1 Initiation

Exercice 1

1. Créez un environnement constitué d'un damier de cases jaunes et noires.

```
ask patches with [pxcor mod 2 = 0 xor pycor mod 2 = 0] [set pcolor yellow]
```

ou encore :

```
ask patches with [pxcor mod 2 != pycor mod 2] [set pcolor yellow]
```

2. Écrivez une procédure permettant de créer ce damier et de donner naissance à 10 tortues. Placez un bouton permettant de lancer cette procédure.

```
to setup
   ca
   ask patches with [pxcor mod 2 = 0 xor pycor mod 2 = 0] [set pcolor yellow]
   crt 10
end
```

3. Créez une procédure permettant à une tortue de se déplacer aléatoirement, puis placez un bouton permettant de déplacer l'ensemble des tortues indéfiniment.

```
to wiggle
   lt random 30
   rt random 30
   fd 1
end
```

Faire ensuite un bouton et le paramétrer en mode "Forever" pour les agents "Turtles"

Exercice 2

1. Créez un environnement de 105×69 patchs de taille 5, non torique.
2. Écrivez une procédure `setup` qui initialise les patchs de façon à dessiner grossièrement un terrain de foot (gazon, avec un rectangle blanc sur les bords, un ligne blanche au milieu, et un cercle blanc de rayon 10).

```
to setup
  ca
  dessiner-terrain
end

to dessiner-terrain
  ask patches [set pcolor green]
  ask patches with [pxcor = max-pxcor or pxcor = min-pxcor
    or pxcor = 0 or pycor = max-pycor or pycor = min-pycor]
    [ set pcolor white ]
  ask patches with [distance patch 0 0 > 9 and distance patch 0 0 <= 10 ]
    [ set pcolor white ]
end
```

3. Modifiez `setup` pour créer 11 joueurs (tortues) rouges et 11 bleus, placés au hasard sur le terrain. Donnez leur une taille de 5.

```
to setup
  ca
  dessiner-terrain
  set-default-shape turtles "person"
  create-turtles 11 [ init-joueur blue ]
  create-turtles 11 [ init-joueur red ]
end

to init-joueur [coul]
  set color coul
  set size 5
  setxy random-pxcor random-pycor
end
```

4. Écrivez une procédure `wiggle` qui déplace chaque joueur aléatoirement, et une procédure `go` qui déplace tous les joueurs.

```
to wiggle
  lt random 30
  rt random 30
  ifelse can-move? 1 [fd 1][rt 180]
end

to go
  ask turtles [wiggle]
end
```

Exercice 3

1. Créez un environnement dans lequel certains patchs contiennent 10 unités de nourriture. On considère qu'un patch a 2 chances sur 5 de contenir de la nourriture. On utilisera pour cela une variable `food` initialisée avec une valeur entière. Le patch sera coloré en jaune si cette variable est supérieure à zéro.

```
patches-own[food]

to setup
   ca
   ask patches [
     ifelse (random 5 < 2)
       [set food 10 set pcolor yellow]
       [set food 0]]
end
```

Comme les variables NetLogo sont initialisées à 0, on peut aussi écrire :

```
to setup
   ca
   ask patches with [random 5 < 2]
       [set food 10 set pcolor yellow]
end
```

2. Créez des tortues de forme `bug` pour représenter des fourmis qui se déplacent aléatoirement et qui mangent une unité de nourriture lorsque le patch où elles se trouve en contient. La fonction `patch-here` permet désigner le patch où une tortue se trouve. On appellera `decide` la procédure exécutée par chaque tortue pour définir son comportement, et `go` la procédure qui définit ce qui se passe à chaque itération dans l'ensemble de la simulation.

```
patches-own[food]

to setup
  ca
  ask patches with [random 5 < 2]
     [set food 10 set pcolor yellow]
  crt 10
  set-default-shape turtles "bug"
end

to wiggle
  lt random 30
  rt random 30
  fd 1
end

to decide
  ask patch-here [if (food > 0) [set food (food - 1)]]
  wiggle
end

to go
  ask turtles [decide]
end
```

3. Faites varier la couleur des patchs contenant de la nourriture en utilisant un dégradé de jaune (grâce à la fonction `scale-color`)

```
il faut rajouter dans "to go"
  ask patches [set pcolor scale-color yellow food 0 10]
```

4. Placez un graphique (composant `plot` dans l'interface) qui affiche la quantité de nourriture totale dans l'environnement.

```
il faut rajouter dans "to go"
  plot sum [food] of patches
Par défaut, si il n'y a qu'un crayon et qu'un graphique, rien d'autre n'est à préciser.
```

2 Simulation n'utilisant que des patchs

Nous allons maintenant programmer un automate cellulaire à deux dimensions à deux états, en l'occurrence le fameux « Jeu de la Vie » de Conway. Le principe en est le suivant :

— chaque cellule peut être vivante ou morte ;
— les cellules évoluent en parallèle (on calcule le nouvel état de toutes les cellules avant d'appliquer les modifications) ;
— les règles d'évolution d'une cellule dépendent du nombre N de voisins vivants parmi les 8 cellules autour d'elle (voisinage de Von Neumann) :
  — si une cellule est vivante, elle survit si elle a exactement $N = 2$ ou $N = 3$ voisins vivants ;
  — si une cellule est morte, elle naît si elle a exactement $N = 3$ voisins vivants ;
  — dans tous les autres cas l'état de la cellule reste inchangé.

Par convention dans la suite un patch vivant sera blanc et un patch mort noir.

1. Changez l'environnement pour disposer de 100×100 cellules de taille 4 dans un monde torique.
2. Écrivez une procédure `setup` permettant d'initialiser aléatoirement l'environnement. La probabilité qu'un patch soit vivant est donnée par une variable `densite-vivants` (entre 0 et 100) réglée au moyen d'un `slider`.

```
to setup
  ask patches [
    ifelse (random 100 < densite-vivants)
      [set pcolor white]
      [set pcolor black]
  ]
end
```

3. Ajoutez une variable `prochain-etat` dans chaque patch et écrivez la procédure `evoluer` qui permet à un patch de calculer la valeur de cette variable en fonction de l'état des patchs voisins.

```
patches-own [ prochain-etat ]
to evoluer
  set prochain-etat pcolor
  let n count neighbors with [ pcolor = white ]
  ifelse (pcolor = white)
    [ if ((n < 2) or (n > 3))
      [ set prochain-etat black ]]
    [ if (n = 3)
       [ set prochain-etat white]]
end
```

4. Écrivez la procédure `go` qui fait évoluer en parallèle tous les patchs puis effectue le changement d'état.

```
to go
  ask patches [ evoluer ]
  ask patches [ set pcolor prochain-etat ]
end
```

3 Simulation n'utilisant que des tortues

On s'intéresse maintenant à un problème de probabilités: le problème de Monty Hall. Un jeu met un candidat face à trois portes fermées (désignées par exemple par 0, 1, 2). Derrière deux d'entre elles, il n'y a rien ; derrière une seule (par exemple 2) se trouve une récompense. Le candidat en choisit une (par exemple 1) : alors, pour augmenter le suspense, l'animateur du jeu (qui sait où se trouve la récompense) ouvre une **autre** porte (dans notre exemple, la porte 0) : elle est vide. On peut noter que l'animateur a toujours une porte vide autre que le choix du candidat à ouvrir : si celui-ci a choisi la bonne porte, les deux autres sont vides, et s'il a choisi une porte vide, il en reste une. L'animateur demande alors au candidat s'il souhaite maintenir son choix initial ou changer en faveur de l'autre porte restée fermée (en l'occurrence ici la 2).

Deux comportements sont possibles et, s'il est possible de déterminer mathématiquement la bonne attitude à adopter, nous allons ici réaliser une étude expérimentale au moyen d'un échantillon de joueurs se conformant à ces deux comportements. L'`observer` de NetLogo va jouer le rôle de l'animateur et jouera avec `nb-joueurs` tortues qui maintiennent leur choix initial (des `obstinés`) et autant de tortues qui changent d'avis (des `versatiles`).

1. Créez une variable globale `porte` correspondant au numéro de la porte menant à la récompense, et dans les tortues deux variables `choix` et `alternative` correspondant respectivement au choix de la tortue et à la porte vide ouverte par l'animateur.

```
globals [ porte ]
turtles-own [ choix alternative ]
```

2. Créez deux espèces NetLogo (`breed`) : les `persévérants` (qui seront dessinés en bleu) et les `versatiles` (en rouge). Écrivez des procédures qui permettent d'initialiser la simulation, c'est-à-dire créer les `nb-agents` tortues de chaque espèce en les plaçant à une position aléatoire, avec la bonne couleur.

```
breed [perseverants obstine]
breed [versatiles versatile]

to setup
  ca
  set-default-shape turtles "circle"
  create-perseverants nb-agents [init-obstine]
  create-versatiles nb-agents [init-versatile]
end

to init-obstine
  setxy random world-width random world-height
  set color blue
end

to init-versatile
  setxy random world-width random world-height
  set color red
end
```

3. On donne la procédure `go` qui définit un cycle de jeu : l'animateur choisit aléatoirement la porte qui contient la récompense (`tirage-au-sort`), puis demande aux tortues de `faire-un-choix`, montre ensuite à chaque tortue une porte vide différente de son choix (`montrer-alternative`), redemande aux tortues si elles veulent changer leur choix ou non (`decision-finale`) et enfin attribue des récompenses (`attribuer-lots`). On décide que la récompense est un changement de taille de la tortue : `+0.01` en cas de victoire, et `-0.01` en cas d'échec (en NetLogo, changer l'attribut `size` redessine immédiatement l'agent). Écrivez les procédures correspondantes puis testez-les. Bien évidemment, pour que l'expérience soit significative, il faut itérer un nombre N de jeux pour savoir quelles sont les tortues qui grossissent.

```
to go
  tirage-au-sort
  ask turtles [ faire-un-choix ]
  montrer-alternative
  ask turtles [ decision-finale ]
  attribuer-lots
end
```

```
to tirage-au-sort
  set porte random 3
end

to faire-un-choix
  set choix random 3
end

to montrer-alternative
  ask turtles [
    ifelse (choix = porte)
      [ set alternative (choix + 1) mod 3 ]
      [ set alternative 3 - (choix + porte) ]
  ]
end

to decision-finale
  if (breed = versatiles)
    [ set choix 3 - (choix + alternative) ]
end

to attribuer-lots
  ask turtles [
    ifelse (choix = porte)
      [ set size size + 0.01 ]
      [ if (size > 0.01) [ set size size - 0.01 ]]
  ]
end
```

4 Une étude expérimentale : « Boys & girls »

Extrait d'un questionnaire pour l'entretien d'embauche chez Google : « Dans un pays où les gens ne veulent que des garçons, les familles continuent d'enfanter jusqu'à ce qu'elles aient un garçon. Si elles ont une fille, elles font un autre enfant. Si elles ont un garçon, elles s'arrêtent. Quelle est la proportion de garçons par rapport aux filles dans le pays ? ».

Proposez un programme NetLogo et une série d'expériences statistiques qui permettent de répondre à cette question.

```
turtles-own [boys girls]

to setup
  ca
  set-default-shape turtles "person"
  crt population [setxy random-xcor random-ycor set color red set boys 0 set girls 0]
end

to go
  ask turtles with [boys = 0] [decide]
  tick
  if not any? turtles with [boys = 0] [stop]
end

to decide
  ifelse (random 2 = 0)
    [ set boys 1 set color blue ]
    [ set girls (girls + 1) ]
end
```

— Vous afficherez un moniteur avec le nombre de garçons et un autre avec le nombre de filles

```
sum [boys] of turtles
sum [girls] of turtles
```

— Vous tracerez la courbe d'évolution du nombre de garçons et de filles
— Vous afficherez dynamiquement l'histogramme du nombre de filles par famille

```
histogram [girls] of turtles
```