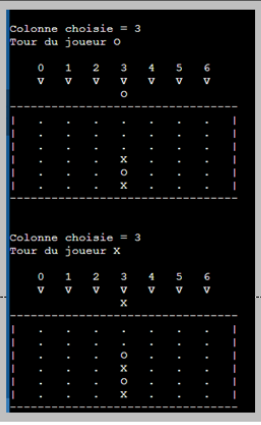
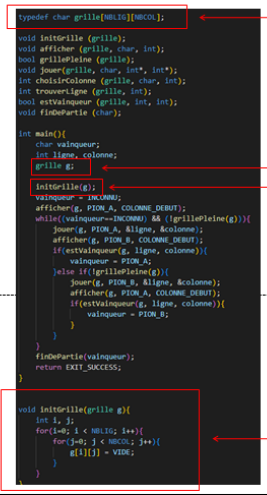
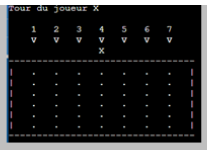
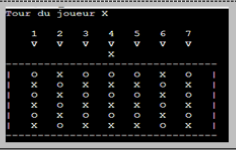
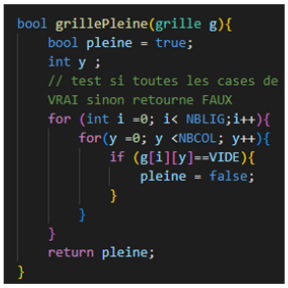
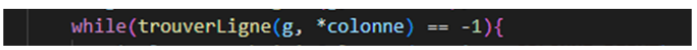
résultat attendu	Capture d'écran	Capture d'écran du code
grille initialisée grille initialisée grille initialisée	 La grille s'initialise qu'une fois avec des cases contenants VIDE puis au fur et à mesure que la partie avance on remplace le contenu des cases par les pions, donc si la grille n'est pas initialisée, elle s'initialise et si elle est déjà initialisé elle n'a pas besoin de se réinitialisé une seconde fois donc on ne met pas la procédure initGrille dans la boucle de jeu.	 On définie un tableau s'appelant grille au format NBCOL collonnes et NBLIG lignes. On initialise une variable g de type grille prenant les mêmes paramètres que le tableau grille et on l'envoie dans initGrille. On initialise le tableau avec des constante VIDE désignant les cases où les pions iront se longer.

grille affichée sans pion		
grille affichée avec 42 pions		
grille affichée		

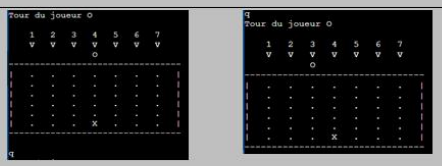
False		
True		

ligne et colonne conformes		
invite à choisir une autre colonne		

redemande la saisie


```
default:
    strcpy(consigne, "Erreur : Entrer 'q' pour aller à gauche, 'd' pour aller à droite, 'SPACE' pour choisir la colonne et 'ENTER' pour valider votre action.\n");
    break;
```

pion déplacé vers la gauche (n° de colonne décrémenté)

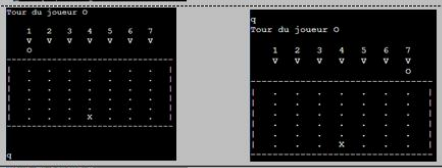



```
case 'q':
    if (col > 0){
        col = col - 1;
```

pion déplacé vers la droite (n° colonne incrémenté)

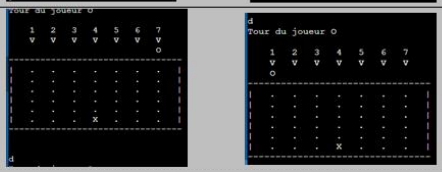

```
case 'd':
    if (col < NBCOL - 1){
        col = col + 1;
```

rien (ou bien boucle en colonne 7)



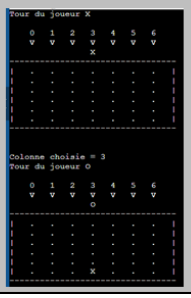

```
case 'q':
    if (col > 0){
        col = col - 1;
    } else {
        col = NBCOL -1;
    }
    break;
```

rien (ou bien boucle en colonne 1)



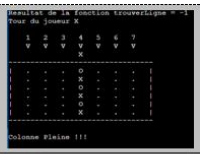

```
case 'd':
    if (col < NBCOL - 1){
        col = col + 1;
    } else {
        col = 0;
    }
    break;
```

retourne le n° de colonne courant

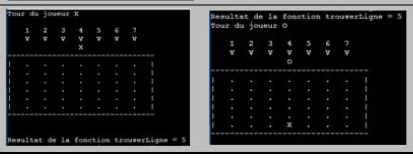



```
    }
    printf("Colonne choisie = %d\n", col);
    return col;
}
```

-1



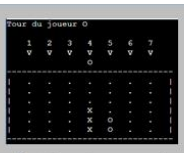
le n° ligne correspond à la première case libre de la colonne




```
int trouverLigne (grille g, int colonne){
    int ligne = -1;

    while (ligne < NBLIG && g[ligne+1][colonne] == VIDE){
        ligne = ligne + 1;
    }
    printf("Résultat de la fonction trouverLigne : %d\n", ligne);
    return ligne;
}
```

False




```
bool estVainqueur (grille g, int ligne, int colonne){
    const int VICTOIRE = 4;
    char pion;
    pion = g[ligne][colonne];
```

```
    /*************** Victoire ? ***************/
    if(som_h >= VICTOIRE || som_v >= VICTOIRE || som_dm >= VICTOIRE || som_dd >= VICTOIRE){
        return true;
    }
    else{
        return false;
    }
}
```

True


```
/*************** Vérification vertical ***************/
/* En Dessous*/
int bottom;
int som_v;

bottom=1;
som_v=1;
while( ligne + bottom < NBLIG && (bottom < VICTOIRE) && (g[ligne + bottom][colonne] == pion)){
    som_v = som_v +1;
    bottom++;
}
if(som_v < VICTOIRE){
    som_v = 1;
}
```

True


```
/*************** Vérification horizontal ***************/
/* Gauche */
int left;
int som_h;

left=1;
som_h=1;
while((colonne - left >= 0) && (left < VICTOIRE) && (g[ligne][colonne - left] == pion)){
    som_h = som_h +1;
    left++;
}
if(som_h < VICTOIRE){
    som_h = 1;
}

/* Droite*/
int right;

right=1;
while((colonne + right < NBCOL) && (right < VICTOIRE) && (g[ligne][colonne + right] == pion)){
    som_h = som_h +1;
    right++;
}
if(som_h < VICTOIRE){
    som_h = 1;
}
```

True


```
/*************** Vérification diagonale descendante ***************/
/* Vers le haut à gauche*/
int top_left;
int som_dd;

top_left=1;
som_dd=1;
while((colonne + top_left) < NBCOL) && (ligne - top_left >= 0) && (top_left < VICTOIRE) && (g[ligne - top_left][colonne + top_left] == pion)){
    som_dd = som_dd +1;
    top_left++;
}
if(som_dd < VICTOIRE){
    som_dd = 1;
}

/* Vers le bas à droite*/
int bottom_right;

bottom_right=1;
while((colonne - bottom_right) >= 0) && (ligne + bottom_right < NBLIG) && (bottom_right < VICTOIRE) && (g[ligne + bottom_right][colonne - bottom_right] == pion)){
    som_dd = som_dd +1;
    bottom_right++;
}
if(som_dd < VICTOIRE){
    som_dd = 1;
}
```

True

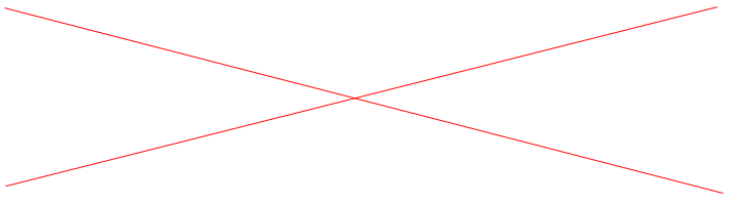

```
/*************** Vérification diagonale montante ***************/
/* Vers le haut à droite*/
int top_right;
int som_dm;

top_right=1;
som_dm=1;
while((colonne + top_right) < NBCOL) && (ligne + top_right < NBLIG) && (top_right < VICTOIRE) && (g[ligne + top_right][colonne + top_right] == pion)){
    som_dm = som_dm +1;
    top_right++;
}
if(som_dm < VICTOIRE){
    som_dm = 1;
}

/* Vers le bas à gauche*/
int bottom_left;

bottom_left=1;
while((colonne - bottom_left) >= 0) && (ligne - bottom_left >= 0) && (bottom_left < VICTOIRE) && (g[ligne - bottom_left][colonne - bottom_left] == pion)){
    som_dm = som_dm +1;
    bottom_left++;
}
if(som_dm < VICTOIRE){
    som_dm = 1;
}
```

False



affiche "X vainqueur"


```
void finDePartie (char vainqueur){
    if(vainqueur == PION_A){
        printf("**********************\n");
        printf("**** FIN DE PARTIE ****\n");
        printf("**********************\n");
        printf("**** VICTOIRE DE %c ****\n", PION_A);
        printf("**********************\n");
    }
    else if(vainqueur == PION_B){
        printf("**********************\n");
        printf("**** FIN DE PARTIE ****\n");
        printf("**********************\n");
        printf("**** VICTOIRE DE %c ****\n", PION_B);
        printf("**********************\n");
    }
    else{
        printf("**********************\n");
        printf("**** FIN DE PARTIE ****\n");
        printf("**********************\n");
        printf("****  MATCH   NUL  ****\n");
        printf("**********************\n");
    }
}
```

affiche "O vainqueur"

affiche "MATCH NUL"