Atelier 2 - KINXIP11 - Info1.Projet

SDD/Affichage

Fraude

Le travail doit être réalisé individuellement. Les programmes seront systématiquement soumis à l'outil MOSS pour détecter toute similarité. En cas de programmes identiques ou très similaires, des sanctions seront appliquées, allant au-delà d'une simple note de 0 pour l'atelier. De plus, des comparaisons seront effectuées entre les programmes des ateliers 2, 3, 4 et 4-bonus pour garantir l'originalité du travail et éviter toute copie entre pairs.

Des capsules d'information essentielles pour ce projet sont disponibles sous Moodle. Vous pouvez les consulter via le lien suivant : **Capsules**

1 Objectif

- Programmer la représentation en Python de 3 configurations distinctes (le départ, le milieu, la fin).
- Développer une fonctionnalité pour afficher la configuration à l'écran.
- Mettre en place un mécanisme de saisie de coordonnées.
- Observer scrupuleusement les règles de Clean Code.

Vous devrez soumettre un unique fichier .py sans y inclure votre nom ni la suite du programme. Assurez-vous que le code est rédigé en Python 3 et reste compréhensible pour vos pairs.

2 Détails

Votre programme doit présenter 3 configurations, en utilisant et en testant les fonctions unitaires `est_dans_grille` et `est_au_bon_format` (voir ci-dessous), tout en autorisant la saisie de coordonnées. Accordez une attention particulière à la clarté du code, en veillant à ce qu'il soit facilement compréhensible par vos relecteurs, en appliquant les principes du Clean Code (consultez les **Capsules** pour plus de détails).

2.1 Représentation

2.1.1 Configurations

On désigne par **configuration** l'état du jeu à un moment donné de la partie. Naturellement, la configuration de "début de jeu" est généralement fournie dans les règles.

La configuration ne se limite pas toujours à la grille et à la position des pions, mais peut également inclure des éléments tels que le tour de joueur, le nombre de pions capturés, ou toute autre information jugée pertinente pour programmer le jeu ou pour jouer.

Pour l'ensemble des ateliers, vous devrez donc proposer 3 configurations :

- Le début de la partie (tel que spécifié dans les règles).
- Une configuration "milieu", à un moment arbitraire de la partie. Cette configuration de test permettra au développeur/évaluateur de vérifier les affichages, les saisies et les déplacements...

— La fin de la partie, ou plus précisément, quelques coups avant la fin. Cette configuration de test permettra au développeur/évaluateur de vérifier les différentes fins de partie possibles.

2.1.2 Structure de données

Il est impératif de spécifier la structure de données, y compris les variables et leurs types, nécessaires à la représentation des éléments du jeu.

Afin d'illustrer votre choix de structures de données, vous devriez implémenter une ou plusieurs fonctions permettant d'assigner des valeurs aux variables au début, au milieu et à la fin de la partie (quelques coups avant). Ces configurations (début, milieu et fin de partie) seront utilisées tout au long du projet, facilitant ainsi les tests de votre programme par les relecteurs sans jouer une partie complète.

La structure suivante est un mauvais choix : il FAUT avoir une variable qui représente la grille. Notamment parce que la grille sera un paramètre pour la plupart des fonctions. Il faut s'imaginer avec une grille 10×10 : ferez-vous 10 listes ? Passerez-vous 10 paramètres ?

```
#definition de la fonction pour l'affichage des grilles
def afficher_grille(grille, grille_mil, grille_fin):
  print("     1   2   3   4   5")
  print("_ _ _ _ _ _ _ _ _ _ _ _")
  print(" A | "+ grille[0]+" | "+ grille[1] + " | "+ grille[2]+" | "+ grille[3]+" | "+ grille[4]+ " |")
  print(" B | "+ grille[5]+" | "+ grille[6]+ " | "+grille[7]+" | "+ grille[8]+" | "+grille[9]+" |")
  print(" C | "+ grille[10]+" | "+ grille[11]+" | "+ grille[12]+ " | "+grille[13]+" | "+ grille[14]+" |")
  print(" D | "+ grille[15]+" | "+ grille[16]+ " | "+grille[17]+" | "+ grille[18]+" | "+grille[19]+" |")
  print(" E | "+ grille[20]+ " | "+ grille[21]+ " | "+grille[22]+" | "+ grille[23]+" | "+grille[24]+" |")
  print("_ _ _ _ _ _ _ _ _ _ _ _")

  print("     1   2   3   4   5")
  print(" A | "+ grille_mil[0]+" | "+ grille_mil[1] + " | "+ grille_mil[2]+" | "+ grille_mil[3]+" | "+ grille_mil[4]+ " |")
  print(" B | "+ grille_mil[5]+" | "+ grille_mil[6]+ " | "+ grille_mil[7]+" | "+ grille_mil[8]+" | "+grille_mil[9]+" |")
  print(" C | "+ grille_mil[10]+" | "+ grille_mil[11]+" | "+ grille_mil[12]+ " | "+ grille_mil[13]+" | "+ grille_mil[14]+" |")
  print(" D | "+ grille_mil[15]+" | "+grille_mil[16]+ " | "+ grille_mil[17]+" | "+ grille_mil[18]+" | "+ grille_mil[19]+" | ")
  print(" E | "+ grille_mil[20]+ " | "+ grille_mil[21]+ " | "+ grille_mil[22]+" | "+ grille_mil[23]+" | "+ grille_mil[24]+" |")
  print("_ _ _ _ _ _ _ _ _ _ _ _")

  print("     1   2   3   4   5")
  print(" A | "+ grille_fin[0]+" | "+ grille_fin[1] + " | "+ grille_fin[2]+" | "+ grille_fin[3]+" | "+ grille_fin[4]+ " |")
  print(" B | "+ grille_fin[5]+" | "+ grille_fin[6]+ " | "+ grille_fin[7]+" | "+ grille_fin[8]+" | "+ grille_fin[9]+" |")
  print(" C | "+ grille_fin[10]+" | "+ grille_fin[11]+" | "+ grille_fin[12]+ " | "+ grille_fin[13]+" | "+ grille_fin[14]+" |")
  print(" D | "+ grille_fin[15]+" | "+ grille_fin[16]+ " | "+ grille_fin[17]+" | "+ grille_fin[18]+" | "+ grille_fin[19]+" |")
  print(" E | "+ grille_fin[20]+ " | "+ grille_fin[21]+ " | "+ grille_fin[22]+" | "+ grille_fin[23]+" | "+ grille_fin[24]+" |")
  print("_ _ _ _ _ _ _ _ _ _ _ _")
```

La structure suivante est un mauvais choix : il ne faut pas mélanger la représentation des données et la représentation graphique. Pour la suite du projet, ce sera infernal d'aller chercher les valeurs des cases et de les déplacer.

```python
def grille_debut_partie():
    grille = []
    l = 2  # l=ligne
    c = 5  # c=colonne
    a = []
    for j in range(c):
        k = "="
        a.append(k)
    grille.append(a)

    for i in range(l):
        a = []
        b="|"
        a.append(b)
        for j in range(c):
            k = " "
            a.append(k)
        a.append(b)
        grille.append(a)

    for i in range(l):
        a = []
        b = "|"
        a.append(b)
        for j in range(c):
            k = "○"
            a.append(k)
        grille.append(a)
    l = 5
    for i in range(l):
        for j in range(c):
            print(grille[i][j], end=" ")
        print()
```

2.1.3 Affichage de l'état de la partie

Vous devez définir une ou plusieurs fonctions qui affichent la configuration : la grille et les informations pour jouer (joueur ou "couleur" dont c'est le tour, lignes, colonnes,...).

C'est à vous de choisir un affichage explicite pour l'utilisateur tout en n'étant pas trop chargé. On vous propose ici 1 deux exemples convenables.

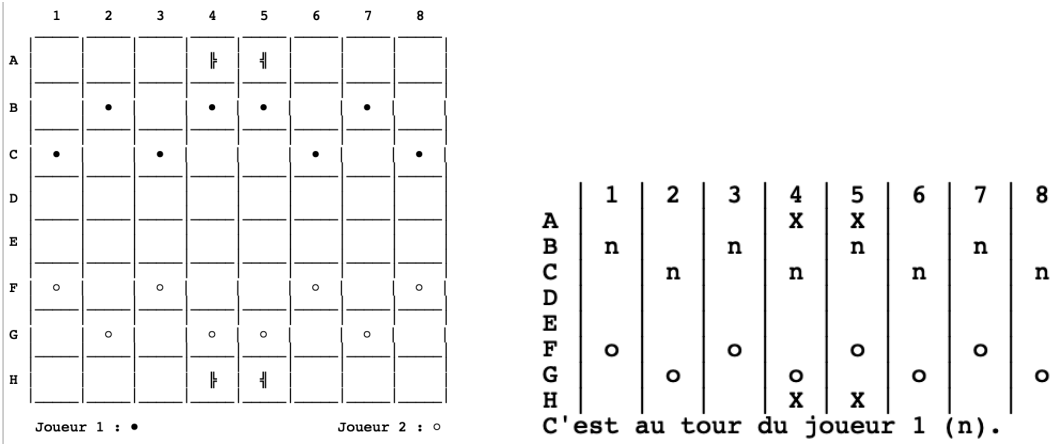

FIGURE 1 – Exemples d'affichage

Le problème de ce troisième affichage ici 2 est qu'il sous-entend que vous avez mélangé les représentations python et graphique. Certains informations sont "fonctionnelles" (la grille) d'autres sont "graphiques" (les A" et "1", les "bords").

```
[' ', '1', '2', '3', '4']
['A', ' ', 'X', ' ', '0']
['B', 'X', '0', ' ', 'X']
['C', 'X', 'X', ' ', ' ']
['D', '0', '0', '0', 'X']
```

FIGURE 2 – Exemples d'affichage : A ne pas faire

Quelques conseils/indications :

— au moins la grille doit être en paramètre
— il est conseillé de décomposer : afficher une grille de taille $n \times n$, c'est afficher n lignes. Afficher une ligne, c'est afficher n cases...
— vous NE devez PAS utiliser de bibliothèque graphique particulière (tkinter ou pygame par exemple).
— pour éviter les erreurs d'encodage (lorsqu'on travail sur des systèmes d'exploitations différents ou qu'on utilise des caractères spéciaux) : ajoutez `# -*- coding: utf-8 -*-` en début de programme. Même commentée cette ligne est "lue" (interprétée) par python qui tolère ainsi les accents et autres caractères encodés en utf-8.

La décomposition peut vous être très utile ici :

— écrire une fonction qui affiche une ligne
— écrire une fonction qui affiche la première ligne
— écrire une fonction qui affiche toutes les lignes

Remarque : il n'y a qu'une seule fonction `afficher_grille`.

2.2 Fonctions unitaires imposées

2.2.1 `est_au_bon_format`

Vous devez définir une fonction `est_au_bon_format(message)` et éventuellement ses fonctions auxiliaires. Cette fonction devra vérifier que le message répondu par le joueur est au bon format. Là où le programmeur attend "A1", que le joueur n'ait pas répondu "11" ou "AB" par exemple.

2.2.2 `est_dans_grille`

Vous devez définir une fonction `est_dans_grille(ligne, colonne, grille)` et éventuellement ses fonctions auxiliaires. Cette fonction devra vérifier que les coordonnées (ligne,colonne) appartiennent à la grille.

Cette fonction servira lors de la saisie, pour vérifier que le joueur n'a pas saisi "n'importe quoi". Par exemple "X22" n'est pas valide dans une grille 4×4.

Selon vos choix d'implémentation, ligne et colonne peuvent être des caractères ou des entiers, il est probablement plus simple qu'il s'agisse d'entiers.

2.2.3 fonctions de tests

Pour chacune de ces fonctions `est_au_bon_format(...)` et `est_dans_grille(...)` vous écrirez une fonction de test `test_est_au_bon_format()` (resp. `test_est_dans_grille()`) qui contiendra les jeux de tests (assert) selon le principe du Test Driven Development (voir Tests unitaires sous **Capsules**).

Pour rappel, ces fonctions sont à écrire AVANT les fonctions qu'elles testent. Voici un exemple de fonction de test pour la fonction qui ferait la sommes des éléments positifs d'une liste :

```
1 def test_somme_positifs():
2     assert somme_positifs([])==0, "liste vide, somme nulle"
3     assert somme_positifs([-1,-2,-3])==0, "liste uniquement negatifs, somme nulle"
4     assert somme_positifs([1,2,-3,1,12])==16, "liste mixte, 1+2+1+12"
```

Code 1 – "Exemple fonction de test"

Ces fonctions de tests seront intégrés dans votre fichier principal (un seul fichier à rendre).

2.3 Saisie de coordonnées

Vous devez définir une ou plusieurs fonctions qui permettent de saisir des coordonnées jusqu'à ce qu'elles correspondent bien au format attendu et à une position **dans** la grille (en utilisant ses fonctions précédentes), sans vous préoccuper de savoir si cette position peut être jouée en respect des règles du jeu (atelier suivant). Cette fonction doit renvoyer les coordonnées vérifiées vis à vis de l'appartenance dans la grille.

Quelques conseils :

— Convertir un entier en lettre et une lettre en entier
Les lettres ont un code "ASCII". Par exemple, la lettre "A" a le code 65, "B" 66 et "a" 97...
Pour passer de l'un à l'autre et permettre des calculs on peut utiliser les fonctions python suivantes : `ord("a")` vaut 97 et `chr(97)` vaut "a".

— Une chaîne de caractères est une liste
On peut facilement manipuler les caractères d'une chaîne de caractère. Par exemple pour `chaine = "A1"` on aura `chaine[0]` qui vaut `"A"` et `chaine[1]` qui vaut `"1"`

3 Evaluation

Pour savoir comme faire l'évaluation, on vous demande fortement de regarder le video "Evaluation par les pairs" sous Capsules. **Lien :** Évaluation par pairs.

Voici une liste non exhaustive des aspects sur lesquels vous serez évalués :

- Respect des consignes (env 1/20) :
  - Anonymat,
  - Un seul fichier, .py
  - Uniquement la partie demandée
  - Compilation : le code ne doit pas comporter d'erreurs (en cas d'erreur l'évaluateur pourra légitimement mettre 0)
- Fonctionnalités :

  L'évaluation portera sur l'existence, la correction et la pertinence du code demandé
  - Structure de données (env 5/20)
  - Affichage (env 3/20)
  - `est_au_bon_format`, `est_dans_grille` et les fonctions de tests associées (env 5/20)
  - Saisie (env. 3/20)
- Clean code (env 3/20) :
  - le nom des variables est explicite et respecte les conventions
  - le programme est composé de fonctions courtes et explicites
  - les fonctions s'appliquent à des paramètres (et pas de variables globales)
  - le programme est commenté pour améliorer la lisibilité et compréhension
  - le programme inclut des tests significatifs pour chaque fonction
  - le programme est essentiellement construit à l'aide de fonctions, il y a un code principal (de moins de 15 lignes)

4 Squelette proposé

A titre indicatif, voici un squelette minimaliste de ce à quoi pourrait ressembler votre programme. Il est souvent plus facile de faire à votre envie mais pour faciliter l'évaluation, le correcteur ne doit pas avoir à corriger votre code pour l'exécuter.

Prévoyez donc un programme principal qui réalise :

— l'appel de l'affichage pour les 3 configurations,

— l'appel des fonctions de test pour les fonctions `est_au_bon_format` et `est_dans_grille`,

— et l'appel d'une saisie.

```
# -*- coding: utf-8 -*- ## Pour s'assurer de la compatiblite entre correcteurs

#### REPRESENTATION DES DONNEES
###initialisation des grilles et autres variables de jeu
...

#### REPRESENTATION GRAPHIQUE
def afficher_grille(grille...) :
    ...

#### SAISIE
###fonctions de verification
#jeux de tests
def test_est_dans_grille():
    assert ...

#verification dans grille
def est_dans_grille(...) :
...

###fonctions de saisie
def saisir_coordonnees(...) :
    ...

#### CODE PRINCIPAL
# execution affichage sur les 3 grilles et autres variables de jeux
afficher_grille(grille...)
afficher_grille(grille...)
afficher_grille(grille...)

#execution test est_dans_grille
test_est_dans_grille() #uniquement pour la mise au point, a conserver pour le correcteur

#affichage des coordonnees saisies
print(saisir_coordonnees( ... ))
```

5 Astuces / conseils complémentaires Python

Ces astuces sont proposées mais ne sont pas forcément nécessaires.

— Afficher un message lors de la saisie : `reponse = input(message)`
Cette ligne est équivalente aux deux suivantes :

```
print(message)
reponse = input()
```

— Afficher un message sans retour à la ligne : `print(message,end="")`
Le paramètre end est un mot clé en python qui précise quel caractère de fin la fonction `print` doit utiliser en fin du message. Par défaut, en l'absence de précision sur ce paramètre, python ajoute un saut de ligne.

Rappels :

— pas d'objet
— pas de récursivité
— pas de try/except
— pas de bibliothèque non standard