PROJET ALGO STRUCTURES DES DONNÉES-2

UNIVERSITÉ DE NANTES

Abdoulaye Djibril Barry E14B533C
Vadim Borisov E160722Y
Groupe: 486M
Formation: L2 Info
Année: 2017/2018

1. Définir les signatures, les rôles et les préconditions des opérations de la structure de données abstraite représentant les pioches:

Soit pioche représente une pioche de cartes.
Soit PiocheT le type d'une liste d'éléments de type T
On a les opérations suivantes :

Signatures	Rôle	Préconditions
creer(out p : PiocheT) detruire(inout p : PiocheT)	Créer la pioche Détruire la pioche	———
est_vide(in p: PiocheT) : **Bool**	Retourne vrai si la pioche *p* est vide (pas de cartes), faux sinon.	Pioche *p* a été créée et pas détruite.
ajouter_carte(inout p:PiocheT, in car : T)	ajoute une carte *T* au sommet de la pioche *p*.	Pioche *p* était créée et pas détruite.
retirer_carte(inout p:PiocheT) : **T**	retire une carte *T* du sommet de *p*, puis la renvoie.	Pioche *p* n'est pas vide
renverser (inout p:PiocheT)	renverse tous les cartes dans la pioche *p*.	Pioche *p* n'est pas vide

2. Dans quel ordre les cartes sont-elles rangées dans une pioche ?En particulier, une pioche se comporte t-elle comme une pile ou une file ? Argumenter:

Une pioche se comporte comme une pile, car nous avons la dernière carte entrée est la première carte sortie de la pioche (LIFO): à chaque fois on ajoute la carte dans la pioche soit on la met au sommet, cela est valable aussi quand on retire la carte, on la prend au sommet de la pioche.

3. Dans quel ordre les cartes sont-elles rangées dans la main d'un joueur ? En particulier, le tas de cartes se comporte t-il comme une pile ou une file ? Argumenter:

Un tas de cartes de joueur se comporte comme une file, car nous avons la première carte entrée est la première carte sortie du tas (FIFO): les cartes récupérées par les joueurs sont toujours mises en bas de son tas. Et à chaque fois que le joueur prend la carte dans son tas, il la prend du sommet de son tas.

4. Dans quel ordre les cartes sont-elles rangées dans un jeu de cartes ? Est-ce nécessaire d'avoir un ordre particulier ? :

Dans un jeu de cartes l'ordre n'a pas d' importance, car on doit mélanger le jeu de cartes à chaque fois avant commencer la partie. Ensuite il n'y a pas d'ordre particulier pour distribuer ces cartes à

chaque joueur et les ajouter dans la pioche, la seule condition est que le nombre de cartes soit égal pour chaque joueur quand la partie commence.

5. Définir la structure de données en mémoire d'un jeu de bataille :

On utilise les types : *carte, jeu_de_cartes, joueur et pioche* en mémoire d'un jeu de *bataille*. La structure de la bataille alors se compose des attributs de la bataille elle-même et des relations qui le lient avec les autres classes.

Type bataille (globale) :
nb_max : Entier // nombre des coups de jeu maximaux pendant une bataille
m : Entier // renverser la pioche chaque m coup
cpt_nmax : Entier //compteur de nb_max coups courants pendant une bataille
cpt_m : Entier //compteur de m coups courants pendant une bataille
nb_jo : Entier // nombre des joueurs – incrémenté chaque fois quand on ajoute un joueur dans la //bataille
nb_jeu : Entier // nombre des jeux – incrémenté chaque fois quand on ajoute le jeu dans la bataille
joueurs : vector<joueur*> // le tableau des pointeurs vers tous les joueurs
jeux : vector<jeu_de_cartes*> // le tableau des pointeurs vers tous les jeux de cartes
pio : pioche //une pioche de la bataille

Pendant la distribution des cartes type bataille communique avec types *pioche, joueurs* et *jeu_de_cartes*:

Type pioche :
nbc : Entier // stocke le nombre des cartes courants dans la pioche
pioche_ : stack<carte*> //pioche représentée par la pile des pointeurs vers les cartes. On utilise les //pointeurs, car il y a une protection contre la recopie de type *carte*
Type joueur :
nom_jo : chaine des car //nom de joueur
nbc : Entier //stocke le nombre des cartes courants dans le tas de joueur
tas : queue<carte*> //tas représenté par la file des pointeurs vers les cartes.

Type jeu_de_cartes :
nb_cartes : Entier // stocke le nombre des cartes courants dans le jeu de cartes
nom_jeu : chaine des car //le nom du jeu
jdc : list<carte*> //liste des pointeurs vers les cartes du jeu

Quand la partie commence, pour chaque coup de la bataille, on a besoin de stocker les données locales :

Type bataille (locale - jouer_un_coup()) :
joueur_max : joueur* //pointeur vers le joueur avec la carte maximale pour le coup courant
carte_max : carte* // pointeur vers la carte maximale pour le coup courant
C : vector<carte*> // pointeurs vers l'ensemble des cartes qui jouent dans le coup courant
Cb : vector<carte*> // pointeurs vers les cartes qui se jouent si il y a plusieurs joueurs max
B : vector<joueur*> // pointeurs vers les joueurs qui se confronte avec les mêmes cartes max
nb_jo_max : Entier //nombre des joueurs qui s'affrontent si les deux ont les valeurs max des //cartes dans le coup courant
C2 : vector<carte*> // pointeurs vers l'ensemble des deuxièmes cartes des tas des joueurs qui //s'affrontent dans la bataille de coup courant

C3 : vector<carte*> // pointeurs vers l'ensemble des troisièmes cartes des tas des joueurs qui
                     //s'affrontent dans la bataille de coup courant

6. Écrire l'algorithme de l'opération jouer un coup(bat : bataille) en utilisant les opérations sur les autres types :

Rem. : comme c'est un algorithme en pseudocode, on suppose qu'il n y a pas la protection des cartes contre la recopie, on manipule ici le type carte directement dans toutes les structures de données nécessaires sans passer par les pointeurs comme en vrai code de C++

```
//fonction supplémentaire :
test_gagnant(in bat:bataille, in joueurs : vector <joueur> ) : Bool
debut :
        si (taille de joueurs == 1) alors
                retourner vrai
        fin si
        retourner faux
fin

jouer_un_coup(in bat : bataille)
var :
        //globales :
        nb_max, m, nb_jo, cpt_nmax, cpt_m : Entier
        joueurs : vector <joueur>       // le tableau des joueurs de la bataille
        pio : pioche                    // pioche de la bataille

        //locales :
        joueur_max : joueur
        carte_max :  carte
        C, C2, C3, Cb  : vector <carte>
        B : vector <joueur>
        nb_jo_max,  i : Entier

debut :
        nb_jo_max = 1
        nb_jo = nb_joueurs(bat)
        // test si il faut renverser la pioche:
        si (cpt_m == m) alors
                renverser(pio)   // renverser la pioche
                cpt_m = 0
        fin si

        carte_max = NULL
        pour i = 1, ... , nb_jo faire
                // test si le joueur a perdu (n'a plus de cartes):
                si (nb_cartes(joueurs[i]) ==  0) alors
                        // supprimer le joueur de la liste des joueurs
                        joueurs.erase(joueurs.begin() + i);
                        nb_jo--
```

```
                i--
                si (test_gagnant(bat, joueurs)) alors
                        retourner       //on arrete la fonction, car on a déjà le gagnant
                fin si
                continue        // on saute a la fin de la boucle
        fin si
        // chaque joueur prend la carte au sommet de son tas
        // C - est l'ensemble de ces cartes
        si (carte_max == NULL) alors
                carte_max = carte_au_sommet(joueurs[i])
                nb_jo_max = 1
                joueur_max = joueurs[i]
                B.clear()       // vider le tableau des joueurs max
                Cb.clear()      //vider tableau des cartes max
                B.push_back(joueurs[i])
                Cb.push_back(carte_au_sommet(joueurs[i]))

        sinon si(valeur(carte_au_sommet(joueurs[i])) > valeur(carte_max) ) alors
                carte_max = carte_au_sommet(joueurs[i])   //A
                nb_jo_max = 1
                joueur_max = joueurs[i]
                B.clear()       // vider le tableau des joueurs max
                Cb.clear()      //vider tableau des cartes max
                B.push_back(joueurs[i])
                Cb.push_back(carte_au_sommet(joueurs[i]))

        sinon si (valeur(carte_au_sommet(joueurs[i])) == valeur(carte_max)) alors
                nb_jo_max++
                B.push_back(joueurs[i])
                Cb.push_back(carte_au_sommet(joueurs[i]));
        fin si
        C.push_back(retirer_carte(joueurs[i]));
fin pour

//si on a seul joueur:
si (nb_jo_max == 1) alors
        // remettre tous les cartes C au tas de joueur_max
        pour i = 1, … , taille de C faire
                ajouter_carte(joueur_max, C[i])
        fin pour

//si plusieurs on a une bataille:
sinon
        // les cartes de C qui ne sont pas dans Cb sont mises en pioche
        pour i = 1, …, taille de C faire
                si (valeur(Cb[0]) != valeur(C[i])) alors
                        ajouter_carte(pio, C[i])
                fin si
        fin pour

        //chaque joueur de B prend une seconde carte au sommet de son tas
        pour  i = 1, … , taille de B faire
```

```
        // si joueur n'a plus de cartes
        si (nb_cartes(B[i]) == 0) alors
                si (!est_vide(pio)) alors     //il prend dans la pioche si elle est pas vide
                        C2.push_back(retirer_carte(pio))

                sinon                                   //sinon le joueur a perdu
                        B.erase(B.begin() + i)
                        i--
                        si (test_gagnant(bat, joueurs)) alors
                                retourner
                        fin si
                        continue
                fin si
        sinon
                C2.push_back(retirer_carte(B[i]))
        fin si
fin pour

//chaque joueur de B prend une troisieme carte au sommet de son tas

nb_jo_max = 1
carte_max = NULL
pour i = 1, …, taille de B faire
        // si joueur n`a plus de cartes
        si (nb_cartes(B[i]) == 0) alors
                si (!est_vide(pio)) alors      //il prend dans la pioche si elle est pas vide
                        C3.push_back(retirer_carte(pio))

                sinon                                   //sinon le joueur a perdu
                        B.erase(B.begin() + i)
                        i--
                        si (test_gagnant(bat, joueurs)) alors
                                retourner
                        fin si
                        continue
                fin si
        sinon
                si (carte_max == NULL) alors
                        carte_max = carte_au_sommet(B[i])
                        joueur_max = B[i]

                sinon si (valeur(carte_au_sommet(B[i])) > valeur(carte_max) ) alors
                        carte_max = carte_au_sommet(B[i])
                        nb_jo_max = 1
                        joueur_max = B[i]
                }

                sinon si (valeur(carte_au_sommet(joueurs[i])) ==valeur(carte_max))
                        nb_jo_max++
                fin si
                C3.push_back(retirer_carte(B[i]-))
        fin si
```

```
                fin pour

                //si on a un seul joueur avec la carte max
                si (nb_jo_max == 1) alors
                        //C3.size() == C2.size() == Cb.size()
                        pour  i = 1, …, taille de C3 faire
                                ajouter_carte(joueur_max, C3[i])
                                ajouter_carte(joueur_max, C2[i])
                                ajouter_carte(joueur_max, Cb[i])
                        fin pour

                //sinon on remet tous les cartes dans la pioche
                sinon
                        pour i = 1, … , taille de C3 faire
                                ajouter_carte(pio, C3[i])
                                ajouter_carte(pio, C2[i])
                                ajouter_carte(pio, Cb[i])
                        fin pour
                fin si
        fin si

        cpt_nmax++;             //on incremente les compteurs
        cpt_m++;

fin
```