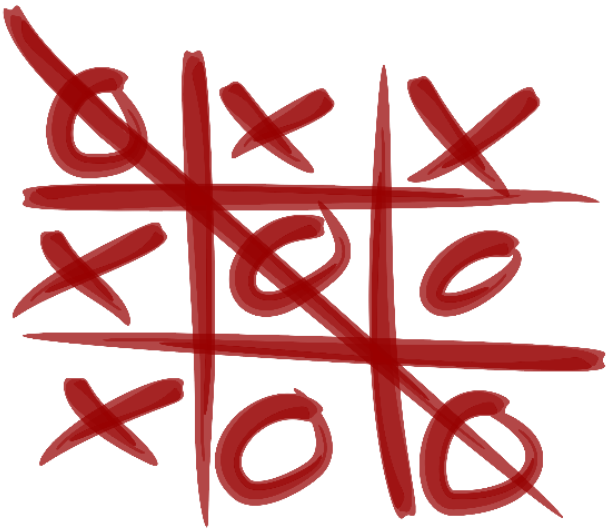
Tic-Tac-Toe

sous Python 3

Dossier rédigé par :
Valentin WITON

Programmé en coopération par :
Jules RICHARD & Valentin WITON



Sommaire :

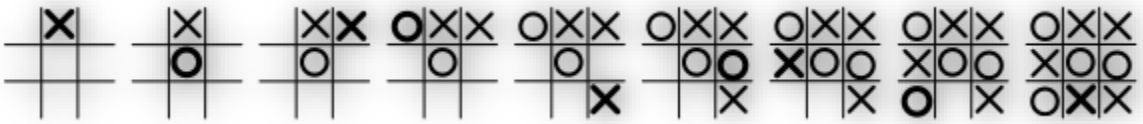
I. Présentation

Le Tic-tac-toe est un jeu basé sur la réflexion, se pratiquant à 2 joueurs qui jouent chacun leur tour et dont le but est de créer un alignement de symbole (généralement croix ou rond) que ce soit à la verticale, à l'horizontale, ou en diagonale. Le jeu se joue sur une grille de 9 cases (3 lignes, 3 colonnes).

Ainsi, il ne faut pas le confondre avec le jeu Morpion, qui lui ressemble par son déroulement, mais dont le but est de former des lignes de 5 symboles et non 3 dans le cas du Tic-tac-toe.

Une partie gagnée par le joueur X. (Image wikipedia.org)

Le nombre de combinaisons de jeu étant limité, l'analyse du jeu est facile à réaliser (pour un être humain). Si les deux joueurs ne se trompent pas dans leurs choix, la partie doit se terminer à tous les coups par un match nul.



Un exemple de match nul. (Image wikipedia.org)

Pour un ordinateur, la tâche n'est pas aussi facile, car, pour jouer parfaitement, l'ordinateur doit connaître les 765 possibilités de gagner et les quelques 26 830 jeux possibles jusqu'à ce que les 9 cases soient remplies.

Dans l'histoire de l'informatique, le Tic-tac-toe a été le premier jeu, doté d'une interface graphique, à fonctionner sur un ordinateur. C'est le développeur A.S. Douglas de l'Université de Cambridge (Royaume-Uni) qui réalisa l'exploit sur l'EDSAC (Electronic Delay Storage Automatic Calculator) en 1952. Plutôt qu'un ordinateur, l'EDSAC, mis en service en 1949, était un immense calculateur électronique qui employait la technique des tubes à vide et pouvait effectuer 15 000 opérations mathématiques par minute, dont 4 000 multiplications.

II. Analyse et déroulement du projet

Lorsque la question du choix du projet s'est posé, mon coéquipier et moi avions pensé au premier abord à réaliser un jeu de Bataille Navale en réseau et plus spécialement à un jeu utilisant une grille.

Quelques jours passèrent et voyant que la programmation de la Bataille Navale allait être compliquée au premier abord, nous avons décidé de nous rediriger vers un jeu plus simple, mais avec une grille pour voir comment fonctionne ce type de jeu et comment cela pourrait se traduire en langage de programmation, notamment en Python, langage sur lequel nous avons travaillé tout au long de l'année. Après avoir pensés à un « Puissance 4 », nous nous sommes rabattus sur le Tic-tac-toe, en gardant en tête la Bataille Navale pour plus tard.

Pour créer le jeu, nous avons d'abord essayés différentes méthodes sans interface graphique qui se sont soldées par un échec. Grâce à l'aide précieuse de notre professeur, nous avons établi le jeu par un tableau constitué de 3 colonnes et de 3 lignes, soit 9 cases remplient de « 0 » à l'initialisation du jeu et changeant selon la case joueur par le joueur. Nous jouions alors en inscrivant le numéro de la case

à jouer dans une fenêtre et ce pour chacun des deux joueurs en alternance. Puis, nous avons ajouté une fonction « Gagnant » qui permet de vérifier à chaque tour si l'un des deux joueurs a gagné en vérifiant le tableau de valeurs : d'abord les lignes, puis les colonnes, puis les diagonales.

Lorsque cette étape fut terminée, nous nous sommes intéressés au module Tkinter, que nous avons appris à maîtriser, du moins dans l'utilisation que nous voulions, ce qui nous a permis de réfléchir à la disposition graphique du jeu.

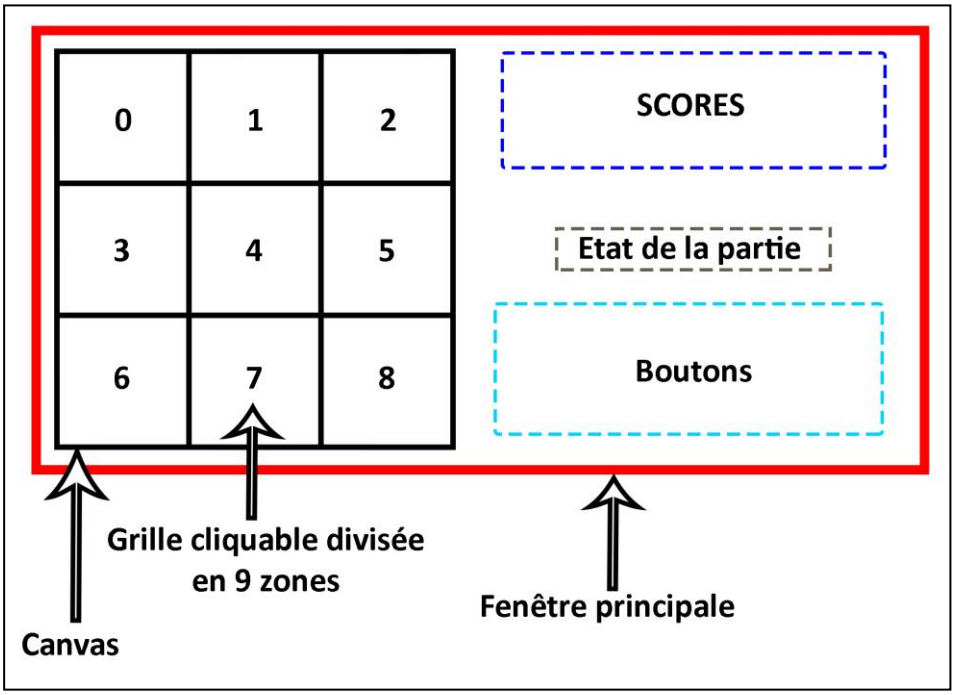


Illustration : Agencement du Tic-tac-toe (Valentin WITON)

L'interface ne faisant pas tout, il nous a fallu appliquer le travail précédemment effectué en parallèle afin que le canvas prenne en compte les paramètres du tableau. Ainsi, nous avons dû créer bon nombre de fonctions et variables, qui nous ont pourtant pris peu de temps. Avec le temps qu'il nous restait, nous étions fin prêt pour passer à l'utilisation du jeu en réseau. Mais n'ayant que peu de connaissances sur ce sujet, nous nous sommes redirigés vers la création d'une sorte de mini Intelligence Artificielle (IA), contre laquelle nous jouons. C'est sur ce principe qu'est basé le programme que nous vous présentons.

III. Problèmes rencontrés

Le premier obstacle a été celui de la façon à obtenir les coordonnées du tableau en fonction de la case choisie. Pour ce faire, nous avions une fonction de près de 20 lignes, et pas forcément très efficace qui était chargée de parcourir le tableau. Il nous fallait donc trouver un autre moyen : celui de la numérotation des cases, auxquelles on fait correspondre des coordonnées (cf : Illustration page 3). Ce numéro obtenu, il suffit de procéder à une division de celui-ci par 3. Le quotient et le reste permettent de retrouver les coordonnées, respectivement la position horizontale et la position verticale de la case.

Ensuite, en ce qui concerne la recherche d'un gagnant ou d'une égalité, nous avons dû ajouter les variables « G » et « sommetab ». Celles-ci nous permettent désormais de vérifier s'il y a un gagnant ou une égalité à chaque instant de la partie. Nous expliquerons leur fonctionnement un peu plus tard.

Après cela, nous sommes passés à l'affichage des scores qui nous a posé plus de problèmes. En effet, il a fallu introduire une nouvelle variable score, qui est en fait un tableau dans lequel les scores des 2 joueurs et les égalités sont comptabilisées. Ce faisant, il a fallu jouer des spécificités de Tkinter, afin de changer le texte au bon moment pour un affichage correct. De plus, nous ne comprenions pas pourquoi lorsque qu'un des deux joueurs gagnait, si l'on cliquait de nouveau dans le canvas, le score du joueur augmentait à chaque clic. Chose que nous avons pu régler en changeant la valeur de la variable « G », correspondant au joueur gagnant.

Autre problème : celui de recommencer une partie et de réinitialiser les scores. Il nous a donc fallu créer deux fonctions, qui se rejoignent plus ou moins l'une et l'autre. La première permet de recommencer la partie en gardant les scores obtenus et est liée au bouton « Recommencer ». L'autre reprend la précédente et se charge également de réinitialiser les scores, ce qui équivaut à relancer le programme depuis le début, mais nous voulions un moyen simple de le faire, sans avoir à quitter le programme justement.

Enfin, entrer le nom du joueur nous a pris un peu de temps supplémentaire. Néanmoins, nous y tenions afin de donner une dimension plus personnelle et plus conviviale à notre programme. Une variable se charge donc de stocker une chaîne de caractères que l'on demande au joueur au démarrage du programme via une fenêtre « Input ». Cette variable est ensuite intégrée au « Label » des scores dans Tkinter pour l'affichage.

IV. Principaux éléments de programmation

4.1. Utilisation du tableau

Comme évoqué précédemment, l'utilisation d'un tableau nous permet de stocker et de modifier des valeurs pour chaque case jouée. C'est la base de notre programme car sans lui, aucun calcul ne serait possible. Il est modifié principalement par 2 fonctions : position_clic() et jeu_ordi() toutes deux liées afin de gérer le jeu du joueur et celui de l'ordinateur.

En ce qui concerne l'utilisateur, la gestion du tableau se fait à travers la variable « event » qui se modifie en fonction de la position du clic dans le canvas. Par la suite, la fonction permet de créer une croix bleue qui témoigne du jeu du joueur et on change la valeur de la case dans le tableau pour indiquer qu'elle est jouée par le joueur 1.

```
def position_clic(event):
    if 0<event.x<100 :#premiere colonne
        if 0<event.y<100 :#premiere ligne
            a=0 #donne le numéro de la case cliquée
            x = 50 #Centre de la croix en X
            y = 50 #Centre de la croix en Y
        if 100<event.y<200 :#deuxieme ligne
            a=3
            x = 50#Centre du cercle ou de la croix en X
            y = 150#Centre du cercle ou de la croix en Y
            ...
    i=a//3 # Calcul des coordonnées pour le tableau
    j=a%3
    if tableau[i][j]==0 and G==0 and J==1: # Condition de jeu
        can1.create_line(x+35, y+35, x-35, y-35, width = 7, fill = "blue") # Création d'une croix bleu (Joueur 1)
        can1.create_line(x-35, y+35, x+35, y-35, width = 7, fill = "blue")
        tableau[i][j]=1 # On fait correspondre le nombre 1 à la case cliquée pour définir que le joueur 1 l'a joué.
```

En ce qui concerne l'ordinateur, nous lui avons attribué une stratégie. Celle-ci consiste à attaquer d'abord, c'est-à-dire à compléter la case manquante pour terminer un alignement dès qu'il le peut et sinon à contrer l'utilisateur pour éviter qu'il ne gagne. Cette stratégie est gérée par la fonction « IA_Analyse », appelée dans la fonction « jeu_ordi » renvoie les coordonnées i et j de la case à jouer en fonction de l'état du jeu. Ensuite, un rond est créé, les variables x2 et y2 étant là pour définir le centre de ce cercle, tout comme x et y le sont pour la croix du joueur 1 dans la fonction « position_clic ».

4.2. La fonction Gagnant

Comme son nom l'indique, la fonction gagnant() permet de vérifier s'il y a un gagnant à la fin de chaque tour de jeu. Pour cela elle est appelée depuis la fonction position_clic() et renvoie une valeur dans la variable G : 0 si pas de gagnant ; 1 si le joueur 1 est gagnant ; 2 si l'ordinateur est gagnant.

```
def gagnant(tab):
    Gagnant=0 # Initialisation de variable
    for i in range(3): # Verification des lignes
        if tab[i][0]==tab[i][1]==tab[i][2] and tab[i][0]!=0:
            Gagnant=tab[i][0]
    for j in range(3): # Verification des colonnes
        if tab[0][j]==tab[1][j]==tab[2][j] and tab[0][j]!=0:
            Gagnant=tab[0][j]
    if tab[0][0]==tab[1][1]==tab[2][2] and tab[2][2]!=0: # Verification des deux diagonales
        Gagnant=tab[1][1]
    elif tab[0][2]==tab[1][1]==tab[2][0] and tab[2][0]!=0:
        Gagnant=tab[1][1]
    return Gagnant
```

4.3. La fonction somme_tableau

Après avoir réussi à créer notre fonction nous permettant de trouver un gagnant lorsqu'il y en a un, nous nous sommes intéressés à la détection d'une égalité. Pour cela nous effectuons la somme des éléments du tableau. Si le joueur 1 commence et que les 9 cases sont remplis, alors celle-ci sera égale à 13 (1*5+2*4), si c'est l'ordinateur qui commence alors elle sera égale à 14 (2*5+1*4). Ainsi si la somme du tableau est égale à 13 ou à 14, cela veut dire qu'il y a une égalité, puisque cela implique qu'il n'y est pas de gagnant (en effet la fonction gagnant l'aurait renvoyée).

```
def somme_tableau():
    sommetab=0
    for i in range (3):
        for j in range(3):
            sommetab+=tableau[i][j]
    return sommetab
```

4.4. La fonction texte_gagnant

La fonction « texte_gagnant » se charge quant à elle d'afficher dans la fenêtre Tkinter le score des joueurs, mais aussi l'état de la partie (ex : joueur x a gagné ; Egalité). Pour cela elle se sert de la valeur de G afin de déterminer les valeurs à renvoyer. Par ailleurs, la fonction se charge également de changer cette valeur de G pour éviter que le score d'un joueur ne défile à chaque clic supplémentaire et pour alterner les joueurs lors de la partie suivante. (Si le joueur 1 gagne la première partie, c'est

l'ordinateur qui commencera la deuxième et vice versa). Et s'il y a égalité, le joueur qui commence la partie suivante est choisi au hasard. (cf 4.5.)

```
def texte_gagnant():
    …
    …
    if G==1 : # Si le joueur 1 gagne
        texte.configure(text="{} a gagné(e)".format(joueur1),font="times 10")
        score[0]+=1
        G=11
    elif G==2 :# Si le joueur 2 gagne
        texte.configure(text="L'Ordinateur a gagné",font="times 10")
        score[1]+=1
        G=22
    elif G==0 and sommetab==13 or sommetab==14:
        texte.configure(text="Egalité")
        score[2]+=1
        if score[2]>=2:
            egalite='Egalités'
        G=3
    label2["text"]="{} : {}".format(joueur1,score[0]) # Mise a jour du score des joueurs et égalitée(s)
    label3["text"]="Ordinateur : {}".format(score[1])
    label4["text"]="{} : {}".format(egalite,score[2])
    return sommetab # Renvoie la valeur de la somme du tableau pour determiner l'égalité dans la fonction position_clic.
```

4.5. Les fonctions effacer et reinitialiser

Elles servent respectivement à recommencer une partie et à recommencer une partie mais avec remise à zéro des scores pour la seconde. Ainsi, la fonction reinitialiser n'est qu'un complément de la seconde. Par ailleurs, le joueur est alterné. En effet, c'est le perdant qui commence la nouvelle partie. En cas d'égalité le joueur qui joue le premier est choisi aléatoirement. Pour éviter les tricheries et les utilisateurs abusant du bouton « Recommencer » jusqu'à ce qu'ils commencent en premier, nous avons rajouté une condition qui permet de repasser au joueur 2, ainsi la triche devient impossible.

```
def effacer():
    can1.delete(ALL) # Effacement du canvas
    global G # variable correspondant au joueur gagnant
    global J
    if J==1 :
        J=2
    if G==11:
        J=2
    elif G==22: # Du coup G==11 si le joueur 1 est gagnant et G==22 si le joueur 2 est gagnant. Ces 2 cas servent à laisser le perdant commencer la partie suivante
        J=1
    elif G==3: # Sert en cas d'égalité. Le joueur qui commence la nouvelle partie est alors choisi aléatoirement
        J=randrange(1,3)
    G=0 # Redéfinition de G à 0, correspondant à pas de gagnant
    for i in range(3):
        for j in range(3):
            tableau[i][j]=0 # Réinitialisation du tableau
    sommetab=0 # Réinitialisation de la somme du tableau
    grille() # Re-création de la grille
    texte.configure(text="") # Remise à Zéro du texte concernant l'état de la partie
    if J==2: # On exécutera la fonction j2_commence si J a été défini à 2, afin de faire jouer l'ordinateur.
        j2_commence()
```

```
def reinitialiser():
    effacer()
    score[0]=0
    score[1]=0
    score[2]=0
    label2["text"]="{} : {}".format(joueur1,0)
    label3["text"]="Ordinateur : {}".format(0)
    label4["text"]="{} : {}".format('Egalité',score[2])
```

V. Jeux de Tests

Avant de lister les différents tests, il est important de noter que lister tous les cas est difficile puisque le Tic-tac-toe comprend 765 manières différentes de gagner et 26 830 possibilités de jeux. Néanmoins, nous avons essayés de tester un grand nombre de cas afin de vérifier que l'ensemble des fonctions et des variables renvoient les valeurs espérées.

- **Le Joueur 1 gagne la partie**

A noter que l'ordinateur contrant bon nombre de nos actions de jeux, il est très difficile de vérifier l'ensemble des cas pour le joueur 1. Ainsi, nous n'avons pas pu tester qu'un ou deux cas par forme de victoire. Cependant, nous avons effectués de nombreux tests avant d'intégrer l'IA qui se sont tous soldés par un succès. Que ce soit en ligne, en colonne, en diagonale, séparé ou non par une case, on peut affirmer à 95% que si le joueur 1 gagne, il sera bien détecté comme gagnant par le programme.

- **L'ordinateur gagne la partie**

En ce qui concerne l'ordinateur, il est plus facile de vérifier les cas puisqu'il suffit de le laisser gagner à la manière où nous le voulons. On peut affirmer avec un pourcentage plus élevé (>98%) que si l'ordinateur gagne, il sera bien détecté comme gagnant par le programme.

- **La partie s'achève par une égalité**

Là-aussi, il nous a été facile d'effectuer un grand nombre de tests, puisque c'est le résultat le plus courant dans une partie et nous n'avons recensé aucun problème.

VI. Développements possibles

- Bien évidemment, nous aurions aimé proposer un Tic-tac-toe en réseau. Cependant par manque de temps, nous n'avons pas pu le faire. Cela reste néanmoins un développement tout à fait envisageable et intéressant. De plus, nous avons déjà le programme sans IA avec alternance des joueurs. Il serait donc aisé de poursuivre dans cette voie.

- Egalement, il serait possible de faire varier l'ordre de la stratégie de défense de l'ordinateur pour rendre encore plus difficile le jeu.

VII. Sources

- Wikipedia.org
- Site ISN du lycée Fénelon : http://isn.fenelon.free.fr/
- Site de Fabrice Sincère – Cours sur Python 3 : http://fsincere.free.fr/isn/python/cours_python.php
- Effbot.org – Tkinter : http://effbot.org/tkinterbook/