# Évolution d'une file de voiture en fonction du comportement des automobilistes.

Damien AUDRAS

CTES 2018/2019 - PROJET MATHÉMATIQUES-INFORMATIQUE

Ce projet a été encadré par Tristan COLOMBO et réalisé par Damien Audras en collaboration avec Abigaël GHOMO TSETEGHO.
*Publié le 30 avril 2019*

# Table des Matières

# 1. Introduction

## 1.1 Description du problème

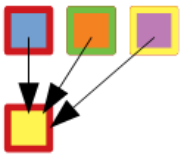
Imaginons une file de voitures et décrivons son évolution possible. Des automobilistes peuvent rentrer dans la file, d'autres en sortir. Certains doivent freiner à cause d'un ralentissement, d'autres accélérer pour retrouver la vitesse autorisée. Un accident peut survenir. Certains ont un comportement nerveux et d'autres pratiquent l'écoconduite. Certains se basent sur la voiture précédente pour anticiper le freinage. Il serait intéressant d'obtenir des statistiques après avoir laissé fonctionner le modèle selon certaines hypothèses. Pour modéliser cette situation on utilise des automates cellulaires à une dimension. Chaque voiture est représentée par une case dans une ligne (la file de voitures) et chaque ligne représente la situation à l'instant t. L'évolution dans le temps se fait du haut vers le bas, on obtient un diagramme en 2D qui s'étend sur une longueur correspondant au temps de la simulation. Les lignes peuvent aussi grandir ou rétrécir en largeur au gré des rentrées et sorties des voitures dans la file. La première ligne peut être remplie aléatoirement avec un nombre aléatoire de voitures, d'au minimum une voiture, sans espace entre les cellules. Pour simuler un feu de circulation ou un rond-point on peut faire entrer un début de ligne une voiture fictive ayant un comportement mémorisé comme freiner, s'arrêter ou ralentir, et sortir.

## 1.2 Présentation d'un automate cellulaire à une dimension

Un automate est une case dans une ligne. Cette case a plusieurs états possibles qui peuvent être représentés par des symboles, des numéros, des hachures ou des couleurs, ou d'autres encore … De l'instant t à l'instant t+1 chaque case peut changer d'état. Ce changement se fait en fonction de l'état des cases environnantes et d'une certaine probabilité. Ce terme de cases environnantes ou de voisinage reste à définir pour bien répondre à la situation. On peut opérer un décalage de certaines cases dans l'une ou l'autre direction de façon à ajouter ou à retirer des cellules avant d'appliquer les

règles de transitions.

La représentation se fait dans une grille dans laquelle une ligne représente un instant et la ligne suivante, l'instant suivant.



Règle de transition de t à t+1

Insertion d'une cellule

Retrait d'une cellule

# 2. Algorithme de transition.

La file de voiture est une liste de cellules de gauche à droite dans le sens de la circulation :la première cellule voit la deuxième et la troisième, la deuxième cellule voit la troisième et la quatrième,...

## 2.1 Description d'une cellule

Une cellule comporte trois états liés au comportement du conducteur et trois états liés à la situation. On utilise la dénomination suivante pour décrire l'état d'une cellule, $e = (c, an, ds, a, v, d)$ avec :

- $c$ est la variable de comportement : $c = 1$ pour un conducteur qui a un objectif de diminution de la consommation, $c = 2$ pour un conducteur standard, dont le seul objectif est de se déplacer et $c = 3$ pour un conducteur nerveux ou aimant jouer avec sa voiture. Pour les cellules spéciales, feu rouge et rond-point, on a $c = 0$.
- $an$ est la variable d'anticipation : $an = 1$ pour un conducteur qui commence à freiner lorsque la voiture qui précède celle qui est devant lui freine ou lorsqu'il aperçoit un feu rouge ou un rond-point en avant de la voiture qui lui précède et $an = 0$ sinon.
- $ds$ est la variable de distance de sécurité : $ds = 1$ si le conducteur respecte les distances de sécurités et $ds = 0$ sinon.
- $a$ est la variable d'accélération : $a = -3$ pour un freinage brutal, $a = -2$ pour un freinage normale, $a = -1$ pour un freinage doux, $a = 0$ pour une garder une vitesse constante, $a = 1$ pour une légère accélération, $a = 2$ pour une accélération normale et $a = 3$ pour une franche accélération.
- $v$ est la variable de vitesse : $v = 0$ pour un véhicule à l'arrêt, $v = 1$ pour une faible vitesse, $v = 2$ pour une vitesse intermédiaire et $v = 3$ pour la vitesse maximale autorisée.
- $d$ est la variable de distance, $d$ indique la distance séparant le véhicule du véhicule précédent. Si $d = 0$ alors il y a un accident, sinon $d$ prend une valeur entière de 1 à 7.

Il y a aussi deux cellules spéciales, les feux rouges et les ronds-points :

— Le feu tricolore à rouge : $e = (c = 0, an = 0, ds = 0, a = -1, v = 0, d = -1)$

— Le rond-point : $e = (c = 0, an = 0, ds = 0, a = -1, v = 1, d = -1)$

$a = -1$ permet de faire freiner le véhicule arrivant sur le feu ou le rond-point.

$d = -1$ permet de ne pas confondre ces cellules avec un véhicule accidenté.

$v = 1$ pour le rond-point permet de différencier les deux cellules et de tenir compte du fait qu'une voiture ne s'arrête pas forcément pour passer un rond-point.

On considère qu'il ne peut y avoir d'accident avec une de ces deux cellules.

## 2.2 Notation de la fonction de transition

On appelle $t$ la fonction qui à l'état $e$ d'une cellule à l'instant $t$ fait correspondre l'état $t(e)$ a l'instant $t+1$ de la même cellule.

On appelle la restriction de $t$ a chacune des composantes de $e$ par la même notation $t$, le contexte permettant de lever les ambiguïtés.

On a $t(e) = (t(c), t(an), t(ds), t(a), t(v), t(d))$

## 2.3 Cause d'un accident

Lorsque le véhicule arrive avec une vitesse élevée trop près du véhicule de devant et que celle-ci freine fort alors il y a un accident et la simulation s'arrête.

**Proposition 2.3.1 — Règle d'accident.** $(v >= 2) \wedge (d = 1) \wedge (a1 = -3) \Longrightarrow t(d) = 0$

## 2.4 Initialisation de la liste de cellules

On place une liste de $n$ cellules normales dans un tableau vide.

On détermine au hasard (probabilité uniforme) les caractéristiques de comportement à l'aide des probabilités p_c1, p_c2, p_an et p_ds , et on prend comme suit les paramètres liés à la situation, pour a un nombre au hasard entre $-1$ et 1 afin d'avoir une faible accélération ou décélération, pour $v$ un nombre au hasard entre 2 et 3 et pour $d$ entre 2 et 4.

— p_c1 : probabilité que $c = 1$

— p_c2 : probabilité que $c = 2$

— p_an : probabilité que $an = 1$

— p_ds : probabilité que $ds = 1$

On utilise les axiomes de Kolmogorov pour déterminer les probabilités des évènements $c = 3$, $an = 0$ et $ds = 0$, c'est-à-dire que la somme des probabilités de tous les évènements élémentaires d'un univers est égale à 1.

On peut par exemple prendre la configuration : p_c1= 0.2, p_c2=0.7,p_an=0.3, p_ds=0.6

## 2.5 Règles de transitions.

Les cellules spéciales ne sont pas concernées par ces règles.

On considère la dernière ligne du tableau des cellules et on appelle $e$ l'état de la cellule dont on cherche à calculer les modifications et $i$ le numéro de sa colonne, $e1$ la cellule de la colonne $i+1$ et $e2$ la cellule de la colonne $i+2$.

$e$ est la modélisation un véhicule, $e1$ du véhicule de devant et $e2$ encore devant.

On accélère ou freine selon que la voiture de devant est proche ou loin, selon qu'on respecte ou non les distances de freinage et selon que la voiture de devant accélère, reste à vitesse constante, freine normalement ou brutalement.

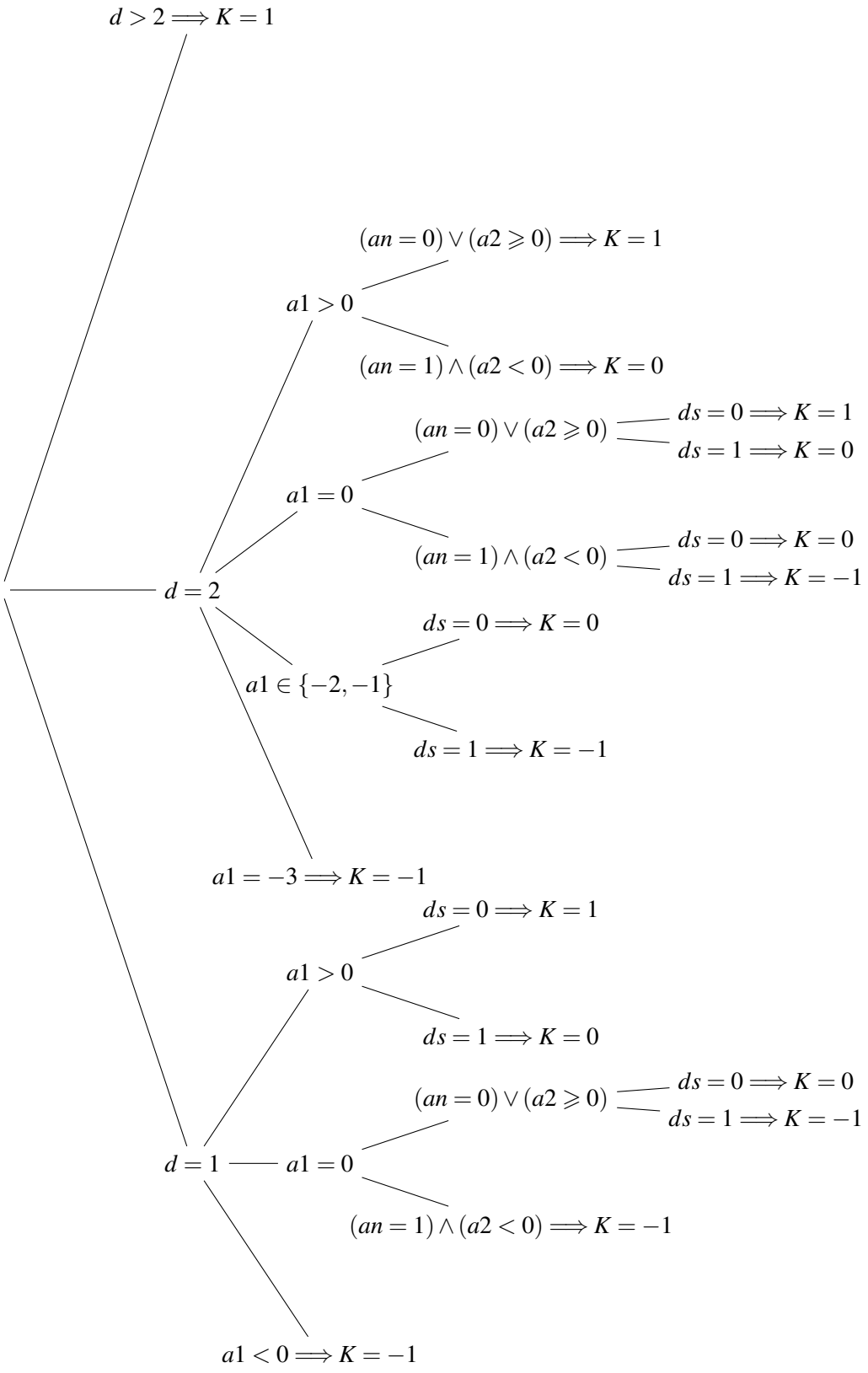
Les règles de transitions représentées dans l'arbre suivant permettent de calculer un coefficient $K$ qui prend pour valeur $1,0$ ou $-1$ et qui multiplié à $c$ donne l'accélération du véhicule.

**Proposition 2.5.1 — bridage de $K$.** –
Un véhicule qui est à la vitesse maximale ne peut pas accélérer et un véhicule qui est à l'arrêt ne peut pas freiner : $((v=3)\wedge(K=1))\vee((v=0)\wedge(K=-1))\Longrightarrow K=0$

**Proposition 2.5.2 — calcul de $t(e)$.** –

$$\begin{cases} t(a)=Kc \text{ et on applique : } t(a)<-3\Longrightarrow t(a)=-3 \text{ et } t(a)>3\Longrightarrow t(a)=3 \\ t(v)=v+a \text{ et on applique : } t(v)<0\Longrightarrow t(v)=0 \text{ et } t(v)>3\Longrightarrow t(v)=3 \\ t(d)=d+v1-v \text{ et on applique : } \begin{cases} (t(d)<1)\wedge(ds=0)\Longrightarrow t(d)=1 \\ (t(d)<2)\wedge(ds=1)\Longrightarrow t(d)=2 \\ t(d)>7\Longrightarrow t(d)=7 \end{cases} \end{cases}$$

On considère ainsi qu'un automobiliste à nécessairement la possibilité de respecter les distances de sécurités.


$d > 2 \Longrightarrow K = 1$
$(an = 0) \vee (a2 \geqslant 0) \Longrightarrow K = 1$
$a1 > 0$
$(an = 1) \wedge (a2 < 0) \Longrightarrow K = 0$
$(an = 0) \vee (a2 \geqslant 0)$
$ds = 0 \Longrightarrow K = 1$
$ds = 1 \Longrightarrow K = 0$
$a1 = 0$
$(an = 1) \wedge (a2 < 0)$
$ds = 0 \Longrightarrow K = 0$
$ds = 1 \Longrightarrow K = -1$
$d = 2$
$ds = 0 \Longrightarrow K = 0$
$a1 \in \{-2, -1\}$
$ds = 1 \Longrightarrow K = -1$
$a1 = -3 \Longrightarrow K = -1$
$ds = 0 \Longrightarrow K = 1$
$a1 > 0$
$ds = 1 \Longrightarrow K = 0$
$(an = 0) \vee (a2 \geqslant 0)$
$ds = 0 \Longrightarrow K = 0$
$ds = 1 \Longrightarrow K = -1$
$d = 1$
$a1 = 0$
$(an = 1) \wedge (a2 < 0) \Longrightarrow K = -1$
$a1 < 0 \Longrightarrow K = -1$

## 2.6 Création d'une nouvelle cellule.

### 2.6.1 Cellule normale

Comme lors de l'initialisation, on détermine au hasard les caractéristiques de comportement à l'aide des probabilités p_c1, p_c2, p_an et p_ds , et on prend pour les paramètres liés à la situation v=1, a=2, d=4 pour garantir une entrée en toute sécurité. On considère que le véhicule entre dans la file avec une vitesse faible mais une accélération normale et à une bonne distance du véhicule devant lui.

### 2.6.2 Cellule spéciale

Les cellules spéciales sont créées selon leurs définitions :

— Le feu tricolore à rouge : $e = (c = 0, an = 0, ds = 0, a = -1, v = 0, d = -1)$
— Le rond-point : $e = (c = 0, an = 0, ds = 0, a = -1, v = 1, d = -1)$

## 2.7 Comment ajouter ou retirer une cellule.

Pour introduire une cellule dans une ligne du tableau :

1. On choisit un véhicule qui va se retrouver derrière le véhicule entrant. On suppose qu'il y a une place suffisante devant ce véhicule pour permettre l'insertion. Pratiquement on prend au hasard un nombre $i$ parmi les numéros de colonne des cellules qui vérifient $d \geqslant 2$, ce qui exclu de fait les cellules spéciales.
2. On place toutes les cellules à partir de la colonne $i+1$ dans la colonne d'indice immédiatement supérieur, en commençant par la fin et après avoir ajouté une place à la fin de la ligne.
3. On génère une cellule normale ou spéciale et on la place dans la colonne d'indice $i+1$.
4. On applique la règle d'insertion sur la cellule d'indice $i$

**Proposition 2.7.1 — règle d'insertion.** Si la cellule entrante est un véhicule on diminue de 1 la distance du véhicule le précédant sinon on ne modifie pas cette distance. On considère qu'un feu ou qu'un rond-point ne prend pas le conducteur par surprise.

Pour retirer une cellule dans une ligne du tableau :

1. On choisit la cellule parmi les cellules spéciales ou normales. On obtient son numéro de colonne $i$.
2. On applique la règle de retrait sur la cellule d'indice $i-1$
3. On place toutes les cellules à partir de la colonne $i+1$ dans la colonne d'indice immédiatement inférieur et on supprime la dernière cellule de la ligne.

**Proposition 2.7.2 — règle de retrait.** Si la cellule précédant la cellule retirée est un véhicule $(c > 0)$ alors on augmente sa distance de 1.

## 2.8 Quand ajouter ou retirer une cellule.

### 2.8.1 Ajout

On insère une cellule normale à chaque itération avec une probabilité p_e et une spéciale avec une probabilité d'entrée de p_eS.

La variable aléatoire qui donne le nombre d'entrées au bout de n itérations suit la loi binomiale, donc le nombre moyen d'entrées de cellules normales au bout de n itérations est np_e et de cellules spéciales de np_eS.

En prenant p_e=0.5, p_eS=0.2 on obtient en moyenne une cellule normale toutes les deux itérations et une spéciale toutes les 5 itérations.

### 2.8.2 Retrait

On retire une cellule normale à chaque itération avec une probabilité de p_r et une cellule spéciale avec une probabilité de sortie de p_rS.

Pour avoir une espérance de vie d'une cellule spéciale de 2 itérations on prend p_rS=1/2=0.5.

Pour que la file de cellules reste stable en longueur on prend p_r=0.5.

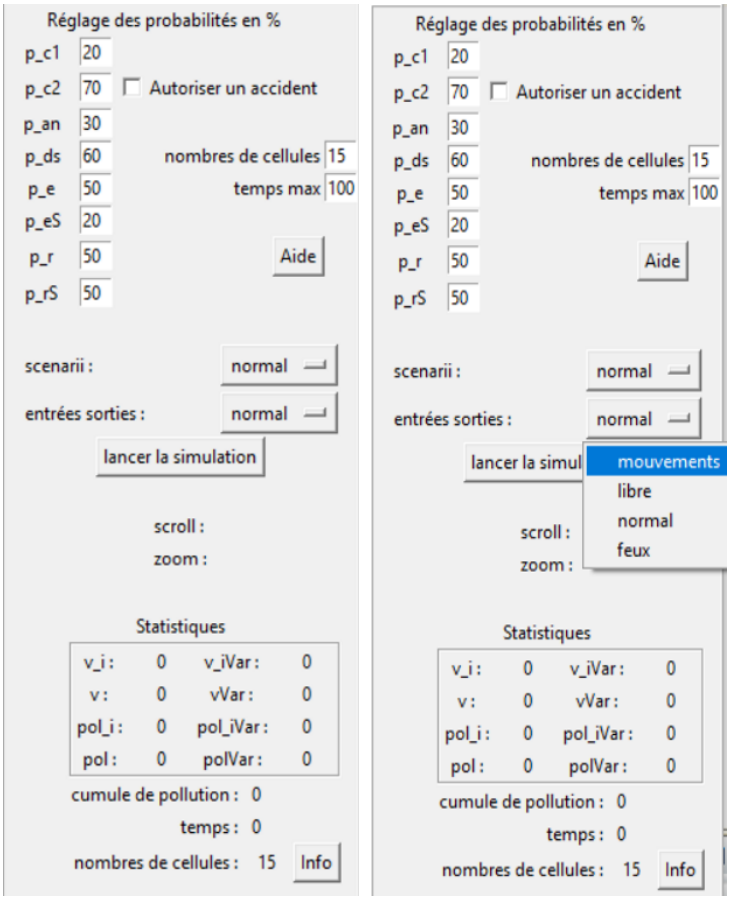
### 2.8.3 Notation

— p_e : probabilité d'entrée d'une cellule
— p_eS : probabilité d'entrée d'une cellule spéciale
— p_r : probabilité de retrait d'une cellule
— p_rS : probabilité de retrait d'une cellule spéciale

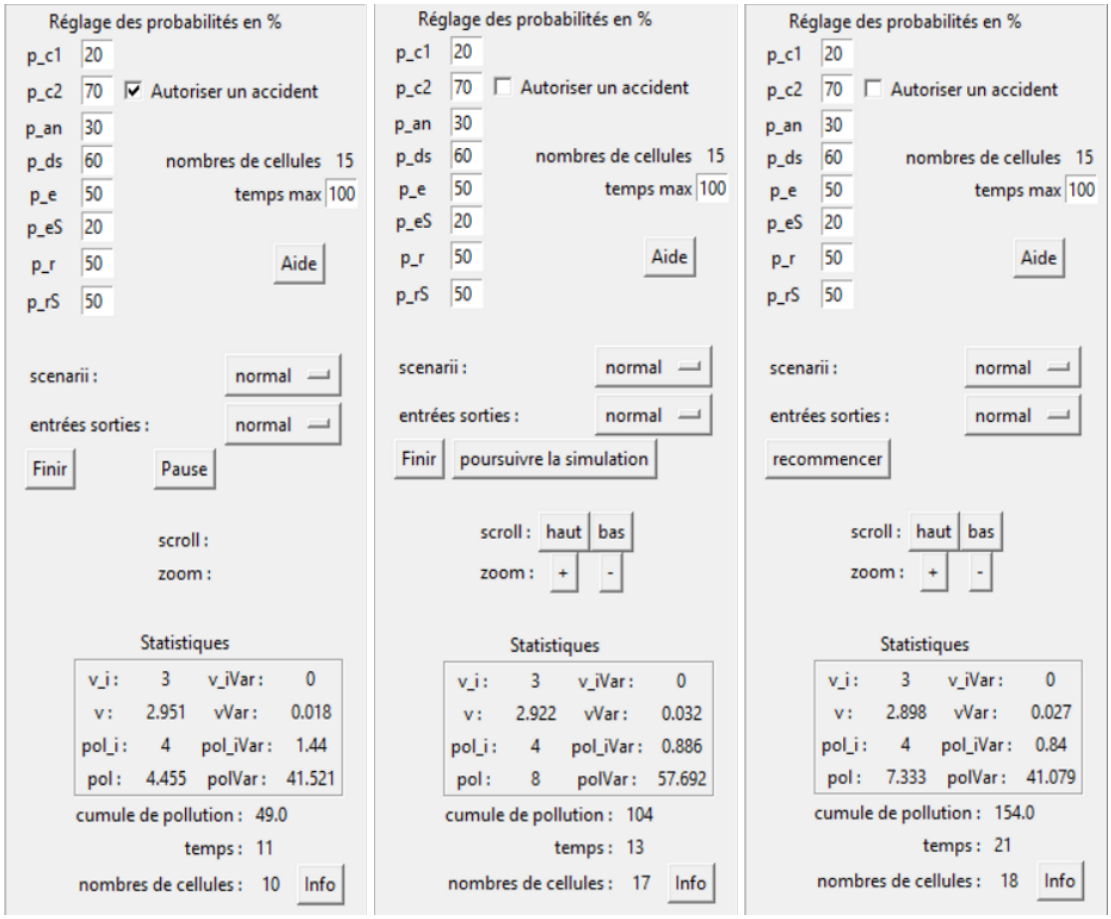
# 3. L'interface graphique.

## 3.1 Description de la fenêtre

La fenêtre se compose d'une partie consacrée à la représentation des cellules, à gauche et d'une partie composée du menu et des résultats statistiques, à droite.
Le menu permet de choisir les valeurs de la répartition de chaque caractéristique de comportement d'une cellule entre ses différentes possibilités.
Il permet également de choisir la fréquence d'apparition et de retrait d'une cellule, spéciale ou non. Il permet de déconnecter la possibilité de repérer un accident.
Les menus déroulants proposent différents scénarios pour utiliser l'application sans avoir à faire ses propres hypothèses.
Il y a aussi des boutons pour faire défiler en haut ou en bas la représentation graphique (scroll) et permettant de faire un zoom sur cette représentation.



## 3.2 Les états de la fenêtre




Le bouton central change d'état :

— État 1 : Lancer la simulation. Lors de l'appui sur ce bouton, le programme de calcul de la matrice des cellules se lance. Les statistiques s'affichent. Le graphisme des cellules est visible dans la partie gauche de la fenêtre. Le scroll et le zoom ne sont pas possibles. Le menu passe à l'état 2.

— État 2 : Pause et Finir. Le scroll et le zoom ne sont pas possibles. L'appui sur le bouton Pause arête l'itération de l'algorithme de transition et le menu passe à l'état 3. L'appui sur Finir arête l'itération de l'algorithme de transition. Le menu passe à l'état 4.

— État 3 : Il est possible de modifier les valeurs des variables, elles seront prises en compte lors de la poursuite de la simulation. Le scroll et le zoom sont possibles. Le bouton Finir a le même effet que dans l'état 2.

— État 4 : La simulation est terminée, Le scroll et le zoom sont possibles. L'appui sur le bouton recommencer initialise toutes les variables. Le menu passe à l'état 1.

## 3.3 Choix de programmation

Suite à la lecture de l'article [3], j'ai décidé de garder l'esprit de concision propre au langage Python et de ne pas définir de classe d'objets. Python autorise à programmer des fonctions locales, placées dans une fonction mère. Elles profitent alors de l'espace des variables propres à la fonction mère, ce qui permet de supprimer le passage de paramètres dans les boutons et simplifie le code. Par exemple, le code :

```
boutonActiver=Button(menuC,command=lancer,
                                    text='lancer la simulation')
```

appelle la fonction lancer() qui va modifier des objets définis dans le corps de la fonction mère menuIhm, notamment boutonActiver.

Pour pouvoir modifier des variables de la fonction mère ou des paramètres de cette fonction, ces variables doivent être mutable, c'est le cas des objets de Tkinter qui possèdent une méthode get ou config, c'est aussi le cas des tableaux fournit en paramètres, notamment tourne qui est défini en tableau pour cette raison par le code :

```
tourne[0]=ihm.after(100*2,iterer,probas,accident,tMax)
```

Cette variable est modifiée à chaque itération et pour pouvoir stopper la boucle il faut disposer de la dernière valeur de tourne[0].

Pour garantir la sécurité des données en mémoire, une ligne de tableau est passé en valeur avec une instruction de type `memoire["listeV"][:]` qui copie la liste ou du type `deepcopy(memoire["cellules"][t])` lorsque qu'il y a plus d'un niveau de profondeur, en l'occurence, `memoire["cellules"][t]` est une liste de listes.

# 4. La représentation des cellules

Elle est basée sur la classe Cellule fournit par Abigaël GHOMO TSETEGHO, deuxième membre du projet.

La fonction `afCellules` s'occupe de placer une lignes de cellules côte à côte au numéro de ligne donné par argument. Les cellules sont disposée de façon à ce qu'il n'y ait pas d'espace entre elles. Le canevas, la liste de la ligne de la matrice cellules à afficher et les dimensions sont données en argument.

La fonction `efCellules` s'occupe d'effacer les cellules de la ligne donnée en argument.

# 5. Les statistiques.

## 5.1 La vitesse

v_i est la vitesse moyenne des cellules représentant des véhicules, dans la ligne courante, c'est la moyenne des vitesses des voitures de la file, on peut alors parler de vitesse instantanée de la file de voitures.

v_iVar est la variance correspondante à v_i.

v est la vitesse moyenne par rapport au paramètre temps de la simulation, c'est-à-dire au nombre d'itérations, de la file depuis le début de la simulation.

vVar est la variance correspondante à v.

## 5.2 La pollution

pol signifie niveau de pollution engendré par le freinage et l'accélération, il correspond à la somme des carrés du paramètre accélération des voitures. Cela suppose que le nombre de particules rejetées par les voitures est proportionnel au carré de leur accélération.

On se base sur l'augmentation ou la baisse d'énergie cinétique provoquée par une accélération ou un freinage. On connaît la formule $E = \frac{1}{2}mv^2$. Lors d'une accélération $a$ en unité de temps discrétisé, la vitesse augmente de $a$. L'accroissement d'énergie est alors :

$$\frac{1}{2}m(v_1^2 - v_2^2) = \frac{1}{2}m(v_1 - v_2)(v_1 + v_2) = \frac{1}{2}ma(a + v_1) = \frac{1}{2}ma^2 + \frac{1}{2}mav_1$$

Cette formule dépend de la vitesse initiale de la voiture et elle ne tient pas compte des frottements liés au démarrage de la voiture, or ils sont très importants. Il suffit de regarder la consommation instantanée sur l'ordinateur de bord d'une voiture pour s'en convaincre. On appelle donc pollution le facteur $a^2$.

pol_i, pol_iVar, pol, polVar ont la même signification que précédemment avec pol à la place de v.

# 6. Le moteur de l'application.

## 6.1 La mémoire

Le fichier main.py contient la mémorisation des données sous la forme de variables globales mutables, des listes ou des objets.

Ces variables sont toutes initialisées au lancement du programme. Certaines variables doivent aussi être réinitialisées lorsque l'utilisateur clique sur le bouton recommencer, c'est le rôle de la fonction init().

La fonction lancement() s'effectue lors d'un appui sur le bouton du même nom. Elle récupère les données de la fenêtre et lance la boucle avec iterer().

## 6.2 La boucle

iterer() construit la boucle, le moteur du processus, par appel récursif décalé dans le temps à l'aide de l'instruction after. Ce décalage est nécessaire pour laisser le temps à l'utilisateur de voir le processus se réaliser devant ses yeux.

Lors de cette boucle les différentes fonctions sont appelées tour à tour pour afficher les statistiques et les cellules, gérer les interruptions liées au temps maximum voulu et à un possible accident, ensuite pour calculer l'évolution de la ligne et incrémenter le compteur temps.

Cette fonction retourne le statut True pour signaler une interruption et False pour poursuivre.

## 6.3 Les algorithmes de scroll et de zoom

Il y a deux structures de données contenu dans le dictionnaire `memoire` : la liste des données décrivant chaque ligne de cellules, appelé `cellules` et la liste des lignes d'objets `Cellules`, appelé

`afCellules`.

Trois variables nous permettent de nous situer dans ces deux listes. Il y a l'objet Tkinter IntVar `temps` dont on connaît la valeur avec la méthode get() et qui donne le nombre de lignes de `cellules`. Il y a la première case de la liste `inAfCel` de `memoire` qui correspond à $i$ et qui donne l'indice de la première ligne vide dans laquelle on doit afficher les cellules, en sachant que la première ligne de cellules affichées à pour indice 0. La troisième variable est la deuxième case de la liste `inAfCel` de `memoire` qui correspond à $n$ et qui donne le nombre maximale de lignes pouvant être affichées.

**Proposition 6.3.1 — relation entre $temps$ et $i$.** On note $t_0$, la variable `tempsAfCel0`, qui a pour valeur l'indice dans la listes `cellules` de la ligne qui précède la première ligne affichée à l'écran, d'indice 0 dans `afCellules`. Au départ, quand une seul ligne est affichée, on convient $t_0 = -1$.

On note $t_{max}$, la variable `tempsAfCelMax`, qui a pour valeur l'indice dans la listes `cellules` de la ligne qui suit la dernière ligne affichée à l'écran, d'indice $n-1$ dans `afCellules`.

On a la relation $temps = t_0 + i$ donc $\boxed{t_0 = temps - i}$

Si l'écran est remplit et qu'il y a au moins une ligne dans `cellules` après la dernière affichée, $i > n$, on a la relation $t_{max} = t_0 + n + 1$ donc $\boxed{t_{max} = temps - i + n + 1}$. Et sinon, il n'y a pas de ligne après la dernière ligne affichée, on garde la formule qui donne $t_{max} > temps$. En effet, $i \leqslant n iff - i + n \geqslant 0 iff t_{max} \geqslant temps + 1$.

1. Algorithme moteur de `iterer()` Valeur initiale : $temps = -1$, $i = 0$, $t_0 = temps - i = -1 + 0 = -1$ est vérifiée.
   Dans `iterer()`, au départ, on initialise la première ligne de cellules, on incrémente $temps$ donc $temps = 0$, on affiche cette ligne, on incrémente $i$ donc $i = 1$, on a alors la relation $t_0 = temps - i = 0 - 1 = -1$ qui est vérifiée.
   Un test d'interruption pour cause de temps dépassé ou d'accident a lieu et les boutons scroll et zoom sont rendus disponibles après utilisation du bouton finir.
   On relance `iterer()` avec un délai. Les boutons scroll et zoom sont rendus disponibles si l'utilisateur utilise le bouton pause.
   L'écran se remplit, $0 < temps < n$ et $1 < i < n + 1$.
   Après avoir affiché la dernière ligne de l'écran, on a $temps = n - 1$ et $i = n$.
   Lors de l'appel suivant, $i = n$ dépasse la capacité de l'écran. Il faut utiliser un scroll vers le bas pour faire apparaître une ligne vide en bas de l'écran : $temps = n - 1$, $i = n - 1$ et $t_0 = temps - i = 0$, en effet la ligne d'indice 0 de `cellules` est cachée et c'est bien la première ligne à pouvoir s'afficher avec un scroll vers le haut.
   Ensuite $temps \geqslant n$.
   Il y a un test pour effectuer un zoom moins si le nombre de cellules à afficher dépasse la largeur de l'écran. En utilisant le même rapport largeur/hauteur pour les cellules et pour le canvas, on a le même nombre maximal de cellules en hauteur qu'en largeur.
2. Algorithme de scroll vers le bas :
   Il permet de libérer une ligne vide en bas de l'écran, il correspond à descendre l'ascenseur dans une page d'un navigateur.
   ne s'effectue que si l'écran n'est pas vide
   effacer la ligne en haut de l'écran
   remonter toutes les lignes d'un rang vers le haut
   Si il n'y a pas de nouvelle cellules à afficher en bas, c'est à dire si $t_{max} > temps$, il faut supprimer la dernière ligne de `afCellules` car cette liste contient les cellules réellement affichées à l'écran.

Sinon il faut afficher la ligne de `cellules` d'indice $t_{max}$, ainsi que les statistiques associées avec cette ligne.
Enfin $i$ prends la valeur $i-1$.
La proposition est vérifiée en sortie d'algorithme.

3. Algorithme de scroll vers le haut :
   Il permet de rappeler la dernière ligne passée en haut de l'écran, il correspond à monter l'ascenseur dans une page d'un navigateur.
   Ne s'effectue que si il y a une ligne dans l'historique, c'est à dire si $t_0 \geqslant 0$
   Effacer la ligne en bas de l'écran si $i \geqslant n$
   Descendre toutes les lignes d'un rang vers le bas et afficher les statistiques associées avec ligne du bas de l'écran.
   Afficher la ligne d'indice $t_0$ de `cellules` tout en haut de l'écran, à l'indice 0 de `afCellules`.
   Enfin $i$ prends la valeur $i+1$.
   La proposition est vérifiée en sortie d'algorithme.
4. Algorithme de zoom plus grand. Il permet d'afficher une ligne de cellules en moins avec des cellules plus grosses.
   On interdit ce zoom si l'écran doit afficher moins de 10 lignes, $n < 10$, ou si une seul ligne est présente à l'écran, $i < 2$.
   La dernière ligne affichée reste fixe, la première ligne d'écran est effacée donc on décrémente $i$ et $n$. L'indice dans `cellules` de la première ligne affichée a augmenté de 1 donc $t_0$ a augmenté de 1. La propriété $t_0 = temps - i$ est donc vérifiée.
5. Algorithme de zoom plus petit. Il permet d'afficher une ligne de cellules en plus avec des cellules plus petites.
   La première ligne affichée reste fixe, on calcul $t_{max}$ et on incrémente $n$.
   Si $t_{max} \leqslant temps$, la ligne d'indice $t_{max}$ de `cellules` doit être affichée ainsi que les statistiques associées avec cette ligne.
   $t_{max}$ augmente de 1. La relation $t_{max} = temps - i + n + 1$ est donc vérifiée.
   $i$, $temps$ et $t_0$ ne sont pas modifiée donc $t_0 = temps - i$ est vérifiée.

## 6.4 Conclusion

On fait tourner le programme dans les différents scénarios proposés, entrée/sortie normal, avec 30 véhicules et sur une durée de 100 itérations.

| scénario | vitesse | pollution |
|---|---|---|
| normal | 2.95 | 593 |
| prudent | 2.95 | 455 |
| éco | 2.93 | 401 |
| nerveux | 2.95 | 1202 |

On remarque le comportement n'influe pas sur la vitesse globale de la file mais influe sur sa consommation et très fortement pour le comportement nerveux.

On peut se demander quelle proportion de nerveux dans la population peut-on tolérer.

En gardant 10% d'éco, on fait varier la proportion de normal, en se basant sur un scénario normal donc sans modifier les distances de sécurité :

| % nerveux | pollution | % d'évolution (base 0%) |
|---|---|---|
| 0 | 583 | 0 |
| 10 | 694 | 19 |
| 20 | 714 | 22 |
| 30 | 895 | 54 |
| 40 | 1289 | 121 |
| 50 | 864 | 48 |
| 60 | 1217 | 109 |
| 70 | 3584 | 515 |
| 80 | 1710 | 193 |
| 90 | 1038 | 78 |

On remarque qu'il est important de ne pas dépasser 20% de comportement trop nerveux pour limiter la pollution et qu'il serait utile de faire à nouveau des campagnes de publicité pour inciter les automobilistes à limiter leurs accélération et freinage en ville.

On met actuellement en valeur les véhicules électriques pour leur importante réactivité. Si les conducteurs nerveux les achètent, cela peut faire baisser la pollution, sauf si cela entraîne un changement de comportement global et que les conducteurs jusqu'à lors mesurés ne se mettent à les imiter.

# Bibliographie


[1] Automates cellulaires, J.-Ph. Rennard -
https ://www.rennard.org/alife/french/ac.pdf

[2] Automates cellulaires 1D : Principes -
https ://deptinfo-ensip.univ-poitiers.fr/ENS/doku/doku.php/stu :automates_cellulaires_1d

[3] Traduction autorisée de Functional Programming in C++ par John Carmack :
https ://cpp.developpez.com/redaction/data/pages/rubrique/cpp/carmack/fonctionnel/