Table des matières

PCP – Travail pratique 02 ..... **2**

Votre tâche ***2***

Détail du travail ***3***

Déroulement attendu d'une exécution du système ***4***

Carte 4

Rats 5

Chasseurs de rats 6

Autres règles ***7***

Recherche de chemins 7

Mouvement 7

Manger un fromage 8

Sortir de la ville 8

Capturer un rat 8

Miauler 8

Entendre un miaulement 8

Fin de l'exécution 8

Forme du diagnostic ***9***

Consignes ***10***

Choix technologiques 10

Format des équipes 11

Remise ***11***

Format de la remise 12

PCP – Travail pratique 02

Pour ce dernier travail pratique, vous êtes conviés à résoudre un problème qui demandera à la fois beaucoup de calcul, beaucoup de communication et beaucoup de synchronisation : le problème de l'invasion des rats.

La situation est la suivante :

- il y a une invasion de rats dans la ville de Rongeurville. C'est un drame, qui met en péril les réserves de fromage de ce paisible hameau;
- en temps normal, un rat sent à distance les fromages et, pris d'un appétit féroce, fonce vers le fromage le plus proche;
- heureusement, il y a aussi des chasseurs de rats, qui ont pour but de… chasser des rats;
- chaque chasseur de rats a une technique avancée pour faire son travail : il imite à la perfection le miaulement d'un chat;
- en entendant le miaulement d'un chat, un rat est pris de panique et cherche à s'éloigner de la source du miaulement;
- l'objectif des rats est de manger la totalité des réserves de fromage de Rongeurville;
- l'objectif des chasseurs de rats est de chasser les rats, ou de les capturer.

Pour planifier vos probabilités de succès, vous souhaiterez évaluer le comportement théorique de rats et de chasseurs de rats. Cela impliquera potentiellement beaucoup de traitement, car tous les rats et tous les chasseurs devront faire de la recherche de chemins sur une carte. Le nombre de rats pouvant être très grand, il vous faut une solution qui ne soit pas liée à un seul ordinateur, comme le serait une implémentation strictement *multithread*.

Un peu de recherche vous a permis de constater que, dans le monde du *High Performance Computing* (HPC), plusieurs développent des solutions multiprocessus, potentiellement destinées à être exécutées en parallèle sur plusieurs ordinateurs, à travers le *Message Passing Interface* (MPI). Le temps presse, alors voilà la piste que vous suivrez.

Votre tâche

Votre tâche sera d'offrir une implémentation multiprocessus du problème des rats et des chasseurs de rats. Cette implémentation comportera (a) une carte de la ville, incluant des fromages et ses sorties, de même que la position officielle des rats et des chasseurs de rats (b) n processus rats et (c) m processus chasseurs de rats.

Cette implémentation sera une forme de simulation où chaque rat et chaque chasseur de rat cherchera à accomplir sa mission. La mission d'un rat est de manger du fromage, alors que la mission d'un chasseur de rat est de chasser les rats ou de les capturer.

Une exécution du système multiprocessus se terminera lorsque les rats ou les chasseurs auront accompli leur mission, et affichera un diagnostic de l'exécution dans un fichier texte (voir Forme du , plus bas).

Détail du travail

Tout d'abord, un avertissement : *ce travail est plus costaud que les précédents*. Il implique plusieurs *a priori* « cachés » :

- représenter une configuration de jeu (la carte de la ville) et en assurer la cohérence;
- comprendre et représenter les actions du système (mouvements, consommation de fromage, départ de la carte, miaulement, capture d'un rat, etc.);
- implémenter un algorithme de recherche de chemin; et surtout
- comprendre et tirer profit du modèle d'exécution de `MPI`, de même que de son modèle de communication. Ceci n'est pas banal, l'`API` de `MPI` montrant un peu son âge.

Déroulement attendu d'une exécution du système

Au lancement d'une exécution du système, vous utiliserez les mécanismes de lancement de `MPI` pour fixer dynamiquement le nombre de processus participant au système. Vous passerez aussi en paramètre au programme le nombre de rats, le nombre de chasseurs, et le nom du fichier contenant la carte de la ville.

Carte

Un processus représentera la carte (je vous suggère d'en faire le processus zéro, pour simplifier le modèle). Celui-ci chargera la carte du disque, la transférera aux autres processus sous la forme d'une chaîne de caractères, et signalera aux autres processus leur rôle et leur position initiale. Pour fins de simplicité, la carte aura les propriétés suivantes :

- elle sera rectangulaire;
- elle sera représentée sur disque comme du texte;
- le symbole `'#'` y représentera un mur;
- le symbole `'R'` y représentera un rat;
- le symbole `'C'` y représentera un chasseur de rats;
- le symbole `'F'` y représentera un fromage;
- le symbole `' '` y représentera un espace « vide », susceptible d'être occupé par un rat ou par un chasseur de rats, sauf en bordure d'une carte où ce symbole représente une sortie de la ville.

Un exemple simpliste de carte sur disque serait le suivant :

```
########################################
#   F    F                C          #   #
#                                      # R #
#     ##################       ########   #
#     #          R     #          F   #   #
#     #  F   #         #     ###      #   #
      #      #####     #     #    #        #
#     #######        ###   ###    #        #
# R                                #   ##  #
#      ####         C          #   #
########################################
```

Dans cette carte, on trouve quatre fromages, deux chasseurs de rats et trois rats. Une sortie se trouve à peu près au centre du mur de gauche de la ville, et une autre se trouve dans le coin inférieur droit.

C'est la carte du donnera aux autres processus un signal indiquant le début officiel de l'exécution. Avant ce signal, les autres processus pourront procéder à leurs diverses initialisations préparatoires, mais ne pourront s'adonner à leurs autres actions comme la recherche de chemins.

Pendant l'exécution du système, la carte sera une sorte de relais : les processus participant à l'exécution lui indiqueront les actions qu'ils comptent poser, la carte acceptera ou non chaque action (elle doit interdire ce qui briserait la cohérence de l'expérience, par exemple une tentative de téléportation, de passage à travers un mur, ou encore une tentative de mouvement par laquelle deux rats ou deux chasseurs de rats en viendraient à occuper un même espace), et diffusera aux processus participant au système chaque changement d'état ayant été accepté.

Le processus représentant la carte est le seul qui détienne la vérité sur l'état de l'exécution. Les copies des cartes envoyées aux autres processus servent à faciliter leur travail de détection de chemins, sans plus, et doivent être considérées pour chacun des processus comme une image du passé, un peu désuète, du fait que ces copies représentent au mieux l'état de la plus récente mise à jour reçue de la carte.

Enfin, c'est la carte qui déterminera le résultat de l'exécution, à savoir une victoire des rats pour cause de consommation de tous les fromages; une victoire des chasseurs du fait que le dernier rat vient de quitter la ville; ou encore une victoire des chasseurs du fait que le dernier rat vient d'être capturé. La carte doit enfin signaler aux autres processus qu'ils doivent s'arrêter, et doit elle-même compléter son travail lorsque le dernier de ses collègues processus aura complété le sien.

Rats

En temps normal, un rat cherche à manger du fromage. Ainsi, son travail est de :

- trouver le chemin le plus court vers le fromage le plus près sur la carte. Par « plus près » ici, nous entendons celui qui serait rejoint par le plus petit nombre de pas de rat. Si deux fromages ou plus sont équidistants d'un rat donné, alors le rat se dirigera vers l'un ou l'autre d'entre eux;
- choisir parmi les cases avoisinant sa case courante l'une de celles le rapprochant le plus de cette destination; et
- signaler à la carte son intention de se déplacer sur cette case.

Un rat se déplace d'un pas de rat à la fois. Un pas de rat mène un rat de sa position courante à une case avoisinante. Pour un rat, une case avoisinante et l'une des huit cases immédiatement voisines (informellement : un rat peut se déplacer à la verticale, à l'horizontale et en diagonale d'une case à la fois). Pour qu'une case avoisinante puisse accueillir un rat, il faut que cette case soit vide (incluant une sortie de la ville) ou contienne un fromage.

Si un rat entend un miaulement à proximité (à proximité signifie ici à sept pas de rats de distance ou moins; ne tenez pas compte des murs pour les fins de ce calcul, car les rats ont de bonnes oreilles), il est pris de peur et son objectif devient de quitter la ville par la sortie la plus proche. Toutefois, si un rat n'a pas entendu de miaulement à proximité au cours de ses cinq derniers envois de demandes de de mouvement à la carte, alors il reprend son plan de manger du fromage et cherche son chemin en ce sens.

Chasseurs de rats

Un chasseur de rats cherche à capture ou à chasser des rats. Ainsi, son travail est de :

- trouver le chemin le plus court vers le rat le plus près sur la carte. Par « plus près » ici, nous entendons celui qui serait rejoint par le plus petit nombre de pas de chasseur de rats. Si deux rats ou plus sont équidistants d'un chasseur de rats donné, alors le chasseur poursuivra l'un ou l'autre d'entre eux;
- choisir parmi les cases avoisinant sa case courante l'une de celles le rapprochant le plus de cette destination; et
- signaler à la carte son intention de se déplacer sur cette case.

Un chasseur de rats se déplace d'un pas à la fois. Un pas de chasseur de rats mène un chasseur de rats de sa position courante à une case avoisinante. Pour un chasseur de rats, une case avoisinante et une des quatre cases immédiatement voisines à la verticale ou à l'horizontale (informellement : un chasseur de rats peut se déplacer à la verticale, à l'horizontale, mais pas en diagonale, d'une case à la fois). Pour qu'une case avoisinante puisse accueillir un chasseur de rats, il faut que cette case soit vide (incluant une sortie de la ville) ou contienne un rat.

Si un chasseur de rats voit un rat à proximité (à proximité signifie ici à dix pas de chasseur de rats de distance ou moins, en ne comptant que les cases sur lesquelles le chasseur de rats pourrait se déplacer), il doit miauler pour faire peur au rat. Un chasseur de rats miaulera si cela est opportun, et seulement avant d'envoyer à la carte une demande de déplacement (pour éviter une cacophonie où tous les chasseurs de rats miauleraient sans cesse).

Autres règles

Pour le bon fonctionnement de l'exécution de ce système, veuillez tenir compte des règles suivantes. J'utiliserai « protagoniste » ci-dessous pour identifier les chasseurs de rats et les rats de manière générale, pour alléger le texte.

Recherche de chemins

Chaque protagoniste effectuera une recherche de chemins sur la base des connaissances qu'il possède à un moment donné. Les chemins investigués doivent tenir compte des règles pour qu'un mouvement soit possible dans chaque cas, de même que de l'objectif courant de chaque protagoniste.

Plusieurs algorithmes existent pour la recherche de chemins, les plus connus étant Dijkstra et A*. Cette partie du travail est intéressante (elle consomme du temps de calcul, ce qui justifie en partie le travail réalisé en parallèle) mais ne devrait pas constituer une difficulté étant l'abondance d'algorithmes documentés que vous pouvez adapter selon vos besoins.

Une fois qu'un chemin a été trouvé, par un protagoniste, celui-ci doit envoyer une demande de mouvement à la carte. Le mouvement n'aura eu vraiment lieu que lorsque la carte l'aura signalé.

Mouvement

Tout protagoniste enverra des demandes de mouvement à la carte, alors que celle-ci enverra des confirmations de mouvement.

Je vous suggère de représenter un demande de mouvement par deux paires $\{x, y\}$ représentant respectivement la case de départ et la case de destination pour ce mouvement, et d'utiliser la même représentation pour les messages envoyés par la carte aux protagonistes pour les informer qu'un mouvement a eu lieu.

Pour signaler un refus de mouvement, la carte peut envoyer un message indiquant les mêmes coordonnées pour le départ et pour l'arrivée.

Il va de la responsabilité de la carte que les mouvements soient cohérents. Par exemple, dans le cas où un rat est à la position $\{0,1\}$ et un chasseur de rats est à la position $\{1,1\}$, il ne doit pas être possible pour le rat de se retrouver à la position $\{1,1\}$ et le chasseur de rats à la position $\{0,1\}$ après un mouvement de part et d'autre, comme s'ils s'étaient passés au travers l'un de l'autre (techniquement, dans une telle situation, il doit y avoir un moment où le chasseur s'est trouvé sur la même case que le rat et l'a capturé).

Les informations quant aux mouvements sont susceptibles de survenir pendant que les protagonistes font leur recherche de chemin. À vous de voir comment vous assurerez la cohérence du processus de recherche de chemin de chaque protagoniste dans ce contexte.

De son côté, la carte des susceptible de recevoir en tout temps des demandes de mouvement, car ce n'est pas une exécution « par tour » que nous faisons ici (un ordinateur plus rapide pourrait générer plus de demandes qu'un autre ordinateur, plus lent, pour une même exécution). À vous de voir comment vous gérerez le tout.

Manger un fromage

Si un rat se trouve superposé à un fromage, alors la carte doit retirer le fromage (il a été mangé) et envoyer aux autres processus un message indiquant le retrait en question. Ceci affectera les possibilités de déplacement des chasseurs de rats de même que la recherche de chemins effectuée par les rats eux-mêmes.

Sortir de la ville

Si un rat se trouve superposé à une sortie de la ville, alors la carte doit retirer le rat (il a quitté) et envoyer aux autres processus un message indiquant le retrait en question. Le processus représentant le rat a alors terminé son travail.

Si le rat nouvellement retiré était le dernier en ville, alors l'exécution doit se terminer. C'est la responsabilité de la carte d'envoyer un message en ce sens aux protagonistes et d'attendre que ceux-ci confirment avoir terminé leur exécution.

Capturer un rat

Si un chasseur de rats se trouve superposé à un rat, alors la carte doit retirer le rat (il a été capturé) et envoyer aux autres processus un message indiquant le retrait en question. Le processus représentant le rat a alors terminé son travail.

Si le rat nouvellement retiré était le dernier en ville, alors l'exécution doit se terminer. C'est la responsabilité de la carte d'envoyer un message en ce sens aux protagonistes et d'attendre que ceux-ci confirment avoir terminé leur exécution.

Miauler

Si un chasseur de rats se trouve en position de miauler (voir Chasseurs de rats, plus haut), alors il doit envoyer un message à la carte indiquant ce miaulement. La carte n'a pas à valider le miaulement (mais ne trichez pas!), et doit relayer à tous les rats la position du miaulement.

Entendre un miaulement

Si un rat entend un miaulement (voir Rats, plus haut), alors il doit avoir pour objectif de quitter la ville et axer sa recherche de chemins en conséquence. Quand le délai de crainte d'un miaulement est expiré, le rat reprend sa recherche de fromage.

Fin de l'exécution

Lorsque la carte constate la fin de l'exécution (consommation du dernier fromage, sortie du dernier rat, ou capture du dernier rat), elle envoie un signal aux protagonistes restants pour les informer de la fin de l'exécution.

Lorsqu'un protagoniste reçoit un signal de fin d'exécution, alors il réaliser les tâches de nettoyage qu'il lui reste à faire (peu importe lesquelles) et envoie un message à la carte indiquant qu'il a conclu son exécution, suite à quoi il termine effectivement son exécution.

La carte termine son exécution suite à la réception d'un message confirmant la fin de l'exécution du dernier protagoniste.

Forme du diagnostic

La carte devra produire un rapport simple en fin d'exécution, relatant les éléments suivants :

- temps total d'exécution entre le lancement de l'exécution (fin de l'envoi par la carte des signaux de démarrage aux protagonistes) et la fin de l'exécution du dernier protagoniste;
- nombre de demandes de mouvement, par protagoniste;
- proportion de mouvements acceptés, par protagoniste (un rationnel dans l'intervalle $[0,1]$);
- chaque capture de rat (quel rat a été capturé par quel protagoniste);
- chaque sortie de rat (quel rat a quitté la carte, à quel endroit);
- chaque consommation de fromage (quel rat a mangé un fromage, à quel endroit);
- chaque miaulement (quel chasseur a miaulé, à quel endroit);
- une « photo » de la carte à intervalles réguliers (disons chaque seconde, mais vous évaluerez ce qui est pertinent en fonction du type de matériel à votre disposition).

Vous pouvez ajouter d'autres informations au fichier de diagnostic si vous le souhaitez. Ce fichier sera un simple fichier texte que l'on pourra examiner ultérieurement, et qui racontera en quelque sorte l'histoire de l'exécution du point de vue de l'un de ses processus.

Consignes

Les consignes quant à l'organisation du travail suivent.

Choix technologiques

Vous pouvez utiliser le langage de votre choix sur la plateforme qui vous convient. La plupart des langages commerciaux réputés (incluant C, C++, Java et C#) ont au moins une implémentation de `MPI`. Cela dit, quelques trucs pour vous aider :

- si vous utilisez un protocole indépendant de la plateforme (par exemple, du texte encodé sur huit ou seize bits ou un format binaire simple à représenter dans plusieurs langages), vous pourrez plus facilement travailler avec des collègues qui utilisent d'autres langages de programmation que vous, ce qui facilitera le débogage;
- définissez rigoureusement le format de vos messages. Si vous utilisez du texte, attention par exemple aux majuscules et aux minuscules (entendez-vous au préalable sur l'impact ou non de ce choix!). Si vous utilisez des nombres, entendez-vous sur la représentation des nombres à virgule flottante et sur les bornes de validité;
- dans tous les cas, définissez les modalités de gestion des erreurs (même d'inattention)!
- faire un protocole est périlleux. Gardez le vôtre simple!
- inspirez-vous des pratiques connues, tout comme des schémas de conception[1] et des idiomes usités dans le domaine. Pensez à des *proxy*, par exemple, vous vous simplifier l'existence… Pensez observateur, pensez `RAII`, pensez `pImpl`. En même temps, n'utilisez pas ces pratiques si elles ne vous apportent pas quelque avantage concret (nous voulons résoudre des problèmes, pas faire du *Name Dropping* architectural);
- envisagez des cas d'utilisation, même informels, pour les interactions entre vos modules. Ça vous épargnera bien des maux de tête;
- choisissez avec soin le format des échantillons, de même que vos structures de données.

[1] Un point de départ : http://h-deb.clg.qc.ca/Sujets/Developpement/Schemas-conception.html

Format des équipes

Le travail peut être fait en équipe de deux à quatre personnes.

Si vous n'aimez pas travailler en équipe, notez que le format du travail vous permet le télétravail et, si vos choix technologiques vous le permettent, ce télétravail ne vous empêchera pas d'interagir à distance avec les modules de vos collègues.

Remise

Je m'attends à un rapport mettant en relief les éléments suivants :

- l'identité et la contribution de chaque membre de l'équipe;
- une description des choix technologiques faits (voir Choix technologiques, plus haut);
- stratégies de synchronisation appliquées dans chaque module, s'il y a lieu;
- problèmes rencontrés et solutions apportées, de même que les bogues restants; et
- des binaires que je peux exécuter sur Windows 7 ou sur Linux, ou une démonstration.

Je veux avoir accès sources de chaque module. Je ne sais pas si je pourrai vous donner une rétroaction détaillée cette fois (faute de temps), mais je veux les avoir à portée de main et pouvoir vous poser des questions au besoin.

Format de la remise

La remise du travail peut être sous forme papier ou sous forme électronique. Dans le cas d'un document électronique, si vous souhaitez une rétroaction signifiante, je dois être en mesure de l'annoter à l'aide de Microsoft Word 2013 (je n'ai *vraiment* rien contre Open Office, mais ce n'est tout simplement pas ce que j'ai sur mes postes de travail). Il est possible mais un peu déplaisant d'annoter un PDF, alors si vous me livrez un document dans ce format, je limiterai ma rétroaction à quelques remarques par courriel, sans plus.

Si vous souhaitez que je teste votre travail chez moi, tenez compte des spécifications techniques de mon ordinateur[2]. Vous devez aussi vous assurer, par souci d'intégrité, de pouvoir me produire les sources de tous vos modules sur demande et de pouvoir me donner une démonstration sur place, au campus où vous suivez ce cours et un jour où je suis présent, si j'en exprime le désir (si je le fais, je vous donnerai quelques jours d'avis).

Ne me rendez pas la vie trop compliquée je vous prie : il est nécessaire que je puisse installer et désinstaller le tout aussi simplement que possible, que ce soit par un installateur automatisé ou à l'aide de consignes simples et claires. Je corrige énormément de travaux chaque session; il est donc impératif que vous me rendiez la tâche facile à l'installation et que mon ordinateur soit *propre* après la désinstallation.

Tout document remis devra présenter un français de qualité. Le code devra utiliser une police de caractères non proportionnelle (quelque chose comme Courier New ou Lucida Console par exemple) pour l'impression de programmes ou d'extraits de programmes, et indentez à l'aide d'espaces (les tabulations ont une influence qui varie d'un éditeur à l'autre, ce qui complique la présentation du code et sa lisibilité). Ces polices conviennent bien à un tel usage, clarifiant l'indentation appliquée[3]. D'ailleurs, même si vous utilisez des outils de génération automatique de code[4], ce code devra être indenté de manière lisible, sinon il vous sera retourné sans que je ne l'aie lu.

Ce travail est à remettre au plus tard le 14 août 2017, à 17 h, par courriel à Patrice.Roy@USherbrooke.ca. C'est le jour de l'examen final, alors on parle vraiment de la limite extrême pour une remise.

[2] PC au processeur Intel à huit cœurs cadencés à 2,7 GHz, 16 Go RAM, *Microsoft Windows 10*, ce qui devrait suffire.

[3] Sur le site de l'enseignant, voir les articles http://h-deb.clg.qc.ca/Sujets/TrucsScouts/Police-pour-code.html et http://h-deb.clg.qc.ca/Sujets/TrucsScouts/Eviter-Tabulations.html pour plus de détails quant aux exigences en lien avec la présentation du code.

[4] Je pense en particulier aux *aficionados* de Delphi ici. http://h-deb.clg.qc.ca/Sujets/Developpement/Pratique-programmation.html#indentation pour des idées. L'important est d'être conséquent à l'interne dans un programme donné, et de viser la lisibilité.