```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>

// les constantes
#define NB_LIGNES 6
#define NB_COLONNES 7
#define PION_A 'X'
#define PION_B 'O'
#define VIDE ' '
#define INCONNU ' '

// les types
typedef int Grille[NB_LIGNES][NB_COLONNES];

// prototypes des fonctions
void initGrille(Grille laGrille);
void afficher(Grille laGrille, char pion);
bool grillePleine(Grille laGrille);
void jouerA(Grille laGrille, char pion, int * ligne, int * colonne);
void jouerB(Grille laGrille, char pion, int * ligne, int * colonne);
int choisirColonne(Grille laGrille, char pion);
int chercherLigne(Grille laGrille, int col);
int alignementMax(Grille, int, int);
bool estVainqueur(Grille laGrille, int lig, int col);
void finDePartie(char vainqueur);
int choisirColonneStrategie1(Grille, char);
int choisirColonneStrategie2(Grille, char);
int choisirColonneStrategie3(Grille, char);
int choisirColonneStrategie4(Grille, char);
int choisirColonneStratPourrie(Grille, char);

// Programme principal. C'est le pion A qui commence Ã  jouer
int main()
{
    srand(time(NULL));
    int i, N, compteurA, compteurB;
    compteurA = 0;
    compteurB = 0;
    N = 100;
    for (i = 0; i < N; i++) {
        Grille laGrille;
        char vainqueur=INCONNU;
        int ligne, colonne;
        initGrille(laGrille);
        afficher(laGrille, PION_A);
        while (vainqueur==INCONNU && !grillePleine(laGrille)){
            jouerA(laGrille,PION_A, &ligne, &colonne);
            afficher(laGrille, PION_B);
            if (estVainqueur(laGrille, ligne, colonne) ){
                vainqueur = PION_A;
                compteurA++;
            } else if (!grillePleine(laGrille)){
                jouerB(laGrille, PION_B, &ligne, &colonne);
                afficher(laGrille, PION_A);
                if (estVainqueur(laGrille, ligne, colonne) ){
                    vainqueur = PION_B;
                    compteurB++;
                }
            }
        }
        finDePartie(vainqueur);
    }
```

```
    printf("X : %d, O : %d", compteurA, compteurB);
    system(EXIT_SUCCESS);
}


void initGrille(Grille laGrille){
    int l, c;
    for (l=0 ; l<NB_LIGNES ; l++){
        for (c=0 ; c<NB_COLONNES ; c++){
            laGrille[l][c] = VIDE;
        }
    }
}

void afficher(Grille laGrille, char pion){
    int l, c;
    system("clear");
    printf("\t");
    printf("  %c\n", pion);
    printf("\t");
    for (c=0; c<NB_COLONNES ; c++){
        printf("----");
    }
    printf("-\n");
    for (l=0 ; l<NB_LIGNES ; l++){
        printf("\t");
        for (c=0; c<NB_COLONNES ; c++){
            printf("| %c ", laGrille[l][c]);
        }
        printf("|\n");
        printf("\t");
        for (c=0; c<NB_COLONNES ; c++){
            printf("----");
        }
        printf("-\n");
    }
    printf("\t");
    for (c=0; c<NB_COLONNES ; c++){
            printf("  %d ",c);
    }
    printf("\n\n");

}

bool grillePleine(Grille laGrille){
    bool pleine = true;
    int c = 0;
    while (pleine && c<NB_COLONNES){
        if (laGrille[0][c] == VIDE){
            pleine = false;
        }
        c++;
    }
    return pleine;
}

void jouerA(Grille laGrille, char pion, int * ligne, int * colonne ){
   *ligne = -1;
    do {
        *colonne = choisirColonne(laGrille, pion);
        *ligne = chercherLigne(laGrille, *colonne);
    }while (*ligne==-1);
    laGrille[*ligne][*colonne] = pion;
}
```

```
void jouerB(Grille laGrille, char pion, int * ligne, int * colonne ){
   *ligne = -1;
    do {
        *colonne = choisirColonneStratPourrie(laGrille, pion);
        *ligne = chercherLigne(laGrille, *colonne);
    }while (*ligne==-1);
    laGrille[*ligne][*colonne] = pion;
}

int choisirColonne(Grille laGrille, char pion){
    int col;

    do{
        printf("Numero de colonne ? ");
        scanf("%d", &col);
    } while (col<0 ||col>6);
    return col;
}

int chercherLigne(Grille laGrille, int col){
    int ligne = -1;

    while (ligne<NB_LIGNES-1 && laGrille[ligne+1][col]==VIDE){
        ligne++;
    }
    return ligne;
}

int choisirColonneStrategie1(Grille laGrille, char pion) {
    int col;
    col = 0;
    while(chercherLigne(laGrille, col) == -1) {
        col++;
    }
    return col;
}

int choisirColonneStrategie2(Grille laGrille, char pion) {
    int indice;
    int max = -2;
    for(int col = 0 ; col < NB_COLONNES ; col++){
        if(chercherLigne(laGrille, col) > max){
            max = chercherLigne(laGrille, col);
            indice = col;
        }
    }
    return indice;
}

int choisirColonneStrategie3(Grille laGrille, char pion) {
    int col, i;
    i = 1;
    col = NB_COLONNES/2;
    while (chercherLigne(laGrille, col) == -1) {
        if (chercherLigne(laGrille, col-i) != -1) {
            col = col - i;
        } else if (chercherLigne(laGrille, col+i) != -1) {
            col = col + i;
        }
        i++;
    }
    return col;
}
```

```
int choisirColonneStrategie4(Grille laGrille, char pion) {
    int col;
    do {
        col = rand() % NB_COLONNES;
    } while (chercherLigne(laGrille, col) == -1);
    return col;
}

int choisirColonneStratPourrie(Grille laGrille, char pion) {
    bool meilleur;
    meilleur = false;
    int col, lig, alignement;
    col = 0;

    //cas de victoire directe
    while (col < NB_COLONNES && meilleur == false) {
        lig = chercherLigne(laGrille, col);
        if (lig > NB_LIGNES) {
            laGrille[lig][col] = pion;
            if (estVainqueur(laGrille, lig, col)) {
                meilleur = true;
            }
        laGrille[lig][col] = VIDE;
        }
        col++;
    }

    //sinon on veut bloquer l'adversaire
    if (!meilleur) {
        col = 0;
        char adverse;

        if (pion == PION_A) {
            adverse = PION_B;
        } else {
            adverse = PION_A;
        }

        while (col < NB_COLONNES && meilleur == false) {
            lig = chercherLigne(laGrille, col);
            laGrille[lig][col] = adverse;
            alignement = alignementMax(laGrille, lig, col);
            if (alignement >= 3) {
                meilleur = true;
            }
            laGrille[lig][col] = VIDE;
            col++;
        }
        alignement = 4;
    }

    //sinon on veut aligner le maximum de pions
    if (!meilleur) {
        int max, meilleurCol;
        max = 0;
        col = 0;
        while (col < NB_COLONNES && meilleur == false) {
            lig = chercherLigne(laGrille, col);
            laGrille[lig][col] = pion;
            if (alignementMax(laGrille, lig, col) > max) {
                max = alignementMax(laGrille, lig, col);
                meilleurCol = col;
            }
```

```
            laGrille[lig][col] = VIDE;
            col++;
        }
        col = meilleurCol+1;
    }
    return col-1;
}

int alignementMax(Grille laGrille, int lig, int col){
    // consiste Ã  regarder si une ligne de 4 pions s'est formÃ©e autour du pion
qui vient de tomber en (lig, col)
    char lePion = laGrille[lig][col];
    int cpt, i, j, max;

    // regarder la verticale, en comptant le nombre de pions au Sud (inutile de
regarder au Nord du pion qui vient de tomber)
    i = lig;
    cpt = 0;
    while (i<NB_LIGNES && laGrille[i][col]==lePion){
        cpt++;
        i++;
    }
    max = cpt;
    // regarder l'horizontale, en comptant le nombre de pions Ã  l'Est et Ã 
l'Ouest
    j = col;
    cpt = 0;
    // on regarde Ã  l'est
    while (j>=0 && laGrille[lig][j]==lePion){
        cpt++;
        j--;
    }
    j = col+1;
    // on regarde Ã  l'ouest
    while (j<NB_COLONNES && laGrille[lig][j]==lePion){
        cpt++;
        j++;
    }
    if (cpt > max) {
        max = cpt;
    }
    // regarder la diagonale descendante, en comptant le nombre de pions au Nord
Ouest et au Sud Est
    i = lig;
    j = col;
    cpt = 0;
    // on regarde au Nord Ouest
    while (j>=0 && i>=0 && laGrille[i][j]==lePion){
        cpt++;
        i--;
        j--;
    }
    i = lig+1;
    j = col+1;
    // on regarde au Sud Est
    while (i<NB_LIGNES && j<NB_COLONNES && laGrille[i][j]==lePion){
        cpt++;
        i++;
        j++;
    }
    if (cpt > max) {
        max = cpt;
    }
```

```
    // regarder la diagonale descendante, en comptant le nombre de pions au Nord
Est et au Sud Ouest
    i = lig;
    j = col;
    cpt = 0;
    // on regarde au Nord Est
    while (j<NB_COLONNES && i>=0 && laGrille[i][j]==lePion){
        cpt++;
        i--;
        j++;
    }
    i = lig+1;
    j = col-1;
    // on regarde au Sud Ouest
    while (i<NB_LIGNES && j>=0 && laGrille[i][j]==lePion){
        cpt++;
        i++;
        j--;
    }
    if (cpt > max) {
        max = cpt;
    }
    return max;
}

bool estVainqueur(Grille laGrille, int lig, int col) {
    if (alignementMax(laGrille, lig, col) >= 4) {
        return true;
    }
    return false;
}

void finDePartie(char vainqueur){
    if (vainqueur != INCONNU){
        printf("Joueur %c vainqueur\n", vainqueur);
    } else {
        printf("MATCH NUL");
    }
}
```