INFO-F404 Operating Systems II - Projet 2

Chapeaux Thomas
Dagnely Pierre

December 17, 2012

1 Introduction

Le but de ce projet était de construire en C++ et MPI une version interactive et multi-processor du jeux cluedo.
En permettant à l'utilisateur de jouer contre plusieurs IA.

2 Choix d'implémentation

2.1 Agencement du programme

Notre programme se compose de trois classes :

Cleudo : C'est la classe principale qui contient le main, elle demande à l'utilisateur combien d'IAs il veut, puis initialise le jeux (et l'environnement MPI) en conséquence. Elle va créer une instance GameMaster et autant d'instances AIs que le joueur désire.
Rem : Au lancement du programme nous demandons directement 6 processus, et c'est l'utilisateur qui choisira combien seront vraiment utilisés.

GameMaster : C'est la classe qui va gérer le jeux et le joueur humain.
Elle s'occupe de toutes les interactions (input/output) avec celui-ci ainsi qu'entre les IAs et lui.
C'est également elle qui va initialiser la partie (distribution des cartes, sauvegarde de la solution, ...)
Ainsi que vérifier les accusations et terminer la partie quand l'une d'entre elle se révèle correcte.

AI : C'est la classe qui va gérer une IA.
Elle contient les méthodes utilisées par les IAs pour faire des suggestions et accusations, répondre à des suggestions, ...

(Voir diagramme de classe en annexe)

2.2 Structures de données

Nous n'utilisons que des std::vector d'entiers, il sont de deux types :

_humanDeck et _aiDeck : Ils représentent les cartes que le joueur humain ou les IAs ont reçut au début du jeu.
Les cartes sont représentées par des entiers compris entre 0 et 20. Cela permet de simplifier les comparaisons et manipulations.

_aiKnowledge : ils sont utilisés par les IAs pour représenter leur connaissance du jeu.
Chaque élément correspond à une carte (ce qui est simplifié par le fait que les cartes sont stockées sous forme d'entier) et la valeur de chaque élément correspond à ce que l'IA sait de cette carte, cette valeur peut être :

- -1 : Cela signifie que l'IA sait que cette carte ne peut faire partie de la solution (car dans son jeux ou désapprouvée par un autre joueur).
- Fonction du type d'IA :
  - default AI : si l'IA ne sait rien d'une carte, sa valeur est mise à 0.
  - listening AI : si l'IA ne sait rien d'une carte, sa valeur est le nombre de fois que cette carte est apparue dans une suggestion ou accusation depuis le début de la partie.

2.3 Implémentation de GameMaster

Le gameMaster sera toujours le processus de rang 0, et sera donc toujours celui qui commencera (c-à-d que l'utilisateur commence toujours)

Le corps du programme se compose d'une boucle tournant jusqu'à ce que quelqu'un trouve la solution ou que l'utilisateur fasse une mauvaise accusation. Le code dans cette boucle se décompose en deux parties :

- C'est le tour de l'utilisateur, le gameMaster lui demande alors de faire une suggestion ou une accusation.
  - S'il fait une suggestion : On appelle une méthode qui "broadcastera" la suggestion à toutes les IAs.
    Puis qui verra successivement avec chacune d'entre-elles si elle peut la désapprouver (et s'arrêtera dés qu'il y aura une désapprobation)
  - S'il fait une accusation : On appelle une méthode qui vérifiera si la suggestion est correcte ou pas. C'est directement vérifié par le gameMaster, sans communications entre processus (Quelque soit la réponse, la partie s'arrête après et le gameMaster stoppe les autres IAs)

  Une fois que l'utilisateur à fini son tour, on préviens l'IA suivante (qui sera toujours celle de rang 1).
- C'est le tour d'une des IAs, le gameMaster (et l'utilisateur) attendent un message, qui peut être de plusieurs types :
  - type = -1 : On demande à l'utilisateur de répondre à une suggestion.
    Il renvoi soit le numéro de la carte qui désapprouve cette suggestion, soit -6 s'il ne peut la désapprouver.

  - type = -2 ou -6 : Une des IAs fait une suggestion(-2) ou une accusation(-6).
    On se contente d'afficher cette suggestion/accusation pour l'utilisateur, mais il ne faut rien y répondre.
  - type = -3 : Les IAs ont finies leurs tours, c'est à nouveau au tour de l'utilisateur.
  - type = -5 : On demande au gameMaster de répondre à une accusation.
    Le gameMaster terminera la partie si l'accusation est correcte, empêchera l'IA de jouer à nouveau sinon.

(Voir l'implémentation de GameMaster en annexe)

2.4 Implémentation de AI

L'IA a un fonctionnement assez semblable à celui du gameMaster.
On retrouve aussi la boucle principale avec les parties actives et passives (attente de messages):

- C'est le tour de l'IA, si l'IA n'a pas fait d'accusation erronée, elle peut alors jouer (sinon elle passe la main à sa voisine immédiatement).
  L'IA regarde alors combien de cartes "inconnues" il lui reste. Si elle n'en a que trois, elle fait une accusation, sinon elle fait une suggestion.
  - Si elle fait une suggestion : elle calcule une suggestion selon le type d'IA qu'elle est.
    Elle appelle ensuite une méthode qui "broadcastera" la suggestion à tous les joueurs.
    Puis qui verra successivement avec chacun d'entre-eux s'il peut la désapprouver (et s'arrêtera des qu'il y aura une désapprobation)
  - Si elle fait une accusation : Elle appelle une méthode qui "broadcastera" l'accusation à tous les joueurs.
    Puis qui verra avec le gameMaster si son accusation est correcte ou pas. Définissant si la partie s'arrête ou si l'IA ne peut plus jouer.

  Une fois qu'elle a fini son tour, elle préviens l'IA suivante.

- C'est le tour d'un des autre joueurs, elle attend un message, qui peut être de plusieurs types :
  - type = -1 ou -2 ou -3 ou -6: L'IA réagit comme doit le faire l'utilisateur.
  - type = -4 : Le gameMaster indique que la partie est finie.

(Voir l'implémentation de AI en annexe)

2.5 Communications entre processus

En permanence on a un processus actifs tandis que les autres processus attendent un message.
Ce message est un array d'entier de taille deux, de la forme : message[2] = {"type du message", "rang de l'envoyeur"}
Lorsqu'il reçoit ce message un processus en attente utilise l'entier représentant le

"type du message" pour savoir comment réagir et dans qu'elles méthodes aller puis le "rang de l'envoyeur" pour savoir à qui répondre.
Les différents processus alternent alors les sends et les receives, en fonction de ce qui leur est demandé :

- Envoi d'une suggestion (ou d'une accusation) à tous les joueurs.
  Le joueur actif envoi alors l'array typeMessage[2] = { -2(suggestion) ou -6(accusation), _myRank } à tous les joueurs pour leur signaler qu'il va envoyer une suggestion/accusation.
  Puis leur envoi l'array suggestion[3] = { suggestion suspect, suggestion arme, suggestion lieu}.
  Ce message ne demande pas de réponse.

- Ensuite le joueur doit chercher un autre joueur qui puisse désapprouver cette suggestion.
  Il va donc envoyer à un joueur l'array typeMessage[2] = { -1, _myRank }.
  Le joueur actif attend alors la réponse de ce joueur, celui-ci ayant déjà reçu la suggestion, il peut directement calculer sa réponse et la renvoyer, via un array reply = { -1 ou le numéro de la carte qui désapprouve}.

- Le joueur courant peut aussi faire une accusation. Sur le même principe que pour les suggestions, il envoi l'array typeMessage[2] = { -5, _myRank} au gameMaster.
  Celui-ci a déjà reçu l'accusation et calcule donc directement la réponse qu'il renvoi avec l'array reply = { 0 (accusation correcte) ou -1 (accusation incorrecte)}.

- En cas d'accusation correcte, le gameMaster envoie ensuite l'array typeMessage[2] = { -4, 0 (Une IA gagne) ou 1 (l'humain gagne)} pour signaler que le jeu s'arrête. Cela ne demande donc pas non plus de réponse.
  Rem : Le deuxième paramètre n'est la que pour afficher un message de fin de partie personnalisé

- Le dernier type de communication a lieu quand un joueur a fini, il envoie alors l'array typeMessage[2] = { -3, _myRank} au joueur suivant.
  La encore ce message ne demande pas de réponse.

Rem : Nous utilisons aussi des broadcast pour l'initialisation du système dans la classe Cleudo, pour envoyer le nombre de joueur voulu par l'utilisateur ainsi que le type d'IA voulu.
Nous n'utilisons plus de broadcast par la suite pour envoyer les suggestion/accusation, car tous les processus n'étant pas à l'écoute (on ne lance que le nombre de processus(// IA) voulu par l'utilisateur) cela aurait nécessité la création d'un communicator dédié, ce qui nous semblait assez lourd pour un gain assez négligeable.
(Voir diagramme de séquence en annexe)

3 Types d'IAs

3.1 Implémentation

L'utilisateur a, au démarrage, le choix entre trois types d'IA :

Default AI : Une IA choisissant les suggestions aléatoirement.

Listening AI : Cette IA prend en considération les suggestions faites par les autres utilisateurs pour faire les siennes.
Chaque fois qu'une suggestion est faite, elle incrémente les valeurs des éléments du vecteur _AiKnowledge correspondants aux trois cartes de la suggestion.
Elle sait donc précisément combien de fois chaque carte a été essayée.

Or on sait que pour chaque suggestion une des carte a été désapprouvée. Chaque carte a donc une probabilité d'un tiers de ne pas faire partie de la solution.
Les cartes les plus souvent "jouées" ont donc une plus grand probabilité de ne pas faire partie de la solution.

L'IA va donc prioritairement choisir les cartes les moins "jouées".
En pratique elle prendra pour faire ces suggestions les cartes dont la valeur correspondante dans le vecteur _AiKnowledge est le plus faible (ou s'il y a égalité, choisira aléatoirement une de ces cartes).

Mix AI : Les IA de rang paires utilisent default AI, les autres utilisent listening AI.

3.2 Statistiques

En faisant jouer les IAs 250 parties entre-elles, on obtient ces statistiques, sur le nombre de tours nécessaires aux IAs pour trouver la bonne solution :

		2 IA	3 IA	4 IA	5 IA
Nombre moyen de tours	Default AI	7.27	8.79	9.61	9.40
	Listening AI	8.60	8.63	9.12	9.40
	Mix AI	7.95	9.95	9.70	10.23
Écart-type	Default AI	2.29	2.93	3.09	3.26
	Listening AI	2.12	2.80	3.26	3.32
	Mix AI	2.27	2.96	3.41	3.18

4 Résultats

On peut voir que le joueur humain a intérêt à jouer contre des listening AI s'il n'y en a que deux, celle-ci étant les moins efficaces.
Mais contre un nombre moyen d'IAs, les listening AI sont plus efficace que les autres et donc plus dures à battre. Elles profitent de l'augmentation du nombre de suggestions pour affiner leurs prédictions.

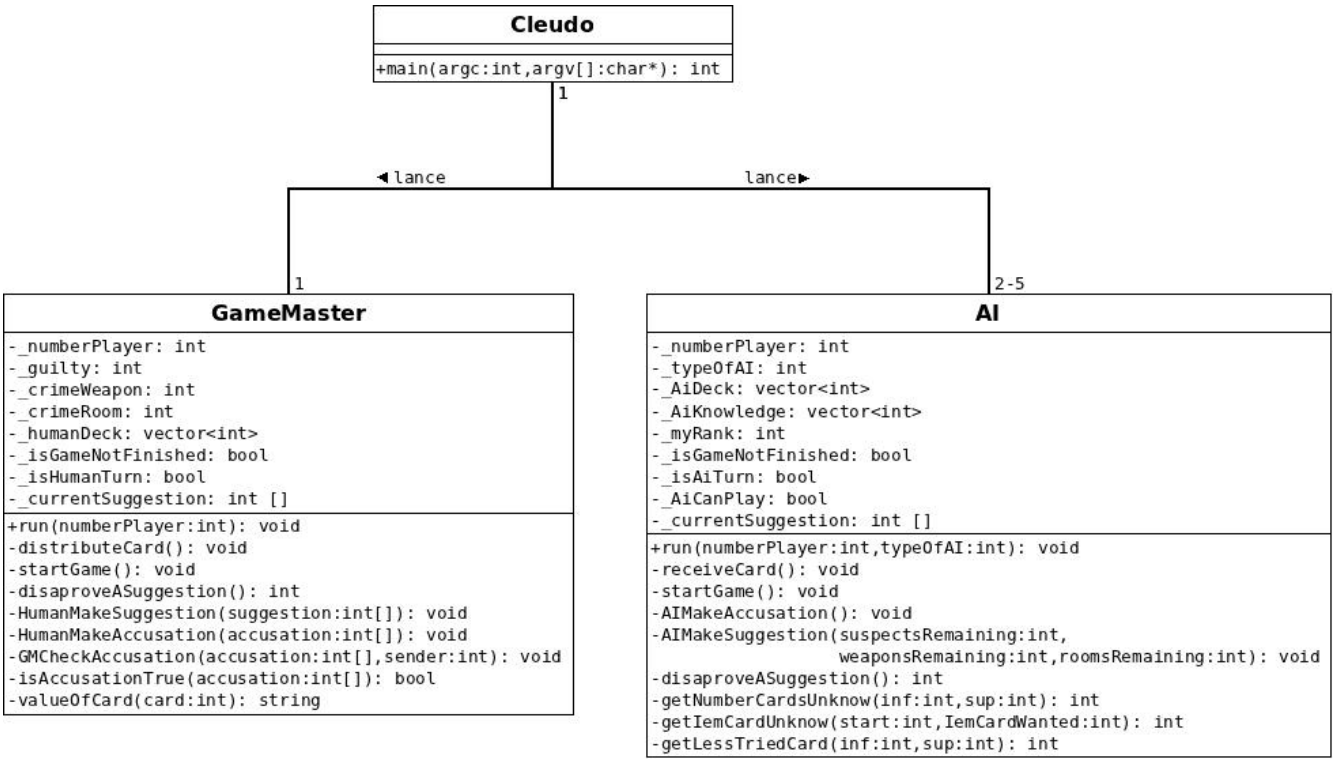
Par contre on peut noter que pour 5 IAs, les listening AI et default AI se valent. Le nombre d'IA fait que la probabilité qu'une d'entre-elles tombe aléatoirement sur la bonne réponse permet aux default AI de compenser l'échange d'information qui a lieu entre les listening AI.

Les Mix AI, s'avèrent quant-à-elle les moins efficaces, et offriront le challenge le plus faible.

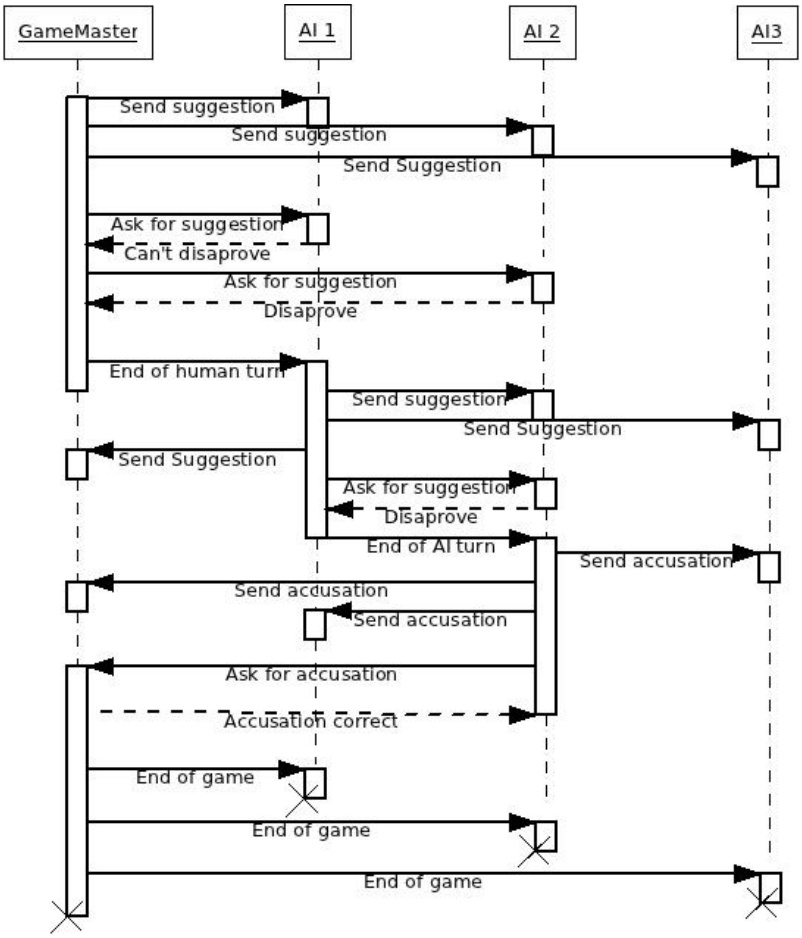
Enfin, on voit qu'aucune différence significative ne se fait jour au niveau des écarts-types.

5 Appendice

5.1 Diagramme de classe



5.2 Diagramme de Séquence



5.3 Implémentation de GameMaster

```
while(_isGameNotFinished)
{
  if(_isHumanTurn)
  {
    // On demande a l'utilisateur s'il veut faire une suggestion ou
    // une accusation
    // Puis on la recupere dans un array : int suggestion[3]
    if(suggestionType == 1) {
      HumanMakeSuggestion(suggestion);
    }
    else {
      HumanMakeAccusation(suggestion);
    }
    _isHumanTurn = false;
    int messageType[2] = { -3, 0 };
    MPI_Send(&messageType, 2, MPI_INT, 1, 0, MPI_COMM_WORLD);
  }
  else {
```

```
19      int typeMessage[2];
20      MPI_Recv(&typeMessage, 2, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
21      if(typeMessage[0] == -1){
22        int reply = disaproveASuggestion();
23        MPI_Send(&reply, 1, MPI_INT, typeMessage[1], 0, MPI_COMM_WORLD);
24      }
25      else if(typeMessage[0] == -2){
26        MPI_Recv(&_currentSuggestion, 3, MPI_INT, typeMessage[1], 0,
              MPI_COMM_WORLD, MPI_STATUS_IGNORE);
27      }
28      else if(typeMessage[0] == -3){
29        _isHumanTurn = true;
30      }
31      else if(typeMessage[0] == -5){
32        int accusation[3];
33        MPI_Recv(&accusation, 3, MPI_INT, typeMessage[1], 0, MPI_COMM_WORLD,
              MPI_STATUS_IGNORE);
34        MJcheckAccusation(accusation, typeMessage[1]);
35      }
36      else // typeMessage[0] == -6
37      {
38        MPI_Recv(&_currentSuggestion, 3, MPI_INT, typeMessage[1], 0,
              MPI_COMM_WORLD, MPI_STATUS_IGNORE);
39      }
40    }
41  }
```

5.4 Implémentation de AI

```
1   while(_isGameNotFinished)
2   {
3     if(_isAITurn){
4       if(_IACanPlay){
5         // We compute the number of card unknow for each type.
6
7         if(suspectsRemaining == 1 and weaponsRemaining == 1 and roomsRemaining
              ==1){
8           IAMakeAnAccusation();
9         }
10        else{
11          // otherwise, the IA dont know the answer and make a suggestion
12          IAMakeASuggestion(suspectsRemaining, weaponsRemaining, roomsRemaining
                );
13        }
14      }
15    }
16    else {
17      int typeMessage[2];
18      MPI_Recv(&typeMessage, 2, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
19
20      if(typeMessage[0] == -1){
21        int reply = disproveASuggestion();
22        MPI_Send(&reply, 1, MPI_INT, typeMessage[1], 0, MPI_COMM_WORLD);
23      }
24      else if(typeMessage[0] == -2) // One of the players brodcast a suggestion
            or accusation
25      {
26        MPI_Recv(&_currentSuggestion, 3, MPI_INT, typeMessage[1], 0,
              MPI_COMM_WORLD, MPI_STATUS_IGNORE);
27        if(_typeOfIA == 1){
```

```
28            // The IA save this information if it's a listening IA (they use it
                  to compute suggestions)
29        }
30      }
31      else if(typeMessage[0] == -3){
32        _isAITurn = true;
33      }
34      else if(typeMessage[0] == -4){
35        _isGameNotFinished = false;
36      }
37      else // typeMessage[0] == -6
38      {
39        MPI_Recv(&_currentSuggestion, 3, MPI_INT, typeMessage[1], 0,
              MPI_COMM_WORLD, MPI_STATUS_IGNORE);
40        if(_typeOfIA == 1){
41            // The IA save this information if it's a listening IA (they use it
                  to compute suggestions)
42        }
43      }
44    }
45  }
```

5.5 Output des programmes

```
[pdagnely@nic143 Cleudo-Final]$ mpirun -mca btl ^openib -np 6 game
How much IA do you want ? (between 2 and 5)
2
Choose the type of AI that you want :
 0 = Default AI         1 = Listening AI              2 = Mix AI
2
You receive the cards :Lounge (20)  Library (17)  Kitchen (12)  Dining room (13)  Mrs. Peacock (4)  Hall (19)
Do you want to make an accusation (type 0) or a suggestion (type 1)  ?
1
Your cards :Lounge (20)  Library (17)  Kitchen (12)  Dining room (13)  Mrs. Peacock (4)  Hall (19)
Liste of the cards
Suspects                    Weapons                          Rooms
--------------------------------------------------------------------------------
0 : Miss scarlet            6 : Candlestick                  12 : Kitchen
1 : Colonel Mustard         7 : Knife                        13 : Dining room
2 : Mrs. White              8 : Lead pipe                    14 : Ballroom
3 : Reverend Green          9 : Revolver                     15 : Conservatory
4 : Mrs. Peacock            10 : Hangman's knot              16 : Billiard room
5 : Professor Plum          11 : Spanner                     17 : Library
                                                             18 : Study
                                                             19 : Hall
                                                             20 : Lounge
--------------------------------------------------------------------------------
Choose the suspect
0
Choose the weapon
6
Choose the room
14
AI 1 refute your suggestion for the card Ballroom (14)
AI 1 make a suggestion for Mrs. Peacock (4) , Hangman's knot (10) , Lounge (20)
you can disaprove this suggestion with some of your card, which one would you use ?
Lounge(20)  Mrs. Peacock(4)  ?
4
AI 2 make a suggestion for Mrs. Peacock (4) , Revolver (9) , Conservatory (15)
this suggestion is disaproved with one of your card : Mrs. Peacock (4)
Do you want to make an accusation (type 0) or a suggestion (type 1)  ?
1
Your cards :Lounge (20)  Library (17)  Kitchen (12)  Dining room (13)  Mrs. Peacock (4)  Hall (19)
```