Alizée Lanon
Elora Vigo

Compte rendu du projet en C

I/ Avancée du projet

Nous avons réalisé le niveau 1 et le niveau 2 de ce projet. Nous avons essayé le bonus 1 mais pas le 2 par manque de temps. De plus, le programme a été pensé pour fonctionner seulement, c'est-à-dire qu'avec plus de temps nous aurions pu essayer de l'optimiser.
Lors de la compilation de notre programme sous Linux, deux warning de type "déclaration implicite de la fonction "affichage sur demande" s'affiche. Sous Mac ce warning se transforme en erreur dû à des normes sur la protection de la mémoire plus importantes. Nous n'avons pas réussi à corriger ce warning malgré nos essais (de nombreuses vérifications de la liaison du fichier main au fichier Affichage). Pour compiler sur Mac nous sommes donc passées par Docker, ce qui a permis de faire compiler le programme sous Mac avec seulement le warning (et pas l'erreur). Ce docker a fonctionné, malheureusement pour une raison inconnue (pas ou peu de modifications faites entre-temps) il ne fonctionne plus actuellement. Une erreur de segmentation est présente (problème d'indice de tableau probablement). Pourtant le code dans le docker est celui sur git, et en se plaçant sur une des branches (hors docker) le programme n'a pas présenté l'erreur malgré les nombreuses exécutions. Ainsi cela ne semble pas être une erreur de code direct. C'est la première fois que nous réalisons un docker, l'erreur peut donc s'expliquer par notre inexpérience dans le domaine. Mais cela paraît étrange que le problème n'ait pas été présent avant.

II/ Organisation des fichiers

Structure.h : création des structures et appel des librairies, ce fichier est nécessaire à la compilation de tous les autres.

fonctionBase.c et fonctionBase.h : ensemble des fonctions utiles pour plusieurs des fichiers suivants, nécessaire à la compilation de tous les .c.

Initialisation.c et Initialisation.h : permet l'initialisation du potager (tomate, puceron et coccinelle).

ListeCaseFiltre.c et ListeCaseFiltre.h : permet de créer un tableau de position adjacente à une case, et de filtrer ce tableau, nécéssaire pour les fonctions de reproduction et mouvement (fichier SimulationTour).

SimulationTour.c et SimulationTour.h : permet de faire les différentes étapes de simulation du tour (reproduction, mouvement,...).

Affichage.c et Affichage.h : permet l'affichage en console du potager.

MainProjet.c : fonction main, nécessite tous les autres fichiers.

makefile : permet la compilation du main (make all), la suppression des fichiers objet (make clean) et de l'exécutable (make mrpropre).

III/ Choix de programmation

1/ Organisation générale du programme

a) Les structures

Nous avons fait le choix de créer cinq structures, ‘coord’ pour les coordonnées des cases du tableau, ‘insecte’ qui sera une coccinelle ou un puceron selon le besoin. Nous n’avons pas trouvé l’utilité de créer deux structures différentes pour les deux types d’insectes, car les deux ont les mêmes objets.
Il y a également la structure ‘tomate’, la structure ‘potager’ composée des informations sur les tomates, les pucerons et les coccinelles, et enfin la structure caseMvt. L’utilisation de cette dernière sera expliquée dans la suite du rapport.

b) Les fonctions de simulations du tour

Chaque action réalisée par un groupe d’entité (les pucerons, les coccinelles ou les tomates) se découpent toutes de la même manière: une première fonction qui applique l’action à un seul élément du groupe et une seconde qui applique cette action à tous les éléments. Par exemple, on retrouve dans la simulation d’un tour des fonctions pour la maturation des tomates (Maturation1Tomate et MaturationTouteTomate), pour les pucerons et coccinelles qui se nourrissent, les mouvements des entités, la reproduction et le vieillissement.

c) Le main

1) Initialisation par l’utilisateur
On demande à l’utilisateur le nombre de tours de simulation qu’il veut réaliser, et à quel tour il veut introduire les coccinelles. Il y a également l’option de choisir quel type d’affichage en console l’utilisateur veut. Il y en a 3 possibles, un affichage du potager à tous les tours, affichage en fonction de la réponse de l’utilisateur à tous les tours (pour afficher petit à petit) ou aucun affichage.
2) Initialisation du potager
On initialise ensuite le potager avec toutes ses composantes: on le remplit dans un premier temps de tomates mûres, puis de pucerons, et si l’utilisateur a choisi de rajouter les coccinelles au tout début, on les introduit également.
La fonction “TousCocciMange” apparaît souvent dans la main pour éviter d’avoir un puceron et une coccinelle sur une même case. Si un puceron est présent sur une case où une coccinelle arrive, cette dernière le mange et devient le seul insecte présent dans la case.
3) Simulation d’un tour
Dans la boucle simulant les tours, on retrouve toutes les fonctions permettant sa simulation. Nous faisons maturer les tomates, bouger les pucerons, bouger les coccinelles, manger les coccinelles, manger les pucerons, reproduire les pucerons (et manger les coccinelles au cas où un nouveau puceron est arrivé sur une case avec une coccinelle), vieillir les pucerons, reproduire les coccinelles (et encore une fois les faire manger) et faire vieillir les coccinelles.
4) Affichage
On affiche ensuite l’état d’un tour, c'est-à-dire le nombre de pucerons et coccinelles vivant à la fin du tour, on affiche ensuite le potager selon l’option choisie au début. Nous avons également choisi d'ajouter une fonction qui affiche le nombre de tomates mûres à la fin de chaque tour pour faciliter la réponse aux questions du sujet.

5) Optimisation possible

Un exemple d'optimisation de notre programme aurait pu être une seconde boucle dans la boucle de simulation d'un tour qui réaliserait toutes les actions pour une entité avant de passer à l'entité suivante. Dans notre cas, nous avons effectué une seule action pour toutes les entités, avant de passer à l'action suivante, ce qui entraîne beaucoup de boucles (non imbriquées) pour parcourir le même tableau de nombreuses fois. Nous n'avons pas pu aller au bout de l'optimisation ici par manque de temps.

2/ Déplacement

Le mouvement d'un puceron sera gardé en mémoire de deux manières, une visuelle (<,>,Λ,v,\,/) et une permettant la distinction entre la diagonale NO et SE, et la diagonale NE de la SO. La seconde manière a donc 8 valeurs possibles, elle prend les valeurs de 0 à 7. Le schéma suivant symbolise la correspondance entre les deux.

\	∧	/	0	1	2
<	X	>	3	X	4
/	v	\	5	6	7

Pour les déplacements nous avons pris l'option de faire un potager sans bordure (bonus 1). Si un puceron est sur le bord gauche du potager et qu'il se dirige à gauche, il arrivera à droite de celui-ci. De même, si un puceron est dans le coin inférieur droit et qu'il se dirige en diagonale vers le bas, il arrivera dans le coin supérieur gauche. Pour coder cela il faut utiliser les 'modulo'. Soit N la longueur et largeur de notre potager, appliquer le modulo N à notre position permet de ne pas dépasser à droite du tableau, puisque l'on récupère le reste de la division par N. On remarque par contre que l'on peut toujours dépasser à gauche du tableau (indice négatif), pour passer en valeurs positives on ajoute donc N à nos coordonnées.

exemple : Si le puceron est en position (x,y), pour aller vers la gauche on soustrait 1 à y (y est la colonne, et x la ligne). Pour récupérer la nouvelle position 'y' sans sortir du tableau on fait donc (y-1 + N)%N.

On remarque que cette méthode permet de ne pas créer de bordure de potager dans le cas où les mouvements sont haut/bas/droite/gauche. Mais lorsque ceux-ci vont en diagonales, le puceron ne réapparaît pas sur la même diagonale de l'autre côté, c'est-à-dire que le potager n'est pas vraiment une sphère mais juste sans bordure. Nous n'avons pas réussi à coder cela, comme c'était assez compliqué (de nombreuses disjonctions de cas) nous avons laissé les mouvements ainsi.

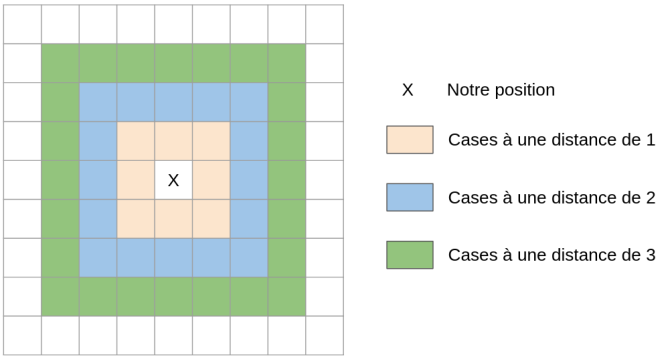
3/ Orientation et mouvement

L'énoncé nous indique de bouger tous les pucerons, puis de les réorienter pour le prochain tour. Nous avons modifié cet ordre pour commencer par orienter le puceron 1, puis le déplacer, réorienter le puceron 2, puis le déplacer… Pour cela nous créons une liste des positions des cases attenantes au puceron. On applique ensuite sur cette liste des filtres permettant d'enlever les cases ne respectant pas certaines conditions.

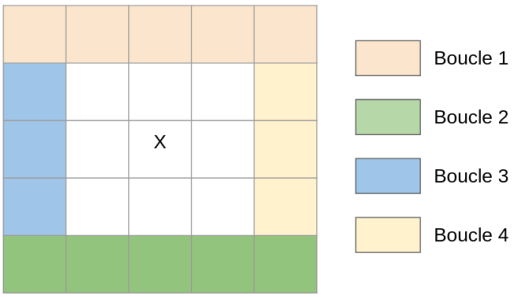
a) Création de la liste des cases attenantes

La fonction 'listaNCase' du fichier 'ListeCaseFiltre' permet la création d'un tableau avec les positions des cases se situant strictement à n case de notre position. En effet, la coccinelle a une vision jusqu'à trois cases, mais on voudra d'abord regarder s'il y a des cases intéressantes à une

case, si ce n'est pas le cas on va regarder celles à deux cases etc. Dans ce cas, pour ne pas étudier les positions à une case plusieurs fois, on regarde seulement celle à une distance égale à n. Le schéma ci-dessous nous indique quelles cases sont renvoyées en fonction de n.



Pour compléter ce tableau de cases, on réalise quatre boucles pour récupérer les coordonnées de la ligne supérieure (boucle 1), ligne inférieure (boucle 2), colonne de gauche (boucle 3) et colonne de droite (boucle 4) (cf figure ci-dessous). L'exemple ci-dessous est celui pour une distance de 2.



On remarque que la boucle 1 et 2 réaliserons 2*n+1 tour (avec n la distance à notre position), et 2*(n-1)+1 pour la 3 et 4.
De plus, pour connaître le mouvement qui sera à réaliser pour arriver jusqu'à la case, on ajoute l'information dans notre tableau. C'est-à-dire que chaque élément de notre tableau de case contiendra ses coordonnées ainsi que son premier mouvement associé pour l'atteindre (structure nommée caseMvt). Si la case d'arrivée est dans la même ligne ou colonne que notre position, le mouvement associé est celui qui y va directement (<,>,∧,v), sinon on se déplace en diagonale. Le schéma ci-dessous résume les directions associées à chaque case (d'une distance de 1 à 3) sous format visuel, dans le programme elles sont encodées par les chiffres de 0 à 7 (voir plus haut). Ainsi, si la coccinelle a vu un puceron sur la première case du schéma ci-dessous, son mouvement sera d'aller en diagonale SO.

\	\	\	∧	/	/	/
\	\	\	∧	/	/	/
\	\	\	∧	/	/	/
<	<	<	X	>	>	>
/	/	/	v	\	\	\
/	/	/	v	\	\	\
/	/	/	v	\	\	\

b) Filtres appliqués

Reproduction d'un puceron

Pour la reproduction d'un puceron, on veut placer le nouvel individu sur une case attenante sans puceron. Si aucune case attenante n'est disponible, il n'y a pas de reproduction. En effet, du point de vue du puceron ils sont nombreux et ils n'est donc pas nécessaire de se reproduire pour la survie de l'espèce. De plus, cela veut aussi dire qu'il n'y aura pas forcément assez de nourriture pour son descendant. Les coccinelles ne sont pas prises en compte pour cette étape. C'est-à-dire que même entouré de coccinelle le puceron se reproduit (pour la survie de l'espèce) et le descendant ira sur une case avec son prédateur. Si une partie des cases attenantes n'ont pas de coccinelle, le descendant peut tout de même atterrir sur une case avec coccinelle puisque celui-ci est encore trop jeune pour être un minimum stratégique.

Reproduction d'une coccinelle

Pour la reproduction d'une coccinelle on applique le même principe, le descendant ira sur une case attenante sans coccinelle, si il n'y a pas de place on ne fait pas la reproduction.

Mouvement d'un puceron

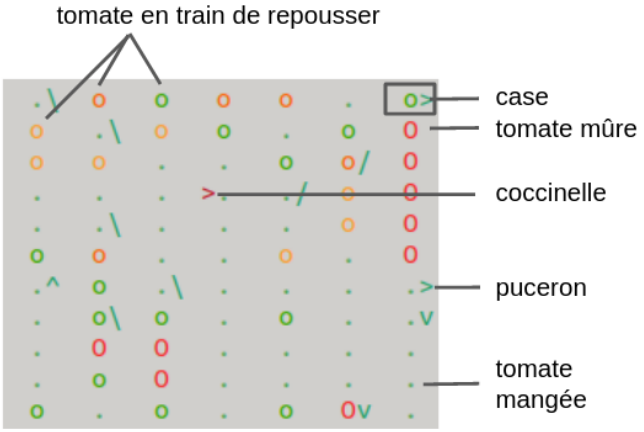
Pour le mouvement d'un puceron on commence par vérifier qu'il n'y a pas de tomate sur sa case. Si ce n'est pas le cas, le puceron continue son mouvement s'il trouve une tomate sur la case d'arrivée. Si la case n'a pas non plus de tomate, il va sur une case attenantes au hasard sans puceron et avec tomate. De plus, pour le niveau 2 on ajoute la condition de ne pas avoir de coccinelle sur la case choisie. S'il est entouré de coccinelle et/ou de pucerons, il ne bouge pas. Parmis les cases restantes libres il ira d'abord sur une case avec tomate, si ce n'est pas le cas il ira sur une case sans tomate. Nous n'avons pas appliqué le fait de le mettre sur une case du potager au hasard car cela lui permettrait de ne pratiquement jamais se faire manger par une coccinelle. En effet, si celui-ci peut aller sur n'importe quelle case du potager si celles attenantes ne lui conviennent pas, il peut échapper plus facilement au coccinelle.

Mouvement d'une coccinelle

Pour le mouvement de la coccinelle, elle ne peut pas aller sur une case avec coccinelle. Si elle est entourée de coccinelle, elle ne bouge pas. Sur les cases attenantes restantes libres elle ira sur une case avec puceron si c'est possible. S'il n'y a aucun puceron, elle va regarder les cases avec puceron à une distance de 2. Si il y a un puceron mais qu'une coccinelle se trouve sur la case qui permet d'aller vers ce puceron : elle ne peut pas y aller (le chemin est bloqué). On vérifie cela en comparant la liste des cases à une distance de 2 avec puceron (cible) et la liste des cases où on peut aller (case à une distance de 1 sans puceron). Si un mouvement est présent dans la liste des cases avec pucerons (liste de structure caseMvt), alors cela signifie que si on avance dans la direction du mouvement on se rapproche d'un puceron mais on arrive sur une case occupée par une coccinelle. Si un mouvement est présent dans les deux listes, cela signifie que le mouvement nous rapproche d'un puceron et que la case pour l'atteindre est libre. Ainsi on récupère la liste des cases avec pucerons à une distance 2 qui sont atteignables. Si possible on va sur une de ces cases au hasard, s'il n'y en a pas on fait de même avec une distance de 3. Enfin dans le cas où la coccinelle n'a toujours pas pu trouver une case qui la rapproche d'un puceron atteignable, elle va au hasard sur une case sans coccinelle attenante.

4/ Affichage

L'affichage des tomates, pucerons et coccinelles est réalisé avec les formes et couleurs indiquées dans l'énoncé. Une case du potager est centrée sur la tomate (de vert à rouge), à sa droite on peut avoir un puceron (vert), et à sa gauche on peut avoir une coccinelle (rouge). La figure ci-dessous est un exemple d'un état du potager, la case entourée contient une tomate en train de repousser avec un puceron. Ce cas de figure est possible lorsqu'il y a reproduction, le nouvel individu n'a pas encore mangé.



IV/ Difficultées rencontrées

- Une des difficultés majeures a été de trouver comment choisir une case attenante au hasard qui respecte les conditions voulues. En effet nous aurions pu faire tourner le programme jusqu'à ce qu'une case aléatoire qui respecte les conditions soit trouvée, mais cela n'était pas optimal. C'est pour cela que l'application de filtre sur la liste de cases possibles est faite. Nous n'avons pas appliqué cette méthode pour placer initialement les pucerons et coccinelles dans le potager, nous les avons placés de manière aléatoire après vérification de la disponibilité de la case. Nous supposons que réaliser la liste des 900 cases et y appliquer un filtre n'est pas le plus optimal sachant qu'on place peu d'éléments (et donc peu de chance d'avoir une case occupée).

- Même après avoir trouver un moyen de choisir une case aléatoire convenable, les fonctions de mouvements étaient compliquées à réaliser.

- Un point important était de bien comprendre l'utilisation des pointeurs sur les structures. Ce n'était pas très compliqué mais il fallait toujours se poser la question de ce qu'il fallait mettre en argument lors de l'appel de fonction (adresse ou valeur) et comment le faire (&, *, & et *, ou rien).

- Un point qui nous a fait perdre un peu de temps, et qui peut complexifier un peu le code, est le fait de modifier une structure ou une fonction après l'avoir potentiellement déjà appelé dans les scripts. Il faut ensuite modifier le reste du code en fonction de la modification. Par exemple, lorsqu'on voulait tuer un puceron, nous avions besoin d'une fonction qui échange la position du puceron mort avec le dernier puceron du tableau (pour ne pas laisser de case vide). C'est à ce moment qu'on s'est rendu compte qu'ajouter un identifiant au puceron était nécessaire. On a donc dû modifier certaines fonctions pour ajouter un identifiant. Nous avions déjà réalisé des fonctions prenant en

argument notre potager (en pointeur) et un puceron du potager (pointeur sur une partie de la structure potager). Nous ne les avons pas changées car cela aurait demandé beaucoup de temps et aurait pu ajouter des erreurs. Les changements à faire auraient été de ne donner en argument qu'un pointeur sur potager et l'identifiant du puceron.

V/ Réponse aux questions

Question 1

En faisant tourner le code 5 fois, on obtient que le tour moyen à partir duquel il ne reste plus que 25% de tomates mûres et le tour 18.8. Au bout de 10 tours il y a 59,6 pucerons en moyenne (sur 5 essais). Au bout de 50 tours il y a 187.8 pucerons en moyenne (sur 5 essais). Au bout de 100 tours, on a une moyenne de 184 pucerons. On remarque donc qu'au bout de 100 tours on a le même nombre de pucerons qu'au bout de 50 tours, on atteint un plateau. C'est le maximum de pucerons que peut contenir le potager en termes ressources alimentaires (en termes de place cela serait 900).

Question 2

Les coccinelles n'arrivent pas à tuer tous les pucerons, elles disparaissent toutes avant. On obtient donc à la fin de 50 ou 100 tours la même chose que précédemment. Elles disparaissent en moyenne au tour 32 (8 essais), c'est-à-dire qu'on a toujours qu'une seule génération de coccinelle (pas de reproduction) dans la grande majorité des cas. De temps en temps une coccinelle arrive à se reproduire, elle disparaît alors 20 tours plus tard (vers le tour 40). Cela signifie donc que les coccinelles ne mangent pratiquement aucun pucerons, car pour se reproduire elles doivent manger 3 fois dans leur 20 tours de vie. Cela peut s'expliquer par les mouvements des pucerons, ceux-ci s'éloignent des coccinelles (on filtre le tableau de cases possibles pour ne pas aller sur une case avec coccinelle). Les coccinelles ont beau courir derrière les pucerons, elles ne les rattrapent pas (ou peu).

Bonus

Les pucerons survivent déjà, il n'y a rien à faire. Pour répondre à la question inverse (comment tuer les pucerons) on peut augmenter le nombre de cases pouvant être parcouru par une coccinelle dans son tour. Elle a une vision de 3 cases mais si un puceron est à une distance de 3 elle va seulement dans sa direction. Le tour suivant, il va bouger, et potentiellement s'éloigner. Ainsi, si les coccinelles peuvent voir et se déplacer à 2 ou 3 cases, elles peuvent cette fois-ci les attraper. Car dans notre cas, la vision à 3 cases n'a pas l'air de les avantager, elles sont donc moins capables de se nourrir que les pucerons. Une autre possibilité serait donc de diminuer la nourriture des pucerons, ils pourraient ne manger que les tomates mûres par exemple (tomate 'O', pas 'o').