TP 02 :
Automates Cellulaires

Master 2 MISC

Exercice 1 : initialisation

Dans ce premier exercice, vous allez initialiser la simulation en créant et affichant l'automate cellulaire dans sa position et configuration initiale.

Chaque patch possède une position fixe avec des coordonnées entières $(pxcor, pycor)$. Le patch au centre de l'espace est l'origine du repère, il a pour coordonnées $(0, 0)$. La première ligne (en haut) correspond à $pycor$ = max-pycor où max-pycor est une constante définie via le bouton setting de l'interface.

Sur la première ligne vous allez représenter l'automate cellulaire dans son état initial (*i.e.* à $t = 0$). Les cellules occuperont toute la largeur disponible (*i.e.* le nombre de cellules sera de $(2*\text{max-pxcor})+1$).

- Définir une variable globale ligne pour représenter la ligne courante :

  ```
  globals [
   ligne
  ]
  ```

  Initialement *ligne* vaut 0

- Définir pour chacun des patchs une variable propre etat :

  ```
  patches-own [
    etat
  ]
  ```

  L'état est une variable binaire qui peut prendre soit la valeur $false$ soit la valeur $true$.

- Analyser le code suivant et exécuter la procédure d'initialisation setup en créant le bouton correspondant dans l'interface.

On pourra pour l'instant conserver la configuration par défaut de l'espace des patchs (max-pxcor= 16, maxpycor= 16, Patch size= 13).

```
to setup
   clear-all
   set ligne 0
   ask patches [
      set pcolor black
```

```
      if pycor = max-pycor - ligne [
         ifelse random 2 = 0 [
           set etat true
         ]
         [
           set etat false
         ]
      ]
   ]
   colorier ligne
end

to colorier [ l ]
   ask patches with [ pycor = max-pycor - l] [
      ifelse etat
      [ set pcolor blue  ]
      [ set pcolor white ]
   ]
end
```

Exercice 2 : fonction locale

Dans cet exercice, vous allez définir la dynamique temporelle de l'automate. A chaque pas de temps (tick) toutes les cellules vont simultanément changer leur état propre. La règle de changement d'état est identique pour toutes les cellules. Le nouvel état d'une cellule à l'instant $t+1$ ne dépend de trois informations : son propre état à l'instant t et de celui de ses deux voisines de droite et de gauche au même instant t. Il y a donc $2^3 = 8$ configurations binaires à spécifier pour définir une règle de transition d'état et donc $2^8 = 256$ règles (ou automates) possibles.

Remarque : on considère que l'espace des patchs est circulaire (cocher World wraps horizontally via le bouton setting de l'interface) et donc que le voisin de droite (resp. gauche) de la cellule qui se trouve à l'extrême droite (resp. gauche) est celle qui se trouve à l'extrême gauche (resp. droite).

Dans cet exercice, vous allez simuler et exécuter la règle de transition suivante (pour faciliter l'écriture, 0 correspond à un état $false$ et 1 à un état $true$.) :

$\{0,1\}^3$	000	001	010	011	100	101	110	111
f	0	1	0	1	0	1	1	0

Le numéro de cette régle est 106.

Une règle sera implémentée en netLogo par une procédure reporter nouvel-etat qui prend en arguments l'état de la cellule à mettre à jour et l'état de ses deux voisines et qui retourne (report) le nouvel état.

1 Compléter en conséquence le reporter nouvel-etat ci-dessous de façon à coder la règle 106.

```
to-report nouvel-etat [g c d]
     ifelse (g =  false and c = false and d = false) [report ...]
    [ifelse (g =  false and c = false and d = true ) [report ...]
    [ifelse (g =  false and c = true  and d = false) [report ...]
    [ifelse (g =  false and c = true  and d = true ) [report ...]
    [ifelse (g =  true  and c = false and d = false) [report ...]
```

```
    [ifelse (g =  true  and c = false and d = true ) [report ...]
    [ifelse (g =  true  and c = true  and d = false) [report ...]
    [     if(g =  true  and c = true  and d = true ) [report ...]
    ]]]]]]]
end
```

2 Analyser le code de la procédure go ci-dessous ; en particulier regarder dans la documentation de rôle de la primitive tick. Créer les deux boutons go et gooo dans l'interface pour lancer la simulation...

```
to go
   ask patches with [pycor = max-pycor - ligne][
      let gauche [etat] of patch-at -1 0 ;; voisin de gauche
      let centre [etat] of patch-at 0 0 ;; ou bien "let centre etat"
      let droit [etat] of patch-at 1 0 ;; voisin de droite
      ask patch-at 0 -1 [
        set etat nouvel-etat gauche centre droit
      ]
   ]
   set ligne (ligne + 1)
   colorier ligne
   tick
end
```

Remarque : vous pouvez par exemple configurer l'interface (bouton setting) avec max-pxcor= 110, maxpycor= 130, Patch size= 2.

Exercice 3 : Programmation des régles

Dans cette question vous allez permettre à l'utilisateur de choisir, parmi les 256 règles de transitions possibles, celle qui sera simulée. Pour cela vous devez déclarer huit nouvelles variables globales via la définition de huit Switchs dans l'interface. Un Switch est un composant de l'interface qui permet de positionner (on/off) une variable booléenne avec les valeurs true or false.

1 Définir 8 Switchs, chacun étant associé à une des variables globales e0, e1, e2, e3, e4, e5, e6, e7. On pourra placer à proximité de chaque Switch un composant Note afin d'indiquer sa signification ; par exemple, au Switch e3 on associera la note 011 (écriture de 3 en binaire).

2 Modifier le reporter nouvel-etat pour prendre en compte le choix de l'utilisateur (via les 8 variables ei).

3 Simuler la dynamique de l'automate :

   a. induite par la règle 0
   b. induite par la règle 255
   c. induite par la règle 247
   d. induite par la règle 90

4 Ajouter un composant Monitor dans l'interface et associer ce composant à une nouvelle procédure reporter code-decimal qui retourne la valeur décimale du nombre binaire (ie. exprimé en base 2) e7e6e5e4e3e2e1e0 (numéro de la régle).

```
to-report code-decimal
   let d 0
   if e0 [set d ...]
   ...
   report d
end
```

5 Simuler la dynamique de l'automate numéro 72, 159, 169, 137, etc.

Exercice 4 : Initialisation et fin

1. Modifier la procédure d'initialisation setup et ajouter un composant Switch afin que l'utilisateur puisse choisir entre une initialisation au hasard (cas précédent) et une initialisation centrée où la seule cellule active (etat=true) est celle du centre.
2. Modifier la procédure principale *go* afin de stopper (utiliser la primitive stop) la simulation lorsque l'on atteint la dernière ligne.
3. Faites afficher (show) la valeur de la variable globale prédéfinie ticks au début de la procédure go : que remarquez-vous ? Pouvez-vous ré-écrire l'ensemble de votre programme sans utiliser la variable ligne ?

Exercice 5 : Jeu de la vie

L'objectif de cet exercice est de vous faire programmer un automate en dimension 2, le fameux jeu de la vie.

- Initialisation : 10% des cellules doivent être actives

```
ifelse (random 10 = 0) [set etat 1] [set etat 0]
```

- Chaque patch à une variable d'état et une variable pour compter le nombre de voisins actifs

```
patches-own [
  etat
  nbVoisins
]
```

- Mise à jour 1 : Pour chaque patch on compte le nombre de voisins actifs

```
set nbVoisins count other neighbors with [etat = 1]
```

- Mise en jour 2 : Chaque patch met à jour son état en fonction du nombre de voisins

```
ifelse (etat = 0  and nbVoisins = 3)
[
  set etat 1
]
[
  if ((etat = 1 and nbVoisins < 2) or (etat = 1 and nbVoisins > 3))
```

```
  [
    set etat 0
  ]
]
```