Programmation Objet
TP C++

Didier Lime

EI3 Info – 2012-2013

Le but du TP est de programmer le jeu classique « Pierre, Feuille, Ciseaux » aussi connu sous le nom de *Shifumi*. Ce texte est sciemment trop long. Au minimum, il faudrait réussir à faire les neuf premières questions.

1 Partie 1 : Les règles du jeu

Le jeu se joue à deux. Chaque joueur choisit simultanément un élément parmi Pierre, Feuille et Ciseaux. Le joueur gagnant est déterminé par les règles suivantes :

1. La Pierre bat les Ciseaux, en les brisant ;
2. La Feuille bat la Pierre, en l'enveloppant ;
3. Les Ciseaux battent la Feuille, en la coupant.

Le joueur qui bat l'autre gagne un point. Le vainqueur est celui qui atteint le premier un nombre de points fixé à l'avance.

Question 1. Définir une classe `Coup` modélisant en général un coup possible, et, **par héritage**, les classes `Pierre`, `Feuille` et `Ciseaux`. Pour différencier ces trois types de coups vous utiliserez une méthode virtuelle `type` renvoyant un objet de type `std::string`. Les valeurs possibles renvoyées seront p. ex. `"Pierre"`, `"Feuille"` et `"Ciseaux"` selon la classe.

Question 2. Définir les opérateurs `==`, `<` et `<<` pour la classe `Coup`.

Si les qualificateurs `const` *posent des problèmes cherchez où vous en avez oublié plutôt qu'où vous pouvez en enlever !*

Question 3. Définir une classe `Joueur` contenant une méthode `obtenir_coup` qui renvoie un pointeur (ou une référence) sur un `Coup`. Les différents types de joueurs (Humain ou Intelligences Artificielles (IA)) seront dérivés de cette classe.

Question 4. Ajouter un attribut de type pointeur ou référence sur un `Joueur` afin de mémoriser, pour chaque coup, le joueur qui l'a joué.

Question 5. Définir une classe `Partie` contenant (au moins) deux joueurs, une méthode `jouer_tour` simulant un tour de jeu (avec mise à jour du score) et une méthode `afficher_scores`.

2 Partie 2 : Joueurs et Intelligence artificielle

Question 6. Dériver la classe `Joueur` pour permettre l'intéraction avec un joueur humain. La méthode `obtenir_coup` sera surchargée et utilisera `std::cin` pour récupérer les coups du joueur.

Question 7. Écrire une fonction `main` mettant une partie en place. Maintenant le programme doit être fonctionnel et permettre le jeu à deux joueurs humains.

Question 8. Ajouter dans chaque classe concrète représentant un coup, un attribut de classe (`static`) appelé p.ex. `nombre` et permettant de compter les instances de ces classes construites depuis le début de la partie. Modifier le programme pour afficher ces nombres à l'issue de la partie.

Question 9. Dériver la classe `Joueur` pour programmer une IA jouant aléatoirement.

Dans l'absolu, face à l'IA de la question 7, il n'y a pas de stratégie permettant d'assurer en moyenne un gain supérieur à 50% des parties. Cependant, les êtres humains ne jouent jamais complètement au hasard. Inconsciemment (ou pas) ils ont tendance à produire et à répéter des séquences. On peut tirer parti de ce fait pour produire une IA qui obtient de (très) bons résultats contre les humains.

Supposons que l'on dispose de l'historique des coups joués jusque là. Par exemple : `cFcFfFpCcF`. `p` désigne la pierre, `f` la feuille et `c` les ciseaux, les majuscules et les minuscules différenciant le joueur 1 du joueur 2. Ici, on remarque qu'après qu'il a joué `F`, le joueur 2 a joué deux fois `F`, une fois `C` et aucune fois `P`. Puisqu'il a joué `F` la dernière fois, on peut supposer qu'il y a deux chances sur trois qu'il joue à nouveau `F` et une sur trois qu'il joue `C`. On va donc choisir un coup aléatoirement selon cette distribution de probabilités.

Question 10. Dériver la classe `Joueur` pour programmer une IA implémentant cette idée. Pour modéliser l'historique, on pourra p. ex. utiliser une `std::list<Coup*>` ou (plus facile) une `std::string` et profiter de la méthode `substr`. Dans le dernier cas, il faudra ajouter une méthode `void memorise(const Coup&)` qui code le coup sous forme d'un caractère et l'ajoute à l'historique.

Dans la question précédente, on s'est basé sur un contexte de taille 1, c'est-à-dire uniquement sur le dernier coup joué. Pour affiner l'analyse on peut regarder ce qu'avait joué le joueur 2, après avoir joué `F` sachant que le joueur 1 avait joué `c` (contexte de taille 2). Cette fois on trouve deux fois `F` uniquement. On va donc jouer `F`.

L'extension naturelle de cette idée est d'étendre la taille du contexte de recherche autant que possible (tant qu'on trouve des occurrences de la sous chaîne correspondante dans l'historique). Une fois que l'on a trouvé le contexte de taille maximale, on détermine la distribution de probabilité en fonction des coups joués après chaque occurrence du contexte dans l'historique.

Question 11. Modifier la classe précédente pour effectuer le choix du coup en fonction d'un contexte de taille maximale.

3 Partie 3 : Améliorations optionnelles

Question 12. Pour que l'IA de la question 9 fonctionne bien il lui faut un historique de grande taille. Modifier le programme pour qu'il demande son nom au joueur humain et sauve l'historique de la partie dans un fichier associé à ce nom. En cas de nouvelle partie du même joueur, le fichier historique de ce joueur sera chargé.

Question 13. Modifier le programme pour pouvoir jouer à plus de deux joueurs. Les règles suivantes sont alors appliquées :

- Si les trois coups possibles sont représentés parmi les coups de tous les joueurs, on ne garde que ceux qui sont en majorité, les autres sont éliminés (s'il n'y a pas de majorité, on recommence) ;
- Si seuls deux des coups possibles sont représentés, le jeu est résolu selon les règles de base, les battus sont éliminés ;
- On recommence jusqu'à ce qu'il n'y ait plus qu'un seul joueur en jeu. Ce joueur gagne un point.