Coordonnees

Attributs

— int x
— int y

Destructeur

— **~Coordonnees**()

Constructeurs

— **Coordonnees**()
— **Coordonnees** (int,int)
— **Coordonnees** (**const** Coordonnees&)

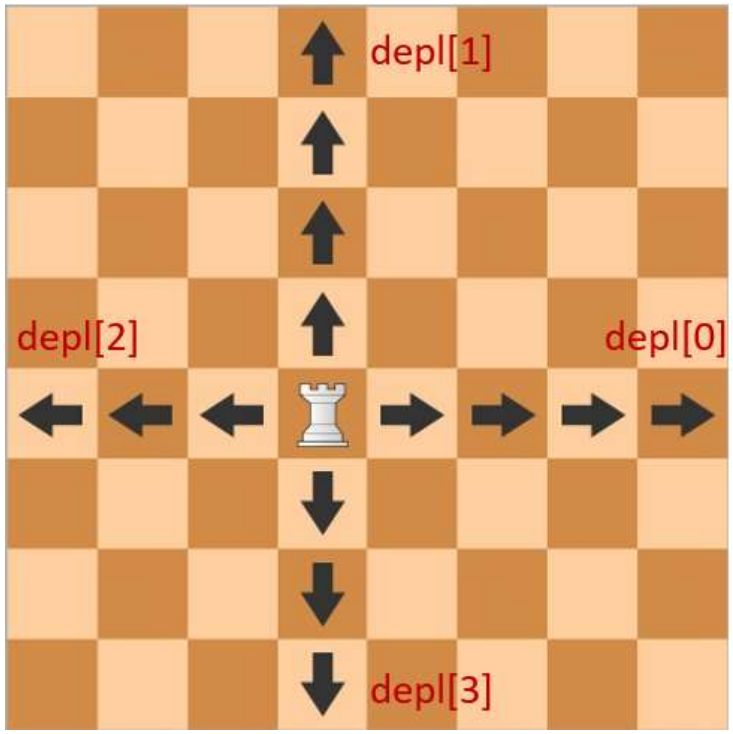
Autres fonctions

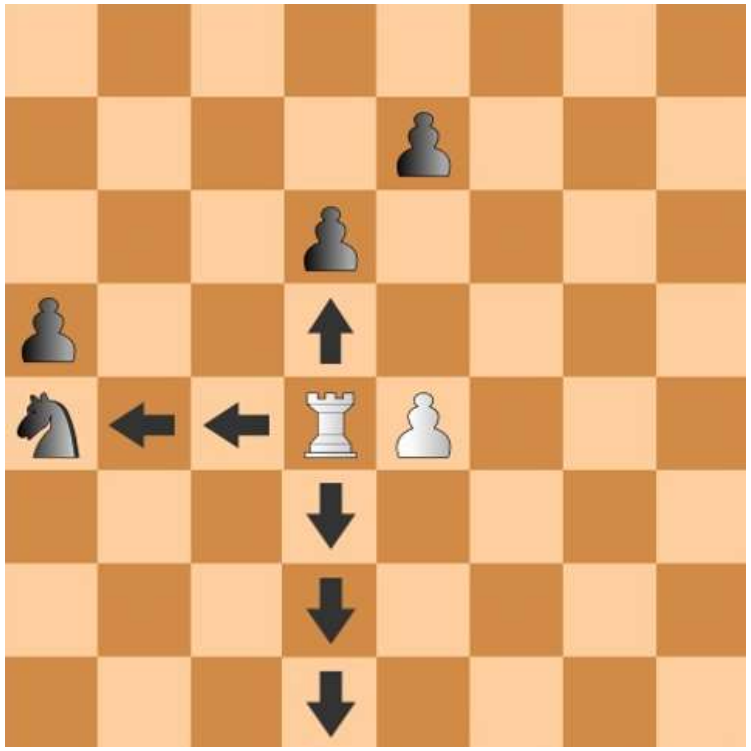
— **affiche**()
— **get_x**()
— **get_y**()
— **set_x**(int)
— **set_y**(int)
— **verif_coord**()

TypePiece

string type	int valeur
"Pion"	1
"Tour"	5
"Fou"	3
"Reine"	9
"Cavalier"	3
"roi"	0

Coordonnees ** depl: tableau des coordonnées relatifs.
int nb_lignes : nombres des lignes de depl.
int nb_colonnes :nombre des colonnes de depl.





Constructeurs

— **TypePiece**(string)
— **TypePiece**()
— **TypePiece**(const TypePiece&)

Destructeur

— **~TypePiece**()

Autres fonctions

— **affiche**()
— **get_type**()
— **get_depl**()
— **get_nb_lignes**()
— **get_nb_colonnes**()
— **change_Type**(string)
— **operator =** un opérateur d’égalité

Piece

TypePiece type

Coordonnees coord

Int couleur

int deplace

Constructeurs

— **Piece**(TypePiece,int,Coordonnees)
— **Piece**()
— **Piece**(**const** Piece &)

Destructeur

— **~Piece**()

Autres fonctions

— **affiche**()
— **get_coul**()
— **get_coord**()
— **set_coord**(Coordonnees)
— **get_TypePiece**()
— **set_type**()
— **get_deplace**()

Coup

Attributs

— int coul
— Piece P
— Coordonnees av
— Coordonnees ap
— int pr
— int sp

Constructeurs

— **Coup**()
— **Coup**(**const** Coup &)
— **Coordonnees** (**const** Coordonnees &)
— **operator =** un opérateur d'égalité

Destructeur

— **~Coup**()

Autres fonctions

— **get_coul**()
— **get_P**()
— **get_av**()
— **get_ap**()
— **get_pr**()
— **get_sp**()

Echiquier

Attributs

— int **dim**
— Piece*** **plateau**

Destructeur

— **~Echiquier**()

Constructeurs

— **Echiquier**()
— **Echiquier**(**const** Echiquier &)

Autres fonctions

— **get_Piece**(int, int)
— **get_plateau**()
— **Affiche**()
— **modif_echiquier**(Coordonnees, Coordonnees)
— **deplace**(Coordonnees, Coordonnees)
— **jouer_coup**(Coup)
— **controle**(int)
— **affiche_controle**(bool **)
— **verif_deplac**(Piece*, Coordonnees)
— **Piece_controle**(Coordonnees)
— **Echec**(int)
— **operator =** un opérateur d'égalité

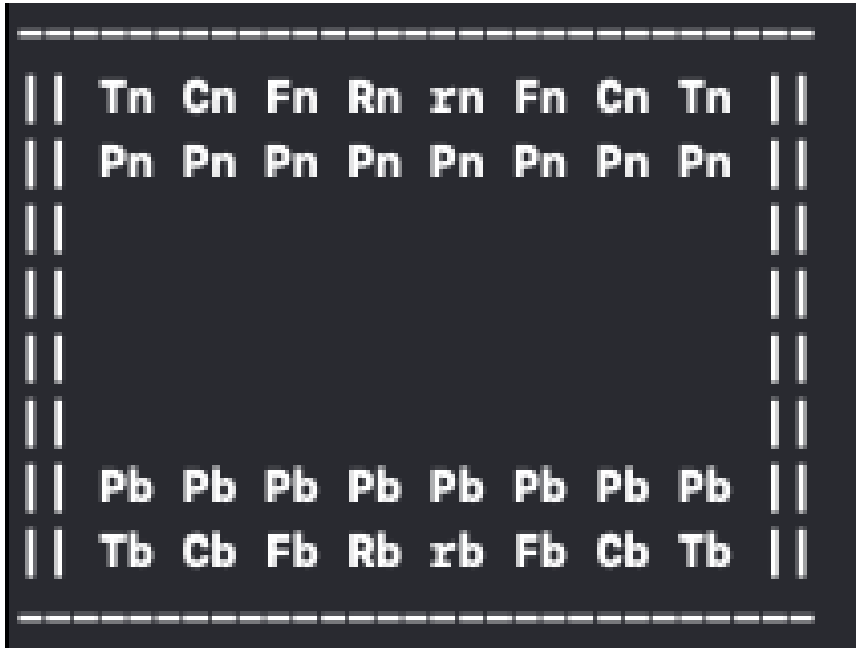
void Affiche()

- Cette fonction sert à afficher notre échiquier sur le terminal, tout en l'initialisant en se basant sur une convention.

```
void Echiquier::Affiche()   // Un affichage simple de l'echiquier (pour l'instant)
{
    cout << endl<<"-----------------------------"<<endl;
    for (int j=dim-1; j>-1; j--)
    {
        cout << "|| ";
        for (int i=0; i<dim; i++)
        {

            Piece *pt = this->get_plateau()[i][j];
            if (pt != NULL)
            {
                if (pt->get_coul() == 0)
                {
                    cout << pt->get_TypePiece().get_type()[0] << "b" << " ";
                }
                else
                {
                    cout << pt->get_TypePiece().get_type()[0] << "n" << " ";
                }

            }
            else
            {
                cout << "   ";
            }
        }
        cout << "||" << endl;
    }
    cout <<"-----------------------------"<<endl;
}
```



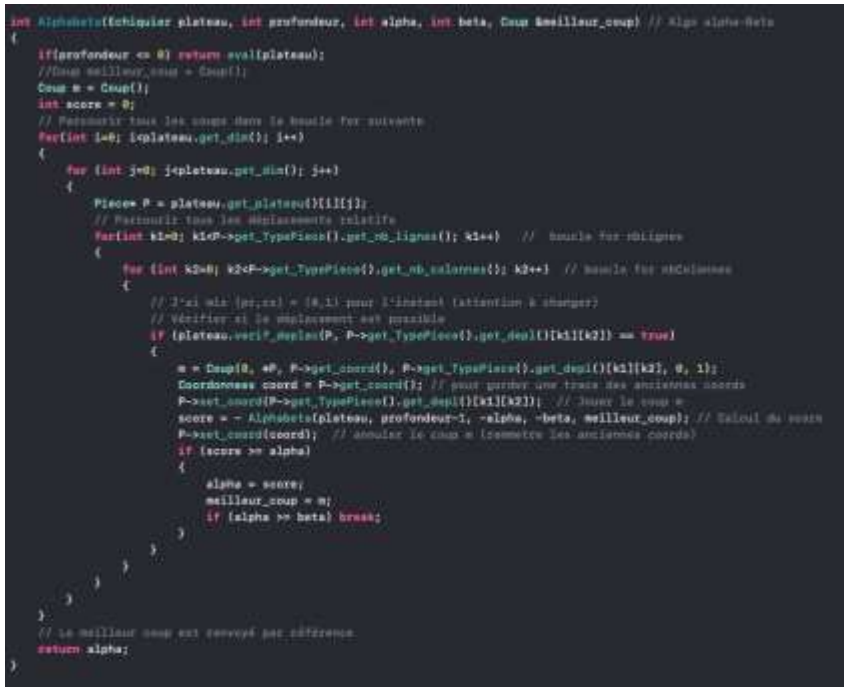
int eval(...)

- Une fonction d'évaluation qui retourne la valeur d'une position de l'échiquier, en utilisant la formule suivante :

$$\gamma_p \left(\text{val\_piece}(j_1) - \text{val\_piece}(j_2) \right) + \gamma_c \left(\text{cont}(j_1) - \text{cont}(j_2) \right)$$

int Alphabeta(…)

- Le principe de la recherche d'un coup par MinMax est un processus de réflexion très naturel que tout un chacun a déjà mis en œuvre.
- Mais les informations ne circulent que dans un sens : des feuilles vers la racine



Ainsi le principe de l'algorithme Alphabeta, est justement d'éviter la génération de feuilles et de parties de l'arbre qui sont inutiles.

Convention : Negamax

```
bool Echiquier::Echec(int coul)
{

    // en échec si le roi se trouve dans une position atteignable par l'adversaire (fonction contrôle)
    // On cherche la position du roi, de couleur coul (puis on applique ce qui écrit au dessus)


    // On parcourt notre plateau
    for (int i=0; i<dim; i++)
    {

        for (int j=0; j<dim; j++)
        {
            // Puis on cherche le roi de couleur coul
            if (plateau[i][j]!=nullptr  && (plateau[i][j]->get_coul() == coul) &&
                (plateau[i][j]->get_TypePiece().get_type() == "roi"))
            {
                // Puis on vérifie si ce dernier est atteignable ou pas
                return Piece_controle(Coordonnees(i,j));

            }
        }
    }
    return false;
}
```

bool Echec(int coul)

- Le but de la fonction Echec() est assez évidente, car elle renvoie un booléen montrant si le joueur contrôlant les pièces de couleur **coul** est en échec (**true**) ou pas (**false**)

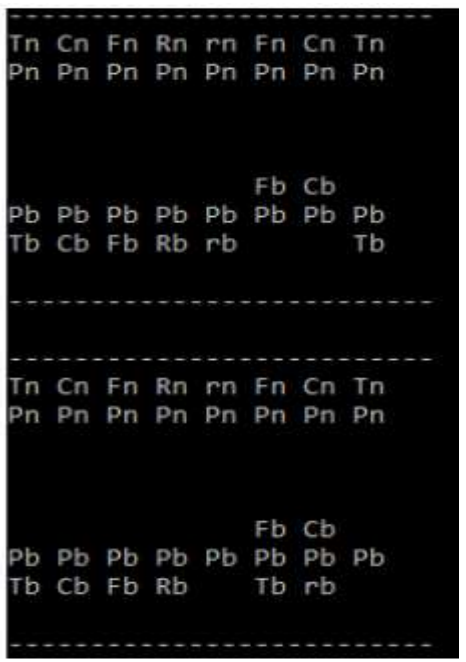
void deplace (Coordonnees av, Coordonnees ap)

Déplacer la Pièce se trouvant aux coordonnées av aux coordonnées ap.

void jouer_coup (Coup C)

Cette fonction permet de jouer un Coup C valide. Elle vérifie si c'est un Coup spécial et s'il y a prise de Pièce ou pas.

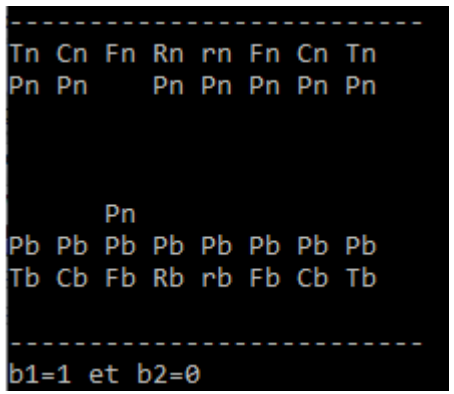
```
Echiquier EC;
EC.deplace(Coordonnees(5,0),Coordonnees(5,2));
EC.deplace(Coordonnees(6,0),Coordonnees(6,2));
EC.Affiche();
TypePiece TP("roi");
Coordonnees av(4,0);
Coordonnees ap(6,0);
Coup c(0,Piece(TP,0,av),av,ap,0,1);
EC.jouer_coup(c);
EC.Affiche();
```



bool verif_deplac (Piece*P, Coordonnees cor)

Cette fonction vérifie si le déplacement de la Piece P vers cor est possible ou non.

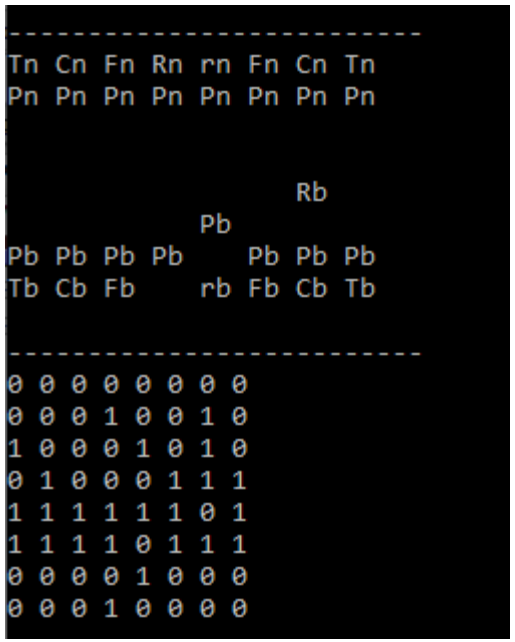
```
Echiquier EC;
EC.deplace(Coordonnees(2,6),Coordonnees(2,2));
EC.Affiche();
bool b1=EC.verif_deplac(EC.get_Piece(1,1),Coordonnees(2,2));
bool b2=EC.verif_deplac(EC.get_Piece(1,1),Coordonnees(1,4));
cout<< "b1="<<b1 << " et b2="<<b2;
```



bool ** controle (int coul)
void affiche_controle(bool **)

Déterminer les cases contrôlées par un joueur de couleur coul.

```
Echiquier EC;
EC.deplace(Coordonnees(4,1),Coordonnees(4,2));
EC.deplace(Coordonnees(3,0),Coordonnees(6,3));
EC.Affiche();
EC.affiche_controle(EC.controle(0));
```



Coup retourne_coup_humain (int tour)

Cette fonction permet de communiquer avec le joueur pour proposer un Coup.

```
Pb Pb Pb Pb Pb Pb Pb Pb
Tb Cb Fb Rb rb Fb Cb Tb

-------------------------
Le tour est au joueur blanc
jouer (1) ou abandonner (0)
1
coordonnees x,y de la piece a jouer
0 1
Confirmer le choix de la piece par 1 ou 0 sinon
1
coordonnees x,y de la nouvelle position
1 2
coordonnees x,y de la nouvelle position
0 3

-------------------------
Tn Cn Fn Rn rn Fn Cn Tn
Pn Pn Pn Pn Pn Pn Pn Pn


Pb


   Pb Pb Pb Pb Pb Pb Pb
Tb Cb Fb Rb rb Fb Cb Tb

-------------------------
```

```
-------------------------
Tn Cn Fn Rn rn Fn Cn Tn
Pn Pn Pn Pn Pn Pn Pn Pn


Rb    Pb

Pb Pb    Pb Pb Pb Pb Pb
Tb Cb Fb    rb Fb Cb Tb

-------------------------
Le tour est au joueur noir
jouer (1) ou abandonner (0)
1
coordonnees x,y de la piece a jouer
3 6
Confirmer le choix de la piece par 1 ou 0 sinon
1
coordonnees x,y de la nouvelle position
3 5
Attention au roi !
coordonnees x,y de la nouvelle position
```

Algorithme humain vs humain

```
int tour=0;
Echiquier EC;
while (1){
    EC.Affiche();
    Coup c=EC.retourne_coup_humain(tour);
    Coordonnees resign=c.get_ap();
    if (resign.get_x()==-1){
        if (tour==0){
            cout<< "le gagnant est le joueur noir" <<endl;
            break;
        }
        else{
            cout<< "le gagnant est le joueur blanc" <<endl;
            break;
        }
    }
    else{
        EC.jouer_coup(c);
    }
    tour=1-tour;
}
```

À vous de jouer !