

# 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()

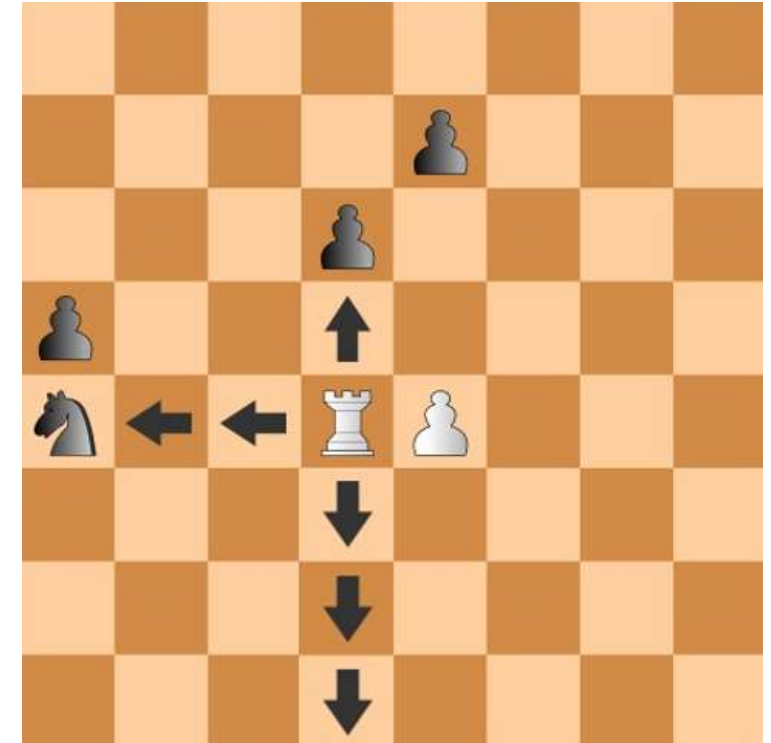
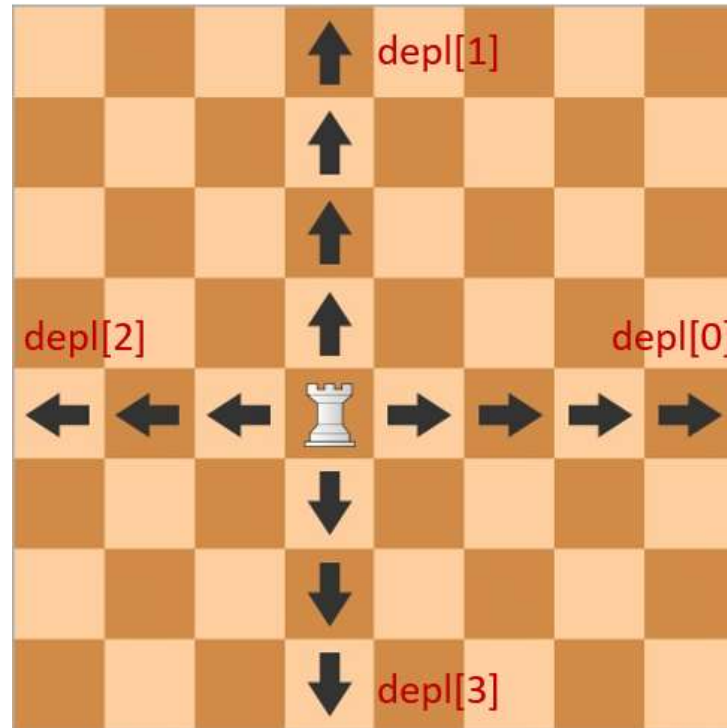
# TypePiece

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**

## Constructeurs

- Echiquier()
- Echiquier(**const** Echiquier &)

## Destructeur

- ~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;
}
```

```
-----
|| Tn Cn Fn Rn xn Fn Cn Tn ||
|| Pn Pn Pn Pn Pn Pn Pn Pn ||
||                               ||
||                               ||
||                               ||
||                               ||
||                               ||
|| Pb Pb Pb Pb Pb Pb Pb Pb ||
|| Tb Cb Fb Rb xb Fb Cb Tb ||
-----
```



## int eval(...)

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

```
double eval(Echiquier plateau)
{
    double gamma_p = .5; // poids arbitraire (à faire varier lors des tests)
    double gamma_c = .5; // poids arbitraire (à faire varier lors des tests)
    int val_h = 0; // valeur des pièces du joueur humain
    int val_c = 0; // valeur des pièces du l'ordinateur
    int cont_h = 0; // nombre de cases contrôlées par l'humain
    int cont_c = 0; // nombre de cases contrôlées par l'ordinateur
    bool == control_humain = plateau.controls[0]; // les cases contrôlées par l'humain (coul blanc)
    bool == control_ordi = plateau.controls[1]; // les cases contrôlées par l'ordinateur (coul noir)

    for (int i=0; i < 8; i++)
    {
        for (int j=0; j < 8; j++)
        {
            if (control_humain[i][j] == true) cont_h++;
            if (control_ordi[i][j] == true) cont_c++;
        }
    }

    bool echec = plateau.chech();

    // retourner des pour une sortie nulle

    if (echec == true) {return +INFINITY;} // Win gagnante du joueur humain = pos p de c
    else if (echec == false) {return -INFINITY;} // Win perdante du joueur humain = pos p de c
    else

        for (int i=0; i<plateau.get_dim(); i++)
        {
            for (int j=0; j<plateau.get_dim(); j++)
            {
                if (plateau.get_plateau(i)[j]->get_coul() == 0) // joueur humain (pièce blanche)
                {
                    val_h += plateau.get_plateau(i)[j]->get_typePiece().get_valeur();
                }
                else // l'ordinateur (pièce noire)
                {
                    val_c += plateau.get_plateau(i)[j]->get_typePiece().get_valeur();
                }
            }
        }

    return gamma_p*(val_h-val_c) + gamma_c*(cont_h-cont_c);
}
```

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

# 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

```
int Alphabeta(échiquier plateau, int profondeur, int alpha, int beta, Coup *meilleur_coup) // Algo alpha-beta
{
    if(profondeur <= 0) return eval(plateau);
    //Donc meilleur_coup = Coup();
    Coup m = Coup();
    int score = 0;
    // Parcourir tous les coups dans la boucle for suivante
    for(int i=0; i<plateau.get_dim(); i++)
    {
        for (int j=0; j<plateau.get_dim(); j++)
        {
            Pieces P = plateau.get_plateau()[i][j];
            // Parcourir tous les déplacements relatifs
            for(int k1=0; k1<P->get_TypePiece().get_nb_lignes(); k1++) // boucle for colonnes
            {
                for (int k2=0; k2<P->get_TypePiece().get_nb_colonnes(); k2++) // boucle for lignes
                {
                    // 2*si m1x {pr,col} = {0,1} pour l'instant (attention à changer)
                    // Vérifier si le déplacement est possible
                    if (plateau.verif_deplacement(P, P->get_TypePiece().get_depl()[k1][k2]) == true)
                    {
                        m = Coup(i, j, P->get_cord(), P->get_TypePiece().get_depl()[k1][k2], 0, 1);
                        Coordonnees coord = P->get_cord(); // pour garder une trace des anciennes coord
                        P->set_cord(P->get_TypePiece().get_depl()[k1][k2]); // Jouer le coup
                        score = - Alphabeta(plateau, profondeur-1, -alpha, -beta, meilleure_coup); // Calcul du score
                        P->set_cord(coord); // annuler le coup et remettre les anciennes coord
                        if (score >= alpha)
                        {
                            alpha = score;
                            meilleur_coup = m;
                            if (alpha >= beta) break;
                        }
                    }
                }
            }
        }
    }
    // Le meilleur coup est renvoyé par référence
    return alpha;
}
```

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 Echec(int coul)

```
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;
}
```

- 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();
```

```
-----  
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 Tb  
  
-----  
  
-----  
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 Tb rb  
  
-----
```

## 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;
```

```
-----  
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  
  
-----  
b1=1 et b2=0
```

```
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));
```

```
-----
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
-----
0 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0
1 0 0 0 1 0 1 0
0 1 0 0 0 1 1 1
1 1 1 1 1 1 0 1
1 1 1 1 0 1 1 1
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 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;
}
```



A close-up photograph of a hand moving a silver chess piece on a reflective board. The hand is positioned on the right side of the frame, with fingers gripping a silver pawn. The chessboard is highly reflective, showing clear reflections of the pieces and the hand. In the background, other silver chess pieces are visible but out of focus. The overall lighting is bright and soft, creating a clean and professional aesthetic. The text 'À vous de jouer !' is overlaid on the left side of the image in a brown, serif font, with a horizontal line underneath it.

À vous de jouer !