Projet Poker

Membres : De Backer Valentin, Broutel Colin, Pernin Clément

Classe : ADI 2.3

Contexte

Après quelques cours d'introduction et de pratique du C/C++, il nous a été demandé de créer un programme permettant de trouver la combinaison la plus forte dans une main de 5 cartes lors d'une partie de poker style « Texas Hold-em », puis de présenter notre production au travers d'un oral et d'un compte rendu le Vendredi 2 décembre.

Contraintes

Pour la réalisation de ce projet, il nous a été donné un cahier des charges :

- Trouver la meilleure combinaison dans une main selon les règles du "Texas Hold-em"
- Réaliser le programme en C++
- Utiliser la méthode de la programmation orientée objet (ou OOP)

Nous étions libres de réaliser la structure du programme comme bon nous semblait tant que les contraintes précédemment évoquées étaient respectées.

Notre solution

La stratégie

Avant de commencer à coder nous avons commencé par établir notre stratégie d'approche du problème

Nous avons d'abord regardé les règles du jeu que nous étions censés programmer.

Chaque joueur obtient une main composée de 5 cartes. En fonction de leur valeur et couleur, on peut en déterminer des combinaisons. Ces combinaisons sont listées et classées de la plus forte à la plus faible dans ce tableau :

Nous avons ensuite listé les différents outils que nous allions avoir à utiliser, à savoir :

- Une fonction qui **crée une main**
- Une fonction qui **trie les cartes** de la main suivant leur valeur (pour pouvoir plus facilement identifier les combinaisons)
- Une fonction qui **compare les couleurs** (pour différencier une couleur, une quinte flush ou une quinte flush royale du reste)
- Une fonction qui **compare les valeurs** (pour déterminer les paires, suites ou brelan et la carte maximale de chaque main)

Puis, nous avons réfléchi d'une manière plus informatique en commençant à écrire en français ou en pseudo-code l'algorithme des différentes fonctions que nous allions utiliser. Ces derniers ont ensuite continué à évoluer au fur et à mesure que le programme avançait.

valactu = valcarte

pair = 0

brelan = 0

carre = 0

pour chaque elm si tab[elm]=tab[elm+1] | alors chaine(init à 1) = chaine + 1

si chaine = 2 : pair(init à 0) = pair + 1

si chaine = 3 : brelan(init à 0) = brelan + 1

si chaine = 4 : carre(init à 0) = carre + 1

tests #----------

si carre = 1 : les autres = 0 || si brelan = 1 : carre = 0 || si carre ou brelan > 1 alors error

---

1ère version de l'algorithme de détection de combinaison

Après avoir préparé notre stratégie vis-à-vis du problème, nous avons tenté de coder le programme de deux manières :

D'abord une version **fonctionnelle**, puis une version **orientée objets**

L'approche fonctionnelle

Nous avons dans un premier temps décidé d'aborder le programme en utilisant de la programmation "fonctionnelle", soit un ensemble de variables et de fonctions appelées dans un *main*. Le but était d'abord de réussir à créer un programme marchant comme prévu en utilisant la méthode de programmation que nous avons l'habitude d'utiliser dans les autres projets.

Ce programme correspond au fichier « Poker.cpp » présent dans le zip envoyé avec notre code. Pour être lancé seul, il faut le lancer avec le *debug* s'appelant « Poker ».

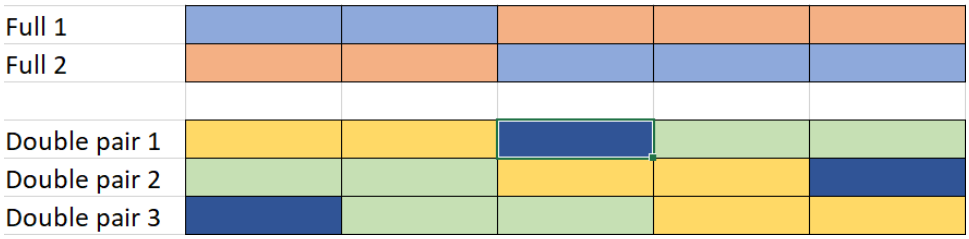
Le programme est organisé comme suit :

1) Définition des variables
2) Définition d'une main de 5 cartes dans un dictionnaire couleur/valeur. Les valeurs et couleurs de chaque carte peuvent être déterminés de manière arbitraires ou aléatoire. Les couleurs sont définies de 1-4 ou COEUR-TREFLE, et les valeurs de 1-14 ou UN-QUATOZE. La multiplicité des possibilités de définition est possible grâce à "enum"
3) On crée un tableau valeur et un tableau couleur à partir de la main

4) Fonctions
    a) Tri : tri en fonction de la valeur de chaque carte dans l'ordre décroissant (tout en gardant associé valeur et couleur), puis on *print* la main sous forme de tableau
    b) Full : comme les cartes sont triés on test si on est dans un des cas de figure du tableau ci-dessous
    c) Double paire : s'il n'y a pas de full, on cherche si on a une double pair suivant les possibilités listées dans le tableau ci-dessous
    d) Flush : On regarde si les 5 couleurs de la main sont identiques
    e) Quinte : On regarde si la valeur de la carte actuelle est la même que celle de la suivante+1, et si c'est valable pour toutes les cartes de la main alors on a une Quinte. Pour savoir si la Quinte est royale, on regarde si la valeur de la première carte vaut 14.
    f) Pair : On vérifie si on n'a ni full ni double pair dans la main, puis on compte le nombre de cartes identiques à la suite pour déterminer si on a une paire, un brelan, un carré ou si on a un cas impossible
    g) Forte : Si on n'a aucun des cas ci-dessus, on cherche la carte la plus forte de la main. Comme le tableau est trié c'est la première valeur de la main qui est la plus forte
    h) Main : c'est la boucle principale du programme. On y appel toutes nos fonctions dans un ordre logique pour que tout fonctionne. On commence par trier la main, puis on vérifie qu'aucune carte n'est identique, avant de commencer les

recherches de combinaisons. On regarde ensuite si on a une quinte flush royale, une quinte flush, un carré, un full, un flush, une quinte, un brelan, une double paire, une paire, ou une carte haute, avant de finalement afficher le résultat.

	Valeur	Couleur
Tri		
Full		
Dpaire		
Flush		
Quinte		
Pairs		
Forte		



Tableaux récapitulatifs du fonctionnement de chaque fonction

Cette version du programme est fonctionnelle et toutes les bases des algorithmes sont posées. Malheureusement cela ne répond pas entièrement aux contraintes car il nous fallait utiliser de la programmation (OOP), or, ce n'était pas le cas à cette étape du projet.

La version OOP

Nous avons tenté d'adapter notre code fonctionnel pour qu'il suive la méthode OOP.

Nous avons eu quelques problèmes avec la création de classes. Les paramètres à entrer sont des tableaux, mais nous n'avons pas réussi à pointer vers eux. Le code ne se lance pas à cause de certaines erreurs, mais la création de la classe et des fonctions de celui-ci fonctionne.

- Création de pointeurs vers valeur et couleur qui sont créés dans le .h
- Appel des fonctions créer dans le .h

Ce programme correspond aux fichiers « Pokerkoul.cpp » et « defpoker.h » présent dans le zip envoyé avec notre code. Pour être lancé ensemble, il faut l'exécuter avec le *debug* d'appelant « Pokerkoul ».

Difficultés

Le projet ne correspond, à l'heure actuelle, pas entièrement à ce qu'il nous a été demandé.

On peut bel et bien trouver la combinaison la plus forte d'une main suivant les règles du "Texas Hold'em", et le tout a bien été réalisé en C++. Mais comme vu précédemment, l'approche "orientée objet" ne fonctionne pas complètement car nous n'arrivons pas à appeler les méthodes avec un tableau en paramètre.

Ce problème est dû à plusieurs raisons :

- Nous avons été assez désorganisés au début.

Nous nous sommes perdus de nombreuses fois au sein du code car le fait de coder tous à la fois nous faisait perdre le fil, nous passions donc au début beaucoup plus de temps que nécessaire à comprendre ou débugger certaines parties du code écrites par d'autres membres de l'équipe.

- Ce projet est aussi le premier que nous avons réalisé en utilisant du C++, nous ne savions donc pas comment bien gérer les classes. Nous avions vu le concept en cours, mais son application à un cas réel est bien plus compliquée qu'il n'y paraît.

Les points positifs

- Notre code renvoie les valeurs attendues en fonction de cas précis

- Nous avons un système de création de main aléatoire ce qui permet de créer du réalisme dans notre code. Nous avons malgré tout gardé des mains définies arbitrairement pour simuler des cas précis qui n'arrivent que rarement dans une distribution aléatoire (une quinte flush royale par exemple)

Ce que nous avons appris

- Développer notre organisation et notre logique de programmation
- Appliquer à un problème donné les connaissances théoriques du C/C++
- Réaliser un projet de programmation à plusieurs

Le code

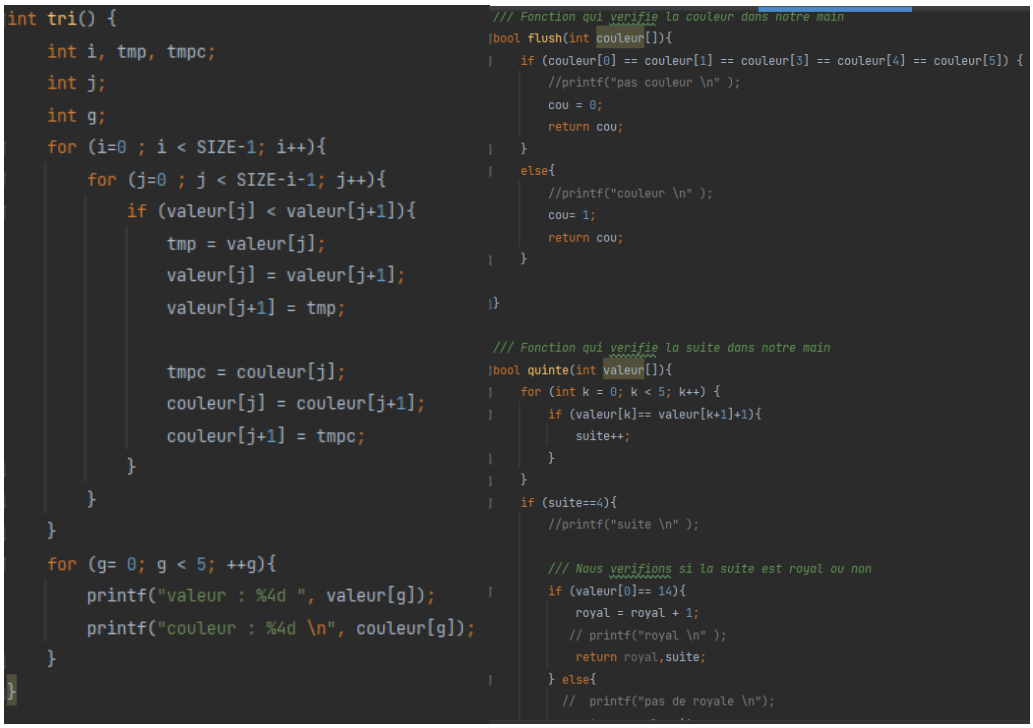
- Création de la classe « Hands » et initialisation de nos méthodes :

```
class Hands {
public:
    //méthodes
    //ctor
    Hands(int valeur[], int couleur[]);  //création d'une main (association couleur/valeur)
    //dtor
    virtual ~Hands();

    //working méthodes
    void tri(int valeur[], int taille); //trier les valeurs des cartes en ordre croissant
    bool flush(int couleur[]); //1 si toutes couleurs identiques
    int pair(int valeur[]); //compte le nombre de paires
    bool quinte(int valeur[]);

    //attributs
    //int couleur[];
    //int valeur[];

};
```

- La méthode tri permet de replacer les valeurs aléatoires dans un ordre décroissant afin de faciliter la création d'autres méthodes.
- La méthode flush permet de définir si un flush est présent dans la main (cela débloque la possibilité d'avoir une combinaison couleur, quinte flush, ou quinte flush royale).
- La méthode paire permet de trouver s'il y a une ou deux paire, un brelan, un full ou un carré.
- La méthode quinte permet de détecter une suite de 5 cartes. Dans cette méthode, si la première carte vaut 14 (soit l'as), ce n'est pas une simple quinte mais une quinte royale.

```
valeur :   13   couleur :    3
valeur :   10   couleur :    3
valeur :    9   couleur :    3
valeur :    4   couleur :    4
valeur :    3   couleur :    2
```

Création de nos tableaux de valeur et de couleur et du tri de valeur



```c
int tri() {
    int i, tmp, tmpc;
    int j;
    int g;
    for (i=0 ; i < SIZE-1; i++){
        for (j=0 ; j < SIZE-i-1; j++){
            if (valeur[j] < valeur[j+1]){
                tmp = valeur[j];
                valeur[j] = valeur[j+1];
                valeur[j+1] = tmp;

                tmpc = couleur[j];
                couleur[j] = couleur[j+1];
                couleur[j+1] = tmpc;
            }
        }
    }
    for (g= 0; g < 5; ++g){
        printf("valeur : %4d ", valeur[g]);
        printf("couleur : %4d \n", couleur[g]);
    }
}
```

```c
/// Fonction qui verifie la couleur dans notre main
bool flush(int couleur[]){
    if (couleur[0] == couleur[1] == couleur[3] == couleur[4] == couleur[5]) {
        //printf("pas couleur \n" );
        cou = 0;
        return cou;
    }
    else{
        //printf("couleur \n" );
        cou= 1;
        return cou;
    }

}

/// Fonction qui verifie la suite dans notre main
bool quinte(int valeur[]){
    for (int k = 0; k < 5; k++) {
        if (valeur[k]== valeur[k+1]+1){
            suite++;
        }
    }
    if (suite==4){
        //printf("suite \n" );

        /// Nous verifions si la suite est royal ou non
        if (valeur[0]== 14){
            royal = royal + 1;
           // printf("royal \n" );
            return royal,suite;
        } else{
          //  printf("pas de royale \n");
```

Définition de nos fonctions (tri, full, flush…)

```c
////////////////////// Appel des fonctions et return du resultat  ////////////////////////////////////////////
/// Fonction qui va appeler toutes nos fonctions et renvoyer la meilleur combinaison
int main(){
    /// Appel des fonctions
    tri();
    Full(valeur);
    Dpaire(valeur);
    flush(couleur);
    quinte(valeur);
    pairs(valeur);
    forte(valeur);

    /// Verification des cartes et renvoie la combinaison la plus forte si toutes les cartes sont différentes
    if (carte1 != carte2 && carte1 != carte3 && carte1 != carte4 && carte1 != carte5 && carte2 != carte3
    && carte2 != carte4 && carte2 != carte5 && carte3 != carte4 && carte3 != carte5 && carte4 != carte5){ //vérification qu'il n'y a pas 2 cartes identiques
        //std::cout << "go jouer" <<std::endl;
        printf("\n" );
        printf("Meilleur combinaison : \n" );
        if (royal == 1 && suite == 4 && cou == 1){
            printf("Quinte flush royal" );
        }
        else{
            if (suite==4 && cou==1){
                printf("Quinte flush" );
            }
            else{
                if (carre==1){
                    printf("Carre" );
                }
                else {
                    if (ful==1){
```

Appel de nos fonctions dans le main et vérification des cartes (sont-elles identiques ?) et Analyse de toutes les combinaisons possibles, puis regroupement de chaque résultat.

Exemple : if couleur == true && suite == true

```
valeur :    8 couleur :    1
valeur :    7 couleur :    1
valeur :    6 couleur :    1
valeur :    5 couleur :    1
valeur :    4 couleur :    1

Meilleur combinaison :
Quinte flush
```

Meilleure combinaison = quinte flush