Nouvelle Calédonie – 2025 – sujet2 - Correction

Exercice 3 (8 points)

Partie A – États et lancers

1. Lancers possibles

Lancers possibles depuis
`e2 = [1, 1, 0, 1, 0, 0]`

- e2[0] = 1 → on peut lancer.
- Emplacements libres (valeur 0) :
  - indice 2
  - indice 4
  - indice 5

Lancers possibles : **2, 4, 5**

---

2. État 3

Lancers possibles depuis `e3 = [0, 1, 1, 0, 1]`

- e3[0] = 0 → aucune balle en main.

Donc seul lancer autorisé : **0**

Lancer possible : **0**

---

3. Fonction `lancer_possible`

Code complété :

```
def lancer_possible(etat, lancer):

    if lancer >= len(etat) or lancer < 0:
        return False

    if lancer == 0 and etat[0] == 1:
        return False

    if lancer > 0:
        if etat[0] == 0 or etat[lancer] == 1:
            return False

    return True
```

👉 *Commentaire:*

On teste :

- borne valide
- cas spécial lancer 0
- présence balle en main
- emplacement libre

---

4. Lancer de 5

Depuis `e2 = [1,1,0,1,0,0]`, lancer 5.

Étape 1 : placement de la balle

On met 1 en position 5 : `[1,1,0,1,0,1]`

Étape 2 : effet de gravité

Décalage à gauche : `[1,0,1,0,1,0]`

État final : **[1, 0, 1, 0, 1, 0]**

---

5. fonction `lancer_balle`

```
def lancer_balle(etat, lancer):

    nouvel_etat = [balle for balle in etat]

    if lancer > 0:
        nouvel_etat[lancer] = 1

    # effet gravité
    nouvel_etat = nouvel_etat[1:] + [0]

    return nouvel_etat
```

🔷 Partie B – Séquences récursives

6. Fonction `liste_lancers_possibles`

```
def liste_lancers_possibles(etat):
    liste = []
    for i in range(len(etat)):
        if lancer_possible(etat, i):
            liste.append(i)
    return liste
```

---

7. Pourquoi la fonction `calcule_sequences` est récursive ?

Elle s'appelle elle-même pour calculer les séquences de longueur n-1.

---

8. Pourquoi termine-t-elle ?

À chaque appel :

- n diminue de 1
- on atteint le cas de base n == 0

Donc terminaison garantie.

---

9. Compléter les lignes manquantes

Code complet:

```
def calcule_sequences(etat, n):

    if n == 0:
        return [[]]

    s_possibles = []

    lancers = liste_lancers_possibles(etat)

    for lancer in lancers:

        nouvel_etat = lancer_balle(etat, lancer)

        suites = calcule_sequences(nouvel_etat, n-1)

        for suite in suites:
            s_possibles.append([lancer] + suite)

    return s_possibles
```

Partie C – Graphe / Automate

10. Représentation de l'automate

Exemple de structure :

```
automate = {
    '11000': [(1,'10100'), (2,'11000'), (3,'10100')],
    '10100': [(1,'11000'), (4,'10010')],
    '10010': [(4,'01100')],
    '01100': [(0,'11000')]
}
```

(Structure indicative conforme au graphe)

---

11. Fonction `lancer_balle_automate`

Code :

```
def lancer_balle_automate(automate, etat, lancer):

    if etat not in automate:
        return ""

    for (l, nouvel_etat) in automate[etat]:
        if l == lancer:
            return nouvel_etat

    return ""
```

👉 Commentaire: On parcourt la liste d'adjacence.

---

12. Fonction `parcours_sequence_depart`

Code :

```
def parcours_sequence_depart(automate, etat_depart, sequence):

    etat_courant = etat_depart

    for lancer in sequence:

        nouvel_etat = lancer_balle_automate(automate, etat_courant, lancer)

        if nouvel_etat == "":
            return None

        etat_courant = nouvel_etat

    return etat_courant
```

👉 Compétences testées :

- parcours séquentiel
- appel fonction auxiliaire
- gestion d'erreur
