École de technologie supérieure Hugues Saulnier, Eric Thé

INF-147 Travail Pratique #2 Remise: Semaine 10

Simulation numérique d'une pandémie Travail en équipes

1 Objectifs

Il y a des situations ou obtenir des données est très difficile, voir impossible. Imaginer de recenser, au cours des années, la population de papillons zébrés sud-africains ou la quantité de mangoustes asiatiques. Pour évaluer de telles populations, on peut à partir de conditions initiales, simuler le comportement d'un écosystème fermé à l'aide d'un automate. Ou bien on peut utiliser certains modèles mathématiques capables de suivre la dynamique d'un système biologique. Par contre, avec des règles simples on peut simuler le modèle mathématique par un automate ou une « simulation numérique ».

2 Description du problème : Simulation de la propagation du virus aviaire

Écrire le programme d'un automate qui simulera un modèle de propagation d'une épidémie du virus aviaire. Le contexte est un petit village fermé (personne n'y entre et personne n'y sort). Les éléments sont les gens de l'ensemble de la population, divisés en 3 catégories : les gens en santé, les malades, et les morts. Tous ensemble dans l'environnement de simulation, ils se déplacent, se rencontrent et possiblement propagent ainsi la maladie. Voici les règles de base de notre système :

- La PROPAGATION : Les gens qui se déplacent aléatoirement entrent en contact avec d'autres gens. Suite à un tel contact, si une des deux personnes est malade et que l'autre est en santé (mais non-rétabli) on génère une probabilité de transmission de la maladie et selon cette probabilité la personne en santé contractera la maladie.
- Le CONFINEMENT : Selon un taux de quarantaine déterminé d'avance, certains gens seront confinés à domicile et donc ne se déplaceront pas dans l'environnement. Un taux de quarantaine de 70% veut dire que 70% des gens restent immobiles. Notez que ces gens peuvent quand même contracter la maladie si une personne malade et non-confinée entre en contact avec eux. Une personne malade non-confinée représente une personne malade mais SANS symptômes ou encore pire, une personne avec symptômes qui ignore les directives de quarantaine.
- Les gens rétablis peuvent contracter la maladie de nouveau (avec une probabilité décroissante).
- La MORT : Suite à la contraction de la maladie on déclenche un compteur du nombre de jours écoulé depuis que cette personne est malade. Après un nombre déterminé de jours (eg. 15 jours) on détermine le « sort » de la personne : soit elle devient rétablie ou soit elle meure. La probabilité de mourir est déterminée selon l'âge de la personne (les aînés sont plus susceptibles de mourir de la maladie). Lorsqu'une personne devient « rétablie » elle redevient non-confinée.
- Le MUR de confinement : Nous allons aussi introduire la simulation d'un « mur » (une barrière physique) de confinement. Le mur sera installé de manière à isoler la moitié des gens de l'autre moitié et la maladie sera initialement propagée d'un seul côté du mur. Ensuite, après un nombre déterminé de jours (eg. 60) on enlève le mur et on observera la propagation dans l'autre moitié.

Le passage du temps sera remplacé simplement par une boucle de commandes séquentielles qu'on appelle un cycle : on déplace les gens non-confinés, on identifie les contacts entre deux personnes avec propagation possible de la maladie, on détermine si les gens qui ont écoulé les 15 jours de maladie meurent ou non et on recommence. Dans notre modèle, chaque cycle représente une heure de temps.

Votre programme devra offrir deux modes d'opération : un mode avec l'affichage de tous les gens qui se déplacent (pour 900 personnes max.) et un mode sans affichages pour permettre une exécution plus rapide (jusqu'à 1500 personnes). Avec le deuxième mode, les données quotidiennes seront inscrites dans un fichier-texte de sortie (un « *logfile* ») pour permettre ensuite d'analyser la progression de l'épidémie. Faites exécuter l'exécutable fourni avec cet énoncé pour voir des exemples de ces 2 modes.

3 Modules à faire

On demande la réalisation de 3 modules : soit le module de gestion d'une personne, le module de l'ensemble des gens et le module de gestion du « mur » de confinement. Les trois modules doivent suivre le principe d'encapsulation de données. C'est à dire que l'accès aux champs d'une structure d'un module doit être fait à partir des fonctions fournies par ce module. Il est donc impératif de créer les fonctions diverses (accesseurs/mutateurs) qui permettront d'interagir avec ce type structuré. Le principe de l'encapsulation des données vous sera expliqué plus en détails en classe.

Module « personne »

Le module « personne » permet de gérer une personne. Utilisez les déclarations données dans le fichier « **personne.h** ». Vous devez compléter le module en écrivant les définitions de toutes les fonctions déclarées pour le fichier d'implémentation « **personne.cpp** ».

Voici quelques précisions sur ces déclarations :

Le type énuméré « **t_etat** » :

Défini les 3 états possibles d'une personne. Vous devez absolument utiliser les 3 constantes énumérées ici partout dans votre code.

Le type structuré « **t_personne** » :

Défini une personne. La position et la vitesse de déplacement (vecteur 2 dimensions) sont des réels et doivent toujours rester des réels. Ce n'est uniquement lorsqu'on dessinera la personne à l'écran (avec un petit cercle) qu'on va temporairement arrondir sa position aux coordonnées entières les plus proches.

Fonction « **init_personne()** » :

Cette fonction reçoit la position de départ ainsi que la probabilité de quarantaine d'une personne. Elle déclare et initialise une variable de type « t_personne » selon les spécifications suivantes :

- On copie les deux coordonnées reçues dans ses deux champs de position.
- Son état de départ sera mis à « EN_SANTE ».
- L'âge de la personne sera choisie au hasard entre [0, 99] (avec la fonction aléatoire « rand_age_canada() » du module utilitaire fourni « **alea_pop.h** »).
- L'état de quarantaine sera déterminé aléatoirement en générant une valeur aléatoire réelle entre [0.0, 1.0] et en la comparant avec la probabilité reçue. Si la valeur aléatoire générée est inférieure à cette probabilité, on met l'état de quarantaine de la personne à 1, sinon elle sera 0.
- La probabilité d'infection au virus est mise à la valeur PROB_CONTAGION.
- Le nombre d'heures en quarantaine ainsi que le nombre d'infections sont initialisés à 0.
- Pour ce qui est de la vitesse (et direction) de départ, commencez par générer un angle aléatoire (en radians) entre [0.5, 1.5]. Avec cette valeur, on obtient la vitesse en X avec « 2 * cos(angle) » et en Y avec « 2 * sin(angle) ». On applique ensuite aléatoirement un facteur de 1 ou -1 indépendamment à chacune des 2 vitesses obtenues (avec probabilité de 50% d'être négative).

La fonction retourne la nouvelle personne ainsi créée.

Fonction « **determiner_mort()** » :

Cette fonction utilise l'âge et le nombre d'infections de la personne reçue pour déterminer si elle va mourir suite à son confinement. On doit comparer l'âge avec les différentes tranches d'âge possibles, soit : 0 à 29, 30 à 39, 40 à 49, 50 à 59, 60 à 69, 70 à 79 et 80 ou plus. Chaque tranche d'âge donne une des 7 probabilités du tableau « PROB_DECES[ ] ». On génère un réel aléatoire entre [0.0, 1.0] et si ce réel est inférieur à (*probabilité de décès de la personne* / (*son nombre d'infections* + 1)), on retourne 1.

Par exemple : si âge = **49** et il a eu **2** infections, on utilisera (PROB_DECES[**2**] / **3**),
et si âge = **16** (inférieur à 29) et il a eu **0** infections on utilisera (PROB_DECES[**0**] / **1**).

Fonction « **contact_personnes()** » :

Reçoit une personne en référence ainsi qu'une coordonnée réelle (px, py). On détermine si la position de la personne reçue est située à moins de (2 * RAYON_1m) de la position reçue. Si oui, on retourne 1. Utilisez le calcul de la distance Euclidienne (avec une sous-fonction privée du module).

Fonction « **deplacer_personne()** » :

Effectue un déplacement de la personne reçue. Simplement ajouter la vitesse à la position actuelle (pour les 2 coordonnées). Suite à ce déplacement, il faudra vérifier si la personne entre en contact avec une des 4 bordures de l'espace de simulation (ou avec le mur de confinement). Un contact avec une des 4 bordures se produit lorsqu'une des deux coordonnées est à moins de RAYON_1m d'une des bordures (soit [0, largeur] en X ou [0, hauteur] en Y). Si une des bordures est atteinte, il faut inverser la vitesse actuelle de la coordonnée fautive (positive à négative ou vice-versa) et appliquer ensuite un autre déplacement dans ce sens. Par exemple, si la position-X de la personne a atteint (largeur – RAYON_1m), on applique « vitx *= -1 » et on effectue un déplacement en X seulement.

Pour la gestion d'un contact avec le mur de confinement, vous pouvez attendre après l'ajout du module « **t_mur** » pour l'instaurer (les 2 derniers paramètres seront donc en commentaires pour l'instant).

Fonction « **inverser_dir_pers()** » :

On reçoit ici 2 personnes et, suite à un contact, on les fera repartir dans deux directions opposées. Simplement choisir un angle aléatoire et déterminer les 2 champs de vitesse de la première personne selon cet angle (exactement comme dans « init_personne() »). La vitesse de la deuxième personne sera l'opposé (positif/négatif) de celle de la première personne.

Par exemple : pers1 : (vitx = -1.64, vity = 1.14) donc, pers2 : (vitx = 1.64, vity = -1.14)

Fonction « **modifier_etat_pers()** :

Modifie l'état et le mode de quarantaine de la personne reçue. Si le nouvel état reçu est « EN_SANTE », on doit diviser sa probabilité d'infection par trois (3) et incrémenter son nombre d'infections.

Module de la liste des personnes

C'est maintenant à vous de développer ce module au complet (*.h et *.cpp). Le module de la liste de personnes définie une structure qui contient un tableau de personnes avec les différents compteurs associés. Ce module devra inclure les modules « **personne.h** » et « **utilitaires_affichage.h** ».

Constantes et structure de la liste :

```
#define FACTEUR  1                 //facteur multiplicatif du nombre de personnes (1 à 5)
#define MAX_PERS (300 * FACTEUR)   //nombre initial de personnes (300, 600, …, 1500)

typedef struct{
  t_personne liste[MAX_PERS];     //tableau statique des personnes
  int  taille;                    //la taille du tableau
  int  nb_pers;                   //nombre de personnes non-mortes dans la liste
  int  nb_malades;                //le nombre de malades
  int  nb_sante;                  //le nombre de personnes en santé
  int  nb_morts;                  //le nombre de morts
} t_liste_personnes;
```

NOTE : Le tableau des personnes sera éventuellement un tableau dynamique.

À vous de définir les fonctions publiques qui vont agir sur la liste des personnes. Ces fonctions recevront toutes la liste en référence. Vous devez définir les fonctions qui vont permettre de :

- Générer la liste initiale de personnes (on reçoit aussi le nombre initial de personnes, la probabilité de quarantaine et les dimensions de l'espace). Il s'agit ici, pour chaque personne, de générer une coordonnée aléatoire dans l'espace qui n'entre pas en contact avec une autre personne déjà dans à la liste. Initialiser une personne en santé avec cette coordonnée et l'ajouter à la liste.

- Vider la liste des personnes (remettre tous les compteurs à zéro).
- Créer le premier malade (le « patient zéro »). On choisit une personne au hasard dans la liste et on met son état à MALADE mais pas en quarantaine. On ajuste ensuite les compteurs de la liste.
- Éliminer une personne morte de la liste. Ici on reçoit également une position-tableau, c'est l'indice de la personne à éliminer. On doit permuter cette personne avec la dernière personne en santé de la liste (on veut tous les morts à la fin de la liste), modifier l'état de cette personne à MORT et ajuster les compteurs de la liste (on aura un malade en moins).
- Traiter toutes les personnes. Ceci est la fonction principale de traitement, elle reçoit aussi la probabilité de quarantaine et les dimensions de l'espace. Voici l'algorithme à implémenter :

  Pour chaque personne encore vivante de la liste :
  - Si cette personne n'est pas en quarantaine ;
    - Déplacer cette personne.
    - Vérifier les contacts avec les autres personnes (**).

    Fin Si
  - Si cette personne est présentement malade ;
    - Augmenter son nombre d'heures de maladie.
    - Si ce nombre d'heures à atteint la limite d'heures en quarantaine (eg. 15 jours) ;
      - Valider si cette personne meurt ou non, si elle doit mourir ;
        - On élimine cette personne de la liste.
        - Décrémenter de compteur de la boucle principale (for) car on vient de permuter une autre personne non-traitée à cette même position-tableau.
      - Sinon, modifier l'état de la personne à « EN_SANTE » et ajuster les compteurs de la liste.

  La fonction retournera le nombre de personnes malades restants dans la liste.
- Dessiner toutes les personnes à l'écran. Pour toutes les personnes encore vivantes dans la liste, on choisit une couleur selon son état actuel, on récupère sa position et on dessine un cercle à cette position arrondie à l'entier le plus près (Ex. [245.7, 134.3] corresponds à [246, 134]). Les couleurs et la fonction de dessin sont disponibles à partir du module « **utilitaire_affichage.h** »
- Finalement, vous aurez besoin de fonctions accesseurs pour les compteurs suivants de la liste :
  - `int get_nb_personnes(const t_liste_personnes * liste_pers);`
  - `int get_nb_malades(const t_liste_personnes * liste_pers);`
  - `int get_nb_sante(const t_liste_personnes * liste_pers);`
  - `int get_nb_morts(const t_liste_personnes * liste_pers);`

****** Vérifier les contacts (sous-fonction privée) : cette sous-fonction reçoit la liste en référence, la personne actuellement traitée (en référence), la probabilité de quarantaine et les dimensions de l'espace.

Pour chaque personne encore vivante de la liste :

- Si ce n'est pas la MÊME personne que celle actuellement traitée ET qu'elle entre en contact avec lui,
  - Si la personne actuellement traitée est MALADE et que l'autre personne est EN_SANTE,
    - Obtenir une valeur entre [0.0, 1.0]. Si cette valeur est inférieure à la prob. d'infection de l'autre personne, modifier l'état de l'autre personne à MALADE (son état de confinement (0/1) sera généré aléatoirement selon la probabilité de quarantaine reçue) et ajuster les compteurs.
  - Sinon, si la personne actuellement traitée est EN_SANTE et que l'autre personne est MALADE,
    - Obtenir une valeur entre [0.0, 1.0]. Si cette valeur est inférieure à la la prob. d'infection de la personne traitée, faire le même traitement mais cette fois-ci appliqué à la personne traitée.
  - Fin Si
  - Inverser les directions des deux personnes concernées.
  - Déplacer la personne actuellement traitée de nouveau.
  - Si l'autre personne n'est pas en quarantaine, la déplacer de nouveau également.

4 Algorithme de la simulation (fichier principal)

Notre simulation se déroulera dans un environnement d'une taille proportionnelle au nombre de personnes. Utilisez les déclarations suivantes pour établir cette taille (selon le FACTEUR multiplicatif) :

```
//dimensions de l'écran d'affichage (en pixels) selon le FACTEUR du nombre de personnes
#define HAUTEUR  (int)( 550 * sqrt(0.8 * FACTEUR))
#define LARGEUR  (int)(1200 * sqrt(0.8 * FACTEUR))
```

Variables du programme principal (ce fichier doit inclure « liste_personnes.h ») :

- La liste des personnes (t_liste_personnes)
- La probabilité de quarantaine (un réel)
- Le nombre de malades, nombre maximal de malades (= 0) et le nombre d'heures (= 0) (3 entiers)

Voici, de façon sommaire, l'algorithme général de la simulation de votre modèle (dans le « main() ») :

- Avec une fonction de validation, saisir la probabilité de quarantaine entre [0.0, 1.0].
- Initialiser le générateur de nombre aléatoires.
- Initialiser le mode graphique et l'écran (avec « init_graph() » et « init_zone_environnement() »).
- Vider la liste de personnes.
- Générer une nouvelle liste de MAX_PERS personnes avec la probabilité de quarantaine et les dimensions d'environnement actuelles.
- Générer le patient-zéro.
- Afficher les compteurs de départ (avec « afficher_infos() »).
- Faire appel à « effacer_zone_environnement() » et ensuite dessiner toutes les personnes.
- Faire une pause-écran (avec « obtenir_touche() »).
- RÉPÉTER :
  - Incrémenter de +1 le nombre d'heures de simulation.
  - Traiter toutes les personnes et récupérer le nombre de malades actuel.
  - Ajuster le nombre maximal de malades si c'est le cas.
  - Faire appel à « effacer_zone_environnement() » et ensuite dessiner toutes les personnes.
  - Faire un petit délai-écran de (30 / FACTEUR) msec. (avec « delai_ecran() »).
  - Si le nombre d'heures écoulé est un multiple de 24 (une journée de terminée), afficher les infos.
  - Si le nombre d'heures écoulé est un multiple de 4, afficher une barre du graphique de progression avec « afficher_graphe() ». Le premier paramètre sera (heures / 4) et les deux autres paramètres sont les proportions réelles (sur MAX_PERS) des gens malades et en santé.
  - On vérifie si une touche-clavier fût pesée (avec « touche_pesee() ») et si oui, on récupère la touche pesée avec « obtenir_touche() ».
- TANT QUE : le nombre de malades est supérieur à 0 ET que la touche pesée n'est pas ESC (= 27).
- On dessine toutes les personnes une dernière fois et on affiche les infos finales.
- Faire une pause-écran (avec « obtenir_touche() »)
- Fermer ensuite le mode graphique et videz la liste de personnes.

➔ Notez que ceci illustre l'algorithme AVEC le mode d'affichage activé et SANS le mur de confinement.

5 Ajout d'un mur de confinement

Le module « **t_mur** » offre deux constantes et le type-enregistrement suivant (dans « t_mur.h ») :

```
#define NB_JOURS_MAX  60          //nombre de jours avant l'ouverture du mur
#define DIST_MUR  (3 * RAYON)     //si on est à moins de 3*5 pixels = contact avec le mur

typedef struct{
    int pos_mur;     //position en X du mur
    int longueur;    //la longueur en Y
    int nb_jours;    //nb. de jours depuis l'instauration du mur de confinement
} t_mur;
```

Ce module doit inclure « **utilitaires_affichage.h** » et offre ensuite 4 petites fonctions très simples :

- **init_mur(..)** : Cette fonction reçoit la position du mur ainsi que sa longueur. Elle déclare et initialise une variable de type « t_mur » avec les deux valeurs reçues et retourne ce nouveau mur ainsi crée.
- **contact_mur(..)** : Reçoit un mur par référence (t_mur *) et la position-X d'une personne (un réel). La fonction vérifie si cette position est à moins de DIST_MUR pixels (distance absolue) et retourne 1 si c'est le cas, 0 sinon.
- **mur_actif(..)** : Reçoit un mur par référence (t_mur *) et incrémente le nombre de jours de confinement de ce mur. Si ce nombre atteint NB_JOURS_MAX la fonction retourne 0 (plus de mur). Sinon elle retourne 1 (le mur est toujours actif).
- **dessiner_mur(..)** : Reçoit un mur par référence (t_mur *) et dessine le mur à l'écran (pour le mode AVEC affichages). Utilisez ici la fonction « afficher_mur() » du module « utilitaires_affichage ».

Modifications dans le reste du programme :

Module « **personne** », fonction « **deplacer_personne()** » : Ajoutez deux paramètres supplémentaires à cette fonction; un « mode_mur » (entier) et un mur par référence (t_mur *). Dans sa définition, ajoutez une validation supplémentaire de contact avec le mur :

```
Si (mode_mur == 1  ET la personne reçue entre en contact avec le mur) {
  -   inverser la vitesse actuelle en X et effectuer un déplacement en X seulement
}
```

Module « **liste_personnes** » :

- **fonction de génération de la liste** : Ajoutez les deux mêmes paramètres à cette fonction; un « mode_mur » (entier) et un mur par référence (t_mur *). Dans le code qui crée une nouvelle personne avant de l'ajouter à la liste, ajoutez la validation supplémentaire de contact avec le mur pour la coordonnée-X de la personne générée aléatoirement. Si il y contact, on doit rechoisir une coordonnée aléatoire pour cette personne (comme quand il est trop près d'une autre personne).

- **sous-fonction qui vérifie les contacts** : Ajoutez les deux mêmes paramètres à cette fonction; un « mode_mur » (entier) et un mur par référence (t_mur *). Ces paramètres sont nécessaires ici car vous devez les ajouter à tous les appels de la fonction « deplacer_personne() ».

- **fonction pour traiter toutes les personnes** : Ajoutez les deux mêmes paramètres à cette fonction; un « mode_mur » (entier) et un mur par référence (t_mur *) car ils sont nécessaires aux appels à « deplacer_personne() » et à la sous-fonction qui vérifie les contacts.

Fichier principal, fonction « main() » : Ajout d'une variable « t_mur » et un entier « mode_mur ». Au départ, demandez le « mode_mur » à l'utilisateur (1 = avec le mur, 0 = sans le mur). Si l'utilisateur choisi le mode avec un mur, créer le mur selon les dimensions « (LARGEUR / 2, HAUTEUR) » de l'écran.

Dans la boucle de simulation, ajoutez le « mode_mur » et le mur comme paramètres supplémentaires à la fonction de traitement des personnes. Ajoutez également une condition « Si (mode_mur == 1 ET une journée de 24 hrs. est terminée) { ..vérifier si le mur est encore actif.. } ». Finalement, lors de l'affichage des personnes, si le « mode_mur » est actif on dessine le mur.

6 Conversion à un tableau dynamique de personnes

Une fois que votre simulateur sera jugé fonctionnel, il faudra modifier légèrement le code du module « liste_personnes » afin de faire utilisation d'un tableau dynamique pour la liste des personnes :

Dans « **t_liste_personnes** » : `t_personne * liste;  //tableau dynamique des personnes`

Le champ « taille » dans cet enregistrement sera maintenant très utile car il remplace la constante « MAX_PERS » au moment de vérifier si la liste est pleine (durant l'ajout d'une nouvelle personne).

L'allocation du tableau dynamique et l'initialisation de sa « taille » doivent être fait dans la fonction qui génère la liste initiale de personnes (selon le nombre initial de personnes reçu en paramètre).

Vous devez aussi libérer ce tableau et remettre la « taille » à zéro dans la fonction qui vide la liste.

7 Écriture dans un fichier

Votre programme doit donner deux options à l'utilisateur, soit un mode avec affichages des personnes ou un mode SANS affichages mais avec écriture des statistiques dans un fichier, après chaque jour complet.

Voici le format désiré pour le fichier-texte des résultats (vous devez inscrire la ligne des titres):

JOUR	POPU	MORTS	SANTE	MALADES	MAX
1	900	0	898	2	2
2	900	0	898	2	2
3	900	0	897	3	3
. . .	. . .	. . .	. . .	. .	. . .
304	850	50	849	1	221

NB. MOYEN d'INFECTIONS: 2.22 (Max. 5, Jamais = 37)

La dernière ligne de données affichée dans le fichier sera obtenue à l'aide d'une dernière fonction de calcul du module « **liste_personnes** ». Cette fonction calcule et retourne la moyenne du nombre d'infections de TOUTES les personnes reçues et aussi récupère le nombre maximal d'infections ainsi que le nombre total de personnes qui n'ont jamais été infectées (i.e. ceux avec « `->nb_inf == 0` »).

```
double nb_moyen_inf(const t_liste_personnes * liste_pers, int * max, int * nb_zero_fois);
```

Cette étape peut être réalisée à la toute fin du projet. Avec un code bien fractionné il est très simple d'ajouter l'option d'écriture dans un fichier sans affichages (sauf pour les appels à « afficher_graphe() »).

8 Dynamique d'un travail en équipe

Dans le monde du travail, vous ne serez jamais le seul et unique maitre d'œuvre d'un programme. Par exemple un membre d'équipe code le module X, un autre le module Y et ensuite le programme principal. Le travail s'exécute souvent en parallèle, donc il est normal qu'un coéquipier utilise des fonctions qui n'ont pas encore été créées par l'autre. Il faut donc s'entendre sur les noms des types définis, des fonctions et les types et noms des paramètres reçus. Pour ce faire, apprenez à bien utiliser « *GitHub* ».

9 Exigences de remise

Chaque équipe va remettre une seule remise avec le programme complet dans une archive ZIP. N'oublier pas de bien commenter chaque module (fichier *.h) et chaque fonction de votre programme. Le travail devra respecter les exigences de remise des travaux pratiques. De plus;

- Un programme qui ne fonctionne ou ne compile pas se voit attribuer la note 0,
- AUCUNE VARIABLE GLOBALE NE SERA ACCEPTÉE (pénalité de 20%),
- Votre projet devra respecter la structure modulaire suggérée dans cet énoncé et en classe,
- La politique du 10% pour la qualité du français dans les commentaires sera appliquée,
- Le plagiat attribuera la note 0 à tous les participants peu importe le degré d'implication.

BON TRAVAIL!