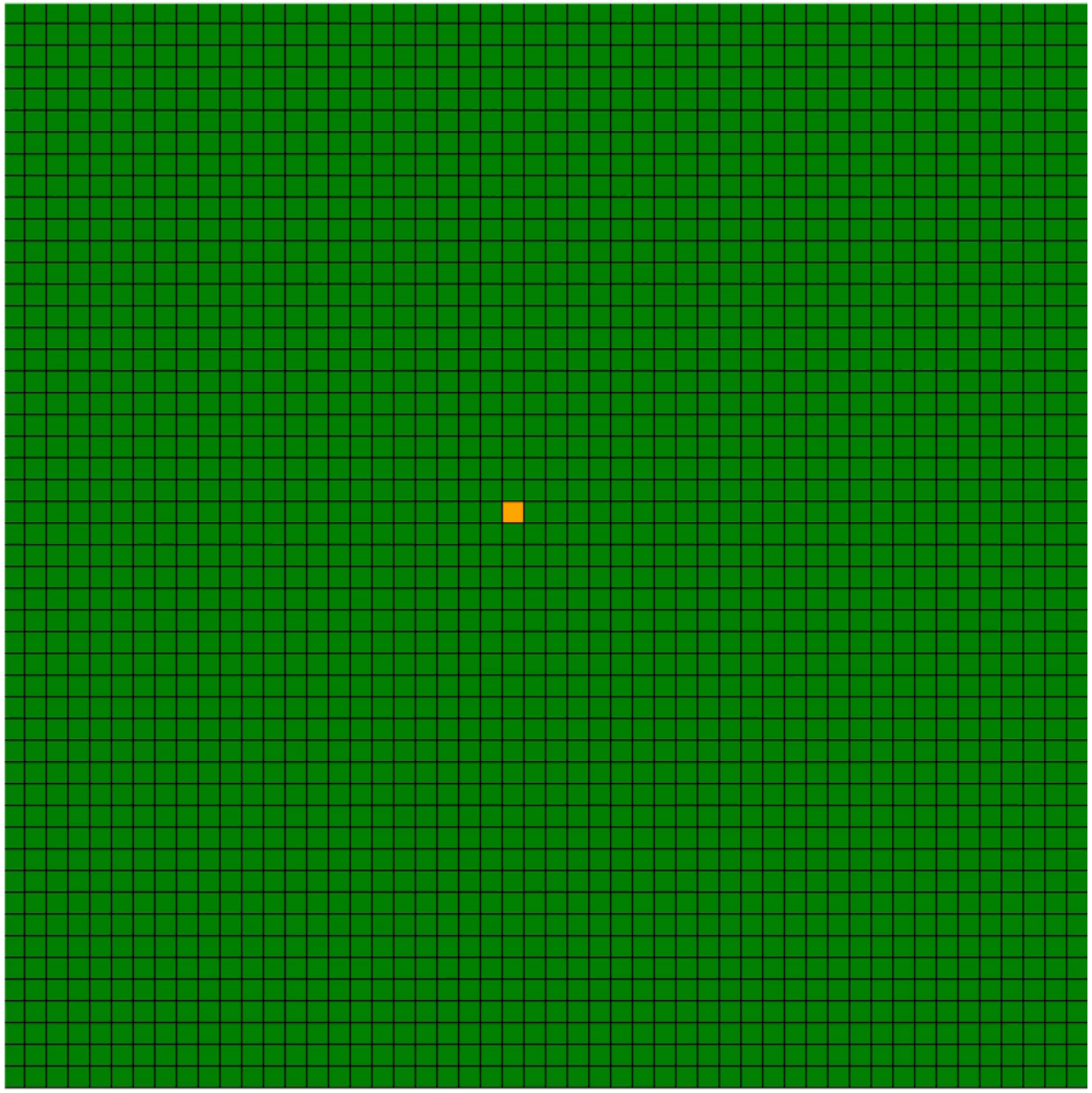
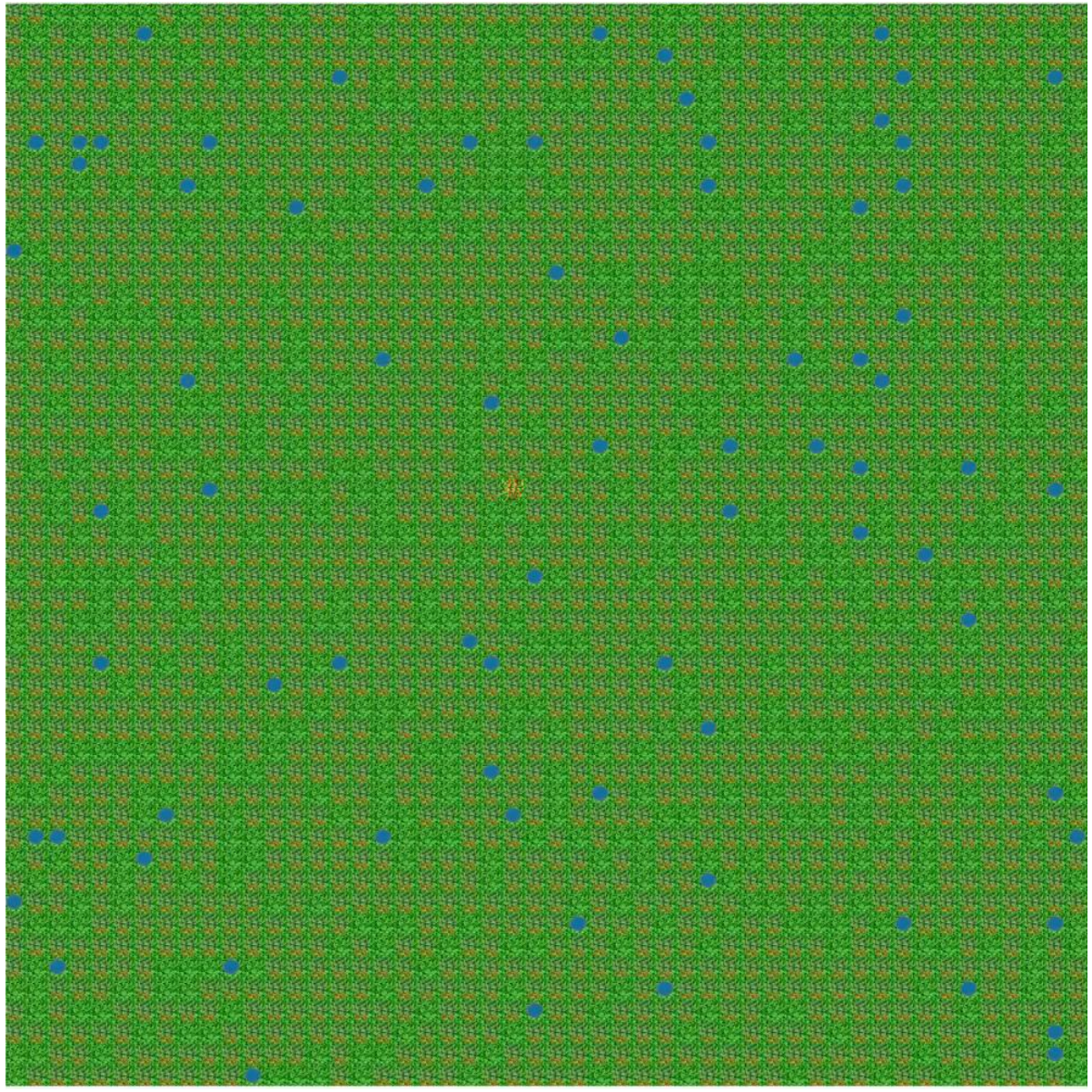
Automates cellulaires



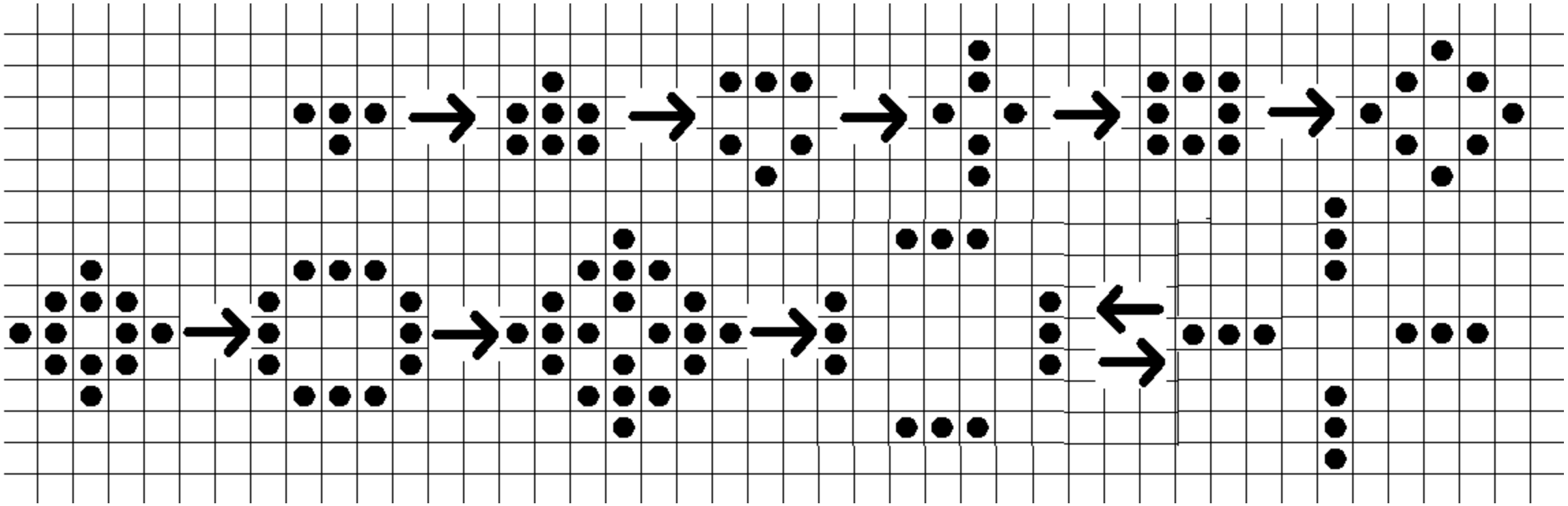


Git Hub

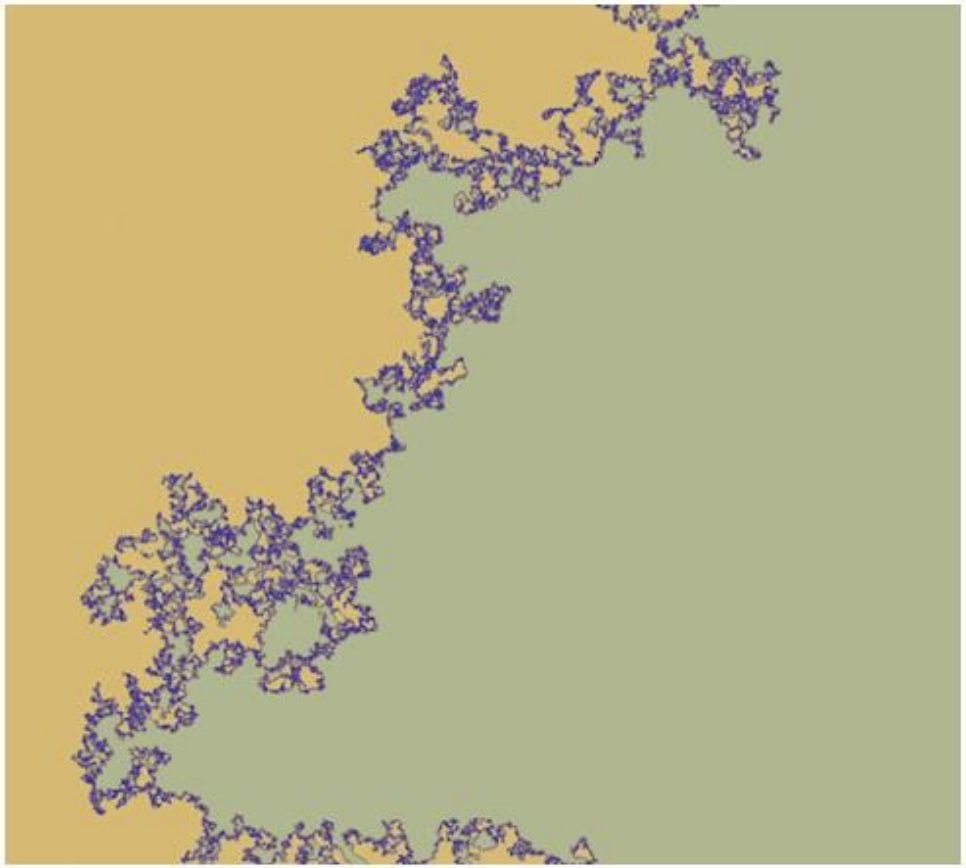
224 commits	2 branches	2 releases	3 contributors

.vscode	commit to pull	5 days ago
__pycache__	Text Box to choice Taille & Proba	6 days ago
ressources projet	Image cool	3 days ago
textures	Small commit	2 months ago
.gitignore	upadate gitignore	19 days ago
AlgoCSV.py	Comment + bug fix + Pro map base	6 days ago
ProAlgoCSV.py	Pro simul	6 days ago
README.md		2 days ago
Résulat_Simulation.png	Rectifications de la fonction enregistrer	9 days ago
algorithmeForet.py	commit to pull	5 days ago
classDialectCsv.py	Commentaires	23 days ago
csv.csv	Create csv.csv	2 days ago
proMap.py	change on prob	5 days ago
projet_tkinter.py	changed int() to float() for proba	2 days ago
varCommunes.py	commit to pull	5 days ago

Les Automates cellulaires

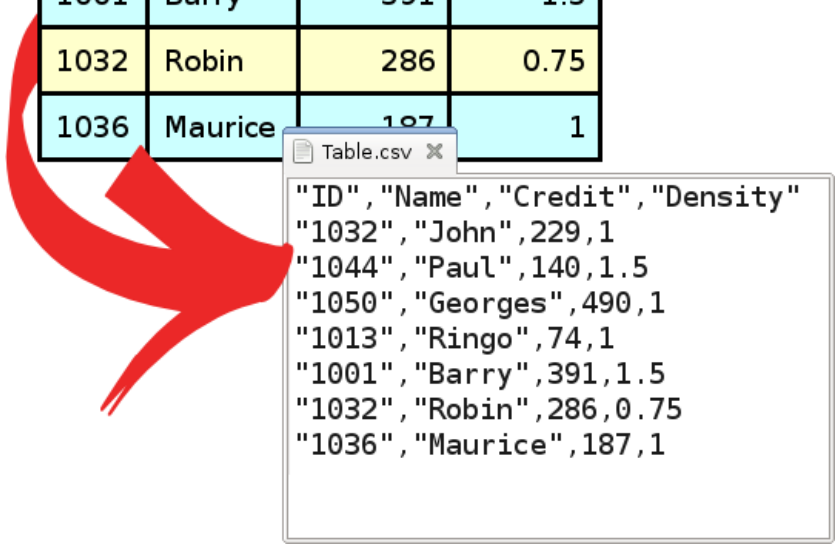


Percolation



CSV

ID	Name	Credit	Density
1032	John	229	1
1044	Paul	140	1.5
1050	Georges	490	1
1013	Ringo	74	1
1001	Barry	391	1.5
1032	Robin	286	0.75
1036	Maurice	187	1



Var Commune

```
class varGlobales():  # Classe stockant quelques '
                      # On évite ainsi les confl
    def __init__(self, largeur, hauteur, nbCell):
        self.hauteurFenetre = hauteur
        self.largeurFenetre = largeur
        self.nbCellules = nbCell          # Nombre d
        self.burnedTrees = 0              # Nombre d
        self.TTtree = 0                   # Nombre d
        self.proba = 0.5
        self.nomCsv  = 'csv.csv'          # Nom du c
        self.listeForet = []              # Liste co
        self.listeCellulesEnFeu = []      # Liste co
        self.listeCellToCheck = []        # Liste co
        self.listeBurnedCell = []         # Liste co
```

```
def getProba(self):
    return self.proba
```

```
def setProba(self, p):
    self.proba = p
```

AlgoCSV

```
class algoCSV():
    def __init__(self, varglobal):
        self.vg = varglobal

    def createCsv(self):
        doubleListe = self.genList()
        with open(self.vg.getNomCsv(), "w", newline="") as f:
            writer = csv.writer(f, classDialectCsv.Dialect())
            for i in range(self.vg.getNbCellules()):
                writer.writerow(doubleListe[i])

    def genList(self):
        doubleListe = []
        for i in range(self.vg.getNbCellules()):
            listX = []
            for j in range(self.vg.getNbCellules()):
                nb = random()
                if nb < 0.75:                   #Abre 75%
                    listX.append(1)
                elif nb < 0.97:                 #Grass 22%
                    listX.append(0)
                else:                           #Eau 3%
                    listX.append(2)
            doubleListe.append(listX)
        return doubleListe
```

Class de Dialect

```
class Dialect(csv.Dialect):
    # Séparateur de champ
    delimiter = ","
    # Séparateur de ''chaîne''
    quotechar = None
    # Gestion du séparateur dans les ''chaînes''
    escapechar = None
    doublequote = None
    # Fin de ligne
    lineterminator = "\r\n"
    # Ajout automatique du séparateur de chaîne (pour ''writer'')
    quoting = csv.QUOTE_NONE
    # Ne pas ignorer les espaces entre le délimiteur de chaîne
    # et le texte
    skipinitialspace = False
```

Create Map

```
def createMap(event):
    algocsv.createCsv()                                     #On créé un csv
    cordY = 0
    gridY = 0
    totalTree = 0
    with open(vg.getNomCsv(), "r", newline='') as f:
        canvas.delete("all")                                # Reset du canvas précédent
        reader = csv.reader(f, classDialectCsv.Dialect())
        for row in reader:                                  # On regarde d'abord les lignes
            cordX = 0                                       # On reset X à chaque nouvelle ligne
            gridX = 0
            for i in row:                                   # Ici c'est la boucle des collones || On met la valeur de
                i = int(i)                                  # Mon reader renvoie un i sous forme de String donc je le
                #On test le i, 0=grass, 1=tree, else=eau
                if i == 1:
                    canvas.create_image(cordX, cordY, anchor=tkinter.NW, image=tree, tag=str(gridX)+","+str(gridY))
                    totalTree += 1
                elif i == 0:
                    canvas.create_image(cordX, cordY, anchor=tkinter.NW, image=grass, tag=str(gridX)+","+str(gridY))
                else:
                    canvas.create_image(cordX, cordY, anchor=tkinter.NW, image=water, tag=str(gridX)+","+str(gridY))
                cordX = cordX+tailleImg                     # On augmente les cords pour afficher l'image au bon endr
                gridX+=1
            cordY = cordY+tailleImg
            gridY+=1
            vg.setTTtree(totalTree)
```

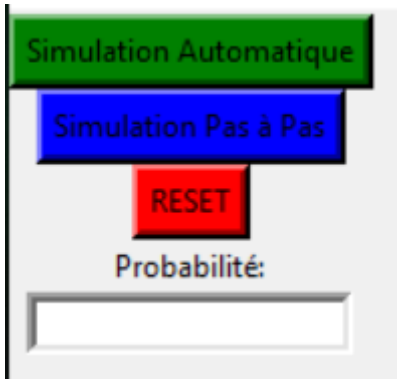
Update Map

```
def updateMap(cellEnFeu, burnedCell):                                   #Ici on update l'affichage
    for i in range(0, len(cellEnFeu), 2):
        canvas.itemconfigure(str(cellEnFeu[i])+","+str(cellEnFeu[i+1]), image=burningTree)
    for i in range(0, len(burnedCell), 2):
        canvas.itemconfigure(str(burnedCell[i])+","+str(burnedCell[i+1]), image=burnedTree)
    stats()
```

Le bouton et les TextBox

```
auto = Button(Fenetre, text = "Simulation Automatique", bg = "green", command = buttonPlusSimu, bd=4)
manuel = Button(Fenetre, text = "Simulation Pas à Pas", bg = "blue", command = pasapas, bd=4)
resetButton = Button(Fenetre, text = "RESET", bg = "red", command = reset, bd=4)
textBoxProba = Entry(Fenetre, justify="center", bd=4)
resultat = Label(Fenetre, text = "")
surfaceBrulee = Label(Fenetre, text = "")
message = Label(Fenetre, text="")

auto.grid(row = 0, column = 0, sticky = "n")
manuel.grid(row = 1, column = 0, sticky = "n")
resetButton.grid(row = 2, column = 0, sticky = "n")
Label(Fenetre, text="Probabilité:").grid(row = 3, column = 0, sticky = "n")
textBoxProba.grid(row = 4, column = 0, sticky = "n")
```



Reset

```
def reset():
    execl(sys.executable, sys.executable, *sys.argv)
```

ButtonPlusSim

```
def buttonPlusSimu():
    if not textBoxProba.get() == "":
        vg.setProba(float(textBoxProba.get()))
    lancer_chrono()
    sim_auto()
```

Reload Tx

```
def refreshTxPath():                                    # Réactualise l'emplacement des Textures ap
    global grass, tree, water, burningTree, burningGrass, burnedTree, burnedGrass, tailleImg
    tailleImg = vg.getLengthCell()
    grass = ImageTk.PhotoImage(Image.open("textures/"+str(tailleImg)+"/grass.png"))
    tree = ImageTk.PhotoImage(Image.open("textures/"+str(tailleImg)+"/tree.png"))
    water = ImageTk.PhotoImage(Image.open("textures/"+str(tailleImg)+"/water.png"))
    burningTree = ImageTk.PhotoImage(Image.open("textures/"+str(tailleImg)+"/burning_tree.png"))
    burnedTree = ImageTk.PhotoImage(Image.open("textures/"+str(tailleImg)+"/burned_tree.png"))
    burningGrass = ImageTk.PhotoImage(Image.open("textures/"+str(tailleImg)+"/burning_tree.png"))
    burnedGrass = ImageTk.PhotoImage(Image.open("textures/"+str(tailleImg)+"/burned_grass.png"))
```

Design Pro

```
def genList(self):
    doubleListe = []
    for i in range(self.vg.getNbCellules()):
        listX = []
        for j in range(self.vg.getNbCellules()):
            listX.append(1)
        doubleListe.append(listX)
    return doubleListe
```

Design Pro

```
def createMap(event):
    if textBox.get() == "":
        vg.setNbCell(50)
    else:
        vg.setNbCell(int(textBox.get()))
    tailleImg = vg.getLengthCell()
    algocsv.createCsv()                                     #On créé un csv
    cordY = 0
    gridY = 0
    with open(vg.getNomCsv(), "r", newline='') as f:
        canvas.delete("all")                                # Reset du canvas précédent
        reader = csv.reader(f, classDialectCsv.Dialect())
        for row in reader:                                  # On regarde d'abord les lignes
            cordX = 0                                       # On reset X à chaque nouvelle ligne
            gridX = 0
            for i in row:                                   # Ici c'est la boucle des collones || On met la valeur
de la case dans i
                canvas.create_rectangle(cordX, cordY, cordX+tailleImg, cordY+tailleImg, fill="green",
tag=str(gridX)+","+str(gridY))
                cordX = cordX+tailleImg                     # On augmente les cords pour afficher l'image au bon
endroit après
                gridX+=1
            cordY = cordY+tailleImg
            gridY+=1
        vg.setTTtree(vg.getNbCellules()*vg.getNbCellules())
```

Design Pro

```
def updateMap(cellEnFeu, burnedCell):                                    #Ici on update l'affichage
par un boucle qui prend une liste de cellule à update
    for i in range(0, len(cellEnFeu), 2):
        canvas.itemconfigure(str(cellEnFeu[i])+","+str(cellEnFeu[i+1]), fill="orange")
    for i in range(0, len(burnedCell), 2):
        canvas.itemconfigure(str(burnedCell[i])+","+str(burnedCell[i+1]), fill="grey")
    stats()

textBox = Entry(Fenetre, justify="center", bd=4)
…
Label(Fenetre, text="Taille de la grille:").grid(row = 4, column = 0, sticky = "n")
textBox.grid(row = 5, column = 0, sticky = "n")
```

Problèmes rencontrés

- Algo CSV
- MainLoop
- Superposition des images

Améliorations possibles

- Lag
- Le CSV