INF1 - Projet

Lucioles

UNIVERSITÉ DE RENNES 1

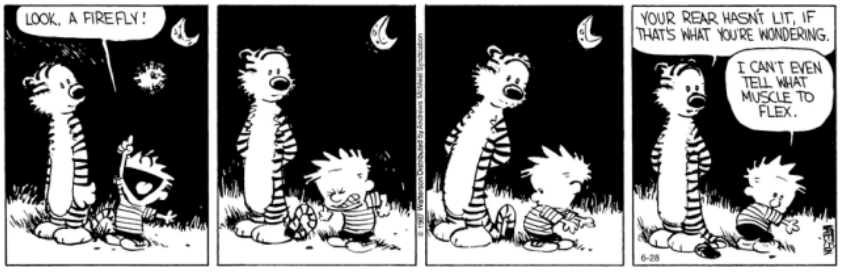

FIGURE 1 – Calvin and Hobbes par Bill Watterson

La famille des lucioles et des lampyres regroupe plus de 2 000 espèces connues de coléoptères produisant presque tous de la lumière. Comme l'explique Bénédicte Salthun-Lassalle, les lucioles mâles d'une même espèce synchronisent leurs signaux lumineux pour que les femelles les reconnaissent parmi les autres espèces [1] :

> En été, une luciole mâle qui peuple les prés et les pelouses émet de la lumière périodiquement. Quand une femelle lui répond, ce mâle entame un comportement nuptial et se dirige vers elle pour s'accoupler. Ce mâle est souvent seul. Mais ce n'est pas le cas chez toutes les espèces de lucioles. Les mâles de certaines espèces « clignotent » en groupe, et leurs signaux lumineux sont alors synchronisés. Andrew Moiseff et Jonathan Copeland, des universités du Connecticut et de Georgie, aux États-Unis, ont montré que ce comportement permet aux femelles de mieux les détecter et de trouver un partenaire de leur espèce.

Cette synchronisation est un comportement émergent qui se produit spontanément : les lucioles n'ont pas de fonction explicite de synchronisation avec leurs voisines, et cette synchronisation ne se produit que lorsque les lucioles sont suffisamment nombreuses. Pour autant, le(s) mécanisme(s) expliquant ce phénomène n'est pas encore bien connu.

Objectif du projet. *Le but de ce projet est de réaliser un simulateur pour une population de lucioles dans une prairie*, afin d'observer les possibles comportements collectifs émergents. Pour ce faire, on utilisera les notions présentées en INF1 : variables, affectation, conditionnelles, définitions de fonctions, tableaux, itération, et affichage.

L'important est de programmer le simulateur. Que vous parveniez ou non à obtenir une synchronisation de vos lucioles n'entrera pas en compte dans votre note. *Ne tombez pas dans le piège de perdre du temps à jouer avec les paramètres.*

1. `https://www.pourlascience.fr/sd/biologie-animale/le-clignotement-synchrone-des-lucioles-10635.php`

1 Déroulé du Projet et Travail à Rendre

Déroulé. On modélisera les lucioles, leur comportement, et leur façon d'intéragir de manière simplifiée. On vous indique la modélisation à réaliser à chaque étape du projet. Les étapes du projet sont progressives, et doivent être réalisées dans l'ordre :

1. Simulation d'une seule luciole. Voir Partie 3.
2. Définition d'une prairie de lucioles. Voir Partie 4.
3. Simulation d'une population de lucioles répartie dans une prairie. Les lucioles évoluent de manière indépendante, sans intéragir entre elles. Voir Partie 4.3.
4. Simulation d'une population de lucioles répartie dans une prairie. Les lucioles évoluent en intéragissant entre elles. Voir Partie 5.

Visualiser les lucioles dans une prairie pourra être fait de deux manières :
— affichage simple de la pairie à chaque pas de temps, en mode texte dans la console.
— animation graphique enregistrée dans un fichier `.gif`. Cet affichage n'est pas la priorité du projet : ne vous y intéressez que si vous avez terminé le reste. Nous vous fournissons des fonctions pour réaliser cet affichage animé.

Travail à rendre. Ce projet à réaliser en binôme : 2 étudiant·e·s, du même groupe de TP. Vous rendrez votre travail sous la forme d'un export de projet Eclipse, au format `.zip`. Pour exporter correctement, reportez-vous au tutoriel Eclipse, en ligne sur Moodle.

Date limite vendredi 16 décembre 2022, 20h. Tout retard sera pénalisé.

Chaque membre du binôme doit déposer un export. **Le nom du fichier doit indiquer les noms des deux membres** : `INF1_Projet_Lucioles_NOM1_NOM2.zip`.

Rendu Page Moodle INF1, devoir "Projet INF1".

Le travail à réaliser est expliqué dans la suite du document. Pour réaliser ce projet, vous pourrez définir toutes les fonctions intermédiaires que vous jugerez utiles. L'introduction de fonctions intermédiaires adéquates fait partie des éléments d'évaluation.

Bibliothèques fournies. Un ensemble de bibliothèques et de fonctions vous est fourni.
— Le fichier `RandomGen.java` vous fournit un générateur pseudo-aléatoire, que vous pourrez utiliser dans le projet pour générer des entiers, des nombres à virgules, ou des booléens de manière aléatoire. Le fichier `DemoAleatoire.java` contient des exemples illustrant son utilisation.
— La bibliothèque `outils.jar` fournit deux fonctions utiles à la génération d'images et la génération de `.gif` animés. Pour utiliser cette bibliothèque, pensez à l'importer dans les fichiers concernés, en spécifiant `import outils.* ;` en haut de fichier. La bibliothèque offre deux fonctions, que l'on vous présentera dans les parties concernées. Des exemples d'images BMP et des GIF animés sont fournis dans les répertoires `img` et `simu` du projet.

2 Configurer votre Environnement

1. Relisez le Tutoriel Eclipse pour les TP disponible dans Moodle ;
2. Importez le fichier `INF1_Projet_Lucioles.zip` ;
3. Les fichiers à compléter sont dans le répertoire `src` du projet.

3 Simulation d'une Luciole : `LucioleSeule.java`

Dans cette partie, on souhaite simuler la variation du niveau d'énergie d'une luciole d'un pas de temps à l'autre. On modélise une seule luciole, en isolation, de la manière suivante :

— un niveau d'énergie courant, de type `double` et positif ou nul. La luciole émet un flash lumineux pendant un pas de temps lorsque son niveau d'énergie dépasse un certain seuil. Après un flash, le niveau d'énergie de la luciole redescend à zéro au pas de temps suivant.

— un delta d'énergie, de type `double` et positif ou nul. Il détermine l'évolution du niveau d'énergie au cours du temps : à chaque pas de temps, le niveau d'énergie augmente du delta. Le delta est fixé pour une luciole, mais peut varier d'une luciole à l'autre.

Le fichier `LucioleSeule.java` définit une constante, appelée `SEUIL`, qui détermine le seuil d'énergie au delà duquel une luciole émet un flash. Vous utiliserez cette constante chaque fois que nécessaire. On supposera donc que le seuil est le même pour toutes les lucioles.

3.1 Modélisation et Affichage

Question 1. *Dans la fonction* `main`*, créer une variable* `lucioleEnergie` *de type* `double`*, et initialisée aléatoirement entre* `0.` *et* `100`*. Pour cela, vous utiliserez les fonctions offertes par le générateur aléatoire disponible dans* `RandomGen.java`*. Vérifier que vous n'obtenez pas uniquement des valeurs comprises entre* `0.` *et* `1.`

Question 2. *Créer une fonction* `symboliseLuciole` *(hors du* `main`*) qui prend en entrée un nombre* `niveauEnergie` *et qui renvoie le caractère* `'*'` *si* `niveauEnergie` *est suffisant pour émettre un flash, et* `'.'` *sinon.*

Question 3. *Créer une fonction* `afficheLuciole` *qui prend en entrée un nombre appelé* `niveauEnergie` *et un booléen appelé* `verbeux`*. La fonction affiche le symbole de luciole correspondant à* `niveauEnergie`*. Si* `verbeux` *est vrai, on affichera ensuite (sur la même ligne) le niveau d'énergie. Enfin, quelle que soit la valeur de* `verbeux`*, l'affichage passe à la ligne suivante.*

Question 4. *Dans la fonction* `main`*, modifier manuellement et plusieurs fois le niveau d'énergie de la luciole, et afficher la luciole après chaque modification.*

3.2 Évolution pour un Pas de Temps

Question 5. *Dans la fonction* `main`*, créer une variable* `lucioleDeltaEnergie` *de type* `double`*, et initialisée aléatoirement entre* `0.` *et* `1.`

Question 6. *Créer une fonction* `incrementeLuciole` *qui prend en entrée deux nombres de type* `double` *appelés* `niveauEnergie` *et* `deltaEnergie`*, et qui renvoie le niveau d'énergie au pas de temps suivant.*

Question 7. *Dans la fonction* `main`*, utiliser maintenant la fonction définie précédemment pour faire évoluer le niveau d'énergie d'une luciole sur plusieurs pas de temps. Conserver les affichages après chaque étape, et vérifier que l'affichage reste cohérent.*

3.3 Simulation d'une Seule Luciole

Question 8. *Créer une fonction* `simuleLucioleNbPas` *qui initialise aléatoirement une luciole, et fait évoluer cette luciole un certain nombre de fois (fourni en paramètre), en affichant la luciole à chaque pas de temps, en mode verbeux. En particulier, vérifier que votre luciole émet un flash au bon moment, et que son niveau d'énergie revient bien à 0 avant d'augmenter à nouveau. Vous testerez cette fonction dans la fonction* `main`*.*

Question 9. *Créer une fonction* `simuleLucioleNbFlashs` *qui initialise aléatoirement une luciole, et qui affiche cette luciole en mode verbeux, jusqu'à ce qu'elle émette son troisième flash (inclus). Vous testerez cette fonction dans la fonction* `main`*.*

4 Une Prairie de Lucioles : `Prairie.java`

Dans cette partie, on s'intéresse maintenant à une population de lucioles, répartie dans une prairie. Pour le moment, les lucioles n'interagissent pas entre elles, et évoluent à chaque pas de temps comme dans la partie précédente.

Chaque luciole est représentée par un tableau de 2 `doubles`. Le premier représente son niveau d'énergie courant. Le second représente le delta d'énergie. Pour clarifier l'accès à ces deux informations, on définit dans le fichier les constantes globales entières `ENERGIE` et `DELTA`. Vous les utiliserez dans la suite comme indices d'un tableau représentant une luciole. Cela permet à la fois de rendre le code plus clair, et d'éviter d'avoir à se souvenir de l'ordre dans lequel ces deux valeurs sont représentées.

On modélisera une population de lucioles par un tableau de lucioles. Chaque luciole de la population est donc identifiée par un numéro, qui est l'index auquel se trouve cette luciole, dans le tableau de la population.

4.1 Création d'une Population de Lucioles

Question 10. *Créer une fonction* `creerLuciole` *qui renvoie un tableau représentant une luciole dont le niveau d'énergie et le delta sont initialisés aléatoirement.*

Question 11. *Créer une fonction* `incrementeLuciole`*, qui étant donnée une luciole, la fait évoluer pour un pas de temps, comme dans la partie précédente. Attention, on demande de modifier la luciole, et pas d'en créer une nouvelle.*

Question 12. *Créer une fonction* `creerPopulation` *qui prend en argument un entier* `nbLucioles` *et qui renvoie un tableau de* `nbLucioles` *lucioles initialisées aléatoirement.*

Question 13. *Dans la fonction* `main`*, créer un exemple de population de lucioles.*

Vérifier que toutes les lucioles sont bien différentes : leur adresse (et non leur numéro dans la population !) est unique, et leurs niveaux d'énergie diffèrent d'une luciole à l'autre.

4.2 Création d'une Prairie

L'écosystème des lucioles est une prairie. On modélise une prairie par une grille de dimensions (*nbLignes*, *nbColonnes*), c'est-à-dire un tableau de *nbLignes* lignes, où chaque ligne est représentée par un tableau de *nbColonnes* entiers. Chaque case de la grille contient un entier :

— si la case n'est pas occupée par une luciole, l'entier est `-1`

— si une luciole occupe cette case, l'entier est son numéro dans la population

Évidemment, chaque case est occupée par au plus une luciole, et une luciole occupe au plus une case.

Question 14. *Créer une fonction* `prairieVide` *qui prend en paramètres deux entiers* `nbLignes` *et* `nbColonnes` *et qui renvoie une prairie dans laquelle il n'y a aucune luciole.*

Question 15. *Créer une fonction* `affichePrairie` *qui prend en paramètres une prairie et une population de lucioles, et qui en réalise l'affichage en mode texte dans la console :*

— *Une case inoccupée de la prairie est représentée par un espace.*

— *Une case occupée par une luciole qui n'émet pas de flash par un point.*

— *Une case occupée par une luciole qui émet un flash est représenté par une étoile.*

— *Les bordures de la prairie sont représentées par des* `#`*.*

Question 16. *Dans votre fonction* `main`*, tester les fonctions précédentes sur des exemples de prairies.*

Question 17. *Créer une fonction* `prairieLucioles` *qui prend en arguments deux entiers* `nbLignes` *et* `nbColonnes`*, ainsi qu'une population de lucioles, et qui renvoie une prairie aux bonnes dimensions, dans laquelle les lucioles de la population sont réparties aléatoirement. Veillez à ne pas placer deux lucioles au même endroit.*

On supposera que les dimensions de la prairie sont suffisantes pour répartir toutes les lucioles de la population.

Question 18. *Dans la fonction* `main`*, créer une nouvelle prairie, et répartissez y une population de lucioles de la bonne taille.*

4.3 Simulation Sans Interactions

On s'intéresse maintenant à la simulation d'une population de lucioles, répartie dans une prairie. On continue de supposer que les lucioles n'interagissent pas entre elles.

Question 19. *Créer une fonction* `simulationPrairie`*, qui simule l'évolution d'une population de lucioles répartie dans une prairie, pour un nombre de pas donnés. La fonction devra afficher, à chaque pas de temps, toute la prairie. Les lucioles évoluent indépendamment, mais évoluent toutes à chaque pas de temps.*

Question 20. *Dans la fonction* `main`*, tester votre simulation sur quelques exemples simples, et contrôler le bon fonctionnement de votre simulateur. Vérifiez que lorsqu'une luciole atteint le seuil d'énergie, elle reste allumée exactement un pas de temps.*

4.4 Simulation Animée Sans Interaction

Attention : ne traiter cette partie que si le reste est réalisé parfaitement !

On souhaite réaliser un autre type d'affichage pour la simulation. À chaque pas de temps, on souhaite générer une image graphique (au format BMP) pour représenter la prairie. Puis, lorsque tous les pas de temps ont été simulés, on souhaite combiner toutes ces images en une animation, dans un fichier GIF animé.

Pour cela, la bibliothèque `outils.jar` fournit deux fonctions, dont on donne la documentation ci-dessous. Vous n'avez pas à programmer ces fonctions ! Elles sont définies dans la bibliothèque `outils.jar` et vous pouvez donc les utiliser.

La première fonction, `BitMap.bmpEcritureFichier`, permet de générer un fichier d'image représentant une prairie peuplée de lucioles.

```
1  /**
2   * Enregistre sur disque, dans le fichier de nom nomFichier.bmp, une image
3   * BMP représentant une prairie de lucioles à un instant donné.
4   *
5   * @param nomFichier nom de fichier, relatif au projet Eclipse courant.
6   * @param prairie    une prairie, de dimensions non nulles
7   * @param population la population de lucioles répartie dans la prairie
8   * @param seuil      seuil d'énergie au delà duquel une luciole émet un
9   *                   flash lumineux
10  */
11 public static void BitMap.bmpEcritureFichier(String nomFichier,
12                              int[][] prairie, double[][] population, double seuil)
```

Pour une prairie de lucioles, le fond de l'image générée sera noir (on observe les lucioles la nuit !). Une luciole présente qui n'émet pas de flash est représentée par un pixel gris, et une luciole qui émet un flash est affichée en vert. Des exemples sont fournis dans le répertoire `img/` du projet.

La deuxième fonction, `GifCreator.construitGIF`, permet de combiner plusieurs images en un GIF animé.

```java
/**
 * Création d'un GIF animé dans un fichier nomFichier. Le nom nomFichier est
 * relatif au répertoire du projet courant. Il faut préciser l'extension de
 * fichier .gif dans nomFichier. Le .gif produit affiche les images dont les
 * noms sont dans fichiersImages, les unes après les autres.
 *
 * @param nomFichier     Nom de fichier dans lequel le GIF est produit
 * @param fichiersImages Noms des fichiers d'images composant le GIF produit
 */
public static void GifCreator.construitGIF(String nomFichier, String[] fichiersImages)
```

Un exemple de création de GIF animé est fourni dans le fichier `DemoGifAnime.java`.

Question 21. *Créer une fonction* `simulationPrairieGIF`*, qui pour un nombre de pas de temps donné, une population de lucioles, et une prairie donnés, réalise la simulation sous forme de GIF animé.*

Vous veillerez à générer les fichiers image BMP dans le répertoire `img` *du projet, et à générer les animations dans le répertoire* `simu` *du projet.*

Question 22. *Dans la fonction* `main`*, réaliser des simulations animées de quelques prairies de lucioles. Vérifier que l'animation est cohérente avec votre premier simulateur.*

5 Simulation avec Interactions : `PrairieInteractions.java`

On s'intéresse maintenant à modéliser un mécanisme d'interaction entre les lucioles. Lorsqu'une luciole émet un flash, elle stimule ses voisines en augmentant leur niveau d'énergie. On supposera que cet apport supplémentaire est le même pour toutes les lucioles, on définit donc une constante `APPORT`.

Une luciole ne peut stimuler que les lucioles qui sont suffisamment proches d'elle dans la prairie. La simulation doit donc considérer une notion de voisinage. Pour définir ce voisinage, on utilisera une constante entière `RAYON`, donnant la valeur du rayon de voisinage à considérer pour calculer les interactions possibles entre lucioles.

5.1 Voisinage d'une Luciole

La Figure 2 illustre la notion de voisinage d'une luciole pour un voisinage de rayon 2. Le voisinage d'une case doit rester dans les bornes de la prairie. Les cases situées près des bords de la prairie ont donc moins de cases voisines que les cases proches du centre. Ainsi, les lucioles en bord de prairie auront donc souvent moins de voisines que les autres. Par exemple, dans la Figure 2, la case $(26; 44)$ a 24 cases voisines occupées par 7 lucioles. Si l'on considère que la prairie s'arrête à la ligne 29, la case $(29; 45)$ occupée par la luciole n^{o}31 n'a que 14 cases voisines, occupées par 3 lucioles.

	...	41	42	43	44	45	46	47
...								
23								7
24				60		84		
25				12				
26					5			
27					63		57	
28			96	36				
29						31		

FIGURE 2 – Portion de la prairie autour de la case (26; 44) et son voisinage de rayon 2 (ici sur fond gris clair). Les cases du voisinage sont situées dans la zone allant de 2 lignes avant à 2 lignes après la ligne 26, et 2 colonnes avant à 2 colonnes après la colonne 44. La case (26 ; 44) contient la luciole n^{o}5. Pour simplifier l'affichage, les cases qui ne contiennent pas de luciole sont ici figurées comme vides. Les lucioles voisines de la luciole n^{o}5 ont les numéros 60, 84, 12, 63, 57, 96 et 36. Les lucioles n^{o} 7 et 31 ne sont pas dans le voisinage de la luciole n^{o}5.

5.2 Pré-calcul du Voisinage

Puisqu'on considère que les lucioles ne se déplacent pas, il est inutile de re-calculer à chaque pas de temps et pour chaque luciole, quelles sont ses voisines. On peut pré-calculer le voisinage de chacune des lucioles de la population, dans un tableau de tableaux d'entiers.

Pour une population de taille N lucioles, le tableau de voisinage est aussi de taille N. Le i^{e} élément du tableau de voisinage contient un tableau (éventuellement vide pour les lucioles isolées) contenant les numéros de lucioles voisines de la i^{e} luciole de la population.

Pour l'exemple de la Figure 2, le 6^{e} élément du tableau de voisinage indique le voisinage de la luciole n^{o}5, c'est-à-dire le tableau {`60, 84, 12, 63, 57, 96, 36`}. L'ordre des numéros de lucioles dans le tableau de voisinage d'une luciole n'est pas significatif.

Question 23. *Créez une fonction* `voisinage`*, qui prend en argument une prairie, et qui renvoie le tableau de voisinage pour cette prairie.*

Vous décomposerez le problème à l'aide de fonctions intermédiaires, pour calculer le nombre de lucioles présentes dans la prairie, le voisinage d'une case donnée de la prairie, ou pour déterminer combien de lucioles sont dans le voisinage d'une autre... Indiquez en commentaire le rôle de chacune de ces fonctions.

5.3 Évolution d'une Luciole pour un Pas de Temps

On rappelle le mécanisme modélisé dans cette partie : une luciole émettant un flash stimule ses voisines en leur apportant un gain supplémentaire d'énergie. Par exemple, pour la Figure 2, où le rayon de voisinage vaut 2, si la luciole n°60 émet un flash, elle doit stimuler les lucioles n°5, 12 et 84. Le gain supplémentaire d'énergie est transmis aux voisines au pas suivant du flash.

Pour la Figure 2, si on suppose un delta d'énegie de 1 pour toutes les lucioles, et un apport d'énergie par flash égal à 5, on obtient les niveaux d'énergie suivants :

Luciole n°	7	60	84	12	5	63	57	96	36	31
Pas t	96	98	98	97	97	96	96	12	12	5
Pas t+1	97	99	99	98	98	97	97	13	13	6
Pas t+2	98	100	100	99	99	98	98	14	14	7
Pas t+3	104	5	5	110	110	99	99	15	15	8
Pas t+4	0	16	21	5	5	110	105	21	21	9

Question 24. *Créer une fonction* `incrementeLuciole` *qui prend en paramètre une population de lucioles, un voisinage, ainsi qu'une luciole et son numéro, et qui modifie la luciole en fonction de son voisinage.*

Pour ne pas bouleverser de manière imprévisible le mécanisme d'interaction entre lucioles, vous aurez besoin d'utiliser une copie de la population de lucioles pour déterminer les niveaux d'énergie au pas suivant.

Contrôler le bon fonctionnement de votre fonction en la testant dans la fonction `main` *sur des exemples de petites prairies. Par exemple, essayer de reproduire l'exemple de la Figure 2.*

5.4 Simulation de la Prairie

On souhaite maintenant définir le simulateur de lucioles en tenant compte de leurs interactions. On réutilisera la fonction programmée en question précédente.

Pour cette simulation encore, vous aurez besoin d'utiliser une copie de la population de lucioles pour ne pas bouleverser de manière imprévisible les interactions entre lucioles.

Question 25. *Créer une fonction de simulation de prairie qui, pour un nombre de pas donné, simule l'évolution de la prairie en tenant compte des interactions de lucioles.*

Vous pourrez en proposer deux versions, réalisant respectivement un affichage simplifé, et un affichage animé sous forme de fichier GIF.

Contrairement à la Partie 4.3, on s'attend à pouvoir observer une synchronisation, si les lucioles sont suffisamment proches les unes des autres.

Question 26. *Créer des fonctions de création de prairie adéquates, qui permettent d'observer des synchronisation de lucioles dans la prairie. Expliquer et justifier votre démarche en commentaires dans le fichier.*

Sauvegarder les meilleures simulations que vous aurez obtenues dans le projet, et indiquez les en commentaires dans la fonction `main`*.*