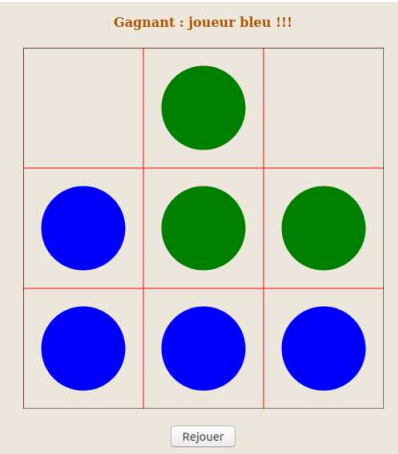
1 Présentation du jeu

Le jeu du Tic Tac se joue à 2 joueurs. Chacun son tour, un joueur pose un pion sur grille 3x3. Si un joueur arrive à aligner 3 pions (colonne, ligne ou diagonale) gagne la partie.
Évidemment ce jeu très simple et très connu conduit à un match nul si les 2 joueurs jouent correctement.



Le but de ce projet est de créer en mode console (au moins dans un premier temps) une version de ce jeu.

2 Analyse du problème

Avant de se lancer dans la programmation de quoi que ce soit, une étape d'analyse préliminaire est indispensable !
Il faut savoir comment seront modélisés les différents éléments intervenant dans le projet : comment est identifié un joueur ? quelle structure de données peut modéliser la grille ? que signifie jouer un pion ? etc.

Avant de lire la suite, essayez de répondre vous-même à toutes ces questions (et d'autres...).

Bien souvent, plusieurs solutions permettent de répondre (plus ou moins efficacement) à ces problématiques. Pour garder une avancée commune dans le projet au sein de la classe, je vous imposerai quelques façons de faire.

La structure principale de ce jeu est certainement le plateau de jeu qui contient les pions. Cette grille sera modélisée par une liste de listes. Chaque "sous-liste" est une ligne de la grille et il y a autant de sous-listes que de lignes dans la grille (3 dans la version de base du jeu). Cette grille est donc un tableau doublement indexé : on accède par exemple à la 3ème case de la 2ème ligne par l'instruction `grille[1][2]`.

Les éléments de ces listes représenteront une case, soit remplie par le pion d'un joueur, soit une case libre.
En mode console, il faut faire abstraction de l'idée qu'on peut se faire d'une grille avec des pions qui sont des formes géométriques, souvent agrémentées de couleurs (cf image donnée en introduction), et trouver une représentation simple de ces données : on propose par exemple

que le chiffre 0 représente une case vide, et les chiffres -1 ou 1 les pions des 2 joueurs.
Avec cette modélisation, la grille présentée plus haut correspond en Python à la liste suivante :

```
[[0, -1, 0], [1, -1, -1], [1, 1, 1]]
```

Au passage, cette structure nous conduit à identifier chaque joueur simplement par un entier : -1 ou 1.

Et jouer un pion correspond simplement au fait d'affecter le nombre -1 ou 1 "au bon endroit" dans la grille (tableau doublement indexé). Si le joueur bleu (identifié par le chiffre 1) joue dans la 2ème case de la 1ère ligne, on écrira : `grille[0][1] = 1`.

Au cours d'une partie, après chaque coup joué, il faudra analyser l'état de la grille pour savoir si le joueur qui vient de poser un pion a gagné : càd tester l'alignement éventuel de 3 pions de sa couleur.

Il faut aussi penser à limiter le nombre de coups à 9 (synonyme de plateau de jeu complet).

On peut proposer l'algorithme suivant pour la gestion complète d'un partie :

Initialiser une grille vierge
Initialiser le 1er joueur
Initialiser le nombre de coups joués
tant que *La grille n'est pas pleine ou qu'un joueur ne gagne pas* **faire**
  Jouer un pion // interaction humaine et maj de la grille
  Incrémenter le nombre de coups joués
  si *Le joueur n'a pas gagné* **alors**
    Changer de joueur
  fin si
fin tq
retourner *Joueur vainqueur (ou match nul)*

3 Spécification des fonctions du jeu

1. Création de la grille :
   Écrire la fonction `creer_grille` :
   Entrées :
   - dim = dimension de la grille carrée
   - vide = caractère symbolisant une case vide
   Sortie : Tableau doublement indexé rempli de caractères vide

   Exemple :
   `creer_grille(3, 0)` vaut `[[0, 0, 0], [0, 0, 0], [0, 0, 0]]`

2. Jouer un pion :
   Écrire la fonction `placer_pion` :
   Entrées :

- grille = tableau doublement indexé représentant la grille
- l = entier représentant le numéro de la ligne
- c = entier représentant le numéro de la colonne
- joueur = identifiant représentant le joueur en cours

Sortie : la grille

Exemple :
`placer_pion([[0, 0, 0], [0, 0, 0], [0, 0, 0]], 1, 2, -1)` vaut
`[[0, 0, 0], [0, 0, -1], [0, 0, 0]]`

3. Tester si un joueur gagne : fonction `gagnant`
   Sur le même modèle que précédemment :
   (a) Écrire la spécification de cette fonction.
   (b) Donner 3 exemples de sorties possibles.
   (c) Écrire cette fonction.

4. Affichage de la grille :
   Pour faciliter le jeu avec des être humains, il faut un affichage « agréable » de l'état de la grille.
   Écrire la fonction `affiche_grille` :
   Entrée :
   - grille = tableau doublement indexé représentant la grille.

   Sortie : affichage (print) sur la sortie standard.

   Exemple :
   `affiche_grille([[0, -1, 0], [1, -1, -1], [1, 1, 1]]` écrit :
   ```
    0      -1      0
    1      -1      -1
    1      1       1
   ```
   Conseils :
   - La fonction `print` peut prendre un paramètre optionnel `end` qui est le caractère à afficher après la chaîne de caractères. Par défaut ce paramètre vaut `'\n'` qui est le caractère de retour à la ligne.
   - Pour écrire du texte centré, ou aligné à gauche ou à droite, en réservant un certain nombres de caractères au texte, on cherchera la documentation à propos du type string, et notamment les méthodes `center`, `rjust` ou `ljust`.
   `https://docs.python.org/fr/3/library/stdtypes.html#text-sequence-type-str`

5. Interface homme machine IHM (saisie des données) :
   Pour savoir où le joueur humain veut placer son pion, on utilisera la fonction `input`.
   `https://docs.python.org/fr/3/library/functions.html#input`
   Cette fonction prend en entrée une chaîne de caractères qui s'affiche sur la sortie standard et renvoie la chaîne de caractères saisie au clavier.
   Remarque : penser à convertir cette chaîne en type numérique lorsque nécessaire (source d'erreur très classique...) !

Pour un jeu de qualité, on prendra soin de tester la validité de la saisie utilisateur avant de poursuivre l'exécution du programme.

(a) Donner les spécifications d'une fonction `saisie` qui doit « interroger » l'utilisateur jusqu'à ce qu'il saisisse les coordonnées d'une case valide pour placer son pion dans la grille.

(b) Écrire cette fonction.

4 Mettre en place le projet

Avec toutes les pistes décrites jusqu'ici, vous avez tout ce qu'il faut pour développer le projet jusqu'à son terme.
Vous pouvez maintenant vous mettre devant un ordinateur et coder les fonctions nécessaires, puis assembler le tout pour pouvoir jouer !

5 Compléments

Un projet n'est jamais fini !
Quelques idées pour poursuivre ce projet :

(a) Faire jouer l'ordinateur.

(b) Faire jouer l'ordinateur avec une stratégie.

(c) Enregistrer dans un fichier les résultats de plusieurs parties entre deux joueurs.

(d) Sauvegarder une partie en cours, et pouvoir la reprendre plus tard.

(e) Développer une interface graphique pour le jeu (ex : bibliothèque par défaut tkinter)

(f) Créer le même projet dans un autre langage.

(g) Faire une version du jeu avec une interface Web : HTML / CSS / JavaScript.

(h) etc.

Un projet n'est jamais fini ... Mais il faut savoir l'arrêter !
Les évolutions d'un projet doivent être calibrées pour tenir dans un délai imparti. Au bout d'un moment il faut savoir figer l'état de son projet et passer à autre chose.