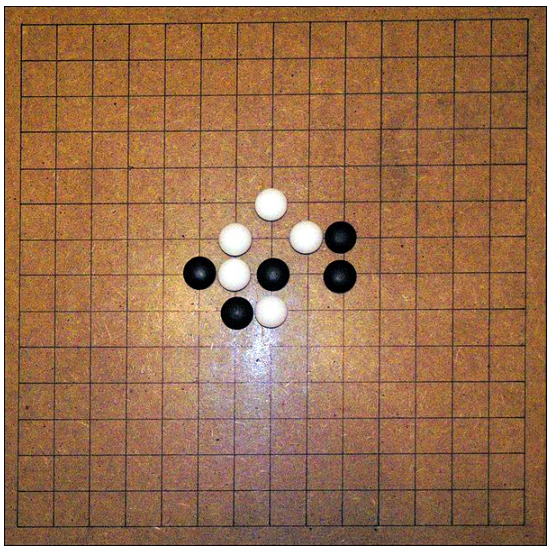
Introduction à la Science Informatique
Jules Collenne

TP Bonus : Développement du jeu Gomoku

Ceci est un TP bonus pour approfondir vos connaissances. Il n'est pas obligatoire et est à faire uniquement si vous désirez approfondir vos connaissances en programmation. Il est à faire à la maison, il n'y a pas de séance consacrée à ce TP.

1. Le jeu Gomoku (五目並べ)

Le Gomoku est un jeu de société traditionnel d'Asie qui se joue sur un plateau de 15x15. Deux joueurs s'affrontent en plaçant alternativement leurs pions, des jetons noirs et blancs, sur le plateau. Le but du jeu est d'aligner cinq pions consécutifs, que ce soit horizontalement, verticalement ou diagonalement.



Exemple de plateau de Gomoku.
Il ressemble au populaire jeu de Go, mais les règles diffèrent.

2. Plan de travail

Pour créer ce jeu, nous allons utiliser plusieurs outils :

- Le langage Python.

- Un environnement de développement intégré (IDE) comme **Spyder**[1], **Visual Studio Code**[2] ou encore **PyCharm (Community Edition)**[3]**.**
- La librairie d'interface graphique **Tkinter** (déjà incluse dans Python).

3. (Petite) introduction à la programmation orientée objet (POO)

Un petit TP n'est pas suffisant pour comprendre tous les détails de la POO. Nous présentons donc ici un petit aperçu pour vous permettre de créer le jeu. En cours, nous avons vu plusieurs structures (qui sont en fait des « **classes** ») : listes, files, piles, matrices… et si je vous disais que l'on pouvait créer absolument tout type de classe que l'on veut ?

Mais d'abord, une classe, qu'est-ce que c'est exactement ? Concrètement, c'est un bout de code qui représente quelque chose et qui contient des variables et des fonctions qui lui sont propres… mais un exemple vaut mieux que mille mots. Voyons cela dans notre cas du Gomoku.

Lors d'une partie de Gomoku entre deux humains, les deux joueurs possèdent des variables et des fonctions quasi identiques : ils ont la fonction « **poser un pion** », les variables « **couleur de pion** » et « **nom du joueur** ». Ces variables n'ont pas les mêmes valeurs d'un joueur à un autre, mais elles représentent la même chose : Enzo et Baptiste ont tous les deux un nom qui leur est propre, mais la valeur n'est pas la même. On dit que Enzo et Baptiste sont deux « **objets** » créés à partir de la classe « **Joueur** ». La classe est un peu comme un schéma de construction, et les objets sont les éléments que l'on crée à partir de ce schéma, avec leurs propres valeurs (on appelle parfois les objets des « **instances** », c'est un synonyme).

Dans notre code, cela peut se traduire de la sorte :

```
class Joueur: # On crée une classe que l'on appelle Joueur
    def __init__(self, nom, couleur, num): # Tout objet a une fonction __init__ qui créé un objet et permet d'initialiser les valeurs de certaines de ses variables. Dans notre cas, on veut que les joueurs aient un nom et une couleur.
        self.nom = nom # On a donné en entrée à __init__ la variable nom. Maintenant, pour la mettre dans notre objet, on va faire cette ligne. Self en anglais signifie soi-même, et quand on l'utilise en python avec self.nom_de_variable, c'est pour dire que l'on fait référence à la variable qui appartient à la classe.
        self.couleur = couleur
        self.num = num

    def joue(self, plateau):
```

1 https://www.spyder-ide.org/
2 https://code.visualstudio.com/
3 https://www.jetbrains.com/pycharm/

```
    ...
    # Ici on codera la fonction pour poser le pion du joueur étant donné un plateau donné en entrée.
    ...
```

Le code d'une classe est indenté et ses fonctions ont un argument « self ». Dans le code d'une classe, quand on veut utiliser ses variables ou fonctions, on utilise self. Par exemple, si l'on veut récupérer la variable « nom » dans le code d'une classe, il faut écrire « self.nom ».

Ensuite, en bas du code, on pourra créer des joueurs comme cela (attention, ce ne sera pas dans la classe, donc il ne faut pas mettre d'indentations) :

```
joueur1 = Joueur("Joueur 1", "black", 1)
joueur2 = Joueur("Joueur 2", "white", 2)
```

joueur1 et joueur2 sont deux instances d'un même objet, avec des valeurs différentes.

Et on pourra même utiliser leurs variables et fonctions de la sorte :

```
print(joueur1.nom)
# joueur1.joue(plateau)
```

Ici, on appelle joueur1, et on lui demande de nous donner sa variable nom. On pourra aussi lancer sa fonction « joue » avec « joueur1.joue(plateau) » quand on aura un plateau.

Vous pouvez déjà copier-coller ces codes dans votre IDE, on s'occupera de finir la classe plus tard.

4. Créer la classe du jeu

On a des Joueurs, maintenant il nous faut un jeu ! Faisons cela de suite. Nous allons créer une classe qui représentera le jeu, gérant le plateau, la logique de jeu et les interactions graphiques. Notre classe aura un plateau de jeu, qui contiendra 0 pour case vide, 1 si la case contient un pion du joueur 1 et 2 si elle contient un pion du joueur 2.

Exercice 1.
Écrivez la classe « **GomokuApp** » en vous inspirant de la classe Joueur mais avec seulement la fonction __init__. Donnez en entrée à son __init_ deux variables p1 et p2, qui seront les deux joueurs. Faites en sorte que GomokuApp ait également un plateau de jeu, composé d'une matrice 15*15 remplie de zéros. Créez également la variable self.current_player qui contient le premier joueur.

5. Utiliser tkinter

Pour utiliser l'interface graphique de Tkinter, il faut dire à Python qu'on a besoin de cette librairie, écrivez tout en haut de votre code :

```
import tkinter as tk
```

Ensuite, allez tout en bas de votre code et créez une fenêtre avec Tkinter en utilisant ces lignes :

```
root = tk.Tk()
root.mainloop()
```

Ici, **root** est une variable qui contient la fenêtre.

Vous pouvez lancer ce code, et vous verrez une jolie petite fenêtre toute vide avec pour titre « tk » ! Bon, ce n'est qu'un début, changeons donc le titre de la fenêtre, et ajoutons un canvas (c'est une zone dans laquelle on pourra interagir et afficher des choses) :

```
root.title("Gomoku")
canvas = tk.Canvas(root, width=600, height=600)
canvas.pack()
```

Constatez que l'on créé un **canvas** de taille 600x600 pixels, rattaché à la fenêtre **root**. De plus, chaque élément de Tkinter contient une fonction « **pack()** » qui permet de l'afficher. Attention : il faut mettre ces lignes avant root.mainloop() ! (d'ailleurs, root.mainloop() sera la dernière ligne du code même une fois fini).

Enfin, le canvas créé sera utilisé par GomokuApp, qui s'en servira pour faire toute la logique graphique.

Exercice 2.
Faites en sorte que la classe GomokuApp ait une variable « canvas » initialisée par un canvas donné en entrée à __init__. Ensuite, entre canvas.pack() et root.mainloop(), créez deux instances de Joueurs et une instance de GomokuApp. Faites en sorte que votre GomokuApp récupère les deux joueurs créés et le canvas.

6. Dessiner avec Tkinter

Pour l'instant, notre fenêtre est un peu triste, ajoutons-y le plateau de jeu en dessinant quelques lignes. Dans la classe GomokuApp, on va créer une nouvelle fonction « **draw_grid** » pour dessiner la grille. Pour dire à Python que la fonction appartient à la classe GomokuApp, il faut simplement l'écrire dedans (avec une indentation) et lui donner « **self** » en entrée, comme dans l'exemple plus haut.

Pour dessiner une ligne dans un canvas en tkinter, il faut faire :

```
canvas.create_line(x1, y1, x2, y2)
```

Avec x1, y1 les coordonnées du début de la ligne et x2, y2 celles de la fin de la ligne.
Attention : en informatique, le point (0,0) se situe en haut à gauche et l'axe y est inversé.

Exercice 3.
Écrivez la fonction « **draw_grid** » dans la classe GomokuApp qui dessine la grille composée de 15 lignes horizontales et 15 lignes verticales. Enfin, à la fin de __init__ de GomokuApp, ajoutez self.draw_grid(). Exécutez le code plusieurs fois jusqu'à ce qu'une grille bien dessinée apparaisse.
Rappel : Notre canvas fait 600x600 pixels, et les lignes parcourent tout le canvas.

7. Poser des pions

Si vous êtes arrivés jusqu'ici, c'est que vous avez réussi à faire une belle grille, bravo ! Maintenant, on va faire en sorte que le joueur puisse poser des pions. Si vous vous rappelez, quand on a créé un canvas, je vous avais dis que ce dernier servait à afficher des choses à l'écran, mais également à interagir avec le programme, c'est exactement ce que nous allons faire. Nous allons dire au programme que self.canvas doit « surveiller » les clics. Si l'utilisateur clique dans le canvas, on va dire au programme d'y déposer un pion de la couleur du joueur. On va commencer par écrire cette ligne à la fin de __init__ de GomokuApp :

```
self.canvas.bind("<Button-1>", self.on_click)
```

Ici, on dit à self.canvas qu'il doit surveiller le « bouton 1 » (le clic gauche de la souris). S'il y a un clic, cela va lancer la fonction self.on_click. Créons donc cette petite fonction dans GomokuApp.

```
def on_click(self, event):
```

Quand un clic est reçu, il va lancer la fonction on_click et lui donner un argument « **event** ». Pour l'instant, dites vous seulement que **event** contient deux variables qui nous intéresse, **event.x** et **event.y**, qui sont les coordonnées du clic dans le canvas.

On va utiliser la fonction **joue** de la classe **Joueur**. Modifiez la pour avoir cette ligne :

```
def joue(self, plateau, event):
```

Dedans, nous allons convertir les coordonnées du clic (sur le canvas, de 0 à 600) en coordonnées du plateau (de 0 à 14).

Exercice 4.
Terminez la fonction **joue** pour qu'elle renvoie les coordonnées **de la case** choisie par le clic. Les coordonnées contenues dans event sont en terme de pixels, nous voulons les convertir en coordonnées représentant nos cases de jeu.
Info : On retourne des -1 dans le cas ou la position choisie est interdite (pion déjà présent).

```
def joue(self, plateau, event):
    x = event.x // ?
```

```
    y = event.y // ?
    if ? <= x < ? and ? <= y < ?:
        if plateau[y][x] == ?:
            return x, y
    return -1, -1
```

Une fois cela fait, on peut utiliser la fonction dans **on_click.** On demande au programme de faire jouer le joueur actuel contenu dans current_player :

```
x, y = self.current_player.joue(self.plateau, event)
```

On vérifie que move n'est pas False :

```
if x != -1 and y != -1:
```

Et on met à jour le plateau :

```
self.affiche(x, y, self.current_player)
```

Mais… on a pas de fonction affiche ? Bien vu ! Écrivons là de suite pour mettre à jour le plateau quand on pose un pion.

Exercice 5.
Corrigez la fonction affiche de **GomokuApp** avec les bonnes valeurs. La fonction **create_oval** dessine un ovale contenu dans le carré (x_hg, y_hg, x_bd, y_bd) avec la couleur donnée à **fill**. x_hg, y_hg designe le point haut-gauche du carré, et x_bd, y_bd le point bas-droite.

```
def affiche(self, x, y, joueur):
    self.plateau[?][?] = ???
    self.canvas.create_oval(x_hg, y_hg, x_bd, y_bd, fill=?)
```

8. Vérifier s'il y a un gagnant

Pour vérifier s'il y a un gagnant, nous pouvons analyser le dernier coup joué et vérifier si ce dernier permet d'aligner 5 pions.

On imagine cette fonction, prenant en paramètre le dernier coup joué (x, y) et le plateau, puis renvoyant True s'il y a un vainqueur, False sinon :

```
def check_winner(x, y, plateau):
    directions = [
        (1, 0),  # Horizontal
        (0, 1),  # Vertical
        (1, 1),  # Diagonal down
        (1, -1)  # Diagonal up
    ]

    for dx, dy in directions:
        count = 1  # Count the current move
        count += count_in_direction(x, y, dx, dy, plateau)
```

```
    count += count_in_direction(x, y, -dx, -dy, plateau)
    if count >= 5:
        return True
return False
```

Pour chaque direction, on compte le nombre de pions alignés. Il nous reste à implémenter **count_in_direction**, qui prend en entrée les coordonnées du pion joué et une direction, et qui renvoie le nombre de pions alignés dans cette direction.

Exercice 6.
Implémentez **count_in_direction** de sorte que **check_winner** fonctionne pour détecter s'il y a un gagnant. Appelez **check_winner** après **self.affiche** dans **on_click** pour vérifier s'il y a un gagnant. Le cas échéant, affichez son nom avec un print() et empêchez tout prochain clic avec **self.canvas.unbind("<Button-1>")**. Faites quelques tests pour vous assurer que tout fonctionne bien. C'est normal si, pour l'instant, il n'y a que le joueur 1 qui joue.
Aide : Vous pouvez faire une boucle while en modifiant au fur à mesure les valeurs de x et y avec dx et dy.

Exercice 7.
Enfin, changez de joueur à chaque fin de tour. À la fin de on_click, vérifiez si c'est le joueur 1 qui joue ; si c'est le cas, changez self.current_player par self.p2, sinon faites l'inverse.

Si tout fonctionne, c'est parfait, bravo à vous ! Vous avez officiellement programmé un petit jeu en python !