Noms des étudiants :

Buland Thomas

Atmani Abdellah

Talhi Sonia

Morpion (tic-tac-toe) avec deux IA.

I . Architecture des données :

Le projet consiste à programmer un jeu du Morpion (Tic-Tac-Toe) en C++, comprenant trois modes de jeu:
Joueur contre joueur
Joueur contre ordinateur
IA1 contre IA2
*Pour cela, une seule classe a été utilisée : **Morpion**, qui regroupe toutes les données et comportements nécessaires au jeu. Aucune variable n'est placée en private, conformément au choix réalisé dans le projet pour simplifier l'accès.*

```
class Morpion
{
public:

   char grille[3][3];  // Grille de jeu 3x3 contenant ' ', 'X' ou 'O'
   char joueur;       // Joueur courant ('X' ou 'O')
   int lastRow;       // Dernière ligne jouée par l'IA2
   int lastCol;       // Dernière colonne jouée par l'IA2

   Morpion();         // Constructeur : initialise la grille et le joueur
```

II . fonctions et méthods :

- methodes de la classe Morpion :

```
void afficherGrille();  // Affiche la grille du morpion au format 3x3 avec séparateurs.

bool jouer(int l, int c);      // Tente de jouer un coup à la position (l, c).

                               //return  false si la case est invalide ou déjà occupée.

bool victoire();      // Teste si le joueur courant possède une ligne, colonne ou diagonale complète.

bool nul();   // Vérifie si toutes les cases sont remplies (= match nul).
```

```
void changerJoueur();     // Alterne entre 'X' et 'O' après chaque tour.

void jouerIA1();     // IA aléatoire : joue un coup au hasard dans une case vide.

void jouerIA2();     // IA intelligente : essaie de jouer autour de son dernier coup.

                    .
```

- fonction globales (modes de jeu) :

```
void jouerJoueurContreJoueur();      // Gère entièrement une partie Joueur vs Joueur.

                                     // Boucle d'affichage, saisie des coups et détection de fin de partie.

void jouerContreOrdi();      // Gère une partie Joueur vs IA2.

                             // Le joueur joue 'X' et la machine joue 'O'.

void jouerIAContreIA();      // Fait s'affronter IA1 (aléatoire) contre IA2 (proximité) en mode automatique...
```

III . validation du programme :

- a) Algorithme du programme principal :

Le `main()` est structuré selon un menu simple :

1. Affiche les choix disponibles
2. L'utilisateur choisit un mode
3. Appelle la fonction correspondant au choix
4. Le mode sélectionné gère toute la partie
5. Le programme se termine lorsque la partie finit

b) Tests effectués

Plusieurs tests ont été réalisés pour garantir la stabilité du programme :

✔ *Tests sur les entrées utilisateur*

- saisie hors grille (ex : 5 12)
- saisie sur une case déjà occupée
  → ✓ Vérification : le programme rejette la saisie et redemande une position

✔ *Tests de détection de victoire*

- victoire en ligne
- victoire en colonne
- victoire en diagonale
  → ✓ Les trois cas sont correctement détectés

✔ *Tests des IA*

- IA1 joue toujours dans une case vide
- IA2 privilégie correctement les cases adjacentes
- IA2 joue aléatoire quand elle est bloquée
  → ✓ Les comportements sont cohérents

✔ *Tests de match nul*

→ ✓ Quand la grille est pleine et aucune victoire n'est détectée, le message "Match nul" apparaît.

c) Résultats obtenus

Le projet n'a pas un objectif statistique comme dans Monty Hall, mais nous pouvons tout de même résumer :

✔ *Comportement de l'IA1*

- totalement aléatoire
- parfois gagne mais sans stratégie

✔ *Comportement de l'IA2*

- plus cohérente : essaie de construire des alignements
- plus performante qu'IA1 mais reste battable

✔ *Mode IA vs IA*

- montre clairement la différence entre une IA aléatoire et une IA basée sur une stratégie locale
- IA2 gagne plus souvent car ses coups sont groupés