L2 Info UPPA
Mai 2018

Gaëtan Croquefer
Antoine Chantelou

Projet de Programmation Orienté Objet

Sommaire

Introduction

Le but de ce projet est de réaliser un programme gérant un jeu de poker de type no limit texas-hold'em sans prise en compte des jetons. Le langage de programmation est le C++ avec la contrainte de ne pas utiliser de bibliothèque externe.

L'objectif étant de mettre a contribution les connaissances apprise en cours de manière concrète dans la réalisation d'un projet en binôme.

Déroulement du projet

Afin de réaliser le projet nous avons commencé par déterminer les étapes pour la conception du projet, quelques règles de programmation, la répartition des tâches ainsi que les priorités.

Nous avons utilisé GitHub pour centraliser nos travaux.

Afin de créer le jeu de carte en lui même, il nous a fallu décider de comment stocker les informations. Nous avons donc créé la classe *Vecteur,* sous forme de *template,* ce qui nous a permis de gérer à un seul endroit la mémoire quelque soit le type de donnée (cartes, joueurs...).

Ensuite nous avons créé la classe carte pour définir chaque carte, la classe deck, qui est le jeu de carte, la classe joueur qui contient les joueurs, et enfin la classe outils.

Toutes ces classes sont définies (fichier.h) et implémentées (fichier.cpp) dans différents fichiers afin de rendre le projet le plus modulaire possible. Toutes les inclusions ont été faites dans les *headers* (fichier.h) de manière a simplifier et à éviter les doubles inclusions.

Afin de simplifier le développement du programme nous avons rapidement été amené a créer un mode DEBUG pour faciliter les tests.

Nous avons fait un *makefile* pour la compilation, et utilisé le maximum de connaissances du cours telles que la *forme canonique de Coplien*, les *templates*, la *surcharge d'opérateur*, les *fonctions « amies »*... Mais nous n'avons pas utilisé *l'héritage*, par exemple, car nous n'en avons pas trouvé l'utilité.

Les différentes classes et leurs fonctions sont décrites en annexe.

Fonctionnement du programme

Le programme commence par demander le nombre de joueurs (entre 2 et 10), puis affiche le déroulement d'un tour étape par étape (preflop, flop, river, turn), une fois le tour terminé, l'utilisateur est invité à décider s'il souhaite recommencer un tour ou non.

Si, lorsque le programme demande le nombre de joueurs on entre 42, le mode DEBUG se lance. Dans ce mode l'utilisateur rentre manuellement les valeurs (1-13) et familles (1-4) des cartes qui composeront le board (jusqu'à un maximum de 7 cartes).

Ensuite le board et la meilleure combinaison parmi les cartes qui le constituent sont affichés.

Conclusion

En définitive ce sujet a été une bonne mise en pratique des connaissances abordées lors des cours et TD/TP. Les plus gros challenges ont été la création de la classe vecteur et la gestion des combinaisons, notamment les quintes. Cependant, l'intérêt que le sujet a suscité nous a permis de surmonter ces difficultés et, même si cela n'était pas demandé dans le projet pour des raisons évidentes de temps, il aurait été intéressant d'introduire la gestion des mises et jetons.

Annexe

Descriptions des différents fichiers et fonctions

vecteur.h :

Ce fichier implémente la classe vecteur, c'est une template, qui sert à stocker les données.

Attibuts privés :

- **Type* _tab :** Tableau qui contient les éléments.
- **int _size :** Entier correspondant à la taille du tableau.

Attibuts public :

- **Vecteur() :** Constructeur par défaut.
- **Vecteur(const Vecteur<Type>& v):** Constructeur par recopie.
- **~Vecteur() :** Destructeur.
- **Vecteur<Type>& operator = (const Vecteur<Type>& v):** Opérateur de recopie.
- **Type operator [] (int index) const :** Permet d'accéder a l'élément a l'indexe donné en lecture.
- **Type& operator [] (int index) :** Permet d'accéder a l'élément a l'indexe donnée en écriture.
- **void resize(int n) :** Permet de redimensionner le tableau a la taille donnée.
- **void push_back(Type t) :** Agrandit le tableau et ajoute l'élément a la fin.
- **Type pop_back() :** Rétrécit le tableau en enlevant le dernier élément et le retourne.
- **int size() const :** Retourne la taille du tableau.

deck.h

Ce fichier implémente la classe deck.

Attributs privés :

- **Vecteur<Carte> _deck :** Vecteur qui contient les cartes du deck.

Attributs public :

- **Deck()** : Constructeur par défaut.
- **Deck(const Deck& j)** : Constructeur par recopie.
- **~Deck() :** Destructeur par défaut
- **Deck& operator = (const Deck& j) :** Opérateur de recopie.
- **void shuffle() :** Mélange le paquet (vecteur) de carte.
- **Carte draw :** "Piocher", enlève la première carte du deck et la renvoie.
- **friend std::ostream& operator << (std::ostream& os, const Deck& c) :** Fonction ami, qui redéfini l'opérateur << pour l'affichage.

joueur.h

Ce fichier implémente la classe joueur.

Attibuts privés :

- **std::string nom :** Nom du joueur.

Attributs public :

- **Joueur() :** Constructeur par défaut.
- **Joueur(const Joueur& j) :** Constructeur par recopie.
- **~Joueur() :** Destructeur.
- **Joueur& operator = (const Joueur& j) :** Opérateur de recopie.
- **Joueur(std::string nom) :** Constructeur surchargé initialisant avec le nom.
- **void draw(Deck& deck) :** Pioche 2 cartes dans le deck passé en paramètre et les ajoute à la main du joueur.
- **Vecteur<Carte> getMain() const :** Renvoie la main du joueur.
- **friend std::ostream& operator << (std::ostream& os, const Joueur& j) :** Fonction ami, qui redefini l operateur << pour l affichage.

carte.h

Ce fichier implémente la classe carte.

Attributs privés :

- int famille : Famille de la carte (1=trèfle, 2=pique 3=carreau, 4=cœur).
- **int valeur :** Valeur de la carte (1=as, 11=valet, 12=dame, 13=roi).

Attributs public :

- **Carte() :** Constructeur par défaut.
- **Carte(const Carte& c) :** Constructeur par défaut.
- **~Carte() :** Destructeur
- **Carte& operator = (const Carte& c) :** Opérateur de recopie.
- **Carte(int valeur, int famille) :** Constructeur surchargé instanciant la carte a la bonne valeur et famille.
- **int getValeur() const :** Retourne la valeur de la cartes.
- **int getFamille() const :** Retourne la famille de la cartes.
- **std::string afficherFamille() const :** Renvoie une chaîne de caractères correspondant à la famille de la carte.
- **std::string afficherValeur() const :** Renvoie une chaîne de caractères correspondant à la valeur de la carte.
- **std::ostream& operator << (std::ostream& os, const Carte& c) :** Redéfini l'opérateur << pour l'affichage.

outil.h

Ce fichier contient le menu debug et les différentes fonctions pour lire le flux d'entrée.

- **Void debugMode() :** Lance le menu de debug.
- **unsigned int readUnsignedInt() :** Demande a l''utilisateur un entier non signé, pour éviter les erreurs de lecture.
- **bool readBool() :** Demande à l'utilisateur un booléen et permet d'éviter les erreurs de lecture.
- **std::string intToString(int number) :** Convertis un entier donné en son équivalent en caractères.

combinaison.h

Ce fichier contient les menus d''affichage des combinaisons et des cartes ainsi que diverse fonctions utile pour la gestion des combinaisons.

- **void afficherCombinaison(const Vecteur<Carte>& cartes)** : Affiche la meilleur combinaison dans les cartes données en paramètre et appelle toutes les fonctions de vérification dans leur ordre hiérarchique. S'arrête lorsque l'une d'elle est trouvée ou affiche la carte haute.
- **void afficherKicker(const Vecteur<Carte>& cartes, Vecteur<int> valeurExclues, int number)** : Affiche le kicker parmi les cartes passées en paramètre, excepté les cartes dont la hauteur se trouve dans valeurExclues (on exclue du kicker les cartes composant la combinaison).
//Affiche un nombre de kicker égal au paramètre "number"

- **void afficherCarteHaute(const Vecteur<Carte>& cartes)** : Affiche la carte haute et 4 kickers.
- **Carte trouverCartesEgales(const Vecteur<Carte>& cartes, int number, int valeurExclue)** : Renvoie la hauteur d'une combinaison composée de "number" cartes si possible, sinon renvoie la carte par défaut.
- **Carte trouverCarteHaute(const Vecteur<Carte>& cartes, const Vecteur<int>& valeursExclues=Vecteur<int>()) :** :Renvoie la carte de hauteur la plus haute parmi les cartes données, excepté les cartes dont la hauteur se trouve dans valeurExclues.

- **bool verifierQuinteFlush(const Vecteur<Carte>& cartes).**
- **bool verifierCarre(const Vecteur<Carte>& cartes).**
- **bool verifierFull(const Vecteur<Carte>& cartes).**
- **bool verifierCouleur(const Vecteur<Carte>& cartes).**
- **bool verifierQuinte(const Vecteur<Carte>& cartes).**
- **bool verifierBrelan(const Vecteur<Carte>& cartes).**
- **bool verifierDoublePaire(const Vecteur<Carte>& cartes).**
- **bool verifierPaire(const Vecteur<Carte>& cartes).**

Pour toute les fonction précédentes : Vérifie si la combinaison concernée (Quinte Flush, Carré, Full, Couleur etc..) est présente dans les cartes passées en paramètre, affiche puis retourne vrai si la combinaison s'y trouve. Retourne faux sinon.

jeu.h

Ce fichier contient les différentes actions de la partie : piocher une carte, afficher les cartes, jouer un tour...

- **void jouerTour(const Vecteur<Joueur>& joueurs) :** Affiche les résultat d'un tour de jeu.
- **void drawStep(const Vecteur<Joueur>& joueurs, Deck& deck) :** Appelle la fonction draw sur chacun des joueurs avec le deck donné.
- **void afficherTable(const Vecteur<Carte>& table) :** Affiche les cartes présente sur le board.
- **void afficherJoueurs(const Vecteur<Joueur>& joueurs, const Vecteur<Carte>& table)** :
Affiche le nom des joueurs, le contenu de leur main et la meilleure combinaison qu'ils ont avec le board passé en paramètre.