```cpp
#include "grille.h"
#include <fstream>
#include <cstdlib>
#include <string.h>

using namespace std;

////////////////// FONCTIONS HORS-CLASSE USUELLE ///////////////////////

bool string_est_int(string coup)        //test si un string est un nombre
{
        if(atoi(coup.c_str()))      //si c'est un int
                return true;
        return false;
}

int string_en_int(string coup)      //Utilisée dans la sauvegarde pour transforme un string en int (utilise par la fonction charger)
{
    int i=atoi(coup.c_str());       //atoi permet de mettre coup en int
    return i;       //on renvoi ce nombre
}

////////////////// CONSTRUCTEUR ET DESTRUCTEUR ////////////////////////

grille::grille()
{
        nb_ligne=0;         //on initialise nb_ligne à 0
        nb_colonne=0;       //on initialise nb_colonne à 0
        nb_jeton=0;         //on initialise nb_jeton à 0
        nb_alignement=0;        //on initialise nb_alignement à 0
        nb_joueur=0;        //on initialise nb_joueur à 0
        al_J=NULL;      //on initialise al_J à NULL (pointeur)
        resultat=0;          //on initialise resultat à 0
        Table=NULL;         //on initialise Table à NULL (pointeur)
        n_jeton=NULL;
        all_jeton=NULL;         //on initialise all_jeton à NULL
}

grille::grille(const grille& G_tmp)
{
        nb_ligne=G_tmp.nb_ligne;
        nb_colonne=G_tmp.nb_colonne;
        nb_jeton=G_tmp.nb_jeton;
        nb_alignement=G_tmp.nb_alignement;
        nb_joueur=G_tmp.nb_joueur;
        al_J=new int[nb_joueur];
        all_jeton=new jeton[nb_joueur];
        Table=new jeton *[nb_ligne];
        for(int i=0; i<nb_ligne; i++)
        {
                Table[i]=new jeton [nb_colonne];
        }
        for(int i=0;i<nb_joueur;i++)
        {
                al_J[i]=G_tmp.al_J[i];
        }

        resultat=G_tmp.resultat;
        for(int i=0;i<nb_ligne;i++)
        {
                for(int j=0;j<nb_colonne;j++)
                {
                        Table[i][j].forme=G_tmp.Table[i][j].forme;
                        Table[i][j].couleur=G_tmp.Table[i][j].couleur;
                        Table[i][j].marquage=G_tmp.Table[i][j].marquage;
                }
        }
        for(int i=0;i<nb_joueur;i++)
        {
                all_jeton[i].forme=G_tmp.all_jeton[i].forme;
```

```
                all_jeton[i].couleur=G_tmp.all_jeton[i].couleur;
                all_jeton[i].marquage=G_tmp.all_jeton[i].marquage;
        }
        n_jeton=new int[nb_joueur];
        for(int i=0;i<nb_joueur;i++)
            n_jeton[i]=G_tmp.n_jeton[i];
}

grille::~grille()
{
        delete[] al_J;      //on detruit al_J
        for(int i=0; i<nb_ligne;i++)
                delete[] Table[i];       //on detruit Table
        delete[] Table;
        delete[] all_jeton;         //on detruit all_jeton
        delete[] n_jeton;
}

////////////////// MANIPULATION DE LA GRILLE ////////////////////////////

void grille::saisie()
{
        bool condition=0;
        do      //on effectue la saisie des données
        {
                system("clear");
                cout << "\E[32;1m           CONFIGURATION DE LA GRILLE DE
                JEU\E[m"<<endl<<endl<<endl;
                if(condition==1)
                        cout<<"la grille est trop petite pour ce nombre d'alignements et la
                        longueur de ces derniers"<<endl<<endl;
                string var;
                cout<<"     nombre de colonnes(minimum de 3, maximum de 8): ";
                cin>>var;
                if(string_est_int(var)==true)
                    nb_colonne=string_en_int(var);
                else
                    nb_colonne=2;
                cout<<endl<<"       nombre de lignes(minimum de 3, maximum de 8): ";
                cin>>var;
                if(string_est_int(var)==true)
                    nb_ligne=string_en_int(var);
                else
                    nb_ligne=2;
                cout<<endl<<"       nombre de jetons à aligner pour gagner(minimum de 3): ";
                cin>>var;
                if(string_est_int(var)==true)
                    nb_jeton=string_en_int(var);
                else
                    nb_jeton=2;
                cout<<endl<<"       nombre d'alignement nécessaire (minimum de 1): ";
                cin>>var;
                if(string_est_int(var)==true)
                    nb_alignement=string_en_int(var);
                else
                    nb_alignement=0;

                if(((nb_alignement+nb_jeton-1)*(nb_alignement+nb_jeton-1))>(nb_ligne*nb_colonn
                e)) //pour eviter des grille trop petite pour des données trop grande
                {
                    condition=1;
                    nb_colonne=2;       //on met une condition d'erreur pour recommencer la
                    saisie
                }
        }
        while ((nb_colonne <3 || nb_colonne >8 ) || (nb_ligne<3 || nb_ligne>8) || (nb_jeton
        < 3|| nb_jeton > 6) || (nb_alignement < 1|| nb_alignement > 9)); //tant que ces
        condition ne sont pas respectées
        preremplir_grille();        //on preremplit la grille grace aux informations saisie

}
```

```
void grille::preremplir_grille()
{
	al_J=new int[nb_joueur];
	all_jeton=new jeton[nb_joueur];		//alocation dynamique de all_jeton
	Table=new jeton *[nb_ligne];		//alocation dynamique de Table (deux dimension)
	for(int i=0; i<nb_ligne; i++)
		Table[i]=new jeton[nb_colonne];
	for(int i=0; i<nb_joueur; i++)		//on intialise les alignement de tout les joueur
	à 0
		al_J[i]=0;
	resultat=0;
	for(int i=0;i<nb_ligne;i++)
	{
		for(int j=0;j<nb_colonne;j++)
		{
			Table[i][j].forme='.';		//on remplit les case vide par .
			Table[i][j].couleur=0;		//ces case auront la couleur 34 (bleu
			foncé)
			Table[i][j].marquage=0;
		}
	}
	for(int i=0; i<nb_joueur; i++)
	{
	    all_jeton[i].forme='?';
	    all_jeton[i].couleur=0;
	    all_jeton[i].marquage=0;
	}
	n_jeton=new int[nb_joueur];
	for(int i=0; i<nb_joueur; i++)
		n_jeton[i]=0;
}

void grille::enlever_marquage_alignement()
{
	for(int i=0;i<nb_ligne;i++)
		for(int j=0;j<nb_colonne;j++)
			Table[i][j].marquage=0;		//on met tout les marqueur de la
			table a 0 (case non visité)
}

void grille::affichage()
{
	system("clear");
	cout << "\E[33;1m
	J\E[m"<<"\E[31;1mO\E[m"<<"\E[36;1mU\E[m"<<"\E[32;1mE\E[m"<<"\E[35;1mZ\E[m"<<endl<<endl
	;
	cout<<"		"<<endl;
	for(int i=0;i<nb_colonne;i++)		//permet de separé de maniere egale les colonne
	{
		if(i<10)
			cout<<"     ";
		else
			cout<<"    ";
		cout<<i+1;
	}
	cout<<endl;
	for(int i=0;i<nb_ligne;i++)
	{
		cout<<"		"<<endl;
		for(int j=0;j<nb_colonne;j++)
		{
			cout<<"     ";
			if(Table[i][j].couleur==1)		//si la case a la couleur 1 c'est
			le joueur1
				cout << "\E[33;1m"<<Table[i][j].forme<<"\E[m";		//donc
				affichage en jaune
			else if(Table[i][j].couleur==2)		//si la case a la couleur 2
			c'est le joueur2
				cout << "\E[31;1m"<<Table[i][j].forme<<"\E[m";		//donc
				affichage en rouge
```

```
                        else if(Table[i][j].couleur==3)
                                cout << "\E[36;1m"<<Table[i][j].forme<<"\E[m";
                        else if(Table[i][j].couleur==4)
                                cout << "\E[32;1m"<<Table[i][j].forme<<"\E[m";
                        else      //case vide
                                cout << "\E[34;1m"<<Table[i][j].forme<<"\E[m";
                                //affichage en bleu fonce
                }
                cout<<endl;
        }
        cout<<endl<<"      nombre jetons necessaires: "<<nb_jeton<<     "   nombe
        d'alignement(s) necessaire(s): "<<nb_alignement<<endl<<endl;
        for(int i=0;i<nb_joueur;i++)        //affiche le nombre d'alignement de chaque joueur
                cout<<"     nb alignment J"<<i+1<<": "<<al_J[i]<<endl;
}

void grille::initialiser_alignement()
{
        for(int i=0; i<nb_joueur; i++)      //on remet tout les alignement a 0
        {
                al_J[i]=0;
                n_jeton[i]=0;
        }
}

void grille::placer(int colonne, jeton le_jeton, int joueur)
{
        for(int j=nb_ligne-1;j>=0;j--)
        {
                if(Table[j][colonne-1].forme=='.')
                {
                        Table[j][colonne-1].couleur=joueur;         //la case choisit prend
                        la couleur du joueur
                        Table[j][colonne-1].forme=le_jeton.forme;       //la case choisit
                        prend la forme du joueur
                        Table[j][colonne-1].marquage=0;
                        break;
                }

        }
}

void grille::pivoter(std::string pivot, jeton le_jeton, int joueur)
{
        if(pivot=="h"||pivot=="horaire")        //deux orthographes autorisées
        {
                vers_la_droite();
                tourner_horaire();
        }
        else if(pivot=="a"||pivot=="antihoraire")
        {
                vers_la_gauche();
                tourner_antihoraire();
        }
}

void grille::vers_la_droite()
{
        for(int i=nb_ligne-1; i>=0;i--)
        {
                for(int j=nb_colonne-1; j>=0;j--)
                {
                        if(Table[i][j].forme!='.')
                        {
                                int k=j;
                                while(k<nb_colonne-1 && Table[i][k+1].forme=='.')
                                //tant que les cases sur la droite sont vides on deplace
                                {
                                        Table[i][k+1].forme=Table[i][k].forme;
                                        Table[i][k+1].couleur=Table[i][k].couleur;
                                        Table[i][k].forme='.';      //on remet la case
```

```
                                        precement utilisé vide
                                        Table[i][k].couleur=0;
                                        Table[i][k].marquage=0;
                                        k++;
                                }
                        }
                }
        }
}

void grille::vers_la_gauche()
{
        for(int i=nb_ligne-1; i>=0;i--)
        {
                for(int j=nb_colonne-1; j>=0;j--)
                {
                        if(Table[i][j].forme!='.')
                        {
                                int k=j;
                                while(k<nb_colonne &&k>0 && Table[i][k-1].forme=='.')
                                //tant que les cases sur la gauche sont vides ont deplace
                                {
                                        Table[i][k-1].forme=Table[i][k].forme;
                                        Table[i][k-1].couleur=Table[i][k].couleur;
                                        Table[i][k].forme='.';      //on remet la case
                                        precement utilisé vide
                                        Table[i][k].couleur=0;
                                        Table[i][k].marquage=0;
                                        k++;
                                }
                        }
                }
        }
}

void grille::tourner_horaire()
{
        grille G_tmp;       //on creer une grille temporaire possedant le meme donne que
        l'ancienne sauf que l'on inverse les valeur de nb_ligne et nb_colonne
        G_tmp.nb_colonne=nb_ligne;
        G_tmp.nb_ligne=nb_colonne;
        G_tmp.nb_joueur=nb_joueur;

        G_tmp.Table=new jeton *[G_tmp.nb_ligne];
        for(int i=0; i<G_tmp.nb_ligne; i++)
                G_tmp.Table[i]=new jeton[G_tmp.nb_colonne];
        G_tmp.al_J=new int[nb_joueur];
        G_tmp.all_jeton=new jeton[nb_joueur];
        G_tmp.n_jeton=new int[nb_joueur];
        G_tmp.nb_jeton=nb_jeton;
        G_tmp.nb_alignement=nb_alignement;
        G_tmp.preremplir_grille();
        for(int i=0;i<nb_joueur;i++)
            G_tmp.al_J[i]=al_J[i];
        for(int i=0;i<nb_joueur;i++)
            {
                G_tmp.all_jeton[i].forme=all_jeton[i].forme;
                G_tmp.all_jeton[i].couleur=all_jeton[i].couleur;
                G_tmp.all_jeton[i].marquage=all_jeton[i].marquage;
            }
            for(int i=0; i<nb_joueur; i++)
                G_tmp.n_jeton[i]=n_jeton[i];
        int l=0;
        for(int i=0; i<G_tmp.nb_ligne; i++)     //on place les jetons dans leur nouveaux
        emplacements
        {
                int k=nb_ligne-1;
                for(int j=0; j<G_tmp.nb_colonne; j++)
                {
                        G_tmp.Table[i][j].forme=Table[k][l].forme;
                        G_tmp.Table[i][j].couleur=Table[k][l].couleur;
```

```
                        k--;
                }
                l++;
        }

        nb_colonne=G_tmp.nb_colonne;        //on remet toute les nouvelle données dans
        l'ancienne grille
        nb_ligne=G_tmp.nb_ligne;
        nb_joueur=G_tmp.nb_joueur;
        nb_jeton=G_tmp.nb_jeton;
        nb_alignement=G_tmp.nb_alignement;

        for(int i=0; i<nb_colonne; i++)
                delete[] this->Table[i];
        delete[] this->al_J;
        delete[] this->all_jeton;
        delete[] this->n_jeton;

        al_J=new int[nb_joueur];
        all_jeton=new jeton[nb_joueur];
        for(int i=0;i<nb_joueur;i++)
                al_J[i]=G_tmp.al_J[i];
        Table=new jeton *[nb_ligne];
        for(int i=0; i<G_tmp.nb_ligne; i++)
                Table[i]=new jeton [nb_colonne];
        for(int i=0; i<nb_ligne; i++)
        {
                for(int j=0; j<nb_colonne; j++)
                {
                        Table[i][j].forme=G_tmp.Table[i][j].forme;
                        Table[i][j].couleur=G_tmp.Table[i][j].couleur;
                }
        }
        n_jeton=new int[nb_joueur];
        for(int i=0; i<nb_joueur; i++)
                n_jeton[i]=G_tmp.n_jeton[i];
}

void grille::tourner_antihoraire()
{
        grille G_tmp;       //on creer une grille temporaire possedant le meme donne que
        l'ancienne sauf que l'on inverse les valeur de nb_ligne et nb_colonne
        G_tmp.nb_colonne=nb_ligne;
        G_tmp.nb_ligne=nb_colonne;
        G_tmp.nb_joueur=nb_joueur;
        G_tmp.Table=new jeton *[G_tmp.nb_ligne];
        for(int i=0; i<G_tmp.nb_ligne; i++)
                G_tmp.Table[i]=new jeton [G_tmp.nb_colonne];
        G_tmp.al_J=new int[nb_joueur];
        G_tmp.all_jeton=new jeton[nb_joueur];
        G_tmp.n_jeton=new int[nb_joueur];
        G_tmp.nb_jeton=nb_jeton;
        G_tmp.nb_alignement=nb_alignement;
        G_tmp.preremplir_grille();
        for(int i=0;i<nb_joueur;i++)
            G_tmp.al_J[i]=al_J[i];
        int l=nb_colonne-1;
        for(int i=0; i<nb_joueur; i++)
                G_tmp.n_jeton[i]=n_jeton[i];
        for(int i=0; i<G_tmp.nb_ligne; i++)     //on place les jetons dans leur nouveaux
        emplacements
        {
                int k=0;
                for(int j=0; j<G_tmp.nb_colonne; j++)
                {
                        G_tmp.Table[i][j].forme=Table[k][l].forme;
                        G_tmp.Table[i][j].couleur=Table[k][l].couleur;
                        k++;
                }
                l--;
        }
```

```
        nb_colonne=G_tmp.nb_colonne;    //on remet les nouvelle données dans l'ancienne grille
        nb_ligne=G_tmp.nb_ligne;
        nb_joueur=G_tmp.nb_joueur;
        nb_jeton=G_tmp.nb_jeton;
        nb_alignement=G_tmp.nb_alignement;
        for(int i=0; i<nb_colonne; i++)
                delete[] this->Table[i];
        delete[] this->al_J;
        delete[] this->all_jeton;
        delete[] this->n_jeton;

        al_J=new int[nb_joueur];
        all_jeton=new jeton[nb_joueur];
        for(int i=0;i<nb_joueur;i++)
                al_J[i]=G_tmp.al_J[i];
        Table=new jeton *[nb_ligne];
        for(int i=0; i<G_tmp.nb_ligne; i++)
                Table[i]=new jeton [nb_colonne];
        for(int i=0; i<nb_ligne; i++)
        {
                for(int j=0; j<nb_colonne; j++)
                {
                        Table[i][j].forme=G_tmp.Table[i][j].forme;
                        Table[i][j].couleur=G_tmp.Table[i][j].couleur;
                }
        }
        n_jeton=new int[nb_joueur];
        for(int i=0; i<nb_joueur; i++)
                n_jeton[i]=G_tmp.n_jeton[i];
}

void grille::echanger(char forme_ori,char forme_dest, int couleur_ori, int couleur_dest)
{
        for(int i=0;i<nb_ligne;i++)
        {
                for(int j=0;j<nb_colonne;j++)
                {
                        if(Table[i][j].forme==forme_ori &&
                        Table[i][j].couleur==couleur_ori)//si on a un jeton du joueur qui
                        fait l'echange, on le remplace par le jeton du joueur vise
                        {
                                Table[i][j].forme=forme_dest;
                                Table[i][j].couleur=couleur_dest;
                        }
                        else if(Table[i][j].forme==forme_dest
                        &&Table[i][j].couleur==couleur_dest )//inverse du cas precedents
                        {
                                Table[i][j].forme=forme_ori;
                                Table[i][j].couleur=couleur_ori;
                        }
                }
        }
}

////////////////// VERIFICATIONS ////////////////////////////////////

void grille::verification(jeton le_jeton, int joueur)
{
        verification_victoire_horizontale(le_jeton, joueur);
        verification_victoire_verticale(le_jeton, joueur);
        verification_victoire_diagonale_1(le_jeton, joueur);
        verification_victoire_diagonale_2(le_jeton, joueur);
}

void grille::verification_victoire_horizontale(jeton le_jeton, int joueur)
{
        for(int i=0; i<nb_ligne; i++)
        {
                int reussite=0;
                for(int k=0; k<nb_colonne; k++)
```

```
                {
                        if(Table[i][k].forme==le_jeton.forme &&
                        Table[i][k].couleur==le_jeton.couleur)
                        {
                                reussite++;
                                if(reussite>n_jeton[joueur-1])      //pour compter la
                                meilleur suite de jetons
                                        n_jeton[joueur-1]=reussite;
                                if(reussite==nb_jeton)
                                {
                                        reussite=0;
                                        al_J[joueur-1]++;
                                        if(al_J[joueur-1]>=nb_alignement)       //si le
                                        nombre d'alignement recherche est atteint
                                        {
                                                resultat=2;
                                        }
                                }
                        }
                        else
                            reussite=0;
                }
        }
}

void grille::verification_victoire_verticale(jeton le_jeton, int joueur)
{
        for(int i=0; i<nb_colonne; i++)
        {
                int reussite=0;
                for(int k=0; k<nb_ligne; k++)
                {
                        if(Table[k][i].forme==le_jeton.forme &&
                        Table[k][i].couleur==le_jeton.couleur)
                        {
                                reussite++;
                                if(reussite>n_jeton[joueur-1])      //pour compter la
                                meilleur suite de jetons
                                        n_jeton[joueur-1]=reussite;
                                if(reussite==nb_jeton)
                                {
                                        reussite=0;
                                        al_J[joueur-1]++;
                                        if(al_J[joueur-1]>=nb_alignement)//si le nombre
                                        d'alignement recherche est atteint
                                        {
                                                resultat=2;
                                        }
                                }
                        }
                         else
                            reussite=0;

                }
        }
}

void grille::verification_victoire_diagonale_1(jeton le_jeton, int joueur)
{
        enlever_marquage_alignement();
        for(int i=0; i<nb_ligne; i++)
        {
                int reussite=0;
                for(int j=0; j<nb_colonne; j++)
                {
                        if(Table[i][j].forme==le_jeton.forme &&
                        Table[i][j].couleur==le_jeton.couleur)
                        {
                                reussite=1;
                                int i_tmp=i;
                                int j_tmp=j;
```

```
                                int k=nb_jeton;
                                if(j-(nb_jeton-1)>=0)
                                {
                                        while(k>1)
                                        {

                                                if(i_tmp+1!=nb_ligne&&j_tmp!=0&&Table[i_tmp+1]
                                                [j_tmp-1].forme==le_jeton.forme &&
                                                Table[i_tmp+1][j_tmp-1].couleur==le_jeton.coul
                                                eur && Table[i_tmp][j_tmp].marquage!=1)
                                                {
                                                        reussite++;
                                                        if(reussite>n_jeton[joueur-1])
                                                        //pour compter la meilleur suite de
                                                        jetons
                                                                n_jeton[joueur-1]=reussite;
                                                        if(reussite==nb_jeton)
                                                        {

                                                                Table[i_tmp+1][j_tmp-1].marqua
                                                                ge=1;
                                                                for(int m=1; m<nb_jeton; m++)
                                                                {

                                                                        Table[i_tmp][j_tmp].ma
                                                                        rquage=1;
                                                                        i_tmp--;
                                                                        j_tmp++;
                                                                }
                                                                reussite=0;
                                                                al_J[joueur-1] ++;

                                                                if(al_J[joueur-1]>=nb_aligneme
                                                                nt)//si le nombre
                                                                d'alignement recherche est
                                                                atteint
                                                                {
                                                                        resultat=2;
                                                                }
                                                        }
                                                        i_tmp++;
                                                        j_tmp--;
                                                        k--;
                                                }
                                            else
                                            {
                                                reussite=0;
                                                break;
                                            }
                                        }
                                }
                        }
                }
        }
}

void grille::verification_victoire_diagonale_2(jeton le_jeton, int joueur)
{
        enlever_marquage_alignement();
        for(int i=0; i<nb_ligne; i++)
        {
                int reussite=0;
                for(int j=0; j<nb_colonne; j++)
                {
                        if(Table[i][j].forme==le_jeton.forme &&
                        Table[i][j].couleur==le_jeton.couleur)
                        {
                                reussite=1;
                                int i_tmp=i;
                                int j_tmp=j;
                                int k=nb_jeton;
```

```
                                if(j+(nb_jeton-1)<=nb_colonne-1)
                                {
                                        while(k>1)
                                        {

                                                if(i_tmp+1!=nb_ligne&&j_tmp+1!=nb_colonne&&Tab
                                                le[i_tmp+1][j_tmp+1].forme==le_jeton.forme
                                                &&
                                                Table[i_tmp+1][j_tmp+1].couleur==le_jeton.coul
                                                eur && Table[i_tmp][j_tmp].marquage!=1)
                                                {
                                                        reussite++;
                                                        if(reussite>n_jeton[joueur-1])
                                                        //pour compter la meilleur suite de
                                                        jetons
                                                                n_jeton[joueur-1]=reussite;
                                                        if(reussite==nb_jeton)
                                                        {

                                                                Table[i_tmp+1][j_tmp+1].marqua
                                                                ge=1;
                                                                for(int m=1; m<nb_jeton; m++)
                                                                {

                                                                        Table[i_tmp][j_tmp].ma
                                                                        rquage=1;
                                                                        i_tmp--;
                                                                        j_tmp--;
                                                                }
                                                                reussite=0;
                                                                al_J[joueur-1] ++;

                                                                if(al_J[joueur-1]>=nb_aligneme
                                                                nt)//si le nombre
                                                                d'alignement recherche est
                                                                atteint
                                                                {
                                                                        resultat=2;
                                                                }
                                                        }
                                                        i_tmp++;
                                                        j_tmp++;
                                                        k--;
                                                }
                                            else
                                            {
                                                reussite=0;
                                                break;
                                            }
                                        }
                                }
                        }
                }
        }
}


////////////////// SAUVEGARDE ET CHARGEMENT ////////////////////////////

void grille::sauvergarde(int joueur)
{
        char str[50];
        char nom[31];
        cout<<"saisir le nom de de la sauvegarde"<<endl;
        cin>>nom;
        strcpy (str,"base_de_donnee/");     //copie "base_de_donnee/" dans str
        strcat (str,nom);       //concataine str et nom dans str
        strcat (str,".txt");        //concatene str et ".txt" dans str. Le nom complet du
        fichier est donc creer dans str
        puts (str);
        ofstream save(str);
```

```
        save<<nb_ligne<<endl;
        save<<nb_colonne<<endl;
        save<<nb_joueur<<endl;
        save<<nb_jeton<<endl;
        save<<nb_alignement<<endl;
        for(int i=0;i<nb_joueur;i++)
        {
                save<<all_jeton[i].forme<<endl;
                save<<all_jeton[i].couleur<<endl;
                save<<all_jeton[i].marquage<<endl;
        }
        for(int i=0;i<nb_joueur;i++)
                save<<al_J[i]<<endl;
        for(int i=0;i<nb_joueur;i++)
                save<<n_jeton[i]<<endl;
        for(int i=0; i<nb_ligne; i++)
        {
                for(int j=0; j<nb_colonne; j++)
                {
                        save<<Table[i][j].forme;
                        save<<endl;
                        save<<Table[i][j].couleur;
                        save<<endl;
                }
        }
        save<<joueur<<endl;
        save.close();
}

int grille::charger(std::string nom)
{
        ifstream fichier(nom.c_str(), ios::in);     // on ouvre en lecture
        if(fichier)     // si l'ouverture a fonctionné
        {
                string contenu;
                getline(fichier, contenu);      //on met dans contenu la ligne selectonnée
                nb_ligne=string_en_int(contenu);
                getline(fichier, contenu);      //on passe a laligne suivante et on fait
                pareil
                nb_colonne=string_en_int(contenu);
                getline(fichier, contenu);
                nb_joueur=string_en_int(contenu);
                getline(fichier, contenu);
                nb_jeton=string_en_int(contenu);
                getline(fichier, contenu);
                nb_alignement=string_en_int(contenu);
                all_jeton=new jeton[nb_joueur];     //on realloue dynamiquement ce qui doit
                etre realloué
                for(int i=0;i<nb_joueur;i++)
                {
                        getline(fichier, contenu);
                        all_jeton[i].forme=contenu[0];
                        getline(fichier, contenu);
                        all_jeton[i].couleur=string_en_int(contenu);
                        getline(fichier, contenu);
                        all_jeton[i].marquage=string_en_int(contenu);
                }
                al_J=new int[nb_joueur];
                for(int i=0;i<nb_joueur;i++)
                {
                        getline(fichier, contenu);
                        al_J[i]=string_en_int(contenu);
                }
                n_jeton=new int[nb_joueur];
                for(int i=0;i<nb_joueur;i++)
                {
                        getline(fichier, contenu);
                        n_jeton[i]=string_en_int(contenu);
                }
                Table=new jeton *[nb_ligne];
                for(int i=0; i<nb_ligne; i++)
```

```
                        Table[i]=new jeton [nb_colonne];
                char forme_du_jeton;
                for(int i=0; i<nb_ligne; i++)
                {
                        for(int j=0; j<nb_colonne; j++)
                        {
                                getline(fichier, contenu);
                                forme_du_jeton=contenu[0];
                                Table[i][j].forme=forme_du_jeton;
                                getline(fichier, contenu);
                                Table[i][j].couleur=string_en_int(contenu);
                        }
                }
                getline(fichier, contenu);
                int joueur_qui_joue=string_en_int(contenu);     //la derniere donnée du
                fichier texte est la personne qui doit jouer
                fichier.close();
                return joueur_qui_joue;
            }
            else
                    cerr << "Impossible d'ouvrir le fichier !" << endl;
            return 0;
}

///////////////// AUTRES METHODES DE LA CLASSE ////////////////////////

int grille::fin()
{
        return resultat;
}

void grille::impose_fin()
{
        resultat=1;     //en cas de volonte de fin de partie resultat deviens 2 et entraine
        la fin d'une partie
}

bool grille::colonne_remplie(int col)
{
    if(Table[0][col-1].forme!='.')  //si la case est vide
            return true;
    return false;
}

int grille::nb_coup_possible()
{
        int nb=0;
        for(int i=0;i<nb_ligne;i++)
                for(int j=0;j<nb_colonne;j++)
                        if(Table[i][j].forme=='.')
                                nb++;
        if(nb==0)       //si il n'y a plus aucune case libre
                resultat=1;     //partie nulle
        return nb;
}

void grille::nombre_joueur(int nombre)
{
        nb_joueur=nombre;
}

int grille::renvoi_nb_joueur()
{
    return nb_joueur;
}

void grille::recupere_jeton_joueur(int joueur, jeton jeton_joueur)
{
        all_jeton[joueur-1]=jeton_joueur;
}
```

```
int grille::renvoie_colonne()
{
        return nb_colonne;
}

int grille::renvoie_ligne()
{
        return nb_ligne;
}

jeton grille::renvoie_jeton_joueur(int joueur)
{
        return all_jeton[joueur-1];
}

int grille::renvoie_nb_jeton()
{
        int nb=0;
        for(int i=0; i<nb_ligne; i++)
                for(int j=0; j<nb_colonne; j++)
                        if(Table[i][j].forme!='.')
                                nb++;
        return nb;
}

void grille::recopie(grille G_tmp)
{
        nb_colonne=G_tmp.nb_colonne;
        nb_ligne=G_tmp.nb_ligne;
        nb_joueur=G_tmp.nb_joueur;
        nb_jeton=G_tmp.nb_jeton;
        nb_alignement=G_tmp.nb_alignement;
        all_jeton=new jeton[nb_joueur];
        n_jeton=new int[nb_joueur];
        al_J=new int[nb_joueur];
        Table=new jeton *[nb_ligne];
        for(int i=0; i<nb_ligne; i++)
                Table[i]=new jeton [nb_colonne];
        preremplir_grille();
        for(int i=0;i<nb_joueur;i++)
                al_J[i]=G_tmp.al_J[i];
        for(int i=0;i<nb_joueur;i++)
        {
                all_jeton[i].forme=G_tmp.all_jeton[i].forme;
                all_jeton[i].couleur=G_tmp.all_jeton[i].couleur;
                all_jeton[i].marquage=G_tmp.all_jeton[i].marquage;
        }
        for(int i=0; i<nb_ligne; i++)
        {
                for(int j=0; j<nb_colonne; j++)
                {
                        Table[i][j].forme=G_tmp.Table[i][j].forme;
                        Table[i][j].couleur=G_tmp.Table[i][j].couleur;
                        Table[i][j].marquage=G_tmp.Table[i][j].marquage;
                }
        }
        for(int i=0;i<nb_joueur;i++)
            n_jeton[i]=G_tmp.n_jeton[i];
}

int grille::renvoi_nb_al_necessaire()
{
        return nb_alignement;
}

void grille::annule_placement(int colonne)
{
        for(int i=0; i<nb_ligne; i++)
        {
                if(Table[i][colonne-1].forme!='.')
                {
```

```
                        Table[i][colonne-1].forme='.';
                        Table[i][colonne-1].couleur=0;
                        break;
                }
        }

}

int grille::renvoie_nb_alignement(int joueur)
{
        return al_J[joueur-1];
}

bool grille::grille_remplie()
{
        for(int i=0; i<nb_colonne; i++)
                if(colonne_remplie(i+1)!=true)
                        return false;
        return true;
}

int grille::renvoie_nb_jeton_aligne(int joueur)
{
        return n_jeton[joueur-1];
}
```