Algorithmique - TP4

IUT 1ère Année

4 décembre 2015

Exercices sur les vecteurs et matrices

Exercice 1. Dans l'exercice suivant, nous utiliserons plusieurs opérations sur les vecteurs et matrices. Les types utilisés sont des vecteurs de réels de dimension 4 et des matrices de réels de dimension 4×4 :

```
typedef double Vector4[4];
typedef double Matrix44[4][4];
```

Télécharger le fichier `matrice44.cpp` et remplir le code des fonctions :

```
void fill(Matrix44& mat, const double val);
// Toutes les entrées de la matrice "mat" ont la valeur "val"

void add(Matrix44& mat, const int x, const int y, const double val);
// l'entrée de mat à la ligne x et la colonne y doit avoir la valeur val

void scalar_multiplication(Matrix44& mat, const double val);
// Toutes les entrées de mat sont multipliées par val

void diag(Matrix44& mat, Vector4& vec);
// Le vecteur vec prend la diagonale de la matrice mat

void print(const Matrix44& mat);
// La matrice est affichée ligne par ligne
```

Exercice 2. Nous étendons les opérations précédentes avec les additions et produits matriciels. Les types utilisés sont :

```
typedef double Vector5[5];
typedef double Matrix55[5][5];
```

Télécharger le fichier `matrice55.cpp` et remplir le code des fonctions :

```
int diagonale(const Matrix55& A, Vector5& D);;
// diag est la diagonale de mat

void somme(const Matrix55& A, const Matrix55& B, Matrix55& C)
// C = A + B

bool estSymmetrique(const Matrix55& A);
// Teste si A est symmétrique (A[i,j] == A[j,i])

int produit(const Matrix55& A, const Vector5& b, Vector5& c);
// c = Ab
// c est donc le vecteur obtenu en applicant la matrice A sur le vecteur b
// (un exercice très similaire a été vu en TD)
```

Exercice 3. Comme nous l'avons vu en TD *Puissance 4* (ou *Connect 4*) est un jeu à deux joueurs constitué d'un plateau vertical de 6 rangées et de 7 colonnes. Le plateau est initialement vide. A chaque tour de jeu, le joueur *rouge* pose un jeton rouge dans une colonne j. Par gravité, le jeton vient au-dessus de la dernière case remplie de la colonne j. Le joueur *jaune* réalise la même opération en posant un jeton jaune dans une colonne.

Nous utiliserons les types suivants :

```
enum Case {rouge, jaune, vide};
typedef Case Plateau[6][7];
```

Implémenter les fonctions suivantes :

```
void init(Plateau& jeu);
// initialiser le jeu (plateau vide)

void afficher(const Plateau& jeu);
// afficher le jeu ligne par ligne

void update(Plateau& jeu, const Case joueur, const int colonne);
// Le joueur pose un jeton dans la colonne qui vient donc au-dessus de la dernière case remplie

bool estRempli(const Plateau& jeu);
// Teste si le plateau est rempli

bool rangee4(const Plateau& jeu, const Case joueur);
// Teste si le plateau contient une rangée de 4 cases consécutives de couleur joueur

bool colonne4(const Plateau& jeu, const Case joueur);
// Teste si le plateau contient une colonne de 4 cases consécutives de couleur joueur

bool diagonale4(const Plateau& jeu, const Case joueur);
// Teste si le plateau contient une colonne de 4 cases consécutives de couleur joueur

bool gagne(const Plateau& jeu, const Case joueur);
// Teste si joueur a gagné

bool terminal(const Plateau& jeu);
// Teste si le jeu est fini (un des joueurs a gagné ou le plateau est rempli)
```

Tester le jeu en utilisant *une boucle événementielle* qui peut se programmer de la manière suivante :

```
int main()
{
        Plateau jeu;

        init(jeu);

        bool tourDeJeu = true;
        while(!terminal(jeu))
        {
                afficher(jeu);

                Case joueur = vide;
                int colonne = 0;

                if(tourDeJeu == true)
                {
                        cout << "Joueur rouge: " << endl;
                        joueur = rouge;
                }
                else
                {
                        cout << "Joueur jaune: " << endl;
                        joueur = jaune;
                }

                cin >> colonne;

                update(jeu, joueur, colonne);

                tourDeJeu = 1 - tourDeJeu;
        }

        return 1;
}
```

Exercice 4. Modifier le jeu Puissance 4 afin que les coups joués soient *légaux* : un joueur ne peut pas placer un pion dans une colonne remplie.

Exercice 5*. Implémenter et tester les fonctions suivantes :

```
void sauvegarder(const Plateau& jeu, bool tourDeJeu);
// sauvegarde dans un fichier l'état du jeu et le tour de jeu courant

void charger(Plateau& jeu, bool& tourDeJeu);
// charge un fichier de sauvegarde afin de reprendre la partie dans l'état courant
```