Exercice 1

1. Si $k > n$, il n'est pas possible d'aligner k symboles, donc un match sera toujours nul.
2. En jouant dans le coin $(0,0)$, les trois coups $(0,1)$, $(1,1)$ et $(1,0)$ sont gagnants. Le joueur 2 ne peut en bloquer qu'un seul des trois, donc le joueur 1 gagne lors de son deuxième coup.
3. En jouant en $(1,1)$, le joueur 1 ouvre une possibilité de gagner dans 3 directions : diagonal, horizontal et vertical. Lors de son deuxième coup, le joueur 1 joue en $(2,2)$, $(1,2)$ ou $(2,1)$, correspondant à une direction où le joueur 2 n'a pas joué. Dès lors, le joueur 1 a deux coups gagnants, de part et d'autre de l'alignement de deux pions, et le joueur 2 ne peut en bloquer qu'un des deux, donc le joueur 1 gagne lors de son troisième coup.

Exercice 2

1. On fait les bons `malloc` et la bonne affectation des valeurs.

```
ttt* init_jeu(int k, int n){
    ttt* jeu = malloc(sizeof(*jeu));
    int* grille = malloc((n * n) * sizeof(*grille));
    for (int i=0; i<n*n; i++){
        grille[i] = 0;
    }
    jeu->k = k;
    jeu->n = n;
    jeu->grille = grille;
    return jeu;
}
```

2. On se contente de libérer le tableau (unidimensionnel) et le jeu.

```
void liberer_jeu(ttt* jeu){
    free(jeu->grille);
    free(jeu);
}
```

3. On commence par créer un tableau de taille trois contenant des zéros. Ensuite, on parcourt la grille pour remplir ce tableau selon les valeurs rencontrées.

```
int* repartition(ttt* jeu){
    int* repart = malloc(3 * sizeof(*repart));
    for (int i=0; i<3; i++) repart[i] = 0;
    for (int i=0; i<jeu->n*jeu->n; i++){
        int val = jeu->grille[i];
        assert (0 <= val && val <= 2);
        repart[val] += 1;
    }
    return repart;
}
```

4. Si aucune case ne contient 0, la partie est terminée. Si les deux joueurs ont le même nombre de coups, c'est au joueur 1 de jouer, sinon c'est au joueur 2. On utilise ici l'opérateur ternaire de test.

```c
int joueur_courant(ttt* jeu){
    int* repart = repartition(jeu);
    if (repart[0] == 0){
        free(repart);
        return 0;
    }
    bool b = repart[1] == repart[2];
    free(repart);
    return b?1:2;
}
```

5. On se contente de calculer le bon indice dans le tableau et de modifier la case si possible.

```c
void jouer_coup(ttt* jeu, int cln, int lgn){
    int i = lgn * jeu->n + cln;
    if (jeu->grille[i] != 0){
        printf("Coup impossible\n");
    } else {
        jeu->grille[i] = joueur_courant(jeu);
    }
}
```

Exercice 3

1. La valeur `di` correspond à :

Direction	Est	Sud-est	Sud	Sud-ouest
`di`	1	$n+1$	n	$n-1$

   En effet, chaque ligne est de taille n.
2. Cela fait redondance avec les directions précédentes.
3. On garde en mémoire la colonne et la ligne courantes pour savoir si on sort de la grille. On calcule le différentiel de lignes et de colonnes en fonction du différentiel d'indices. On vérifie alors qu'il y a bien k cases du joueur dans cette direction.

```c
bool alignement(ttt* jeu, int i, int di, int joueur){
    int k = jeu->k, n = jeu->n;
    int cln = i % n, lgn = i / n;
    int dc = ((di + 1) % n) - 1, dl = (di + 1) / n;
    for (int j=0; j<k; j++){
        if (cln < 0 || cln >= n || lgn < 0 || lgn >= n){
            return false;
        }
        i = lgn * n + cln;
        if (jeu->grille[i] != joueur){
            return false;
        }
        cln += dc; lgn += dl;
    }
    return true;
}
```

4. On parcourt l'ensemble des cases possibles, en testant s'il existe un alignement pour les 4 directions possibles (stockées ici dans un tableau statique).

```
bool gagnant(ttt* jeu, int joueur){
    int n = jeu->n;
    int tabdi[4] = {1, n - 1, n, n + 1};
    for (int i=0; i<n*n; i++){
        for (int j=0; j<4; j++){
            if (alignement(jeu, i, tabdi[j], joueur)){
                return true;
            }
        }
    }
    return false;
}
```

Exercice 4

1. C'est une écriture d'un entier en base 3 :

```
int encodage(ttt* jeu){
    int cle = 0, n = jeu->n;
    for (int i=0; i<n*n; i++){
        cle = 3 * cle + jeu->grille[i];
    }
    return cle;
}
```

2. On commence par encoder le jeu, puis par vérifier si la clé est ou non dans le dictionnaire. Si elle n'y est pas, on fait les calculs nécessaires pour l'ajouter. On commence par vérifier si l'état est terminal (l'un des joueurs a gagné ou la grille est remplie). Si ce n'est pas le cas, on calcule l'attracteur auquel appartient chacun des voisins de la configuration courante récursivement, et on calcule à quel attracteur appartient la configuration selon le résultat. Pour explorer les voisins, on parcourt toutes les cases, et pour chaque case vide, on joue un coup, qu'on annule juste après avoir calculé l'attracteur (pour éviter de copier la configuration).

```c
int attracteur(ttt* jeu, dict* d){
    int cle = encodage(jeu);
    int n = jeu->n, joueur = joueur_courant(jeu);
    if (!member(d, cle)){
        if (gagnant(jeu, joueur)){add(d, cle, joueur);
        } else if (gagnant(jeu, 3 - joueur)){
            add(d, cle, 3 - joueur);
        } else if (joueur == 0){
            add(d, cle, 0);
        } else {
            int tab[3] = {0, 0, 0};
            for (int i=0; i<n*n; i++){
                if (jeu->grille[i] == 0){
                    jeu->grille[i] = joueur;
                    int att = attracteur(jeu, d);
                    jeu->grille[i] = 0;
                    tab[att] += 1;
                    if (att == joueur){
                        break;
                    }
                }
            }
            if (tab[joueur] != 0){ add(d, cle, joueur);}
            else if (tab[0] != 0){ add(d, cle, 0);}
            else { add(d, cle, 3 - joueur);}
        }
    }
    return get(d, cle);
}
```

3. On calcule l'attracteur de la position courante, et on cherche une configuration voisine qui a le même numéro d'attracteur.

```c
int strategie_optimale(ttt* jeu, dict* d){
    int att = attracteur(jeu, d);
    int n = jeu->n, joueur = joueur_courant(jeu);
    for (int i=0; i<n*n; i++){
        if (jeu->grille[i] == 0){
            jeu->grille[i] = joueur;
            int att2 = attracteur(jeu, d);
            jeu->grille[i] = 0;
            if (att == att2){
                return i;
            }
        }
    }
}
```

Exercice 5

1. On écrit une petite fonction qui affiche une ligne de séparation dans la grille :

```c
void afficher_ligne_sep(int n){
    printf("\n ");
    for (int i=0; i<n; i++){
        printf("+-");
    }
    printf("+\n");
}
```

Ensuite, l'idée est d'afficher l'entête, puis chaque ligne avec les bons symboles.

```c
void afficher(ttt* jeu){
    int n = jeu->n;
    char tab[3] = {' ', 'X', 'O'};
    printf(" ");
    for (int cln=0; cln<n; cln++){
        // affichage des numéros de colonnes
        printf(" %d", cln);
    }
    afficher_ligne_sep(n);
    for (int lgn=0; lgn<n; lgn++){
        printf("%d|", lgn);
        for (int cln=0; cln<n; cln++){
            int i = n * lgn + cln;
            printf("%c|", tab[jeu->grille[i]]);
        }
        afficher_ligne_sep(n);
    }
    printf("\n");
}
```

2. On commence par une première boucle permettant de déterminer qui commence (la boucle ne s'arrête pas tant qu'on n'a pas saisi la bonne valeur). Ensuite, on crée le dictionnaire permettant de garder en mémoire les attracteurs, et on lance la partie, qui correspond à une boucle `while` qui tourne tant que la partie n'est pas terminée. On commence par afficher la grille. Ensuite, si c'est à l'IA de jouer, on détermine son coup avec la stratégie optimale. Sinon, on demande la saisie au joueur. On pense à recalculer le joueur courant (notamment pour savoir si la partie est terminée).

```c
void jouer_partie(int k, int n){
    ttt* jeu = init_jeu(k, n);
    char c;
    int IA, cln, lgn;
    while (true){
        printf("Voulez-vous commencer ? (o/n) ");
        scanf("%c", &c);
        if (c == 'o'){
            IA = 2;
            break;
        } else if (c == 'n'){
            IA = 1;
            break;
        }
    }
    dict* d = create();
    int joueur = 1;
    while (joueur != 0 && !gagnant(jeu, 1) && !gagnant(jeu, 2)){
        afficher(jeu);
        if (joueur == IA){
            int i = strategie_optimale(jeu, d);
            cln = i % n;
            lgn = i / n;
            printf("L'IA joue ligne %d, colonne %d\n", lgn, cln);
            jouer_coup(jeu, cln, lgn);
        } else {
            printf("C'est a vous de jouer\n");
            printf("Saisir la ligne : ");
            scanf("%d", &lgn);
            printf("Saisir la colonne : ");
            scanf("%d", &cln);
            if (cln < 0 || cln >= jeu->n || lgn < 0 || lgn >= jeu->n){
                printf("Ces coordonnees ne sont pas possibles.\n");
            } else {
                jouer_coup(jeu, cln, lgn);
            }
        }
        joueur = joueur_courant(jeu);
    }
    afficher(jeu);
    if (gagnant(jeu, IA)){
        printf("L'IA a gagne !\n");
    } else if (gagnant(jeu, 3 - IA)){
        printf("Vous avez gagne !\n");
    } else {
        printf("C'est un match nul !\n");
    }
    liberer_jeu(jeu);
    dict_free(d);
}
```