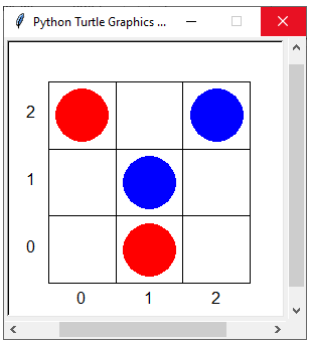
1NSI	Langages et programmation Projet zéro	Programmation d'un jeu de morpion en python



I - Présentation du travail

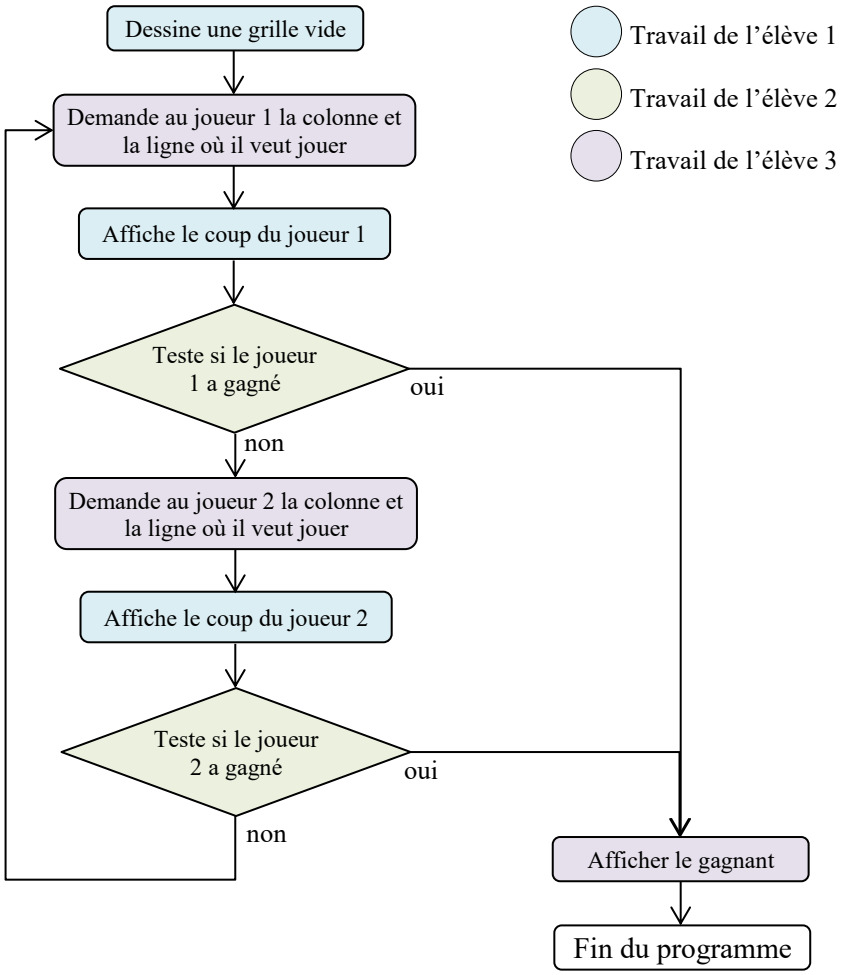
L'objectif de ce travail est de réaliser un jeu de morpion en python.

On réalisera une version où les deux joueurs sont humains et jouent tour à tour.

La tâche va être divisée en trois partie distinctes, chaque élève ayant la charge d'une seule partie :

- L'élève 1 est responsable de la partie graphique.
- L'élève 2 est responsable de la fonction qui détermine si un joueur a gagné.
- L'élève 3 est responsable de l'interaction avec l'utilisateur et de la gestion de la partie. Son travail s'appuie sur celui des deux premiers élèves.

Principe du jeu :



II - Représentation mémoire de la grille :

On utilisera un tableau de `str` (chaîne de caractère) à deux dimensions pour représenter la grille de jeu.
Chaque case du tableau grille contient un tableau qui représente une ligne de la grille en partant par la ligne la plus basse.

Les tableaux représentant les lignes sont des tableaux de 3 éléments de type `str` qui représentent l'état des cases de chaque colonne de la ligne.

- Une case vide correspond au caractère `' '` (espace)
- Une case occupée par un pion du joueur 1 correspond au caractère `'X'`
- Une case occupée par un pion du joueur 2 correspond au caractère `'O'`

Exemples :

`grille = [['X', 'O', ' '], [' ', 'X', ' '], ['O', ' ', 'X']]`
correspond à la représentation :

O		X
	X	
X	O	

`grille = [[' ', 'X', 'X'], ['O', ' ', ' '], ['O', ' ', 'O']]`
correspond à la représentation :

O		O
O		
	X	X

III - Précisions

Cette partie donne des précisions sur le travail attendu pour chaque élève. A chaque fois on donnera plusieurs étapes qui doivent être accomplies les unes après les autres (ne pas vous lancer directement sur la dernière version !). Chaque version devra être rendue dans un fichier séparé ayant comme nom « Elève *x_Nom de l'élève*_v*n* .py» (ex : « Elève 2_MARTIN_v2.py »).

Important : Il faut bien respecter à la lettre les spécifications de votre code (nom exact des fonctions et procédures et ordre des paramètres) pour que le programme complet fonctionne quand on assemblera les différentes parties.

1) Elève 1

a. Version 1

Utiliser le module turtle pour les graphiques.

- ⇨ Programmer une procédure `trace_grille()` qui ne prend pas d'argument. Elle efface l'écran et trace une grille vide de 9 cases. Cette procédure sera appelée par le programme de l'élève 3.
- ⇨ Programmer une procédure `trace_pion(ligne, colonne, type_pion)` qui prend en argument la ligne, la colonne et le type d'un pion et le représente au bon endroit sur la grille. Cette procédure sera appelée par le programme de l'élève 3. Le type du pion peut être `'X'` ou `'O'`. A vous de choisir comment vous voulez représenter chaque pion (forme, couleur …).

  Aide : Il y a quelques calculs à faire pour déterminer l'endroit de la fenêtre où représenter le pion. Pour le dessin du pion, on peut utiliser la fonction `dot` du module turtle.

- ⇨ Ecrire une procédure `partie_gagnee(type_pion)` qui prend en argument le type du pion ayant gagné et ne fait rien (pour l'instant) à part clôturer la partie turtle du programme avec la fonction `done()` (donc la seule instruction de cette procédure est `turtle.done()` pour l'instant).

- ⇨ De même, écrire une procédure `partie_nulle()` qui ne fait rien pour l'instant à part appeler la procédure `done()` de turtle.

Testez vos procédures en utilisant un programme minimal qui les appelle.

b. Version 2

⇨ Améliorer la procédure `trace_grille()` pour qu'elle affiche en bout de ligne et de colonne les numéros de ligne et de colonne pour aider le joueur à savoir où il doit jouer (voir la procédure `write` du module turtle).

⇨ Améliorer les fonctions `partie_gagnee` et `partie_nulle` pour qu'elles affichent dans la console (`print`) un message indiquant le gain de la partie ou la partie nulle.

c. Version 3

⇨ Améliorer la procédure `partie_gagnee(type_pion)` pour qu'elle affiche qui a gagné avec des effets graphiques agréables.

d. Version 4

⇨ Améliorer la procédure `partie_nulle()` pour qu'elle affiche un effet graphique agréable différent de celui d'une partie gagnée.

2) Elève 2

a. Version 1

⇨ Programmer une fonction `verifie_si_gagne(grille)` qui prend en argument la grille de jeu et qui renvoie `True` si un joueur (quel qu'il soit) possède 3 pions alignés horizontalement et `False` dans le cas contraire.

b. Version 2

⇨ Améliorer la fonction `verifie_si_gagne(grille)` pour qu'elle teste aussi les pions alignés verticalement dans la grille.

c. Version 3

⇨ Améliorer la fonction `verifie_si_gagne(grille)` pour qu'elle teste aussi les pions alignés en diagonale dans la grille.

Tester la fonction en utilisant un programme minimal qui définit une grille puis appelle la fonction et affiche son résultat.

d. Version 4

⇨ Programmer une fonction `coup_ordinateur(grille, type_pion)` qui prend en argument la grille de jeu et le type de pion qui doit jouer (`'X'` ou `'O'`) qui renvoie un tuple `(ligne, colonne)` correspondant à la case où l'ordinateur va jouer (exemple : `(1,0)` si l'ordinateur doit jouer dans la case de la deuxième ligne (numéro 1) et la première colonne).

Pour ce programme qui se veut assez simple, on se contentera de choisir une case au hasard parmi les cases libres. Il y a plusieurs façons de faire cela.
Si vous êtes suffisamment à l'aise vous pouvez envisager de ne pas choisir la case au hasard, mais en fonction de critères qui doivent amener l'ordinateur à gagner.

3) Elève 3

a. Version 1

⇨ Ecrire un programme qui réalise la fonction décrite dans l'organigramme de la page 1 en s'appuyant sur les fonctions et procédures des autres élèves (voir dans leurs parties respectives pour connaître les spécifications exactes (nom, arguments, retour) de chaque fonction/procédure).

Le travail des élèves 1 et 2 n'étant pas encore réellement disponible, on utilisera les fonctions et procédures du fichier « Suppletif_eleve3.py ». Pour que celles-ci fonctionnent correctement il faut donc faire « `from Suppletif_eleve3 import *` » comme première ligne du programme (cet import sera remplacé ensuite par l'import des fichiers des autres élèves).

b. Version 2

⇨ Améliorer le programme pour qu'il s'arrête une fois que la grille est remplie (on a joué 9 coups) si aucun joueur n'a gagné avant. Dans le cas où la grille est remplie avant qu'un joueur ne gagne, le programme appelle la procédure `partie_nulle()` de l'élève 1.

c. Version 3

⇨ Améliorer le programme pour qu'il vérifie qu'un joueur ne joue pas dans une case déjà occupée. Si c'est le cas il redemande jusqu'à ce que le coup soit valide.
Aide : Ecrire une fonction `demande_coup(grille)` qui demande la ligne et la colonne, vérifie qu'il n'y a rien et sinon redemande. Cette fonction renvoie un tuple (voir cours précédent avec les exercices) `(ligne, colonne)` une fois que le coup est valide.

d. Version 4

⇨ Améliorer le programme pour qu'il puisse gérer le jeu contre l'ordinateur. Au début du programme, on demande le nombre de joueurs humain qui peut valoir 0, 1 ou 2. A chaque fois qu'un joueur doit donner la position de son coup, on appelle soit la fonction `demande_coup(grille)` programmée précédemment si le joueur est humain, soit la fonction `coup_ordinateur(grille, type_pion)` programmée par l'élève 2 si le joueur est joué par l'ordinateur. Dans le cas où on a zéro joueurs (l'ordinateur joue contre lui-même), on pourra rajouter une petite pause entre chaque coup (avec la fonction `sleep` du module `time`) pour qu'un humain puisse suivre la partie.

IV – Critères d'évaluation

Les règles d'écriture sont les mêmes que pour les TP : Pas d'utilisation de fonctions non vues en classe sauf exception (à demander au professeur), commentaires pour chaque fonction et à l'intérieur du code des fonctions.
Il sera tenu compte de la qualité du code (éviter les répétitions en faisant des boucles par exemple ou découper les tâches longues en plusieurs sous-fonctions), du niveau d'avancement (v1, v2, v3 ou v4) et bien sûr du fonctionnement du programme et de l'absence de bogues.