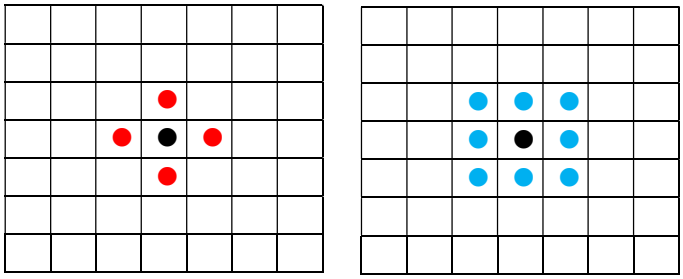
LIFAMI – TD / TP : Dynamique des populations

Objectifs : Applications en biologie
Manipulation des structures, des boucles, des tests, etc.

Automate cellulaire et jeu de la vie

Un **automate cellulaire** est une grille régulière de « cellules » contenant chacune un « état » choisi parmi un ensemble fini d'états et qui peut évoluer au cours du temps. L'état d'une cellule au temps *t+1* est fonction de l'état au temps *t* d'un nombre fini de cellules appelé son « voisinage ».

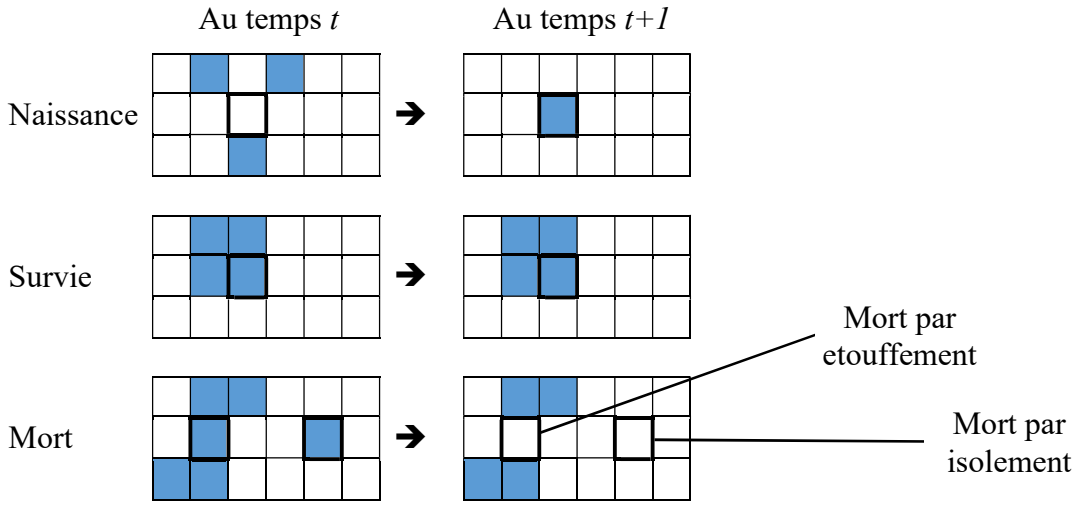


Le **jeu de la vie** est un automate cellulaire imaginé par le mathématicien britannique John Horton Conway en 1970 dans lequel chaque cellule peut prendre l'un des deux états « vivant » ou « mort » et est entourée de **huit** cases susceptibles d'accueillir d'autres cellules.

Les règles d'évolution entre le temps *t* et le temps *t+1* sont les suivantes.

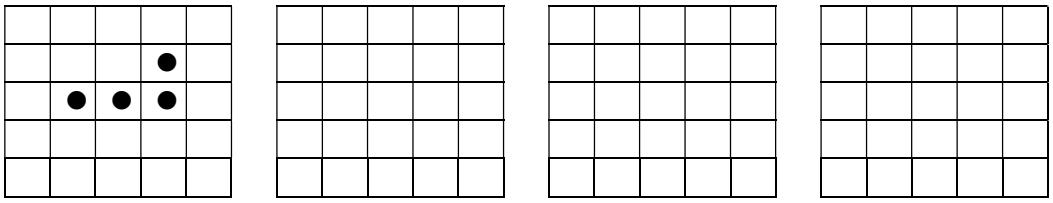
- **La survie / la stance** : chaque cellule ayant deux ou trois cellules adjacentes vivantes survit jusqu'à la génération suivante.
- **La mort** : chaque cellule ayant quatre cellules adjacentes vivantes ou plus disparaît, ou meurt, par surpopulation / étouffement. Chaque cellule n'ayant qu'une ou aucune cellule adjacente meurt d'isolement.
- **La naissance** : chaque emplacement adjacent ayant exactement trois cellules, fait naître une nouvelle cellule pour la génération suivante.

Toutes les naissances et toutes les morts ont lieu en même temps au cours d'une génération.



A faire en TD

1. En appliquant les règles énoncées précédemment, prévoyez l'évolution du système suivant.



2. Ecrivez la structure `Jeu_de_la_vie` qui contiendra une grille 2D d'états (0 pour morte et 1 pour vivante), la taille de la grille utilisée `dx` et `dy`, et le nombre de cellules vivantes dans la grille `alive`.

3. Ecrivez le sous-programme `init` permettant de fixer les paramètres du jeu (taille de la grille et nombre de cellules vivantes) et de remplir la grille avec des cellules mortes.

4. Ecrivez le sous-programme `etat_initial` permettant de positionner aléatoirement des cellules vivantes dans la grille.

5. Ecrivez enfin le sous-programme `etat_suivant` qui à partir d'une configuration calcule l'état suivant en fonction des règles énoncées en préambule.

A faire en TP en complément de ce qui précède

6. Pour permettre un affichage de la grille sous Grapic, ajouter des champs `Img_Dead` et `Img_Alive` de type `Image` à votre structure `Jeu_de_la_vie`. Ajoutez à votre dossier `data` deux images de votre choix symbolisant la `vie` et la `mort`.

7. Implémentez en adaptant à la nouvelle structure les procédures `init`, `etat_initial`, et `etat_suivant`.

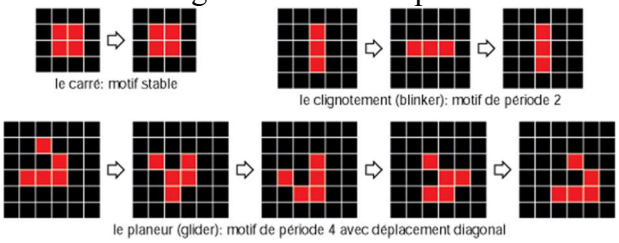
8. Ecrivez la procédure d'affichage de la grille de jeu en fonction des états des cellules contenues dans la structure `Jeu_de_la_vie`.

9. Ecrivez enfin la fonction principale qui appellera successivement les différents sous-programmes en laissant quelques secondes de latence entre l'affichage de deux états successifs.

Pour aller plus loin … en TP !

10. Implémentez la configuration décrite au tout début du TD et vérifiez que ce que vous aviez prédit est correct.

11. Cherchez et tester d'autres configurations "remarquables".



Proies et prédateurs : version analytique

L'écologie mathématique est née dans les années 1920 avec les travaux d'Alfred Lotka (1880-1949) et de Vito Volterra (1860-1940) qui ont proposé indépendamment l'un de l'autre le premier modèle décrivant une interaction de type « proie-prédateur » ou, plus généralement, de type « ressource-consommateur ».

Les équations dites de Lotka -Volterra s'écrivent fréquemment :

$$\begin{cases} \frac{dx(t)}{dt} &= x(t)\left(\alpha - \beta y(t)\right) \\ \frac{dy(t)}{dt} &= -y(t)\left(\gamma - \delta x(t)\right) \end{cases}$$

où

- t est le temps ;
- x(t) est l'effectif des proies en fonction du temps ;
- y(t) est l'effectif des prédateurs en fonction du temps ;
- les dérivées dx(t)/dt et dy(t)/dt représentent la variation des populations au cours du temps.

Les paramètres suivants caractérisent les interactions entre les deux espèces :

- α est le taux de reproduction des proies (constant, indépendant du nombre de prédateurs) ;
- β est le taux de mortalité des proies dû aux prédateurs rencontrés ;
- γ est le taux de mortalité des prédateurs (constant, indépendant du nombre de proies) ;
- δ est le taux de reproduction des prédateurs en fonction des proies rencontrées et mangées.

A faire en TD

12. Un état d'équilibre de la population est observé quand aucune des deux populations en présence n'évolue, c'est-à-dire quand les dérivées correspondantes sont nulles. Ecrivez ce nouveau système d'équation et trouvez ses solutions.

13. En appliquant les coefficients $\alpha = 0.045$, $\beta = 0.001$, $\gamma = 0.025$ et $\delta = 0.0002$ à une population initiale de 120 proies et 40 prédateurs, écrivez les équations permettant de calculer le nombre de proies et de prédateurs au temps t+1 en fonction du temps t.

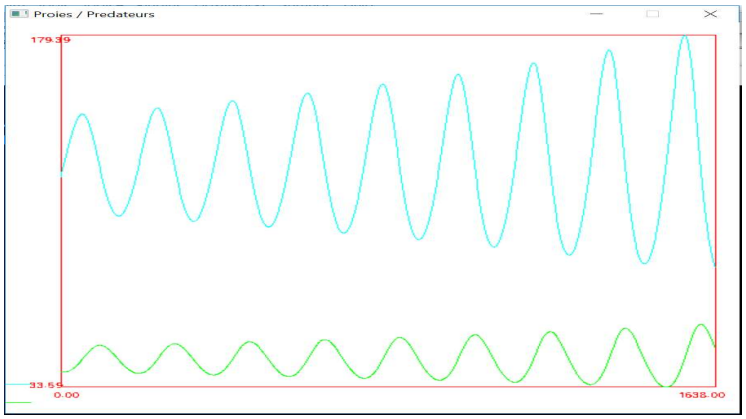
14. Nous allons maintenant décrire les structures nécessaires à l'implémentation du modèle proies / prédateurs en C/C++. Définissez une structure `Ecosysteme` qui dans sa version initiale contiendra les champs suivants : `nb_proies`, `nb_predateurs`, ainsi que les 4 coefficients réels des équations de Lotka-Volterra notés `alpha`, `beta`, `gamma` et `delta`.

15. Ecrivez une fonction `evolution_ecosysteme` qui simule l'évolution de chacune des populations au cours du temps. Les résultats obtenus (nombre de proies et de prédateurs à chaque instant) seront dans un premier temps stockés dans un tableau.

A faire en TP

16. Après avoir défini les structures nécessaires à la modélisation du système proies / prédateur et initialisé les paramètres avec les valeurs précédentes, modifiez la procédure `evolution_ecosysteme` en faisant afficher à chaque pas de temps le nombre de proies et de prédateurs sur un graphique. Vous utiliserez pour cela les fonctions Grapic :
    - `plot_setSize (Plot &p, const int n)`
      qui définit le nombre de valeurs conservées (si n<0 une infinité, cas par défaut).
    - `plot_add (Plot &p, float x, float y, int curve_n)`
      qui ajoute un point (x,y=f(x)) à la courbe numéro `curve_n`.
    - `plot_draw (const Plot &p, int xmin, int ymin, int xmax, int ymax, bool clearOrNot)`
      qui dessine la courbe dans le rectangle `(xmin,ymin,xmax,ymax)`; et efface le contenu du rectangle si `clearOrNot` est à `true`.

Vous devriez obtenir les courbes suivantes :



Proies vs. Prédateurs : version expérimentale !

Nous allons dans cette partie simuler le système proies / prédateurs en nous basant sur des règles de mort / survie telles qu'elles se produisent (approximativement) dans la nature. L'objectif est de suivre l'évolution de chaque population au cours du temps comme dans la version analytique basées sur les équations de Lotka-Volterra et de vérifier qu'on obtient un comportement similaire. Pour chaque individu on énoncera des règles (plus ou moins complexes et réalistes) de naissance, reproduction, survie et mort (par prédation, faim, ou encore vieillesse).

A faire en TD

17. Définissez une structure `Individu` qui contiendra les champs `type_individu` (proie, prédateur ou herbe), `duree_vie` et `duree_jeune`.

18. La représentation visuelle du système se fera au moyen d'une grille 2D dans laquelle seront présent des individus (proie ou prédateur). Définissez la structure `Ecosysteme` qui contiendra une grille 2D d'individus (ainsi que les paramètres de taille de la grille `dx` et `dy`), le nombre de proies et de prédateurs et pour chaque population une image (proie, prédateur et herbe).

19. Ecrivez la procédure `evolution_ecosysteme` qui prédit le devenir de chaque individu en fonction de son environnement (voisinage) avec les deux règles suivantes.
    - Si deux proies sont dans des cases adjacentes, elles se reproduisent et donnent donc naissance à un nouvel individu de type proie placé dans une case libre.
    - Si un prédateur a dans son voisinage une proie, il la mange, s'il a un autre prédateur dans son voisinage il se reproduit uniquement s'il reste de la place dans le voisinage.

A faire en TP

20. Définissez les constantes `DUREE_VIE_PROIE`, `DUREE_VIE_PREDATEUR`, `MAX_JEUNE_PROIE`, et `MAX_JEUNE_PREDATEUR`.

21. Définissez les structures `Individu` et `Ecosysteme`. Choisissez une image pour chaque type d'individu (proie, prédateur et herbe) et enregistrez-les dans le dossier `data`. Ces images doivent stockées dans la structure `Ecosysteme`.

22. Ecrivez la procédure `init_ecosysteme` qui initialisera le système avec des valeurs par défaut (grille de taille 10 par 10, avec 40 proies et 10 prédateurs). Vous remplirez ici toute la grille avec de l'herbe puis positionnerez aléatoirement vos proies et prédateurs.

23. Ecrivez la procédure `draw_ecosysteme` qui affiche le contenu de la grille avec les images correspondantes dans une fenêtre *Grapic*.

24. Implémentez la version de base de la procédure `evolution_ecosysteme` et passez à chaque itération le nombre de proies et de prédateurs à une courbe `Plot` de *Grapic*.

25. Affichez la courbe d'évolution des deux populations et comparez-la à la courbe théorique obtenue grâce aux équations de Lotka-Volterra.

Modification des règles (pour aller plus loin…)

26. Rajoutez / modifiez les règles d'évolution afin de rendre le système plus réaliste. On pourra par exemple ajouter les règles suivantes.
    a. Les prédateurs peuvent mourir de faim (au-delà de `MAX_JEUNE_PREDATEUR` itérations de jeûne, par exemple 4), ou de vieillesse (au-delà de `DUREE_VIE_PREDATEUR` itérations, par exemple 10).
    b. Les proies peuvent également mourir de faim (si plus de nourriture à proximité).

27. Donnez à chaque individu la faculté de se déplacer dans son voisinage ; les proies en sens inverse de leurs prédateurs et les prédateurs en direction des proies.

28. Comparez les nouvelles courbes aux courbes théoriques.