

Prompts testés et leur résultat associé (réponse et code)

Yan Rabefaniraka, stagiaire de Meriem Ouederni et David Brunet

18 Juin 2025

Contents

Consommation + ou - passive	1
Émulation de la consommation passive	1
ChatGPT	1
Commentaires sur le code obtenu	5
Github Copilot (Claude 3.7 Thinking)	5
Commentaires sur le code obtenu	10
Création de contenu	10
Émulation	10
ChatGPT	10
Commentaires sur le code obtenu	12
Github Copilot	12
Commentaires sur le code obtenu	14
Co-création	14
Atteindre la co-création	14
ChatGPT	14
Commentaires sur le code obtenu	17
Conclusion sur la co-création pour une tâche simple (et avec sujet connu)	17
Tests réalisés avec ChatGPT et Github Copilot.	

Consommation + ou - passive

Prompt passif:

- Code moi le jeu des 13 allumettes en Python. Un joueur peut jouer contre un ordinateur.

Émulation de la consommation passive

Pour émuler une consommation passive de l'IA, je me suis basé sur la rubrique d'évaluation de Francoise Vaillant et Pascal Mériaux. J'ai ainsi:

- **Prompté** avec une requête simple sans technique de prompt et sans mobiliser mes connaissances du sujet ni de *Python*.
- Volontairement ignoré la majorité de la réponse de l'IA.
- Évité de relire le code fourni et n'ai pas vérifié qu'il fonctionne comme attendu.

ChatGPT

Interactions: → voir ce lien

Résultat du code:

```

import random

def afficher_allumettes(n):
    print(f"\nIl reste {n} allumette{'s' if n > 1 else ''} sur le tas.")

    # Affichage ligne par ligne
    lignes = [
        ' '.join(['o'] * n),
        ' '.join(['|'] * n),
        ' '.join(['|'] * n),
        ' '.join(['|'] * n)
    ]
    for ligne in lignes:
        print(ligne)

def demander_prise(max_possible):
    while True:
        try:
            prise = input("Combien voulez-vous en prendre ?\n> ")
            nb = int(prise)
            if nb < 1:
                print(f"Impossible ! Prise invalide. ({nb} < 1).")
            elif nb > 3:
                print(f"Impossible ! Prise invalide. ({nb} > 3).")
            elif nb > max_possible:
                print(f"Impossible ! Prise invalide. ({nb} > {max_possible}).")
            else:
                return nb
        except ValueError:
            print("Impossible ! Prise invalide. La prise doit être un entier.")

def strategie_distrain(n):
    nb = random.randint(4, 10)
    print("> " + str(nb))
    if nb > n or nb > 3 or nb < 1:
        print("Oups ! Je me suis trompé... Je recommence.")
        return strategie_naif(n)
    return nb

def strategie_naif(n):
    return random.randint(1, min(3, n))

def strategie_rapide(n):
    return min(3, n)

def strategie_expert(n):
    return n % 4 if n % 4 != 0 else random.randint(1, min(3, n))

def coup_ordinateur(n, strategie):
    print("À mon tour.")
    if strategie == "distrain":
        nb = strategie_distrain(n)
    elif strategie == "naif":
        nb = strategie_naif(n)
    elif strategie == "rapide":
        nb = strategie_rapide(n)

```

```

elif strategie == "expert":
    nb = strategie_expert(n)
else:
    nb = strategie_naif(n) # par défaut

print(f"Je retire {nb} allumette{'s' if nb > 1 else ''} au tas.")
return nb

def jeu():
    print("Début de partie. Voulez-vous jouer en premier ? [y/N]")
    reponse = input().strip().lower()
    joueur_commence = reponse == 'y'

    if joueur_commence:
        print("Vous avez choisi de jouer en premier.")
    else:
        print("Vous avez choisi de laisser l'ordinateur commencer.")

    # Choix de stratégie
    strategies = ["distrain", "naif", "rapide", "expert"]
    strategie = random.choice(strategies)
    print(f"\nL'ordinateur joue en mode **{strategie.upper()}**.\n")

    allumettes = 13
    tour_joueur = joueur_commence

    while allumettes > 0:
        afficher_allumettes(allumettes)

        if tour_joueur:
            nb = demander_prise(allumettes)
            print(f"Vous retirez {nb} allumette{'s' if nb > 1 else ''} au tas.")
        else:
            nb = coup_ordinateur(allumettes, strategie)

        allumettes -= nb
        tour_joueur = not tour_joueur

    # Fin de partie
    afficher_allumettes(allumettes)
    if tour_joueur:
        print("\nVous avez perdu !")
    else:
        print("\nVous avez gagné !")

if __name__ == "__main__":
    jeu()

```

Test du programme:

```

Début de partie. Voulez-vous jouer en premier ? [y/N]
y
Vous avez choisi de jouer en premier.

L'ordinateur joue en mode **EXPERT**.

```

```

Il reste 13 allumettes sur le tas.
o o o o o o o o o o o o o
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
Combien voulez-vous en prendre ?
> 4
Impossible ! Prise invalide. (4 > 3).
Combien voulez-vous en prendre ?
> 0
Impossible ! Prise invalide. (0 < 1).
Combien voulez-vous en prendre ?
> 1
Vous retirez 1 allumette au tas.

Il reste 12 allumettes sur le tas.
o o o o o o o o o o o o o
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
À mon tour.
Je retire 3 allumettes au tas.

Il reste 9 allumettes sur le tas.
o o o o o o o o o o
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
Combien voulez-vous en prendre ?
> bou
Impossible ! Prise invalide. La prise doit être un entier.
Combien voulez-vous en prendre ?
> 3
Vous retirez 3 allumettes au tas.

Il reste 6 allumettes sur le tas.
o o o o o o
| | | | | |
| | | | | |
| | | | | |
À mon tour.
Je retire 2 allumettes au tas.

Il reste 4 allumettes sur le tas.
o o o o
| | | |
| | | |
| | | |
Combien voulez-vous en prendre ?
> 3
Vous retirez 3 allumettes au tas.

Il reste 1 allumette sur le tas.
o
|
|

```

```

|
À mon tour.
Je retire 1 allumette au tas.

Il reste 0 allumette sur le tas.


Vous avez perdu !

```

Commentaires sur le code obtenu

- ChatGPT a considéré par défaut qu'on voulait une IA experte. Comme les réponses sont non-déterministes, on peut s'attendre à d'autres résultats pour l'exact même prompt.
- Par ailleurs, malgré avoir rappelé dans sa réponse que le perdant était celui qui retirait la dernière allumette, il a sorti un code qui implémentait la variante inverse. Ainsi, on gagne en retirant la dernière. . .
- De même le code est très peu commenté, là où on pourrait attendre d'un étudiant à qui on demande un code de le commenter en détail pour expliquer son raisonnement (raffinages notamment).
- Il a utilisé un paradigme fonctionnel/impératif qui sied bien à du Python et qui rend le code très lisible et globalement factorisé.
- Pour autant, la solution pour l'affichage final (que ce soit celui des allumettes ou des valeurs de prises invalides) est très douteux: c'est peu efficace et répétitif et pourrait profiter de la simplicité de la syntaxe Python (très permissive) pour être plus concise. Aussi, les allumettes ne sont pas groupées comme attendu.
- À noter tout de même que l'affichage est **exactement** comme attendu excepté les allumettes et la fin où il affiche 3 lignes vides car la fonction d'affichage est encore appelée.
- Il n'a pas du tout compris ce qu'on demande pour la stratégie "Distrain".

→ Globalement, tous ces problèmes sont évitables en promptant mieux, en connaissant mieux le sujet et en relisant l'*output* pour pouvoir interagir correctement avec l'IA en revenant sur les erreurs/manquements.

Github Copilot (Claude 3.7 Thinking)

Interactions: → voir ce fichier

Résultat du code:

```

import random

def afficher_allumettes(nombre):
    """Affiche les allumettes restantes avec le format demandé."""
    print(f"Il reste {nombre} allumette{'s' if nombre > 1 else ''} sur le tas.")

    # Groupe les allumettes par 5
    lignes = ["", "", "", ""]
    for i in range(nombre):
        if i > 0 and i % 5 == 0:
            lignes[0] += "  "
            lignes[0] += "o "
            lignes[1] += "| "
            lignes[2] += "| "
            lignes[3] += "| "

        for ligne in lignes:
            print(ligne)

def tour_joueur(nombre_allumettes):

```

```

"""Gère le tour du joueur."""
while True:
    try:
        choix = input("Combien voulez-vous en prendre ?\n> ")
        try:
            choix = int(choix)
            if choix < 1:
                print(f"Impossible ! Prise invalide. ({choix} < 1).")
            elif choix > 3:
                print(f"Impossible ! Prise invalide. ({choix} > 3).")
            elif choix > nombre_allumettes:
                print(f"Impossible ! Prise invalide. ({choix} > {nombre_allumettes}).")
            else:
                print(f"Vous retirez {choix} allumette{'s' if choix > 1 else ''} au tas.")
                return choix
        except ValueError:
            print("Impossible ! Prise invalide. La prise doit être un entier.")
    except Exception:
        print("Saisie invalide, veuillez réessayer.")

def tour_ordinateur_distrain(nombre_allumettes):
    """L'ordinateur veut retirer entre 1 à 3 allumettes même si ce n'est pas possible."""
    # Choisit un nombre entre 1 et 3 même si ça dépasse le nombre d'allumettes
    choix_initial = random.randint(1, 3)

    if choix_initial > nombre_allumettes:
        print(f"> {choix_initial}")
        print(f"Oops! Je ne peux pas prendre {choix_initial} allumettes car il n'en reste que {nombre_allumettes}")
        choix = nombre_allumettes
    else:
        choix = choix_initial

    print(f"> {choix}")
    print(f"Je retire {choix} allumette{'s' if choix > 1 else ''} au tas.")
    return choix

def tour_ordinateur_naif(nombre_allumettes):
    """L'ordinateur retire au hasard entre 1 et le maximum possible."""
    choix = random.randint(1, min(3, nombre_allumettes))
    print(f"> {choix}")
    print(f"Je retire {choix} allumette{'s' if choix > 1 else ''} au tas.")
    return choix

def tour_ordinateur_rapide(nombre_allumettes):
    """L'ordinateur retire systématiquement le nombre maximum d'allumettes possible."""
    choix = min(3, nombre_allumettes)
    print(f"> {choix}")
    print(f"Je retire {choix} allumette{'s' if choix > 1 else ''} au tas.")
    return choix

def tour_ordinateur_expert(nombre_allumettes):
    """L'ordinateur joue de la meilleure manière possible."""
    # Stratégie optimale: laisser un multiple de 4 plus 1 allumettes
    if nombre_allumettes % 4 == 1:
        # Position perdante, prend un nombre aléatoire
        choix = random.randint(1, min(3, nombre_allumettes))

```

```

else:
    # Position gagnante, calcule le nombre à prendre
    choix = (nombre_allumettes - 1) % 4
    if choix == 0:
        choix = random.randint(1, min(3, nombre_allumettes))

print(f"> {choix}")
print(f"Je retire {choix} allumette{'s' if choix > 1 else ''} au tas.")
return choix

def jeu_des_13_allumettes():
    """Fonction principale du jeu."""
    print("Bienvenue au jeu des 13 allumettes!")
    print("Le joueur qui prend la dernière allumette perd.")

    # Sélection de la stratégie de l'ordinateur
    print("\nChoisissez la difficulté:")
    print("1. Distrain - L'ordinateur fait parfois des erreurs")
    print("2. Naif - L'ordinateur joue au hasard")
    print("3. Rapide - L'ordinateur prend toujours le maximum")
    print("4. Expert - L'ordinateur joue parfaitement")

    while True:
        choix_difficulte = input("Votre choix (1-4): ")
        try:
            difficulte = int(choix_difficulte)
            if 1 <= difficulte <= 4:
                break
            print("Veuillez entrer un nombre entre 1 et 4.")
        except ValueError:
            print("Veuillez entrer un nombre valide.")

    # Assigner la stratégie de l'ordinateur en fonction du choix
    if difficulte == 1:
        tour_ordinateur = tour_ordinateur_distrain
    elif difficulte == 2:
        tour_ordinateur = tour_ordinateur_naif
    elif difficulte == 3:
        tour_ordinateur = tour_ordinateur_rapide
    else:
        tour_ordinateur = tour_ordinateur_expert

    # Choix de l'ordre de jeu
    tour_du_joueur = None
    while tour_du_joueur is None:
        reponse = input("Début de partie. Voulez-vous jouer en premier ? [y/N] ")
        if reponse.lower() in ['y', 'yes', 'oui']:
            tour_du_joueur = True
            print("Vous avez choisi de jouer en premier.")
        elif reponse.lower() in ['n', 'no', 'non', '']:
            tour_du_joueur = False
            print("Vous avez choisi de laisser l'ordinateur commencer.")
        else:
            print("Réponse non reconnue. Veuillez répondre par y ou N.")

nombre_allumettes = 13

```

```

while nombre_allumettes > 0:
    print("") # Ligne vide pour séparer les tours
    afficher_allumettes(nombre_allumettes)

    if tour_du_joueur:
        nombre_prises = tour_joueur(nombre_allumettes)
    else:
        print("À mon tour.")
        nombre_prises = tour_ordinateur(nombre_allumettes)

    nombre_allumettes -= nombre_prises

    # Si plus d'allumettes, on a un gagnant
    if nombre_allumettes == 0:
        if tour_du_joueur:
            print("\nVous avez perdu !")
        else:
            print("\nVous avez gagné !")
        break

    tour_du_joueur = not tour_du_joueur

    # Demander si le joueur veut rejouer
    rejouer = input("\nVoulez-vous rejouer (o/n)? ").lower()
    if rejouer in ['o', 'oui', 'yes', 'y']:
        jeu_des_13_allumettes()

# Lancer le jeu
if __name__ == "__main__":
    jeu_des_13_allumettes()

```

Test du programme:

```

Bienvenue au jeu des 13 allumettes!
Le joueur qui prend la dernière allumette perd.

Choisissez la difficulté:
1. Distrain - L'ordinateur fait parfois des erreurs
2. Naif - L'ordinateur joue au hasard
3. Rapide - L'ordinateur prend toujours le maximum
4. Expert - L'ordinateur joue parfaitement
Votre choix (1-4): 4
Début de partie. Voulez-vous jouer en premier ? [y/N] y
Vous avez choisi de jouer en premier.

Il reste 13 allumettes sur le tas.
o o o o o   o o o o o   o o o
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
Combien voulez-vous en prendre ?
> 3
Vous retirez 3 allumettes au tas.

Il reste 10 allumettes sur le tas.
o o o o o   o o o o o

```



```

| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
À mon tour.
> 1
Je retire 1 allumette au tas.

Il reste 9 allumettes sur le tas.
o o o o o   o o o o
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
Combien voulez-vous en prendre ?
> 3
Vous retirez 3 allumettes au tas.

Il reste 6 allumettes sur le tas.
o o o o o   o
| | | | | |
| | | | | |
| | | | | |
À mon tour.
> 1
Je retire 1 allumette au tas.

Il reste 5 allumettes sur le tas.
o o o o o
| | | | |
| | | | |
| | | | |
Combien voulez-vous en prendre ?
> 3
Vous retirez 3 allumettes au tas.

Il reste 2 allumettes sur le tas.
o o
| |
| |
| |
À mon tour.
> 1
Je retire 1 allumette au tas.

Il reste 1 allumette sur le tas.
o
|
|
|
Combien voulez-vous en prendre ?
> 3
Impossible ! Prise invalide. (3 > 1).
Combien voulez-vous en prendre ?
> 0
Impossible ! Prise invalide. (0 < 1).
Combien voulez-vous en prendre ?
> bau

```

```

Impossible ! Prise invalide. La prise doit être un entier.
Combien voulez-vous en prendre ?
> 1
Vous retirez 1 allumette au tas.

Vous avez perdu !

Voulez-vous rejouer (o/n)? n

```

Commentaires sur le code obtenu

- Affichage presque réussi. L'IA n'a pas ajouté la délimitation en tas pour toutes les lignes. Outre ça, l'affichage textuel est exactement comme attendu.
- Globalement, le programme fait exactement ce qu'on attend de lui.
- Le code est excellent et chaque fonction a sa *docstring* fournie avec quelques commentaires.
- Cependant, il y a une sensation d'*over-engineering*— qui est certainement inhérent à Claude Sonnet 3.7— avec des répétitions qui ne sont pas nécessaires par endroits.
- L'IA a ajouté le choix de difficulté, qui n'est pas explicitement demandé. C'est une bonne chose.

→ Même conclusion, en promptant mieux on aurait pu faire ça en un seul prompt avec exactement ce qu'on veut.

Création de contenu

Prompt:

- J'ai pour mission de coder en Python le jeu des allumettes. Il y a 13 allumettes dans le tas initial dans lequel le joueur puis l'ordinateur retirent tour à tour entre 1 et 3 allumettes. Le perdant est le dernier à avoir pris une allumette. On doit donc faire jouer un humain contre un ordinateur: On demande à l'humain lors de son tour le nombre d'allumettes qu'il veut retirer, tandis que la difficulté de l'ordinateur sera choisie par le joueur en début de partie. Ces difficultés seront "Naif" où il choisit au hasard entre 1 et le nombre de retrait maximum possible, "Rapide" où il retire systématiquement le maximum d'allumettes possible, "Distrait" où il essaie de retirer entre 1 à 3 allumettes jusqu'à ce que ça fonctionne, et "Expert" où il joue du mieux que possible. Pour cette dernière, en analysant les tours possibles il semblerait qu'il faut toujours réussir à laisser un nombre d'allumettes congru à 1 modulo 4 à son adversaire. Un affichage particulier est attendu. Je te le partagerai juste après. Peux-tu me faire la structure du code Python ? Je voudrai faire ça de manière fonctionnelle afin d'éviter le moins de répétition possible. Je me chargerai du gros du code, écris-moi juste la structure à compléter.

Émulation

Pour émuler une consommation créative de l'IA, je me suis encore basé sur la rubrique d'évaluation de Françoise Vaillant et Pascal Mériaux. J'ai ainsi:

- **Prompté** avec une requête plus complète en utilisant mes connaissances du sujet et de *Python* afin de détailler ce que je veux exactement.
- Intéragi et pris en compte les réponses de l'IA pour corriger/améliorer ce qu'elle donne.
- Demandé à obtenir un squelette plutôt qu'un code complet.

ChatGPT

Interactions: → voir ce lien

Résultat du code:

```

import random

# ----- Paramètres du jeu -----

```

```

NB_ALLUMETTES_INIT = 13
RETRAIT_MIN = 1
RETRAIT_MAX = 3

# ----- Fonctions de l'IA -----

def ia_naif(nb_allumettes):
    """IA naïve : choix aléatoire valide"""
    pass

def ia_rapide(nb_allumettes):
    """IA rapide : retire le maximum possible"""
    pass

def ia_distrain(nb_allumettes):
    """IA distrain : tente jusqu'à réussir un retrait valide"""
    pass

def ia_expert(nb_allumettes):
    """IA expert : stratégie optimale pour gagner"""
    pass

# ----- Fonction de sélection de l'IA -----

def choisir_strategie_ia(nom_strategie):
    """Retourne la fonction IA correspondant au niveau choisi"""
    strategies = {
        "Naif": ia_naif,
        "Rapide": ia_rapide,
        "Distrain": ia_distrain,
        "Expert": ia_expert
    }
    return strategies.get(nom_strategie, ia_naif)

# ----- Fonction de tour du joueur humain -----

def tour_joueur(nb_allumettes):
    """Demande et valide le nombre d'allumettes retirées par le joueur"""
    pass

# ----- Fonction d'affichage (à personnaliser) -----

def afficher_allumettes(nb_allumettes):
    """Affiche les allumettes restantes d'une manière visuelle particulière"""
    pass

# ----- Fonction principale de jeu -----

def jouer_partie():
    nb_allumettes = NB_ALLUMETTES_INIT

    # Sélection de la stratégie
    niveau = input("Choisissez la difficulté de l'ordinateur (Naif, Rapide, Distrain, Expert) : ")
    ia = choisir_strategie_ia(niveau)

    joueur_courant = "Humain" # Alternance "Humain" / "IA"

```

```

while nb_allumettes > 0:
    afficher_allumettes(nb_allumettes)

    if joueur_courant == "Humain":
        retrait = tour_joueur(nb_allumettes)
    else:
        retrait = ia(nb_allumettes)
        print(f"L'ordinateur retire {retrait} allumette(s).")

    nb_allumettes -= retrait

    # Vérifie la fin du jeu
    if nb_allumettes == 0:
        print(f"{joueur_courant} a perdu !")
        break

    # Changer de joueur
    joueur_courant = "IA" if joueur_courant == "Humain" else "Humain"

# ----- Lancement du jeu -----

if __name__ == "__main__":
    jouer_partie()

```

C'est un code à compléter.

Commentaires sur le code obtenu

- Avec le prompt fait, l'IA donne uniquement un code complétable qui a une structure plutôt bonne et qui inclut ce dont on a besoin. On pourrait lui faire faire l'affichage du texte comme il s'agit d'une tâche laborieuse et répétitive. En fait, il s'agit d'une sorte de raffinement qu'il nous a fait.

Github Copilot

Interactions: → voir ce lien

Résultat du code:

```

import random

def initialiser_jeu():
    """Initialise le jeu et récupère les paramètres de départ"""
    nb_allumettes = 13
    print("Bienvenue dans le jeu des allumettes!")
    print("Choisissez le niveau de difficulté de l'ordinateur :")
    print("1. Naïf")
    print("2. Rapide")
    print("3. Distrain")
    print("4. Expert")

    # À compléter : code pour récupérer le choix de difficulté

    return nb_allumettes, difficulte

def afficher_allumettes(nb_allumettes):
    """Affiche l'état actuel du jeu avec les allumettes restantes"""
    # À compléter : code pour afficher les allumettes selon le format demandé

```

```

pass

def tour_joueur(nb_allumettes):
    """Gère le tour du joueur humain"""
    # À compléter : demande au joueur combien d'allumettes il veut retirer
    # Avec vérification de la validité du choix (entre 1 et min(3, nb_allumettes))
    pass

def strategie_naif(nb_allumettes):
    """Stratégie où l'ordinateur choisit aléatoirement entre 1 et le max possible"""
    max_retrait = min(3, nb_allumettes)
    return random.randint(1, max_retrait)

def strategie_rapide(nb_allumettes):
    """Stratégie où l'ordinateur retire toujours le maximum possible"""
    return min(3, nb_allumettes)

def strategie_distrain(nb_allumettes):
    """Stratégie où l'ordinateur essaie de retirer entre 1 et 3 jusqu'à ce que ça fonctionne"""
    # À compléter : simulation d'un ordinateur qui fait des erreurs
    pass

def strategie_expert(nb_allumettes):
    """Stratégie optimale basée sur la théorie des jeux (modulo 4)"""
    # À compléter : implémentation de la stratégie gagnante
    pass

def tour_ordinateur(nb_allumettes, difficulte):
    """Gère le tour de l'ordinateur selon le niveau de difficulté choisi"""
    if difficulte == 1: # Naïf
        retrait = strategie_naif(nb_allumettes)
    elif difficulte == 2: # Rapide
        retrait = strategie_rapide(nb_allumettes)
    elif difficulte == 3: # Distrain
        retrait = strategie_distrain(nb_allumettes)
    else: # Expert
        retrait = strategie_expert(nb_allumettes)

    # À compléter : affichage du coup de l'ordinateur

    return nb_allumettes - retrait

def jouer():
    """Fonction principale qui gère la boucle de jeu"""
    nb_allumettes, difficulte = initialiser_jeu()
    tour = 0 # 0 pour le joueur, 1 pour l'ordinateur

    while nb_allumettes > 0:
        afficher_allumettes(nb_allumettes)

        if tour == 0: # Tour du joueur
            nb_allumettes = tour_joueur(nb_allumettes)
            if nb_allumettes == 0:
                print("Vous avez pris la dernière allumette. Vous avez perdu!")
                break
        else: # Tour de l'ordinateur

```

```

        nb_allumettes = tour_ordinateur(nb_allumettes, difficulte)
        if nb_allumettes == 0:
            print("L'ordinateur a pris la dernière allumette. Vous avez gagné!")
            break

        # Changement de tour
        tour = 1 - tour

        # Demander si le joueur veut rejouer
        # À compléter

if __name__ == "__main__":
    jouer()

```

Commentaires sur le code obtenu

- Pareillement que pour ChatGPT, la structure est ce dont on attendrait d'un raffinement, quoique beaucoup de structures de contrôle sont imbriquées. Comme d'habitude, Claude a tendance à *over-engineer* un peu, puisqu'ici il mâche même un bout du code à l'utilisateur. Dans ce cas d'utilisation spécifiquement, ce n'est peut-être pas si mal pour atteindre la suite de notre projet de **co-crédation**.

Co-crédation

Prompt:

- Tu es un programmeur fonctionnel. Ton objectif est d'implémenter en Python le jeu des 13 allumettes. Il s'agit d'un jeu à deux joueurs où on place un tas initial d'allumettes (13 par défaut) dans lequel chaque joueur va prendre tour à tour entre 1 et 3 allumettes. Le perdant est le joueur ayant pris la dernière allumette. Tu codes le jeu de sorte qu'il s'agit d'un affrontement entre un joueur humain et un ordinateur. Tu pourras demander en début de partie si le joueur veut commencer ou laisser la main à l'ordinateur. Ce dernier aura 4 niveaux de difficultés qu'on nous demandera de choisir en début de partie: Naif où l'ordi prend un nombre entre 1 et le maximum possible (3 ou moins s'il reste moins de 3 allumettes), Rapide où l'ordi prend toujours le nombre maximum d'allumettes que lui permet le tas, Distract ou il prend un nombre entre 1 et 3, et Expert où il joue de la meilleure manière possible. On attend un affichage particulier dont tu laisseras la fonction vide pour le moment. Aussi, tu codes avec un code facilement changeable avec des valeurs par défaut qu'on peut modifier. L'architecture de l'application doit aussi permettre d'ajouter une nouvelle difficulté sans modifier grand chose au code déjà existant. Enfin, tu évites au maximum d'imbriquer des structures de contrôle: Si plusieurs structures s'imbriquent c'est qu'on peut en faire plusieurs fonctions.

Atteindre la co-crédation

Pour atteindre la co-crédation, j'ai d'abord role prompté ChatGPT pour spécialiser un peu plus ses réponses. J'ai aussi assimilé les consignes du projet et les ai clairement dictées à l'IA, tout en détaillant ce que j'attendais exactement, tout en laissant de la place pour des sections que je pourrais coder seul.

ChatGPT

Interactions (importante pour voir les changements dans le code): → voir ce lien

Résultat du code (après interactions et corrections):

```

import random
from typing import Callable

# --- Configuration par défaut ---
DEFAULT_TAS = 13          # Nombre d'allumettes initial
DEFAULT_PRISE_MAX = 3     # Nombre max d'allumettes qu'un joueur peut retirer

```

```

# --- Affichage (à personnaliser selon besoin graphique ou console) ---
def afficher_etat(nb_allumettes: int):
    # Cette fonction est prévue pour afficher le nombre d'allumettes restantes
    # Exemple d'affichage graphique : | | | ... ou juste un nombre
    pass

# --- Entrée du joueur humain ---
def demander_choix_joueur(max_possible: int) -> int:
    """Demande au joueur combien d'allumettes il veut prendre, avec gestion des erreurs."""
    while True:
        try:
            choix = int(input(f"Combien voulez-vous en prendre ? "))
            if 1 <= choix <= max_possible:
                return choix
            # Message d'erreur personnalisé selon l'erreur
            raison_invalide = f"{choix}"
            raison_invalide += f"< {max_possible}" if choix > max_possible else f"> 1"
            print(f"Impossible ! Prise invalide. ({raison_invalide})")
        except ValueError:
            print("Entrée non valide. Entrez un nombre.")

# --- IA : Stratégies de l'ordinateur selon la difficulté choisie ---
def strategie_naive(tas_actuel: int, prise_max: int) -> int:
    """L'ordinateur prend un nombre aléatoire possible d'allumettes."""
    return random.randint(1, min(tas_actuel, prise_max))

def strategie_rapide(tas_actuel: int, prise_max: int) -> int:
    """L'ordinateur prend le maximum d'allumettes possible."""
    return min(tas_actuel, prise_max)

def strategie_distrain(tas_actuel: int, prise_max: int) -> int:
    """L'ordinateur prend un nombre complètement aléatoire entre 1 et 3 (ou max restant)."""
    return random.randint(1, min(tas_actuel, prise_max))

def strategie_expert(tas_actuel: int, prise_max: int) -> int:
    """
    Stratégie optimale basée sur le modulo :
    Idéalement, on laisse toujours un multiple de (prise_max + 1) à l'adversaire.
    """
    prise = (tas_actuel - 1) % (prise_max + 1)
    return prise if prise != 0 else 1

# --- Enregistrement des stratégies IA disponibles ---
strategies: dict[str, Callable[[int, int], int]] = {
    "naif": strategie_naive,
    "rapide": strategie_rapide,
    "distrain": strategie_distrain,
    "expert": strategie_expert,
}

# --- Lancement de la partie ---
def jouer_partie(tas_initial_allumettes: int = DEFAULT_TAS, prise_max: int = DEFAULT_PRISE_MAX):
    """
    Fonction principale qui orchestre le jeu :
    - Choix de la difficulté
    """

```

```

- Choix du joueur qui commence
- Boucle de jeu alternée
- Détection du perdant
"""
niveau = demander_difficulte()
strategie_ordi = strategies[niveau]

joueur_commence = demander_si_joueur_commence()

tas_allumettes = tas_initial_allumettes
joueur_actuel = "joueur" if joueur_commence else "ordi"

# Boucle principale du jeu
while tas_allumettes > 0:
    afficher_etat(tas_allumettes)
    prise = get_prise(joueur_actuel, tas_allumettes, prise_max, strategie_ordi)
    tas_allumettes -= prise
    joueur_actuel = "ordi" if joueur_actuel == "joueur" else "joueur"

# Le dernier joueur à avoir joué a perdu
afficher_etat(tas_allumettes)
perdant = "Vous avez perdu !" if joueur_actuel == "joueur" else "Vous avez gagné !"
print(perdant)

# --- Menu : Choix de la difficulté IA ---
def demander_difficulte() -> str:
    """Affiche et demande à l'utilisateur de choisir un niveau de difficulté IA."""
    print("Choisissez un niveau de difficulté :")
    for i, nom in enumerate(strategies):
        print(f"{i + 1}. {nom.capitalize()}")

    while True:
        try:
            choix = int(input("Entrez le numéro du niveau : "))
            if 1 <= choix <= len(strategies):
                return list(strategies.keys())[choix - 1]
        except ValueError:
            pass
        print("Choix invalide.")

# --- Menu : Choix du joueur qui commence ---
def demander_si_joueur_commence() -> bool:
    """Demande si le joueur souhaite commencer en premier."""
    choix = input("Début de partie. Voulez-vous jouer en premier ? (y/N) : ").lower()
    res = choix.startswith("y")
    if res:
        print("Vous avez choisi de jouer en premier.")
    else:
        print("Vous avez choisi de laisser l'ordinateur commencer.")
    return res

# --- Détermination de la prise de chaque joueur ---
def get_prise(joueur_actuel: str, tas_allumettes: int, prise_max: int, strategie_ordi: Callable[[int, int],
    """
    Donne le nombre d'allumettes prises par le joueur courant.
    Appelle soit la saisie du joueur humain, soit la stratégie IA.

```



```

"""
if joueur_actuel == "joueur":
    prise = demander_choix_joueur(min(tas_allumettes, prise_max))
    print(f"Vous retirez {prise} allumette" + ("s" if prise > 1 else "") + " du tas.")
else:
    prise = strategie_ordi(tas_allumettes, prise_max)
    print(f"L'ordinateur prend {prise} allumette(s).")
return prise

# --- Lancer le jeu depuis le terminal ---
if __name__ == "__main__":
    jouer_partie()

```

Commentaires sur le code obtenu

Ce code vient après interaction avec l'IA pour corriger les erreurs et affiner le code. On peut en retenir que: - Le code est plutôt bien structuré et suit les principes de la programmation fonctionnelle. Il est facile à lire et à comprendre, même si quelques choix de programmation rendent le code chargé. Aussi, j'ai eu à raffiner un peu plus après le premier passage, puisque l'IA a parfois oublié de ne pas imbriquer des structures de contrôle. De même, le code manquait de commentaires et de *docstrings*. Sinon, l'IA a globalement bien compris les consignes, malgré quelques stratégies incomprises comme "Distrain" qui n'est pas implémentée comme on l'attendait. De même, il semblerait que l'IA ait confondu le perdant et le gagnant, puisque le code est écrit pour que le joueur gagne en retirant la dernière allumette, alors que c'est l'inverse qui est attendu. C'est une erreur qui se répète avec ChatGPT alors que le prompt est clair. **Ca vient probablement de son apprentissage où la plupart des données apprises sont des jeux où le gagnant est celui qui retire la dernière allumette.**

→ On remarque de nouveau que sans connaissance/expertise au préalable du sujet, on ne peut pas exploiter pleinement les capacités de l'IA et on s'en remet à elle pour des tâches qui sont parfois simples. Je confirme avec ce test là qu'il faut un regard critique sur le code produit pour être capable de le corriger ou rebondir dessus, sans quoi on ne va non seulement rater le cahier des charges et finir avec un code qui parfois ne fonctionne pas, mais aussi apprendre largement moins et faire trop peu travailler notre intellect.

Conclusion sur la co-crédation pour une tâche simple (et avec sujet connu)

Globalement, on constate que la co-crédation dans les exercices d'apprentissage de programmation est possible avec les LLMs qui peuvent produire un code de qualité, mais qu'il y a nécessité en premier lieu de savoir interagir judicieusement avec l'IA, faire du role prompting et surtout connaître son sujet, sans quoi l'utilisateur manquera d'agentivité et laissera l'IA le mener au lieu de s'en servir comme assistant de réflexion. Ses connaissances sur le sujet et cette agentivité lui permettront aussi de corriger les erreurs éventuelles de l'IA et d'affiner/améliorer le code produit. De plus, sur une observation plus personnelle, les IAs généralistes comme ChatGPT manquent définitivement de spécialisation (*fine-tuning*) pour des tâches précises et semblent halluciner plus facilement (c.f le code du gagnant/perdant qui est incohérent par rapport à ce qu'écrit textuellement l'IA), là où Github Copilot (ici spécifiquement Claude Sonnet 3.7) est plus consistant et précis dans les réponses et évite ce genre d'erreur "bêtes". On peut probablement mettre ça sur le dos de l'apprentissage et du fine-tuning puisque Copilot est entraîné sur l'entiereté du code public disponible sur Github. Enfin, cela n'empêche pas Copilot sous Claude Sonnet notamment d'*over-engineer* sa production et de faire bien plus ce qu'on demande (ça peut être frustrant!), ce qui est encore une fois synonyme de tous les biais liés aux LLMs dont les utilisateurs doivent absolument être conscients sans quoi ils ne pourront pas exploiter pleinement les capacités de l'IA.