Projet Programmation Concurrente – année 2022-23
Polytech'Nice Sophia – SI4
M. Riveill

Durée indicative : entre 15 et 30 heures dont uniquement 6 heures seront faites lors des séances de TD. A faire par groupe de deux (éventuellement 3).

Vision d'ensemble du projet : il s'agit de modéliser de manière simple le déplacement d'une foule. Nous sommes dans le cadre d'un cours programmation concurrente et donc ce qui est important ce n'est ni l'IHM, ni l'évolutivité du code mais bien l'identification des contraintes de synchronisation et y apporter une réponse adéquate.

Vous devez vous servir du notebook ci-joint en Python et le considérer comme un rapport exécutable.

Spécification de la mise en œuvre

La description du terrain est faite dans un fichier de description qui a le format suivant :

```
120 100   #[1] taille de la grille
# position initiale des personnes (X, Y) et case de sortie (X, Y)
2 2 0 0
3 3 0 9
6 6 9 0
7 7 9 9
```

Dans l'exemple ci-dessus, la grille est un terrain de 10 * 10 et il contient 4 personnes :
1. La personne dans la position initiale 2,2 doit rejoindre la position 0,0 pour sortir de la grille
2. La personne dans la position initiale 3,3 doit rejoindre la position 0,9 pour sortir de la grille
3. Etc.

Toutes les positions de sortie sont sur le bord de la grille. Une fois la position de sortie atteinte alors la personne disparaît. Quand toutes les personnes sont sorties du terrain la simulation s'arrête.

Pour le rendu du projet prévoyez une grille de la taille de 120 * 100 et 2^10 (1024) personnes. En effet on souhaite mesurer l'évolution du temps d'exécution et de la place mémoire occupée en fonction de l'évolution du nombre de personne sur la grille, de 2 à 1024 selon les valeurs suivantes : 2^0, 2^2, 2^4, 2^6, 2^8 et 2^10.

Déplacement d'une personne

Il est volontairement simple :
- On se déplace sur une case vide qui peut être à droite, à gauche, en bas ou en haut,
- On essaie toujours de se rapprocher de la sortie,
- On choisit toujours le meilleur déplacement possible.

[1] # signifie un commentaire

Arrêt du simulateur

On arrête la simulation quand toutes les personnes sont sorties.

Les principaux problèmes :

- Détection et traitement de situation classique en programmation concurrente : interblocage et section critique
- L'objectif est de mettre en œuvre une solution qui conserve le maximum de parallélisme. Le parallélisme max de votre station est lié au nombre de cœur dont elle dispose.

Comme l'on va comparer les différentes mises en œuvre pour un nombre variable de personne, il est souhaitable qu'à chaque exécution, les personnes partent de la même place.

Je suis plus intéressé par la manière dont vous allez présenter vos solutions que par le code qui va les mettre en œuvre.

Scénario 1 : approche itérative – un seul thread → Pour le 15 octobre.

Principal problème à traiter : 2 personnes devant occuper la même case. Il faut alors mettre en œuvre une stratégie que nous allons voir en cours : le thread la plus jeune (celle qui a l'ID le plus petit est détruite, la ressource qu'elle occupait i.e. la case est libérée et elle redémarre – tout de suite ou ultérieurement à sa position initiale).

On comprend aisément dans le scénario ci-dessous que si la personne à gauche ne peut aller que vers la droite, et que la personne à droite ne peut aller que sur la gauche alors une situation de blocage va se mettre en place.

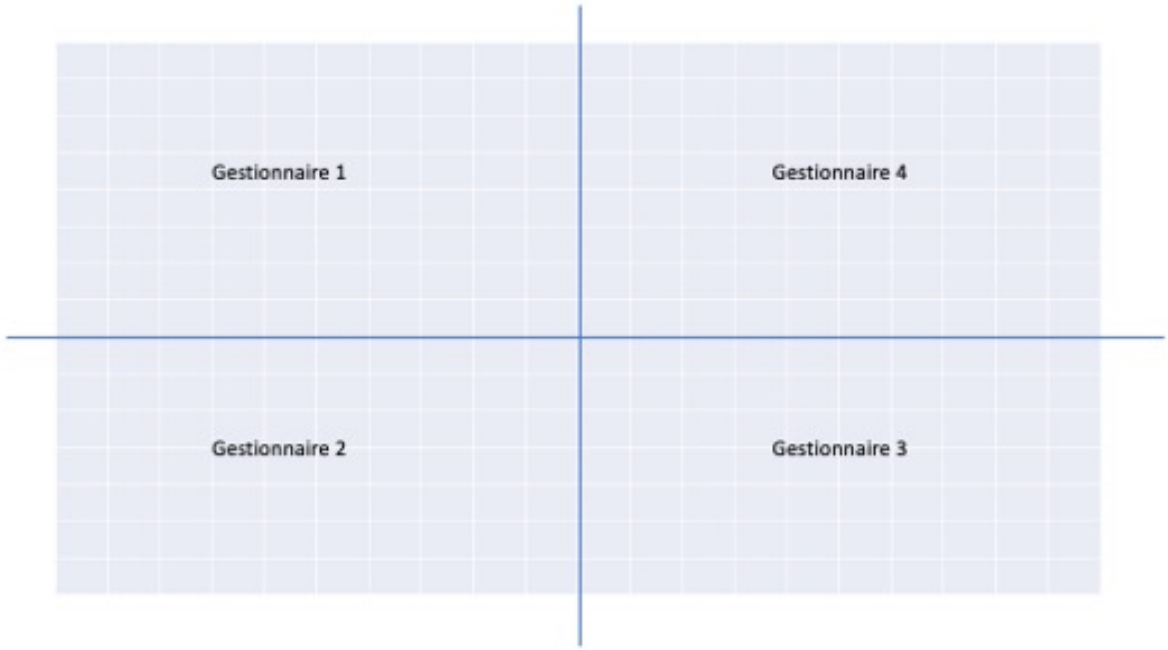
Scénario 2 : approche autant de thread que de personnes → Pour le 30 novembre.

Principaux problèmes à traiter :

- Accès en section critique à la portion de grille pertinente
- Le problème du scénario 1
- La terminaison du simulateur

Scénario 3 : approche 4 threads → Pour le 31 décembre.

La grille est divisée en quartier. Chaque quartier à un gestionnaire qui gère de manière séquentielle les threads (cf. scénario 1).



Principaux problèmes à traiter :

- Trouver un mécanisme simple permettant aux gestionnaires de « s'échanger » des personnes quand celle-ci change de « partie » de la grille
- Les problèmes du scénario 3.

Que rendre, à chacune des étapes

- Une archive zip se décompressant dans un répertoire portant le nom des membres du groupe
    - L'archive de l'étape N doit contenir les éléments de l'étape N-1
- L'archive contient
    - Un répertoire java source
    - Un répertoire java compilé
    - Un document pdf décrivant
        - Un comparatif des mécanismes de manipulation de threads / outils de synchronisation associé à chacune des étapes pour les langages : Java / Python / C-Posix. La liste des éléments à comparer sera mise à jour sur LMS.
        - Votre solution pour chacune des étapes
        - Une comparaison entre les différentes étapes en particulier vis à vis de l facilité de mise en œuvre, du temps d'exécution en fonction du nombre de personnes et de la place mémoire occupée.
        - La manière d'exécuter votre code (rappel : ce qui m'intéresse est la manière dont vous utilisez et justifiez l'utilisation de tel ou tel mécanisme – la partie graphique est en dehors du travail demandé – inutile donc d'y passer un temps démesuré).