Polynésie – 2025 – sujet1 - Correction

Exercice 2 (6 points)

---

1. Baguenaudier à 4 cases, les trois dernières remplies

État initial :
`[False, True, True, True]`

La première case remplie est la case d'indice **1**.

Règles :

- On peut toujours jouer la case 0.
- Si le jeu n'est ni vide ni rempli, on peut jouer la case qui suit la première case remplie.

Donc :

- Case 0 jouable
- Case 2 jouable (1 + 1)

États après les coups possibles :

- Jouer case 0 → `[True, True, True, True]`
- Jouer case 2 → `[False, True, False, True]`

👉*Commentaire: Il est essentiel d'identifier correctement la première case remplie avant d'appliquer la règle.*

---

2. Fonction `initialiser`

```
def initialiser(n):
    return [False] * n
```

👉 *Commentaire: Solution simple et efficace grâce à la multiplication de liste.*

---

3. Fonction `victoire`

Compléter lignes 3, 4 et 5 :

```
def victoire(tab):
    for etat_case in tab:
        if etat_case == False:
            return False
    return True
```

👉 *Commentaire:*

- *On s'arrête dès qu'on trouve une case vide (optimisation naturelle).*
- *On aurait aussi pu écrire* `if not etat_case:`.

---

4. Fonction `indice_premiere_case_occupee`

```
def indice_premiere_case_occupee(tab):
    for i in range(len(tab)):
        if tab[i]:
            return i
    return None
```

👉 *Commentaire: On retourne* `None` *si aucune case n'est remplie.*

---

5. Fonction `coup_valide`

```
def coup_valide(tab, case):
    if not isinstance(case, int):
        return False
    if case < 0 or case >= len(tab):
        return False

    if case == 0:
        return True

    indice = indice_premiere_case_occupee(tab)

    if indice is None:
        return case == 0

    if all(tab):
        return case == 0

    return case == indice + 1
```

👉 *Commentaire:*

- On vérifie d'abord la validité de l'indice (bonne pratique).
- Attention aux cas particuliers : jeu vide ou jeu rempli.

---

6. Fonction `changer_case`

```
def changer_case(tab, case):
    if coup_valide(tab, case):
        tab[case] = not tab[case]
    return tab
```

👉 *Commentaire:*

- `not` permet d'inverser simplement True / False.
*- Ici la modification se fait **en place**.*

---

7. Compléter la ligne 6 de `vider`

```
print("Vider case", n)
```

---

8. Affichage produit par `vider(3)`

En supposant :

- `remplir(1)` affiche `Remplir case 1`
- `remplir(0)` ne fait rien

Déroulement :

```
Vider case 1
Vider case 3
Remplir case 1
Vider case 2
Vider case 1
```

👉 *Commentaire*: *Il faut suivre soigneusement l'arbre d'appels récursifs.*

---

9. Fonction récursive `remplir`

Par symétrie avec `vider` :

```
def remplir(n):
    if n == 1:
        print("Remplir case 1")
    elif n > 1:
        remplir(n-1)
        vider(n-2)
        print("Remplir case", n)
        remplir(n-2)
```

👉*Commentaire:*

- Structure miroir de `vider`.
- C'est une récursion croisée (remplir appelle vider et inversement).

---

10. Baguenaudier de 2000 cases

Ce code **n'est pas adapté**.

- La profondeur de récursion peut dépasser la limite Python (~1000 appels).
- Complexité exponentielle.
- Risque de `RecursionError`.

👉*Commentaire:*

- Pour 2000 cases, il faudrait une approche itérative ou augmenter la limite de récursion (ce qui reste dangereux).
- Le nombre d'opérations devient gigantesque.
