École de technologie supérieure (été 2025) Hugues Saulnier, Eric Thé

INF-147 Travail Pratique #1

Remise à venir sur moodle

Propagation d'un virus grippal (partie 1)

Travail en équipe

1 Objectifs

Utiliser des types structurés, autant pour un objet que pour une collection d'objets dans une simulation numérique.

Expérimenter sérieusement le développement modulaire d'un logiciel de simulation

Tout le développement voulu dans ce TP1 et le suivant doit se faire en C, toute forme de C++ est strictement interdite.

2 Mise en situation générale

Obtenir des données sur une population est souvent difficile, voire impossible. Recenser et observer une population de papillons zébrés sud-africains ou de manchots sur les iles australes reste encore difficile sinon impossible. De façon différente on peut à partir de conditions initiales assumées, résoudre le comportement d'un écosystème fermé avec un modèle mathématique capable de refléter la dynamique du système biologique (*essentiellement avec des systèmes d'équations différentielles, ++math 265++*). Finalement, on peut utiliser un automate défini par une liste finie de règles simples et ainsi approcher le comportement du modèle théorique dans une simulation numérique. C'est ce que nous allons réaliser.

3 Description du problème obtenir un modèle simplifié de l'action d'un virus grippal

Concevoir et tester un logiciel qui simule la propagation d'un virus grippal dans une population limitée. Le contexte initial de simulation est un quartier fermé *(personne n'y entre et personne n'en sort)*
Toutes les personnes présentes demeurent dans l'environnement de simulation (`hauteur` et `largeur` définies dans le plan cartésien), les personnes se déplacent, se rapprochent et possiblement propagent la maladie. Voici les règles primaires de l'automate

1. Dans notre modèle, le temps est simulé par un passage dans la boucle principale de simulation, chaque passage simule un cycle d'une heure dans la vie de la population

2. La **population** se divise en 3 états mutuellement exclusifs { sain , malade, mort } définis par trois constantes du logiciel. Au départ de la simulation, tous sont en santé *(une fonction dédiée y choisira plus tard le patient0)*

3. La **propagation,** les personnes se déplacent et peuvent entrer en contact avec d'autres. *Lors d'un contact (défini par la distance qui les sépare) d'une personne malade et d'une en santé, on teste l'infection possible avec la probabilité à s'infecter de la personne en santé. La personne en santé contractera ou non la maladie.*

4. Les personnes rétablis d'une infection peuvent recontracter la maladie *mais avec une probabilité nettement plus faible de fois en fois. La probabilité initiale est constante pour tous, la probabilité actuelle d'infection d'une personne se définit avec (probabilité initiale) / (constante ^(nombre d'infections))*

5. La **mortalité,** dès qu'une personne s'infecte, on initialise un compteur à 0 chez le nouveau malade qui va s'incrémenter à chaque passage dans la boucle principale de simulation. Ce compteur en atteignant la constante `NB_HRS_MALADIE`, va déclencher une fonction spécialisée qui détermine le « sort » du malade.

6. La fonction du point précédent détermine la mort ou la survie du malade*. La probabilité de mort dépend de l'âge de la personne et du nombre d'infections vécues. Un tableau de probabilités constantes à la première infection*

vous est fourni dans le module m_personne Cette probabilité décroit avec le nombre d'infections, la probabilité s'actualise avec ((probabilité initiale) / (nombre d'infections))

7. Le **confinement,** avec un taux constant de confinement souhaité, une partie de la population initiale est mise **symboliquement** en quarantaine, elle aura une faible probabilité de se déplacer. Un taux de confinement de 75% veut dire que 75% de la population restera peu mobile mais que le reste de la population le sera bien plus. *Notez que mobile ou pas, toute personne en santé risque de contracter la maladie si une personne malade entre en contact avec elle*.

8. La probabilité de se déplacer d'une `t_personne` est aléatoire et sera donnée à chaque `t_personne` lors de son initialisation.

9. Les actions séquentielles d'un cycle dans la boucle principale de simulation sont décrites très sommairement ici :
   - incrémenter les heures de maladie des malades et déterminer le sort de ceux qui ont écoulé toute la période de maladie *(pour mourir ou se rétablir)*
   - Déplacer les personnes
   - Identifier les contacts entre deux personnes avec chaque fois la possible propagation de la maladie aux personnes saines *(le couple sain, malade est toujours à surveiller)*

4 Trois modules cette semaine

Cette première semaine ne fait que mettre en place des modules essentiels à la simulation numérique, les premiers résultats obtenus des tests ne seront en rien représentatifs d'une pandémie. C'est au cours des prochaines semaines de développements et de tests que votre modèle va s'affiner.

Un premier module utilitaire, je vous ai donné un module complet `m_R3` des points et vecteurs de l'espace ordinaire avec toutes les fonctions nécessaires à un bon fonctionnement *(vous avez même les macros-fonctions)*. Avant tout autre développement, chaque équipe va dériver de `m_R3` son propre module `m_R2` qui servira à positionner et à déplacer les personnes. *Votre module m_R2 correct et testé est prioritaire à tout autre développement.*

On demande maintenant la réalisation de deux modules hiérarchisés spécifiques à la simulation. Le premier complètement et le second partiellement cette semaine

Un quartier rectangulaire du plan cartésien est défini par une `largeur` et une `hauteur` en mètres que vous donnerez dans votre main de test, cette région du plan contient toute la population durant la simulation. Chaque personne aura une position et une vitesse représentée par un vecteur position et un vecteur vitesse de R2 *(comme la tradition l'exige)*

Le module `m_personne` de gestion d'une personne sera réalisé au complet mais on vous en offre déjà une interface cohérente sous forme initiale. Votre équipe **réalise** son implémentation et **teste** toutes les fonctions dans un `main` *imaginons-le dans un fichier* `pandemie.c`

Une fois le module `m_personne` bien testé viendra la réalisation primaire du module `m_liste_personnes` qui va nous donner un premier container bien nécessaire de nos `t_personnes` dans la simulation. Vous n'en ferez cette semaine que la création et le remplissage initial + une action pandémique, l'introduction du premier malade *(dit patient0)*

5 Un mandat absolu

Tous les modules de l'ensemble de cette simulation doivent suivre le **principe d'encapsulation** des propriétés. C'est dire ainsi que **l'accès aux champs d'une structure doit être fait uniquement à l'intérieur des fonctions fournies par son module**. Il est donc impératif de créer les fonctions diverses *(constructeur / informatrices-accesseurs / mutatrices)* qui permettront d'interagir civilement avec le type structuré.

Je tiens à en faire la remarque tout de suite : Ne pas respecter le principe d'encapsulation vous fera automatiquement perdre le tiers de la valeur totale du TP, c'est tellement énorme que vous vous y tiendrez.

Module « m_personne »

Le module `m_personne` permet d'interroger ou de modifier une ou deux `t_personne` L'interface du module vous est fournie et vous devez développer son implémentation un nouveau fichier `m_personne.c` qui va contenir les définitions de toutes les fonctions déclarées dans l'interface.

Voici quelques précisions sur son interface

Le type énuméré `t_etat` définit les 3 états possibles d'une personne. Vous devrez utiliser ces constantes énumérées d'état dans votre code `{SAIN, MALADE, MORT}`

Le type structuré `t_personne` définit une personne, il est impératif de bien comprendre le sens donné à chacun de ses membres, champs, propriétés 😊

Certaines fonctions du module sont décrites plus en détail ici mais le commentaire dans l'interface est vraiment poussé et ce qui est ici ne peut vous nuire. Lisez avec attention tout le commentaire dans l'interface qui souvent suffit amplement.

Fonction `init_personne` reçoit les dimensions de la région et la proportion de confinement désiré dans [0.0, 1.0] alors :

- Initialiser les champs qui dépendent des constantes
- obtenir aléatoirement les deux coordonnées du vecteur position, px dans [0.0, largeur] et py … 😊
- initialiser son état SAIN
- initialiser son âge avec `rand_age_canada`
- La probabilité de mouvement sera définie aléatoirement en 2 temps. Si `(randf()<` proportion de confinement), elle sera basse aléatoire dans [0, PROB_BASSE_MAX] sinon elle sera haute dans [PROB_HAUTE_MIN, 1.0]
- Pour déterminer aléatoirement les composantes initiales de la vitesse, vous avez le module `m_r2` mais si vous voulez le faire à la main, voilà une recette : générer un angle aléatoire (en radians) entre [0.5, 1.5] , obtenir la vitesse en abscisse avec « 2 * cos(angle) » et en ordonnée avec « 2 * sin(angle) » , appliquer aléatoirement un changement de signe (*= -1) à chacune des 2 composantes de la vitesse (avec prob. 0.5 pour chacune d'être modifiée).

La fonction retourne la nouvelle personne

Fonction `determiner_mort_ou_retabli` reçoit une personne, si elle est `MALADE` et que ses heures de maladie atteignent ou dépassent NB_HRS_MALADIE, elle devra changer d'état pour `SAIN` ou `MORT`. Tester si `randf()` < sa probabilité de décès alors il meurt ou il se rétablit.

Fonction `deplacer_personne` reçoit une personne et les dimensions de la région, elle additionne au vecteur position de la personne son vecteur vitesse. Si la position de la personne déborde la hauteur ou de la largeur, il faut inverser le signe de la composante du vecteur vitesse responsable du débordement

Fonction `inverser_vitesses` reçoit 2 personnes pour leur donner des vecteurs vitesse dans deux directions opposées. Obtenir un premier vecteur vitesse aléatoire `(V)` pour une personne puis donner le vecteur opposé `(V * -1)` à la seconde personne.

Module d'une liste des personnes

C'est maintenant à vous de commencer le développement complet du module `m_liste_personnes (.h et .c)` Le module définit une structure publique qui contient minimalement un tableau dynamique de `t_personne` avec différents compteurs associés

Type de cette liste de personnes

```
typedef struct{
  t_personne * liste ;            //tableau dynamique de t_personnes
  int  taille;                    //la taille du tableau précédent

  int  nb_personnes;               //nombre de personnes dans la liste
  int  nb_malades;                //le nombre de malades dans la liste
  int  nb_sante;                  //le nombre de personnes en santé dans la liste
  int  nb_morts;                  //le nombre de morts dans la liste
} t_liste_personnes;
```

À vous de déclarer et de définir les premières fonctions publiques du module

En tout premier, la fonction qui va **construire** une `t_liste_personnes` vide. Elle reçoit en paramètre la taille du tableau dynamique voulu. Elle réalise l'allocation dynamique pour initialiser les membres `liste` et `taille` tous les autres champs sont fixés à 0. La fonction retourne la nouvelle `t_liste_personnes`

Maintenant, toutes les autres fonctions du module auront une référence sur une `t_liste_personnes` en premier paramètre et c'est juste normal.

La fonction `ajouter_des_personnes` est terriblement importante, elle reçoit cinq paramètres, la référence à la liste puis le nombre de t_personnes à ajouter, la largeur et la hauteur de la région, la proportion de personnes en confinement. Just do it avec une boucle d' init_personne 😊 et ajuster les compteurs. Elle retourne le nombre de personnes ajoutées à la liste

Comme dans le module `m_personne,` ajouter les informatrices `get_` assez évidentes ici

La fonction `creer_patient_zero` choisit aléatoirement une `t_personne` saine dans la `t_liste_personnes` qui doit être sans malade, elle change l'état de la personne choisie, ajuste les compteurs et retourne 1, sinon elle retourne 0 peu importe la raison.

Comme pour le module `m_personne`, il serait pas mauvais de créer une fonction d'affichage de l'état de la liste pour débugger parfois plus différemment qu'avec les `breakpoint et assert`

Voilà pour la première semaine

Bon travail