Jeu du Mastermind

Fiche d'aide

Cette fiche contient des indications et des stratégies pour vous aider à surmonter les difficultés du projet Mastermind. Elle ne donne pas les réponses complètes, mais des pistes de réflexion et des exemples pour vous débloquer.

① Génération de la combinaison secrète

1.1 Comprendre la contrainte « sans répétition »

Conseil. Vous avez déjà travaillé, dans le devoir maison de préparation, sur le tirage aléatoire sans répétition (méthode manuelle avec randint et del , puis sample). Le principe est identique ici !

Remarque 1. Pour éviter les répétitions, deux approches principales :

1. **Méthode manuelle** : copier la liste des couleurs, tirer une couleur au hasard, la retirer de la copie, recommencer.
2. **Méthode directe** : utiliser random.sample() qui fait tout cela en une ligne.

Si COULEURS = ["J","B","R"] et NB_PIONS = 2 :

```
from random import sample
combinaison = sample(COULEURS, NB_PIONS)
# Resultat possible : ["R", "J"]
```

② Saisie de la combinaison du joueur

2.1 Valider la saisie

Conseil. Beaucoup d'élèves se demandent comment vérifier que chaque couleur saisie est valide. Pensez à l'opérateur in , ou à la fonction trouve_position() du devoir maison !

```
def saisir_combinaison():
    """Demande au joueur de saisir une combinaison."""
    proposition = []
    for i in range(NB_PIONS):
        couleur = input(f"Pion {i+1} : ")
        # Vérifier que couleur est dans COULEURS
        while couleur not in COULEURS:
            print("Couleur invalide !")
            couleur = input(f"Pion {i+1} : ")
        proposition.append(couleur)
    return proposition
```

③ Fonction `evalue_proposition` : le cœur du jeu

C'est LA fonction la plus difficile du projet. Voici comment l'aborder par étapes.

3.1 Comprendre le problème des doubles comptages

Remarque 2. Le piège principal : un pion peut être compté comme « bien placé » OU « mal placé », mais pas les deux ! De plus, si une couleur apparaît plusieurs fois, il faut faire attention à ne pas la compter plusieurs fois.

Exemple 1. Secret : ["J","R","V","O"]
Proposition : ["J","O","R","N"]
Analyse étape par étape :

- Position 0 : « J » est bien placé → marque "#"
- Position 1 : « O » n'est pas à la position 1 dans le secret, mais il est à la position 3 → mal placé → marque "X"
- Position 2 : « R » n'est pas à la position 2, mais il est à la position 1 → mal placé → marque "X"
- Position 3 : « N » n'est pas dans le secret → marque "0"

Résultat : ["#","X","X","0"]

3.2 Stratégie en deux passes

Conseil. **Passe 1 – Compter les bien placés :**

- Parcourir les indices de 0 à NB_PIONS − 1
- Pour chaque indice i , comparer proposition[i] et secret[i]
- Si égalité : marquer « bien placé » et « neutraliser » ces pions pour ne pas les recompter

Passe 2 – Compter les mal placés :

- Pour chaque pion de la proposition non encore traité
- Chercher s'il existe dans le secret (à une autre position) et n'a pas déjà été compté
- Si oui : marquer « mal placé » et neutraliser ce pion du secret

3.3 Comment « neutraliser » un pion déjà compté ?

Remarque 3. Pour éviter de recompter les pions, on peut :

1. Faire des **copies** de la proposition et du secret
2. Remplacer les pions traités par une valeur impossible (exemple : None ou "")

```
secret_copie = secret[:]  # Copie pour modifications
prop_copie = proposition[:]

# Passe 1 : bien placés
reponse = []
for i in range(NB_PIONS):
    if prop_copie[i] == secret_copie[i]:
        reponse.append(SYMBOLE_BIEN_PLACE)
        prop_copie[i] = None   # Neutraliser
        secret_copie[i] = None
    else:
        reponse.append("")  # À compléter en passe 2

```

```
# Passe 2 : mal placés
for i in range(NB_PIONS):
    if reponse[i] == "":  # Pas encore traité
        # Chercher dans secret_copie...
```

3.4 Gérer la recherche d'un pion mal placé

Conseil. Pour vérifier si un pion de la proposition existe ailleurs dans le secret :

1. Parcourir les positions du secret (copie)
2. Si on trouve la couleur ET que ce n'est pas None (déjà utilisé)
3. Alors c'est un mal placé : le marquer et neutraliser cette position du secret

```
# Pour chaque pion non traité de la proposition
if reponse[i] == "":
    couleur = prop_copie[i]
    trouve = False
    # Chercher dans le secret
    for j in range(NB_PIONS):
        if secret_copie[j] == couleur:
            reponse[i] = SYMBOLE_MAL_PLACE
            secret_copie[j] = None  # Neutraliser
            trouve = True
            break  # Important : on s'arrête dès qu'on trouve
    if not trouve:
        reponse[i] = SYMBOLE_ABSENT
```

④ Fonction `manche_de_mastermind`

4.1 Structure de la boucle principale

Conseil. Cette fonction doit :

1. Générer une combinaison secrète
2. Initialiser un compteur d'essais
3. **Boucler** tant que : le joueur n'a pas gagné ET il reste des essais
4. À chaque tour : saisir, évaluer, afficher
5. Renvoyer le nombre d'essais

```
def manche_de_mastermind():
    secret = genere_combinaison_secrete()
    nb_essais = 0
    gagne = False

    while not gagne and nb_essais < NB_MAX_ESSAIS:
        print(f"\nEssai {nb_essais + 1}/{NB_MAX_ESSAIS}")
        proposition = saisir_combinaison()
        nb_essais += 1
```

```
        reponse = evalue_proposition(proposition, secret)
        print(f"Réponse : {combi_to_str(reponse)}")

        # Vérifier si gagné
        if reponse == [SYMBOLE_BIEN_PLACE] * NB_PIONS:
            gagne = True

    # Affichage final
    if gagne:
        print(f"Bravo ! Trouvé en {nb_essais} essais !")
    else:
        print(f"Perdu ! Le secret était : {combi_to_str(secret)}")

    return nb_essais
```

4.2 Comment détecter la victoire ?

Conseil. Le joueur gagne quand **tous** les symboles de la réponse sont des « # ».
On peut comparer la réponse à une liste contenant uniquement des « # ».

```
if reponse == [SYMBOLE_BIEN_PLACE] * NB_PIONS:
    # Le joueur a gagné !
```

5 Conseils méthodologiques

5.1 Tester vos fonctions au fur et à mesure

Remarque 4. Ne codez pas tout d'un coup ! Testez chaque fonction individuellement avec des exemples simples avant de passer à la suivante.

```
# Définir les constantes pour les tests
SYMBOLE_BIEN_PLACE = "#"
SYMBOLE_MAL_PLACE = "X"
SYMBOLE_ABSENT = "O"

# Test 1 : tous bien placés
print(evalue_proposition(["J","R","V","O"], ["J","R","V","O"]))
# Attendu : ["#","#","#","#"]

# Test 2 : aucun correct
print(evalue_proposition(["B","N","J","R"], ["V","O","L","G"]))
# Attendu : ["O","O","O","O"]

# Test 3 : mélange
print(evalue_proposition(["J","O","R","N"], ["J","R","V","O"]))
# Attendu : ["#","X","X","O"]
```

5.2 Utiliser des affichages de débogage

Debug avec `print`. Si votre fonction evalue_proposition ne fonctionne pas :

— Ajoutez des print() pour voir l'état des copies à chaque étape

— Affichez les valeurs des variables de boucle

— Vérifiez que les neutralisations fonctionnent bien

5.3 Commencer simple, puis complexifier

Remarque 5.

1. D'abord, codez avec des constantes simples : 3 couleurs, 3 pions, 5 essais
2. Testez que tout fonctionne dans ce cas simple
3. Puis passez aux valeurs réelles du Mastermind
4. Enfin, testez les cas limites (tous mal placés, aucun correct, etc.)

6 Questions fréquentes

6.1 « Ma fonction renvoie trop de mal placés »

Conseil. Vous oubliez probablement de neutraliser les pions du secret après les avoir comptés. Vérifiez que vous remplacez bien par None dans secret_copie .

6.2 « Mon programme ne s'arrête jamais »

Conseil. Vérifiez :

— Que nb_essais est bien incrémenté à chaque tour

— Que la condition de victoire est correcte

— Que votre boucle while a bien les deux conditions

6.3 « J'ai une erreur 'list index out of range' »

Conseil. Vous essayez probablement d'accéder à un indice qui n'existe pas. Vérifiez :

— Que vos boucles vont bien de 0 à NB_PIONS − 1

— Que vous n'utilisez pas len(liste) comme indice (rappelez-vous : le dernier indice est len(liste) − 1)

7 Pour aller plus loin

Une fois que votre jeu fonctionne, vous pouvez :

— Améliorer l'affichage (couleurs, émojis, formatage)

— Ajouter un système de score sur plusieurs manches

— Créer un menu pour choisir le niveau de difficulté

— Implémenter le mode avec répétitions de couleurs
— Proposer un mode « aide » qui affiche le nombre de solutions possibles restantes

N'hésitez pas à demander de l'aide si vous êtes bloqué après avoir essayé ces pistes !