Projet d'Automne : Me too ?

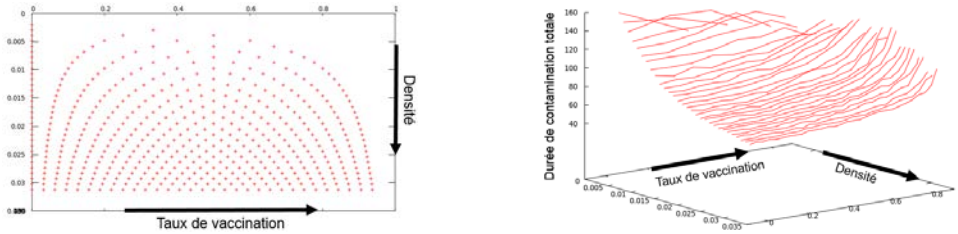

Figure 1: deux illustrations de la sortie obtenue avec gnuplot (section 3.7) pour le fichier test12.txt (sauf nbSim: 1000). A gauche : ensemble des contextes (densité, taux de vaccination) ; à droite : les durées de contamination simulées.

1. Introduction

Le projet se propose d'évaluer l'influence du taux de vaccination sur la durée de contamination d'une population mobile dans un monde très simplifié. La visualisation des résultats de simulation se fera avec le logiciel open source gnuplot comme dans la Fig 1.

Le projet se décompose en deux étapes, chacune ayant un rendu noté (Fig 2). Le **premier rendu** (semaine 8) a pour but d'évaluer l'organisation visuelle d'un code complexe comportant des structures de contrôle. Pour le **rendu final** (semaine 12) il faudra partiellement ré-écrire ce code mais surtout le compléter à l'aide de fonctions en mettant en oeuvre les *principes d'abstraction et de ré-utilisation*.

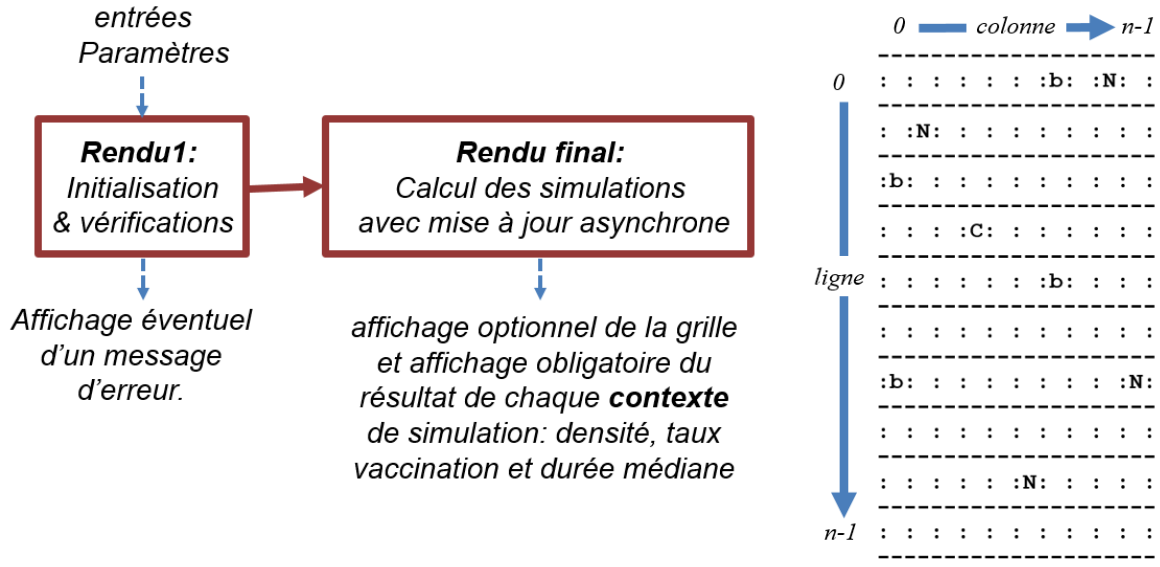

Fig. 2 : gauche : décomposition du projet en 2 rendus
droite : affichage de la grille pour les données initiales décrites en 2.1 ; la lettre **C** désigne l'unique personne initialement *contaminée***,** les autres étant non-vaccinées (lettre **N**). Le but de chaque personne est indiqué par la lettre minuscule **b** (éventuellement masquée par une personne). On utilise la lettre **V** pour désigner une personne vaccinée (aucune dans cette illustration).

2. Spécifications

Une **exécution** du projet va conduire à une seule lecture des paramètres (rendu1) suivie par un grand nombre de simulations structurées comme suit:

- au plus haut niveau, on veut explorer **l'ensemble des contextes** obtenus en faisant varier la *densité* et le *taux de vaccination* pour le jeu de données fourni en entrée (détails en 2.5). On appelle *contexte de référence* l'association de la *densité* initiale et d'un *taux de vaccination* initial nul. Chaque contexte va servir à estimer une seule *durée de contamination*.

- au niveau intermédiaire d'évaluation d'un seul **contexte *(densité, taux de vaccination)***: le hasard joue un rôle important dans l'évolution d'une simulation. C'est pourquoi, chaque contexte effectue **nbSim** simulations distinctes à partir des mêmes conditions initiales (détails en 2.4).

- au niveau simulation : une simulation se déroule sur plusieurs **cycles** de mise à jour. Chaque cycle applique les règles qui modélisent le déplacement et la contamination de **nbP** personnes dans un monde qui est un espace sans obstacles représenté par une grille de **n** x **n** cases (Fig 2 droite)**.** Chaque personne occupe **une case** de la grille (pas plus d'une personne par case). Chaque personne doit se diriger vers un but individuel selon le plus court chemin. Une fois atteint ce but, un nouveau but est déterminé aléatoirement. La contamination se fait par contact.

2.1 Ordre et format des données en entrées:

Ces informations sont fournies dans la phase d'initialisation dans l'ordre suivant :

- Paramètre ***verbose*** d'affichage de texte (détails en 3.3 et 3.5)
- Affichage optionnel de la grille : 0 = pas d'affichage, autre valeur = affichage (2.3.4)
- le nombre de simulations **nbSim** (nbSim > 0) pour estimer une durée de simulation
- La taille **n** d'un côté de la grille carrée représentant le monde (n > 1)
- Le nombre de personnes **nbP** (**1 < nbP ≤ n^2**)
- pour chaque personne, sur une ligne :
  - **position** (**i_ligne, j_colonne**) et celle de son **but** (**i_ligne, j_colonne**) compatibles avec la représentation d'un tableau en langage C, avec les indices compris entre **0** et **n-1**.

Par défaut, on considère que c'est toujours la première personne qui est contaminée tandis que toutes les autres sont non-vaccinées.

données	*Signification de la donnée*
0	ne pas afficher de message d'invitation (3.5 et 3.7)
1	montrer la grille comme dans la Fig2_droite (2.3.4 et 3.7)
1	faire une seule simulation (= **nbSim**)
10	le monde est une grille carrée de 10 cases de côté (= **n**)
5	nombre de personnes dans le monde (= **nbP**)
3 3 4 6	La première personne de la liste est celle qui est **contaminée**.
1 1 2 0	
0 8 0 6	Une seule personne par ligne avec sa position (ligne, colonne)
6 9 6 0	puis son but (ligne, colonne)
8 5 8 5	

Fig. 3 : colonne de gauche : les données correspondant à la Fig 2_droite. Les fichiers de test seront organisés de cette manière. Il est indispensable de respecter cet ordre.

Le rendu1 consiste à lire et vérifier que les données respectent les conditions indiquées plus haut. La section 4.1 décrit comment procéder lorsqu'une erreur est détectée.

La suite de la section 2 commence par décrire le niveau simulation (2.2 et 2.3) puis le niveau intermédiaire (2.4) et le plus haut niveau (2.5).

2.2 Modélisation du déplacement et de la contamination: Chaque personne a un but à atteindre dans la grille et avance progressivement vers ce but, en se déplaçant seulement d'une case à la fois. Elle peut bouger dans une des 8 cases contiguës.
2.2.1 Rebouclement du monde sur lui-même : les côtés droit et gauche de la grille se touchent, tout comme le haut et le bas de la grille. Cette propriété devra être exploitée pour ***se déplacer selon le plus court chemin vers son but***. Par exemple, dans la Fig 2, la personne initialement en (6,9) se trouve en fait juste à côté de son but de position (6,0) grâce au rebouclement des côtés droit et gauche. Il lui suffit de se déplacer d'une seule case vers la droite pour l'atteindre.

2.2.2 Gestion de blocage : Si une personne est bloquée par une autre qui se trouve sur son « plus court chemin » alors elle peut choisir temporairement une des 7 autres cases à sa disposition même si elle ne respecte pas le critère du plus court chemin. Si aucune case n'est disponible, alors elle choisit un nouveau but et reste immobile pour ce cycle de mise à jour.

2.2.3 Choix du but : Si une personne se trouve sur son but elle se donne un autre but qui doit être différent de l'endroit où elle est actuellement. Ce choix est effectué aléatoirement (section 3.6). Il n'y a pas de problème si cette destination est actuellement occupée ou si elle est aussi choisie par une autre personne.

2.2.4 Modélisation de la contamination : une personne contaminée n'est pas consciente de son état et se comporte comme les autres personnes. Si une personne non-vaccinée se trouve au contact d'une personne contaminée, c'est-à-dire dans l'une des *8 cases voisines*, alors elle devient elle-même contaminée. Il faut prendre en compte le rebouclement du monde sur lui-même (2.2.1).

2.2.5 Incubation *: une personne qui est contaminée pendant le cycle de mise à jour courant ne devient contagieuse qu'au cycle de mise à jour suivant.*

2.3 Structure d'une simulation
2.3.1 Vérification de contamination initiale (rendu final): il se peut que certaines personnes soient en contact initial. Il faut donc effectuer une étape de vérification de contamination *avant le lancement de la première simulation*.

2.3.2 Pseudocode d'une simulation avec mise à jour asynchrone:

```
Tant_que la condition d'arrêt n'est pas remplie
    Affichage optionnel de la grille
    // mise à jour asynchrone
    Pour chaque personne
        Analyser l'état actuel (but, voisinage)
        Mise à jour éventuelle du but
        Déplacement éventuel
        Test de contamination
    Fin_Pour

    Evaluation de la condition d'arrêt
Fin_Tant_que

Affichage optionnel de la grille
Mémorisation de la durée de contamination totale
```

2.3.3 Condition d'arrêt : la simulation se termine lorsque toutes les personnes initialement non-vaccinées ont été contaminées ou si le nombre de cycles atteint la valeur de **MAX_CYCLES.** Le résultat d'une simulation se traduit par une ***durée de contamination*** (le nombre de cycles de mise à jour au moment de l'arrêt) qui est mémorisée.

2.3.4 Mise à jour **asynchrone :** dès qu'on a déplacé une personne, on vérifie s'il y a contamination des personnes voisines, même si celles-ci n'ont pas encore effectué leur déplacement. Et ainsi de suite. Un cycle de mise à jour se termine lorsque toutes les personnes ont été traitées (boucle « Pour chaque personne » dans le pseudocode).

2.4 Résultat pour un contexte donné (densité, taux de vaccination)

Le résultat d'une simulation est dépendant du hasard qui décide du choix des buts. Ce résultat pouvant varier beaucoup pour les mêmes conditions initiales de *densité* et de *taux de vaccination*, on décide de calculer et mémoriser le résultat de **nbSim** simulations. Pour un contexte donné, on mémorise ainsi **nbSim** valeurs de durée de contamination. Ensuite on retient seulement la valeur médiane des **nbSim** durées pour disposer d'une information moins sensible aux valeurs extrêmes que leur moyenne.

Pour obtenir la valeur médiane, il faut trier le tableau des **nbSim** durées jusqu'à l'indice du milieu du tableau qui fournit la valeur médiane recherchée. Le résultat pour un contexte donné est l'affichage dans le terminal des trois valeurs suivantes, séparées par un espace :

densité taux_de_vaccination valeur_médiane_des_nbSim_durées_de_contamination

2.5 Exploration des contextes possibles

On désire explorer systématiquement les **contextes** possibles avec les données fournies en entrée, à l'aide d'une double boucle (Fig 4) :

- Boucle externe sur la *densité* : le nombre de personnes **nbP'** est initialisé à **nbP** et décroit de 1 à chaque passage, jusqu'à un minimum de 2 (axe violet vers la droite dans Fig. 4; **on enlève en commençant par la fin de la liste fournie en initialisation**).
- Boucle interne sur le *taux de vaccination* : le taux est initialisé à zéro et augmente à chaque passage en ajoutant une personne (axe orange vers le bas dans Fig 4; **on vaccine en commençant immédiatement après la personne contaminée par défaut**). Le taux de vaccination maximum est toujours **(nbP'-2)/nbP'** car la première personne est déjà contaminée et on conserve une dernière personne non-vaccinée à contaminer.

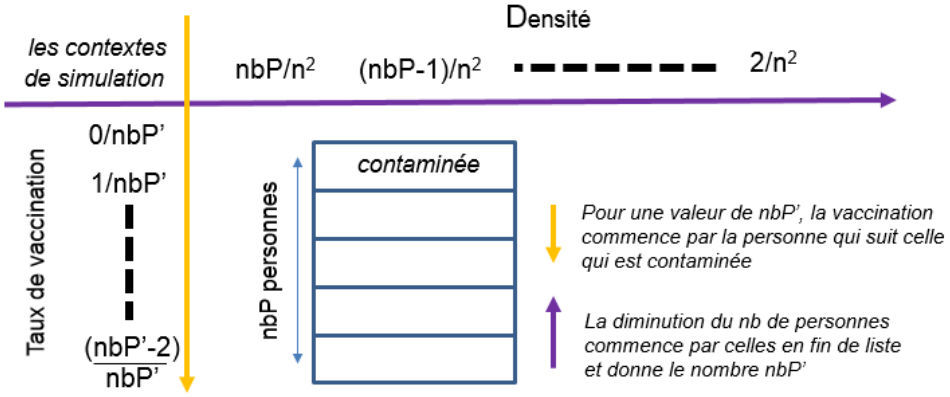

Fig. 4 : Le nombre initial de personnes nbP sert à définir la densité de référence pour laquelle on étudie nbP taux de vaccinations (entre 0/nbP et (nbP-2)/nbP). Ensuite on diminue le nombre de personnes en ignorant la dernière de la liste ; cela nous donne une nouvelle quantité nbP' qui sert à définir le nombre de taux de vaccination à étudier. Et ainsi de suite jusqu'à 2 personnes.

3. Mise en œuvre en langage C

Nous mettons à disposition ***un fichier à compléter*** ; il contient une fonction **main()** et des ***fonctions qu'ils faudra utiliser pour l'affichage des messages d'erreurs*** (Table 1).

3.1 Clarté et structuration du code avec des fonctions

La clarté de votre code est toujours prise en compte dans le barème de tous les rendus.
Etant donné l'avancement du cours, le premier rendu peut être écrit simplement en complétant la fonction **main()**. Les étudiants maîtrisant déjà les fonctions peuvent en utiliser pour structurer leur premier rendu (4.1) mais en travaillant avec un seul fichier. Dans tous les cas, l'affichage des messages d'erreurs doit seulement être fait avec les fonctions fournies, sans les modifier.

3.2 Mémorisation et structuration des données

Etant donné l'avancement du cours, l'outil recommandé pour mémoriser les données est le tableau de taille variable (appelé **VLA** pour *Variable Length Array*) à une ou plusieurs entrées. Il convient parfaitement pour transmettre, et éventuellement modifier, des données dans des fonctions. Exemples d'utilisation de tableau VLA :

- Les données des **nbP** personnes
- Tout autre tableau qui vous semblera utile, par exemple pour des paramètres

On utilisera **enum** avec profit pour créer les symboles permettant d'accéder à chaque élément de tels tableaux au lieu d'avoir recours à des *magic numbers* pour les valeurs d'indices.

Seuls les étudiants qui maîtrisent déjà les notions de structures, pointeurs et l'allocation dynamique de mémoire peuvent s'en servir s'ils le désirent.

3.3 Variables locales ou globales ?

Toutes les variables ou tableaux utilisés pour ce projet seront déclarés ***localement*** et transmis en paramètre aux fonctions, seulement lorsque c'est nécessaire. La seule exception admise et recommandée en tant que variable globale est le booléen **`verbose`** (détails 3.5).

3.4 Redirection des entrées-sorties et fichiers de tests

Les informations fournies au clavier peuvent être rassemblées dans un fichier test qui est redirigé sur l'entrée standard. Ce mécanisme simple de *redirection* est vu en cours en semaine 5. Nous mettons ainsi à disposition un ensemble de *fichiers tests*.

Vous pouvez également rediriger la sortie standard vers un fichier texte pour facilement mettre au point votre projet et/ou disposer d'un fichier de données pour gnuplot.

Concernant la **syntaxe** des fichiers de test **: il est possible d'avoir des séparateurs supplémentaires entre les données, en fin de ligne ou sur des lignes supplémentaires (espace, passage à la ligne, etc).** On utilisera scanf() pour ne pas être perturbé par ces séparateurs supplémentaires.

3.5 Rôle du booléen `verbose`

Le booléen **`verbose`** sert à contrôler l'affichage des messages d'invitation dans le terminal (les messages d'erreurs sont TOUJOURS affichés). En effet il est plus pratique pour l'utilisateur que le programme affiche un texte pour l'informer des données à fournir. Par contre cet affichage n'est pas désirable quand nous voulons noter vos programmes avec la redirection des entrées-sorties. C'est pourquoi cette variable booléenne **`verbose`** est initialisée avec la

première valeur fournie au programme (pas de message d'invitation pour demander cette valeur); cette valeur est visible au début des fichiers tests :

- La valeur 0 (FAUX) interdit l'affichage des messages d'invitations.
- Toute autre valeur sera interprétée comme une autorisation d'affichage des messages

Tous les fichiers tests contiennent la valeur 0. Cependant, si on la remplace par 1 votre programme doit montrer des messages d'invitation comme dans le programme de démo.

3.6 Mise en œuvre du hasard pour le choix du but

La fonction **rand()** renvoie un nombre aléatoire (entier positif) entre **0** et **RAND_MAX** (cette constante est définie dans **stdlib.h**). Tous les entiers de cet intervalle ont la même chance d'être renvoyés par **rand()**. Pour obtenir une valeur aléatoire entière d'**indice** dans l'intervalle [**0, n-1**], il suffit de renormaliser la valeur fournie par **rand()** comme suit :

indice = ((double) rand()/RANDMAX)*n et d'exclure le cas où on obtient **n**.

3.7 Format pour la sortie destinée à la visualisation avec gnuplot

Si on veut obtenir une visualisation du même type que la Figure 1 avec **gnuplot**, l'affichage optionnel de la grille doit être désactivé. De plus, du fait de la redirection de l'affichage vers un fichier, tous les messages d'invitation doivent commencer par le caractère **#** pour être considérés comme des commentaires par gnuplot. Seul ce qui est décrit en 2.5 est nécessaire pour gnuplot. Supposons que le fichier de donnée obtenu par redirection (3.4) s'appelle **xyz.dat**. Tout d'abord, dans le terminal écrire la commande : **gnuplot**
Ensuite, dans gnuplot, écrire la commande :

splot "xyz.dat" using 1:2:3 with line

4. Rendus :

Au semestre d'automne nous fournissons des fichiers tests pour les 2 rendus du projet pour vous sensibiliser à l'étendue minimale des tests nécessaires pour valider un programme. En effet il ne suffit pas qu'un programme compile pour qu'il soit correct… Construire un ensemble de scénarios de tests d'un programme, pour lesquels on connait la solution, est très important pour valider votre programme vous-mêmes.

4.1 Rendu intermédiaire (semaine 7): seulement le TEST DES VALEURS DES PARAMETRES

L'accent est mis sur le *respect des* **conventions de programmations** (noms de variable, symbole…) et sur *l'organisation visuelle du code* : indentation, alignement, clarté générale. Notre document sur les conventions à respecter vous donne aussi la signification des codes que nous utiliserons pour vous donner un feedback (ex : E14 indique comment écrire un nom de variable composé de plusieurs mots). Le but « technique » de ce rendu est de *vérifier si les paramètres indiqués dans la Table1 sont corrects* pour les fichiers test 1 à 6. Si une des erreurs de la Table1 est détectée, il faut appeler la fonction indiquée (elle est fournie sous forme de code source). Cette fonction provoque la fin du programme.

Barème indicatif (10 pt)

(4pt) clarté du listing (respect de la police de caractère, taille, mise en page, lisibilité, indentation, ligne de code sans wrapping = 87 caractères max avec geany)
(3pt) respect des conventions de programmation du cours (nom des variables, etc…)
(3pt) votre programme s'exécute correctement pour les fichiers de test 1 à 6.

Fichier testX.txt	*Correct* / Incorrect	Fonctions fournies / *commentaire*
1	Incorrect	`erreur_nbSim(int nbSim)` */ n'est pas strictement positif*
2	Incorrect	`erreur_taille_monde(int n)` */ n'est pas strictement supérieur à 1*
3	Incorrect	`erreur_nbP(int nbP, int n)` */ doit être >1 et inférieur ou égal à n*n*
4	Incorrect	`erreur_indice_ligne_colonne(int indice, int indicePersonne)` `indice` *est la* ***première valeur incorrecte*** *d'indice de ligne ou colonne qui n'est pas comprise dans l'intervalle* **[0, n-1]** *pour la position ou le but d'une personne.* `indicePersonne` *désigne une personne. Sa valeur commence à* **0** *pour la première personne fournie en entrée et varie jusqu'à* **nbP-1** *pour la dernière personne fournie en entrée*
5	Incorrect	`erreur_superposition(int indiceP_A, int indiceP_B)` `indiceP_A` *et* `indiceP_B` *sont les indices des deux Personnes superposées. leur valeur commence à* **0** *pour la première personne fournie en entrée et varie jusqu'à* **nbP-1** *pour la dernière personne fournie en entrée.*
6	*Correct*	c'est un fichier correct qui ne produit aucun message dans le terminal pour le rendu1. Voir aussi **Rappel 1**.

Table 1 : fichiers tests avec les erreurs à détecter pour les fichiers incorrects.

IMPORTANT : lorsque **verbose** est FAUX (section 3.5), il ne faut PAS afficher d'autres messages que ce qui est produit par les fonctions indiquées dans cette table car cela nous empêche d'automatiser la correction, et si nous devons intervenir manuellement pour noter votre rendu, cela vous fait perdre des points.

Rappel1 (3.4): tous les fichiers de test peuvent contenir des séparateurs supplémentaires. Cela ne doit pas faire échouer l'étape de lecture de votre programme.

Remarque : Pour les personnes ayant déjà des bases, nous autorisons votre code à avoir de l'avance mais 1) cela ne rapporte aucun points, 2) le code sera noté concernant le style, les conventions, et sa lisibilité, 3) le programme du rendu1 ne doit **PAS** afficher plus que ce qui est demandé pour le rendu1.

Il faut fournir :

- Le code source de votre projet doit être téléchargé sur moodle avec l'outil visible en semaine 7 du cours, au plus tard **le jeudi 2 novembre à minuit**.
    - **Le nom du fichier** doit être votre ***numéro SCIPER*** suivi de l'extension « ***.c*** ».
        - Par exemple M Meylan ayant pour numéro SCIPER 274628 va télécharger un fichier nommé 274628.c

- Vous devez aussi fournir les éléments **« papier »** suivants **avec votre nom/prénom/SCIPER**, au plus tard en INJ 141 (boites à droite de la porte) le 3 novembre à 12h00: **code source imprimé** avec une mise en page **PORTRAIT**, les **numéros de ligne** comme avec geany, la police de caractères utilisée par défaut par geany, ou **Courrier New** de **taille 10**. Votre objectif est de ne pas avoir de passages à la ligne parasites (=ajouté par l'impression si la ligne de code contient plus de 87 caractères = *wrapping*) ni de fin de ligne coupée. Vous devez conserver les **4 espaces** par défaut pour chaque indentation. Vous pouvez vérifier votre mise en page sans l'imprimer grâce à la commande **Print Preview** de **Geany** (sélectionner d'abord **Print** dans le menu **File**).Nous vous rendrons ce document avec des indications sur des aspects principaux à améliorer pour le rendu final.

4.2 Rendu final (semaine 12):

Nous évaluerons surtout l'usage des ***fonctions*** pour mettre en oeuvre les *principes d'abstraction et de ré-utilisation*.

barème (25 p)

(4pt) clarté du code source dans geany (largeur PORTRAIT, lisibilité, indentation, mise en page, ligne de code sans wrapping = 87 caractères max avec geany, en particulier, idéalement la taille d'une fonction ne doit pas dépasser une page écran de geany mais nous relaxons cette contrainte jusqu'à la taille d'une page imprimée avec geany, c'est-à-dire *57* lignes)
(3pt) respect des conventions de programmation du cours (nom des variables, etc…)
(4pt) décomposition du code : **mise en œuvre des principes d'abstraction et de ré-utilisation.**

(1pt) votre programme s'exécute correctement avec les fichiers 1 à 6
(6pt) votre programme s'exécute correctement pour les fichiers 7 à 12
(2pt) votre programme s'exécute correctement pour 2 fichiers corrects mais non fournis

Fichier testX.txt	*commentaire*
7	contamination immédiate, avant simulation
8	contamination immédiate, avant simulation à cause du rebouclement
9	contamination après 1 cycle de mouvement, un seul contexte, nbSim :1
10	Monde 6 x 6, nbP:2 -> un seul contexte, nbSim: 3 pour calcul de la valeur médiane
11	Monde 10 x 10, nbP: 3, nbSim: 1, pour avoir plusieurs contextes
12	Monde 32 x 32, nbP: 32, nbSim:100, peut prendre 2-3 minutes de calculs

Table 2 : fichiers corrects de tests supplémentaires pour évaluer le rendu final.

Rapport :
(3pt) analyse du résultat
(2pt) dessin du graphe des appels de fonctions

Il faut fournir :

- Le code source de votre projet doit être téléchargé sur la database moodle avec l'outil visible en semaine 12 du cours, au plus tard **le jeudi 7 décembre à minuit.**
    - **Le nom du fichier** doit être votre ***numéro SCIPER*** suivi de l'extension « ***.c*** ».

- Vous devez aussi fournir à votre assistant de TP une version imprimée du **RAPPORT**, au plus tard en TP le vendredi 8 décembre à 10h00:

Le Rapport est nominal ; il ne contient PAS de page de titre, ni de table des matières
Le Rapport contient :

Résultat de la phase d'analyse (max 1 page, soyez concis) :
Décrire l'organisation générale du programme en faisant ressortir la mise en oeuvre des principes *d'abstraction* et de *ré-utilisation*.

Donner le pseudocode de votre **algorithme** pour **l'évaluation des contextes (2.5)** ; utiliser une notation d'appel de fonction avec paramètres pour l'évaluation d'un contexte individuel (2.4). En faisant l'énorme supposition que le coût calcul d'un contexte (2.4) ait une durée constante, quel est l'ordre de complexité de votre algorithme pour l'évaluation des contextes (2.5) en fonction de nbP ?

Fournir une capture d'écran de l'image produite par gnuplot pour le fichier de test **test12.txt.** Votre simulation parvient-elle à reproduire l'influence de la densité et du taux de vaccination sur la durée médiane de contamination comme dans la figure 1 ?

Dessin du *graphe des appels de fonctions* : ne PAS inclure les fonctions de la bibliothèque standard (relire les notes de cours chap 6). Une liste d'outils est visible ici : http://en.wikipedia.org/wiki/Call_graph.

Annexe 1: contenu du fichier DEMO17.zip

1 fichier source à compléter pour votre projet : projet17.c
Ce nom de fichier vous permet d'utiliser le fichier script ci-dessous pour tester votre programme pour le premier rendu. **Ce nom devra être changé avec le numéro de SCIPER pour le téléchargement sur moodle** comme expliqué en 4.1 et 4.2.

2 exécutables de démo compilés et fonctionnant sur la machine virtuelle du cours. C'est le moyen le plus efficace de vérifier un aspect de la donnée qui ne vous semble pas clair : exécutez d'abord le programme de démo car c'est lui qui fait foi pour les rendus :

- **inter_demo17.x** réalise seulement la vérification des données (rendu intermédiaire)
- **final_demo17.x** est le programme de démo complet

12 fichiers tests :

- De **test1.txt** à **test6.txt** pour le rendu intermédiaire (**inter_demo17.x**)
- De **test1.txt** à **test12.txt** pour le rendu final (**final_demo17.x**)

1 fichier script : test_projet17_inter.sh Il s'agit d'un fichier texte écrit dans le langage de commande du système d'exploitation. Avec ces commandes il est possible d'automatiser les tests comme vous pouvez le constater en l'ouvrant dans geany. Si vous voulez utiliser le script il ne faut *pas modifier pas la structure du dossier ni les noms de fichier*. Le script s'exécute comme un programme s'il a le droit en *exécution* (ajouter au besoin) :

./test_projet17_inter.sh

Annexe 2: Projet individuel et méthode de travail

Dernier rappel : le projet d'automne est INDIVIDUEL. Selon les recommandations du SAC, le détecteur de plagiat sera utilisé pour comparer les projets entre eux et avec ce qui est visible sur internet.
Faites attention quand vous travaillez à plusieurs pour vous entre-aider en cas de bug (c'est OK) ; le détecteur de plagiat trouve régulièrement des projets de personnes qui, à force de discuter entre amis, finissent par avoir exactement la même structure de solution (c'est pas normal parce que cela démontre un travail d'équipe).