— Algorithmique & Programmation – D.E.S.S. Imoi —

Synthèse du projet

N'ayant pas obtenu avant la semaine dernière la seconde partie du projet de certains des étudiants, je n'ai pas encore pu corriger celle-ci. Cependant, la correction de la partie algorithmique me permet d'avoir une idée assez précise des méthodes qui ont été utilisées. Il me semble important, avant de lire le corrigé que je propose, de comprendre comment il était possible de parvenir à la solution mise en œuvre dans celui-ci.

1 Partie algorithmique

1.1 Gestion d'une mémoire des demandes

Le problème majeur de ce projet était la gestion de la mémoire des cartes demandées par chacun des autres joueurs. Cette gestion se résume principalement à deux actions : ajouter une donnée (couple carte-joueur) et consulter une donnée (obtenir le joueur associé à une carte, s'il existe). Le point clé est que la première de ces actions sera effectuée par la partie, alors que la seconde sera effectuée par un des joueurs-ordinateur intelligents.

Le graphe de dépendance permet de constater que la classe définissant la partie dépend de celle définissant les joueurs-ordinateur intelligents. Le plus simple est par conséquent de définir la mémoire au sein des joueurs-ordinateur, et de la rendre accessible à la partie en définissant une méthode publique d'ajout de donnée. L'autre solution (malheureusement choisie par la quasi-totalité des étudiants), nécessite plus de travail : il faut alors passer en paramètre des méthodes la mémoire des demandes, ce qui implique de modifier les méthodes virtuelles définies dans la classe abstraite commune à tous les joueurs.

La mise en œuvre de la mémoire n'est pas vraiment un problème : il s'agit en fait de choisir entre rapidité et place occupée en mémoire vive. Ce choix n'est finalement pas très important ici. Par contre, l'utilisation de cette mémoire pour définir les méthodes `cherche` et `demande`, imposées par la classe des joueurs dans la classe des joueurs-ordinateur intelligents, a été plus difficile.

1.2 Définition du joueur-ordinateur intelligent

Tout d'abord, on peut remarquer qu'il aurait été plus judicieux de définir pour les joueurs une unique méthode donnant la carte cherchée et le numéro du joueur auquelle elle est demandée. En effet, si les choix de la carte et du joueur peuvent être dissociés pour les joueurs-utilisateur et les joueurs-ordinateur simples, ils ne sont pas sensés l'être pour les joueurs-ordinateur intelligents (problème de cohérence). *Mea culpa*, j'aurais dû me consacrer plus tôt à la partie algorithmique du projet, et corriger cela avant le TP 4.

.../...

Dans cette situation, pour garantir la cohérence entre le choix de la carte et celui du joueur, et pour éviter de faire deux recherches, chaque joueur-ordinateur intelligent mémorise la carte et le numéro du joueur trouvés lors de la précédente recherche, ainsi que deux champs booléens indiquant si ces données ont été consultées. On remet à jour conjointement les deux données dès qu'on a besoin de l'une d'entre elle qui n'est plus à jour (qui a déjà été consultée). **Seul inconvénient**, les deux méthodes `cherche` et `demande` ne laissent plus constant l'objet sur lequel elle sont appelées. Cette modification est cependant plus minime que celle imposée par l'autre solution (ajout d'un paramètre correspondant à la mémoire des demandes, paramètre non utilisé pour les autres joueurs).

1.3 Traitement du numéro du joueur demandé

Pour finir, considérons le problème de l'interprétation du numéro du joueur auquel on demande une carte. Ce problème n'a pas été convenablement traité par la plupard des étudiants, alors qu'il date du TP 4. Le problème consiste à obtenir un numéro parmi n joueurs, en vérifiant que ce numéro n'est pas celui du joueur auquel on en fait la demande. Le plus simple me semblait être de demander un nombre entre 1 et $n-1$, puis d'ajouter ce nombre au numéro du joueur auquel on a fait la demande, et de prendre le résultat modulo n sur l'intervalle $\{1,\ldots,n\}$. Peu d'étudiant semblent avoir compris que faire du nombre retourné (compris entre 1 et $n-1$).

Quoi qu'il en soit, le traitement étant le précédent et la mémoire des demandes contenant le numéro du joueur ayant une carte, un joueur-ordinateur intelligent doit déterminer la distance entre ce joueur et lui-même, pour que le traitement précédent rétablisse correctement le numéro voulu. Par conséquent, le joueur-ordinateur doit connaître son numéro, d'où un constructeur un peu différent de celui des joueurs-utilisateur et des joueurs-ordinateur simples.

2 Partie programmation

La réflection précédente permet d'aborder simplement la partie programmation de ce projet. Les joueurs-ordinateur intelligents ayant le même comportement de mémorisation, il peuvent partager la même mémoire des demandes. Celle-ci est donc définie comme un champ `static` dans la classe `intell_ordi`. Les méthodes de remise à zéro, d'ajout de donnée et de libération de cette mémoire sont elles aussi statiques. Le reste est très classique.

La modification de la classe partie se limite à quelques lignes du fichier source :

- il faut libérer la mémoire des demandes lors de la destruction de la partie (la méthode `mariage::~mariage()` se termine par un appel à `intell_ordi::libere_memoire();`);
- avant de distribuer les cartes pour une nouvelle partie, il faut créer des joueurs-ordinateur non plus simples mais intelligents, et remettre à zéro la mémoire des demandes ;
- enfin, lorsqu'on joue le tour d'un joueur, la demande de ce joueur doit être non seulement affichée mais aussi ajoutée à la mémoire des demandes.