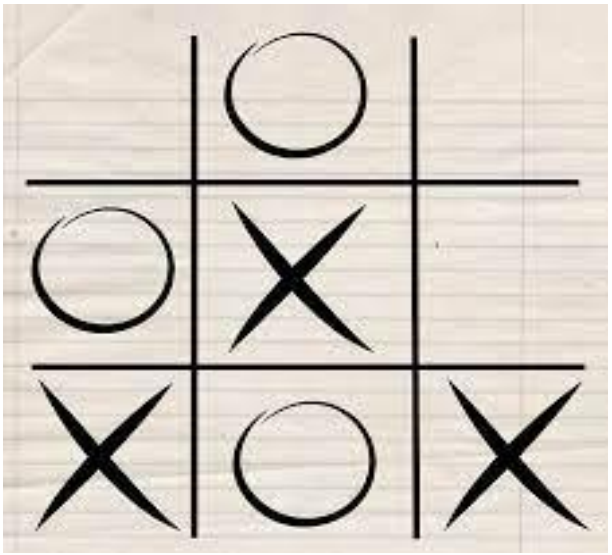
Université de Lille

***Projet* : Jeu de Morpion**



*Réalisé En **Python** par :*

ZEGGANE Yathmas

EL FAKHOURI Yasser

I - <u>Cahier des charges</u> :

Ce jeu se déroule soit entre 2 utilisateurs soit entre un utilisateur et l'IA, et consiste à aligner trois (croix / ronds) sur une matrice composée de 9 cases répartis 3X3 dans des cellules qui forment une grille.

Il existe trois types d'alignements gagnants :

- *alignement vertical.*
- *alignement horizontal.*
- *alignement diagonal.*

L'interface utilisateur du jeu de morpion est développée en utilisant la bibliothèque *Tkinter* de *Python*. Elle est organisée en plusieurs éléments graphiques :

- Une fenêtre principale affichant la page d'accueil avec 3 boutons
  - *<u>Partie rapide</u>* : Une partie entre deux utilisateurs.
  - *<u>Configurer AI</u>* : Une partie entre un utilisateur et l'AI en personnalisant le niveau, le symbole et la couleur de l'AI.
  - <u>Quitter</u> : Quitter le jeu

- La grille de jeu sera composée de neuf cases qui seront initialisées à vide. Lorsqu'un joueur clique sur une case vide, celle-ci sera remplie avec le symbole correspondant au tour du joueur.
- Les symboles utilisés pour le jeu de morpion seront représentés par les caractères 'X' et 'O'.
- Un message s'affiche en haut de la fenêtre pour indiquer le tour du joueur, avec le symbole correspondant.
- Un message de résultat s'affiche en haut de la fenêtre lorsque la partie est terminée, indiquant le gagnant ou la partie nulle.

Lorsque l'utilisateur clique sur "<u>Configurer AI</u>", une nouvelle fenêtre s'affiche avec les options suivantes :

- Niveau de difficulté : Facile, Moyen, Difficile.
- Symbole : 'X' ou 'O'.
- Couleur

L'utilisateur peut choisir son niveau de difficulté, le symbole qu'il souhaite jouer ainsi que la couleur de son symbole.

En résumé, l'interface utilisateur du jeu de morpion sera simple et intuitive, permettant aux utilisateurs de jouer facilement à deux joueurs ou contre une IA avec différents niveaux de difficulté. Le jeu sera également équipé d'options pour permettre aux joueurs de relancer une nouvelle partie ou de quitter le jeu avec pour chacune un bouton.

Le gagnant de la partie est le premier joueur qui parvient à réaliser le premier alignement correct.

II- Plans de tests :

Avant de lancer le jeu, plusieurs tests unitaires et d'intégration doivent être effectués pour s'assurer que tout fonctionne correctement.

Les tests unitaires ont pour but de tester chaque unité de code individuellement, c'est-à-dire chaque fonction ou méthode. Pour cela, on utilise *Doctest* et la méthode *testmod*() pour exécuter automatiquement les tests écrits dans la documentation de chaque fonction.

Les tests d'intégration, quant à eux, permettent de tester le fonctionnement des différentes parties du programme ensemble. On a utilisé la bibliothèque *Pytest* pour écrire et exécuter des tests d'intégration dans un script qui représente un scénario de jeu qui comprend la vérification de la partie nulle, la partie avec un gagnant ainsi que les fonctions utilisées pour réaliser ces fonctionnalités.

Voici un exemple de fonction qui vérifie si la partie nulle :

```
65      def test_of_tied_game(self, gameboard):
66          size = gameboard.get_game().get_sizeOfBoard()
67          gameboard.play(0,0)
68          assert gameboard.get_game().has_Win(gameboard.get_cells()) == False
69          assert gameboard.get_game().tied_game(gameboard.get_cells()) == False
70          gameboard.play(0,1)
71          gameboard.play(0,2)
72          gameboard.play(1,0)
73          gameboard.play(1,1)
74          assert len(gameboard.get_game().possibleMovements(gameboard.get_cells())) == (size * size) - 5
75          gameboard.play(2,2)
76          gameboard.play(2,1)
77          gameboard.play(2,0)
78          gameboard.play(1,2)
79          assert len(gameboard.get_game().possibleMovements(gameboard.get_cells())) == (size * size) - 9
80          assert gameboard.get_game().tied_game(gameboard.get_cells()) == True
81          assert gameboard.get_game().has_Win(gameboard.get_cells()) == False
```

Ces tests comprennent :

- Vérifier que les symboles 'X' et 'O' sont correctement attribués aux joueurs.
- Vérifier que le choix de difficulté de l'AI fonctionne correctement.
- Vérifier que les alignements gagnants sont correctement détectés.
- Vérifier que le jeu peut détecter une partie nulle correctement.
- Vérifier que le message de fin de partie s'affiche correctement.

III- <u>Conception</u> :

Dans cette section, on définit l'architecture du programme, les algorithmes utilisés, ainsi que les différents composants du jeu en incluant des captures d'écran de l'interface utilisateur, des diagrammes de classes. Pour chaque composant, son rôle et son fonctionnement, ainsi que les interactions avec les autres composants seront expliqués.

a- <u>Structures de données</u> :

1- <u>Listes</u> :

possibleMovements : une liste qui représente les mouvements possibles des 2 utilisateurs, le mouvement est possible lorsque la case est vide, elle ne contient ni 'X' ni 'O'.

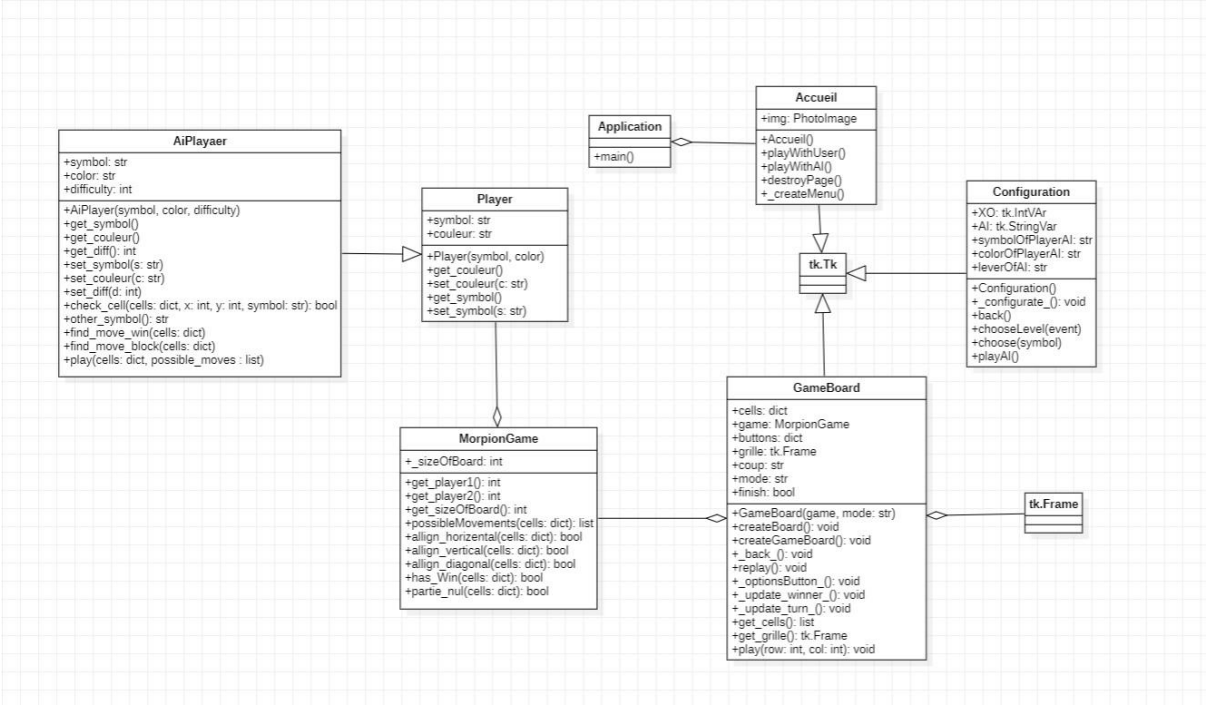
2- <u>Dictionnaire</u> :

Dans la classe *GameBoard*, on retrouve deux dictionnaires:

1- ***cells*** qui a pour clé est les coordonnées de la cellule, et pour valeur valeur le coup joué : *X* ou *O*.

2- ***Buttons*** qui a pour clé est les coordonnées du bouton, et pour valeur valeur le bouton lui-même.

Voici le diagramme UML du projet :



1- ***<u>Architecture du programme</u>***: Le programme est écrit en Python. Il utilise la bibliothèque graphique Tkinter pour l'interface utilisateur. Le programme est divisé en plusieurs fonctions:

- La fonction `play` qui gère le déroulement du jeu (un clic de l'utilisateur sur un bouton).
- La fonction `possibleMovements` qui vérifie si le mouvement est possible en consultant les cellules et vérifiant si elle est vide ou pas.
- `allign_horizental` qui vérifie si un alignement horizental gagnant a été réalisé, de même pour `allign_vertical et allign_diagonal.`
- La fonction `tied_game` qui vérifie si la partie est nulle.
- La fonction `has_Win` qui vérifie si la partie est nulle.

- La fonction `playAI` fait jouer l'AI.

2- *Algorithmes* :

3- Manuel de l'interface :

Dans cette section, vous pouvez décrire les différentes fonctionnalités de l'interface utilisateur, ainsi que les instructions pour les utiliser. Vous pouvez également inclure des captures d'écran pour illustrer les différentes étapes. Pour chaque fonctionnalité, vous pouvez expliquer comment elle fonctionne, les paramètres éventuels à configurer, ainsi que les messages d'erreur qui peuvent être affichés en cas de problème.

1. **Bouton "Partie rapide"**
    - Cette fonctionnalité permet de jouer rapidement à une partie de morpion contre un autre utilisateur.
    - Pour jouer une partie rapide, cliquez sur le bouton "Partie rapide".
    - Vous serez redirigé vers la page de jeu, où vous pourrez jouer en utilisant les boutons de la grille.
    - Vous pouvez également cliquer sur le bouton "Rejouer" pour jouer une nouvelle partie rapide.
    - Si un problème survient lors de l'exécution de cette fonctionnalité, un message d'erreur s'affiche pour vous informer de la nature du problème.
2. **Bouton "Configurer AI"**
    - Cette fonctionnalité permet de configurer les paramètres de l'ordinateur contre lequel vous jouez.
    - Pour configurer l'AI, cliquez sur le bouton "Configurer AI".
    - Vous serez redirigé vers la page de configuration de l'AI, où vous pourrez ajuster les paramètres tels que la difficulté de l'AI et son symbole.
    - Une fois que vous avez terminé de configurer l'AI, vous pouvez cliquer sur le bouton "play" pour enregistrer vos modifications et ensuite jouer.
3. **Bouton "Quitter"**

- Cette fonctionnalité permet de quitter le jeu de morpion.
- Pour quitter le jeu, cliquez sur le bouton "Quitter".

IV- Problèmes rencontrés :

Les problèmes éventuels qui pourraient être rencontrés lors de la mise en œuvre de ce projet sont :

- Difficultés pour implémenter correctement l'algorithme Minimax pour l'IA : Minimax est un algorithme couramment utilisé pour développer des intelligences artificielles pour les jeux à deux joueurs avec des règles simples, comme le jeu de morpion. Les principales difficultés peuvent être liées à la compréhension de la théorie derrière l'algorithme, à la mise en place de la logique pour évaluer les mouvements possibles et à l'optimisation des performances de l'IA.

- Difficultés pour détecter le gagnant ou la partie nulle : Bien que la détection du gagnant ou de la partie nulle soit relativement simple pour le jeu de morpion, elle peut poser des problèmes si elle n'est pas implémentée correctement. Les erreurs les plus courantes incluent des erreurs de vérification des alignements gagnants, des erreurs de gestion des entrées utilisateur, ou des erreurs de logique pour déterminer si la partie est nulle.

VI- Points à améliorer :

Pour améliorer le jeu de morpion, voici quelques suggestions :

- Ajouter un mode de jeu en ligne pour permettre à des joueurs de différents endroits de jouer ensemble : cela peut être un ajout

intéressant pour augmenter l'attrait de votre jeu de morpion. Pour implémenter cette fonctionnalité, vous aurez besoin d'utiliser des bibliothèques ou des outils spécifiques pour les communications en réseau et pour les interfaces utilisateur.

- Ajouter des options de personnalisation, telles que des thèmes graphiques différents ou la possibilité de choisir la taille de la grille de jeu, puisqu'elle est par défaut de taille 3X3 : ces fonctionnalités permettent aux joueurs de personnaliser leur expérience de jeu en fonction de leurs préférences. L'ajout de nouveaux thèmes graphiques peut rendre le jeu plus attractif visuellement, tandis que la possibilité de choisir la taille de la grille de jeu peut augmenter la difficulté et la variété de jeu. Pour implémenter ces fonctionnalités, vous devrez ajouter des options de configuration dans l'interface utilisateur et mettre à jour les algorithmes en conséquence.