

RAPPORT SAE 1.01

Table des matières :

I : La description du projet :	2
II : Les fonctions utiles dans tout le projet :	2
La procédure “nombrelignehonizontale” :	2
La procédure “effacer_console” :	3
La fonction “inputCustom” :	5
III : La fonction de lancement des mini-jeux :	7
IV : Les menus :	8
Les différents menus et leur fonction :	8
Explication du code :	9
V : Gestion des scores :	11
Les fonctions :	11
La fonction “listejoueur” :	11
La fonction “enregistrer_score” :	12
La fonction “lire_scores_par_joueur” :	14
La fonction “affiche_score_final” :	15
La fonction “reinitialiser_scores” :	17
La fonction “afficher_scores_total” :	18
Essais et exemples :	19
La fonction liste joueur :	19
La fonction enregistrer_score :	20
La fonction lire scores par joueur :	20
La fonction score final :	20
La fonction réinitialiser score :	21
La fonction scores total :	22
VI : Jeu 1 : Devinette	24
Sa fonction principale :	24
Explication du code :	26
Jeux d'essais :	27
VII : Jeu 2 : Allumette	30
Sa fonction principale :	30
Explication du code :	32
Jeux d'essais :	33
VIII : Jeu 3 : Morpion	35
Sa fonction principale :	35
Explication du code :	41
Jeux d'essais :	48
IX : Jeu 4 : Puissance 4	56
Sa fonction principale :	56
Explication du code :	59
Jeux d'essais :	60

I : La description du projet :

Ce projet est donc un condensé de plusieurs mini-jeux dans un seul projet. Ce sont des mini-jeux qui se jouent dans la console, ils n'ont donc pas d'interface graphique. Il y a le jeu des devinettes, le jeu des allumettes et un morpion.

Pour préciser dès le début, notre projet fonctionne correctement dans la console Visual Studio Code ou si l'on exécute le fichier via le cmd avec :

```
C:\Users\NOM\Documents\SAE-1.01-S1>py main.py  
OU  
C:\Users\NOM\Documents\SAE-1.01-S1>python3 main.py
```

Mais nous avons rencontré un problème auquel nous n'avons pas trouvé de solution. Si l'on l'ouvre directement avec le clic droit sur la console python proposée dans le "ouvrir avec" dès que l'on a fini de rentrer les joueurs cela se ferme sans information.

II : Les fonctions utiles dans tout le projet :

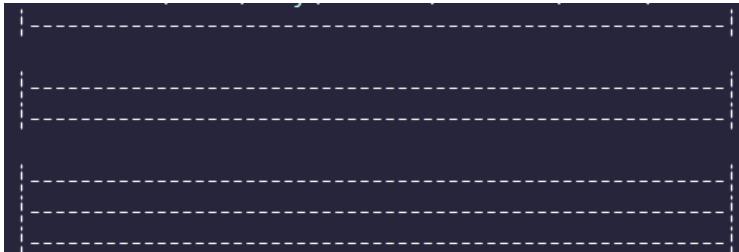
La procédure “nombrelignehorizontale” :

```
def nombrelignehorizontale(nombre : int, longueur : int) :  
    """  
        Fonction pour afficher un nombre de lignes horizontales  
    Args:  
        nombre (int) : Nombre de lignes  
        longueur (int) : Longueur de la ligne  
    Returns:  
        None : None  
    """  
  
    for i in range(nombre) :  
        print("|" + '-' * longueur + "|")  
        i += 1
```

La procédure **nombrelignehorizontale** sert à afficher plusieurs lignes horizontales constituées de tirets, encadrées de symboles " | " aux extrémités.
Elle possède 2 paramètres en entrée, le **nombre** qui correspond au nombre de lignes horizontales que l'on souhaite, et la **longueur** qui se multiplie avec le ‘-’ pour la longueur de la ligne souhaitée, les deux paramètres sont des entiers.
Elle ne retourne donc aucune valeur, elle n'effectue que les “print” des différentes lignes.

```
nombrelignehorizontale(1, 55) # Affiche une ligne horizontale de longueur 55
print()
nombrelignehorizontale(2, 55) # Affiche deux lignes horizontales de longueur 55
print()
nombrelignehorizontale(3, 55) # Affiche trois lignes horizontales de longueur 55
```

Ces appels produiront donc les résultats ci-dessous :



La procédure “effacer_console” :

```
def effacer_console() :
    """
        Fonction pour effacer la console.

    Args:
        None : None.

    Returns:
        None : None.
    """
    print("\033c", end="")
```

Cette procédure sert donc d'où son nom à vider la console pour éviter une surcharge d'information à l'écran, donc pour une meilleure compréhension des mini-jeux.
Elle n'a besoin d'aucun paramètre en entrée et ne retourne rien non plus.

La partie “**\033**” utilise un code **ANSI** pour effacer la console. Voici comment cela fonctionne :

- **\033** représente le caractère d'échappement (ESC), qui signale le début d'une commande spéciale dans les terminaux.
- **c** est une commande qui demande au terminal de **réinitialiser** (ou effacer) son affichage.

En écrivant dans un “print” **\033c**, la console va interpréter cela comme une commande pour effacer tout le texte à l'écran. Le “**end=**” à la fin permet d'éviter un retour à la ligne.
Cette fonction évite de devoir importer la bibliothèque **os**.

```
print("Bonjour")
print("salut")
print("coucou")
print("coucou")
print("coucou")
print("bonsoir")
```

Bonjour

salut

coucou

coucou

coucou

bonsoir

PS C:\Users\didry\OneDrive\Documents\GitHub\SAE-1.01-S1> 

```
print("Bonjour")
print("salut")
print("coucou")
print("coucou")
print("coucou")
print("bonsoir")
```

```
effacer_console()
```

TERMINAL

PORTS

COMMENTAIRES

PROBLEMES

SORTIE

CONSOLE DE DÉBOGAGE



Si l'on rajoute donc l'appel de la fonction il n'y a plus rien dans la console.

La fonction “inputCustom” :

```
def inputCustom(Texte: str, Type: type, Error: str, min_val: float = -float('inf'), max_val: float = float('inf')):
    """
    Demande à l'utilisateur une valeur de type Type (int, float, str) et valide cette valeur.

    Args:
        Texte (str): Le message à afficher pour demander la saisie à l'utilisateur.
        Type (type): Le type de la valeur attendue (int, float, str).
        Error (str): Le message d'erreur à afficher en cas de saisie incorrecte.
        min_val (int/float): La valeur minimale acceptée (par défaut -infini).
        max_val (int/float): La valeur maximale acceptée (par défaut infini).

    Returns:
        value: La valeur saisie et validée par l'utilisateur.
    """
    value = None
    while True:
        try:
            value = Type(input(Texte)) # Conversion basée sur le type fourni
            # Vérification des bornes uniquement pour les types numériques
            if isinstance(value, (int, float)):
                if value < min_val or value > max_val:
                    print(f"\033[31m La valeur doit être comprise entre {min_val} et {max_val}\033[0m")
                    continue
            return value
        except ValueError:
            print("\033[31m " + Error + "\033[0m")
```

Cette fonction sert à éviter d'avoir plein de test indépendant pour vérifier si les données à saisir sont dans le bon type ainsi que dans le bon intervalle.

Elle doit posséder de 3 à 5 paramètres en entrée.

Le premier est une **chaîne de caractères** qui sera le texte affiché dans le “input”.

```
entier = inputCustom("Saisir un entier : ", int, "La valeur doit être un entier", 1, 10)
print(f"Entier saisi : {entier}")
```

Dans cet exemple le premier paramètre est “saisir un entier”, on obtient donc ceci à l'affichage :

Saisir un entier :

Ensuite le deuxième paramètre doit être un type comme **int** ou **str**. Il permet donc de restreindre la saisie au type choisi.

Dans l'exemple précédent on est donc obligé de saisir un entier.

Saisir un entier : Salut
La valeur doit être un entier

Peu importe ce que l'on écrira si ce n'est pas un entier, cela renverra le paramètre **Error** que l'on aura donc saisi à l'appel de la fonction

Donc le 3e paramètre étant celui de l'erreur, il est une chaîne de caractère qui sera donc affichée si le type choisi ne correspond pas à la saisie de l'utilisateur. Ici, “**la valeur doit être un entier**”.

Enfin les 2 derniers paramètres sont pour gérer l'intervalle dans lequel la valeur doit être saisie.

```
entier = inputCustom("Saisir un entier : ", int, "La valeur doit être un entier",1,10)  
print(f"Entier saisi : {entier}")
```

Dans notre exemple, l'intervalle est (1;10). Si l'on saisit au-dessus ou en-dessous cela renverra :

```
Entrez un entier : -5  
La valeur doit être comprise entre 1 et 10.
```

III : La fonction de lancement des mini-jeux :

```
from fonction import *
from devinette import devinette
from allumette import allumette
from morpionTEST import morpion

def main () :
    choix : int
    menu("main")
    menu("jeux")
    choix = int(inputCustom("Votre choix 0 / 1 / 2 / 3 : ", int, "La valeur doit être un entier", 0, 3))

    if choix == 0 :
        lancement("devinette")
        devinette()

    if choix == 1 :
        lancement("allumette")
        allumette()

    if choix == 2 :
        lancement("morpion")
        morpion()

    if choix == 3 :
        menu("quitter")

#####
# Lancement du programme #
#####
if __name__ == "__main__":
    effacer_console()
    main()
```

La fonction “main” est donc la fonction pour le lancement des mini-jeux. Je vais donc décrire la fonction “menu” ainsi que la fonction lancement dans cette partie. Car la fonction main ne repose que sur les autres fonctions, l’unique chose dans le “main” est le choix du jeu ou le fait de quitter.

Elle possède aussi l’importation de chaque jeu ainsi que du fichier fonction comportant toutes les fonctions “autres” utilisées dans les programmes.

IV : Les menus :

Les différents menus et leur fonction :

```
def menu (cas : str) :
    """
    Fonction pour les menus
    Args:
        cas (str) : Nom du cas
    Returns:
        None : None
    """

    listejeux = ["Le jeux des devinettes.", "Le jeu des allumettes.", "Le jeu du morpion.", "Quitter les mini-jeux."]
    listedevinette = ["Plus grand", "Plus petit", "C'est gagné !"]

    if cas == "main" :
        print("Bienvenue dans le HUB, ici vous pouvez choisir ce que vous voulez faire parmis les choix suivants : ")

    if cas == "jeux" :
        switch(listejeux)

    if cas == "devinette" :
        switch(listedevinette)

    if cas == "quitter" :
        print("\033[33m Êtes-vous sûr de quitter les mini-jeux ? \033[0m")
        x = inputCustom("\033[33m 0 : Oui / 1 : Non --> \033[0m", int, "La valeur doit être un entier", 0, 1)

        if x == 0 :
            print("Merci d'avoir joué, à bientôt !")
            effacer_console()
            exit()

        else :
            from main import main
            main()

def switch(cas : list[str]) :
    """
    Fonction pour afficher un menu \n
    Paramètres : cas -> list[str] : Liste des choix possibles \n
    Retourne : None
    """

    for i in range(len(cas)) :
        print(str(i) + " : " + cas[i])
```

La fonction “menu” a pour principal objectif d'afficher les menu en fonction du jeu passé en paramètre.

Par exemple si l'on passe “main” en paramètre comme dans le programme principal on vient donc d'afficher :

```
Bienvenue dans le HUB, ici vous pouvez choisir ce que vous voulez faire parmis les choix suivants :
```

Ensuite nous avons un menu pour afficher la liste des jeux, qui utilise la fonction switch mise au-dessus pour afficher une liste défini dans la fonction de manière claire et non pas de manière brute comme peut le faire python à la base.

Celui des devinettes est utilisé dans le jeu éponyme pour afficher “plus grand ou plus petit ou c'est gagné”.

Enfin celui qui est différents des autres est le **menu**("quitter") car si il est appelé nous aurons donc 2 choix :

```
Êtes-vous sûr de quitter les mini-jeux ?  
0 : Oui / 1 : Non -->
```

Si l'on saisit "0" cela quittera donc le projet complètement en écrivant "merci d'avoir joué et à bientôt" cela effacera la console ensuite et quittera le programme grâce à la fonction python **exit**.

Si l'on saisit "1" cela relancera le "main" et donc relance le menu principal donc le jeu tout entier.

Explication du code :

La fonction "menu" utilisant la fonction "switch" je vais commencer par la fonction switch.

```
def switch(cas : list[str]) :  
    """  
        Fonction pour afficher un menu \n  
        Paramètres : cas -> list[str] : Liste des choix possibles \n  
        Retourne : None  
    """  
  
    for i in range(len(cas)) :  
        print(str(i) + " : " + cas[i])
```

Elle prend donc pour paramètre une liste de chaînes de caractères et ne retourne rien.

Elle utilise un for pour parcourir toute la liste donnée en paramètre grâce au "len(cas)" qui lui permet de savoir jusqu'à combien d'éléments il doit aller.

A chaque occurrence de la boucle on écrit dans la console l'indice de l'occurrence ainsi que la valeur correspondant à cet indice.

Dans le cas de l'appel du menu des devinettes on obtient :

```
0 : Plus grand  
1 : Plus petit  
2 : C'est gagné !
```

Les indices à gauche et leur valeur à droite.

```

def menu (cas : str) :
    """
    Fonction pour les menus \
    Paramètres : cas -> str : Choix du menu \
    Retourne : None
    """

    listejeux = ["Le jeu des devinettes.", "Le jeu des allumettes.", "Le jeu du morpion.", "Quitter les mini-jeux."]
    listedevinette = ["Plus grand", "Plus petit", "C'est gagné !"]

    if cas == "main" :
        print("Bienvenue dans le HUB, ici vous pouvez choisir ce que vous voulez faire parmis les choix suivants : ")

    if cas == "jeux" :
        switch(listejeux)

    if cas == "devinette" :
        switch(listedevinette)

    if cas == "quitter" :
        print("\033[33m Êtes-vous sûr de quitter les mini-jeux ? \033[0m")
        x = inputCustom("\033[33m 0 : Oui / 1 : Non --> \033[0m", int, "La valeur doit être un entier", 0, 1)

        if x == 0 :
            print("Merci d'avoir joué, à bientôt !")
            effacer_console()
            exit()

        else :
            from main import main
            main()

```

La fonction “menu”, elle, est toute simple. Elle repose sur des “if”, en fonction de quel paramètre est passé dans l’appel de la fonction cela affichera différentes choses. Et dans le cas de quitter on demande une saisie pour être sûr que l’utilisateur veut quitter le programme.

V : Gestion des scores :

Les fonctions :

La gestion des scores possède 5 fonctions. Une pour enregistrer les joueurs, une pour enregistrer les scores dans un fichier texte, une pour lire le fichier texte, une pour afficher les scores des différents mini-jeux et la dernière pour réinitialiser les scores complètement.

La fonction “listejoueur” :

```
def listejoueur(jeux: str) -> List[str]:
    """
    Fonction pour créer une liste de joueurs pour un jeu donné.
    Args:
        jeux (str): Nom du jeu
    Returns:
        List[str]: Liste des noms des joueurs
    """
    compteur: int = 0
    joueurs: List[str] = []

    print("\033[33mVous devez d'abord rentrer les noms des joueurs : \033[0m")
    print("Saisissez le nom des deux joueurs : ")

    # Collecter les noms des joueurs
    while compteur < 2:
        nom = str(inputCustom(f"Entrez le nom du joueur {compteur + 1} : ", str, "Le nom doit être une chaîne de caractères."))
        while nom == "": # Vérifier si le nom est vide
            print("\033[33mErreur : le nom du joueur ne peut pas être vide. Veuillez réessayer.\033[0m")
            nom = str(inputCustom(f"Entrez le nom du joueur {compteur + 1} : ", str, "Le nom doit être une chaîne de caractères."))
        # Normalisation du nom : enlever les espaces, mettre en minuscules et capitaliser
        nom_normalise = nom.strip().capitalize()
        joueurs.append(nom_normalise)
        compteur += 1

    # Enregistrer les joueurs et leur score initial (0) dans le fichier
    for joueur in joueurs:
        enregistrer_score_binaire(jeux, joueur, 0)

    return joueurs # Retourner directement la liste des joueurs
```

Cette fonction sert à rentrer à chaque lancement de partie, qui sont les joueurs qui jouent. Si l'on rentre un nom vide le code nous demande de ressaisir le nom. Ensuite on normalise le nom avec .strip et .capitalize pour retirer les espaces ainsi que mettre la 1ere lettre en majuscule et les autres en minuscules. Enfin on utilise la fonction enregistrer_score pour rentrer dans le fichier binaire les joueurs avec aucune addition de score. Et on retourne une liste pour être utilisable à la sélection aléatoire des joueurs dans chaque jeux.

La fonction “enregistrer_score” :

```
def est_un_nombre(val: str) -> bool:
    """
    Fonction pour vérifier si une chaîne de caractères est un nombre.
    Args:
        val (str): Chaîne de caractères à vérifier
    Returns:
        bool: True si la chaîne est un nombre, False sinon
    """
    try:
        int(val)
        return True
    except ValueError:
        return False

#####
#####

def enregistrer_score_binaire(nom_jeu: str, nom_joueur: str, nouveau_score: int) -> None:
    """
    Fonction pour enregistrer un nouveau score pour un joueur dans un fichier binaire.
    Args:
        nom_jeu (str): Nom du jeu
        nom_joueur (str): Nom du joueur
        nouveau_score (int): Score à ajouter
    Returns:
        None
    """
    nom_fichier = f'./score/{nom_jeu}.bin'

    # Lire les scores existants
    scores_liste = lire_scores_par_joueur_binaire(nom_jeu)

    # Traiter et valider les données existantes
    scores_valides : List[str]
    scores_valides = []

    joueur_trouve = False

    for joueur_score in scores_liste:
        if ':' not in joueur_score:
            print(f"Entrée ignorée (format invalide) : {joueur_score}")
            continue
        joueur, score_str = joueur_score.split(':', 1)
        if not joueur or not score_str or not est_un_nombre(score_str):
            print(f"Entrée ignorée (format invalide) : {joueur_score}")
            continue
        if joueur == nom_joueur:
            # Mettre à jour le score pour le joueur trouvé
            score = int(score_str) + nouveau_score
            scores_valides.append(f'{joueur}:{score}')
            joueur_trouve = True
        else:
            scores_valides.append(joueur_score)

    # Ajouter un nouveau joueur s'il n'est pas trouvé
    if not joueur_trouve:
        scores_valides.append(f'{nom_joueur}:{nouveau_score}')

    # Écrire tous les scores mis à jour dans le fichier binaire
    with open(nom_fichier, 'wb') as fichier:
        pickle.dump(scores_valides, fichier)

    print(f'Score de {nouveau_score} ajouté pour le joueur {nom_joueur} dans le fichier {nom_fichier}.')
```

1. Fonction `est_un_nombre` :

- Cette fonction est utilisée pour vérifier si une chaîne de caractères représente un nombre valide. Elle essaie de convertir la chaîne en entier avec `int(val)` et si cela échoue (lance une `ValueError`), elle retourne `False`, sinon `True`.
- Exemple d'utilisation :
 - Si `val = "25"`, `est_un_nombre("25")` retourne `True`.
 - Si `val = "abc"`, `est_un_nombre("abc")` retourne `False`.

2. Fonction `enregistrer_score_binaire` :

- Cette fonction enregistre ou met à jour le score d'un joueur dans un fichier binaire.
- Elle appelle `lire_scores_par_joueur_binaire` pour récupérer la liste des scores existants.
- Ensuite, elle utilise `est_un_nombre` pour vérifier que chaque score dans la liste est valide. Si un score n'est pas valide, il est ignoré, et un message d'erreur est affiché.
- Si le joueur est trouvé, son score est mis à jour en ajoutant le `nouveau_score` au score existant. Sinon, un nouvel enregistrement pour le joueur est ajouté.
- Enfin, les scores sont réécrits dans le fichier binaire.

Interaction des deux fonctions :

- **Validation des scores :**
 - Lors de la lecture des scores existants (via `lire_scores_par_joueur_binaire`), chaque score est vérifié avec `est_un_nombre` pour s'assurer qu'il est valide.
 - Si un score est mal formaté ou contient des caractères non numériques, il est ignoré et un message est affiché.
- **Mise à jour ou ajout d'un score :**
 - Si le joueur est déjà dans la liste des scores, son score est mis à jour avec le `nouveau_score`. La fonction `est_un_nombre` permet de s'assurer que seuls des scores valides sont considérés pour cette mise à jour.
 - Si le joueur n'existe pas encore dans la liste, il est ajouté avec son score initial.

La fonction “lire_scores_par_joueur” :

```
def lire_scores_par_joueur_binaire(nom_jeu: str) -> List[str]:  
    """  
        Fonction pour lire les scores depuis un fichier binaire.  
        Args:  
            nom_jeu (str): Nom du jeu  
        Returns:  
            List[str]: Liste des scores sous forme de chaîne (nom_joueur:score)  
    """  
    nom_fichier = f'./score/{nom_jeu}.bin'  
    try:  
        with open(nom_fichier, 'rb') as fichier:  
            return pickle.load(fichier)  
    except (FileNotFoundException, EOFError):  
        # Retourne une liste vide si le fichier n'existe pas ou est vide  
        return []
```

Cette fonction, `lire_scores_par_joueur_binaire`, lit des scores d'un fichier binaire et les retourne sous forme d'une liste de chaînes, où chaque chaîne représente un joueur et son score.

1. **Paramètre d'entrée :**
 - Elle prend un nom de jeu (`nom_jeu`) comme argument.
2. **Fichier cible :**
 - Elle construit le chemin du fichier binaire où les scores sont enregistrés, par exemple : `./score/nom_jeu.bin`.
3. **Lecture du fichier :**
 - Elle ouvre le fichier binaire en mode lecture ('rb') et utilise la bibliothèque `pickle` pour lire les données qui ont été enregistrées sous forme d'objet Python (ici une liste).
4. **Gestion des erreurs :**
 - Si le fichier n'existe pas (`FileNotFoundException`) ou est vide (`EOFError`), elle retourne une **liste vide**.
5. **Résultat :**
 - Elle renvoie une liste contenant les scores sous forme de chaînes (exemple : "Alice:100").

Cette fonction est utile pour récupérer facilement les scores enregistrés pour un jeu spécifique.

La fonction “affiche_score_final” :

```
def afficher_scores_final(nom_jeu: str) -> None:
    """
        Fonction pour afficher les scores finaux des joueurs d'un jeu.
        Affiche les scores dans un format clair et lisible et détermine le(s) gagnant(s) en fonction
        du score le plus proche de 0 si le jeu est "allumette", sinon le(s) gagnant(s) a/ont le score le plus élevé.

    Args:
        nom_jeu (str): Nom du jeu
    Returns:
        None : None
    """

    # Initialisation des variables
    gagnants: List[str] = [] # Liste pour stocker les gagnants
    score_max: int = -1 # Valeur de départ pour le jeu normal (score le plus élevé)
    score_proche_0: int = 1000 # Valeur de départ pour le jeu "allumette" (score le plus proche de 0)

    # Lire les scores depuis le fichier binaire
    scores_liste: List[str] = lire_scores_par_joueur_binaire(nom_jeu)

    if not scores_liste:
        print(f"Aucun score n'est disponible pour le jeu {nom_jeu}.")
        return

    print("\n--- Scores finaux ---")

    # Affichage des scores
    for joueur_score in scores_liste:
        joueur, score_str = joueur_score.split(':')
        score = int(score_str)
        print(f"{joueur}: {score} points")

    # Si le jeu est "allumette", on cherche le score le plus proche de 0
    if nom_jeu == "allumette":
        # Nouveau gagnant si score plus proche de 0
        if abs(score) < abs(score_proche_0):
            gagnants = [joueur]
            score_proche_0 = score

        # Ajouter à la liste si égalité avec le score le plus proche de 0
        elif abs(score) == abs(score_proche_0):
            gagnants.append(joueur)

    else:
        # Sinon, on cherche le score le plus élevé
        if score > score_max:
            gagnants = [joueur]
            score_max = score
        elif score == score_max:
            gagnants.append(joueur)

    # Vérification que des gagnants ont été déterminés
    if not gagnants:
        print("Erreur : Aucun gagnant n'a été déterminé.")
        return

    # Affichage du/des gagnants et de leur score
    if nom_jeu == "allumette":
        gagnants_str = ", ".join(gagnants)
        print(f"\Le(s) gagnant(s) pour le jeu des allumettes est/sont {gagnants_str} avec {score_proche_0} points (proximité à 0)!")
    else:
        gagnants_str = ", ".join(gagnants)
        print(f"\Le(s) gagnant(s) pour le jeu {nom_jeu} est/sont {gagnants_str} avec {score_max} points!")

    print("--- Fin des scores ---\n")
```

Cette fonction afficher_scores_final qui affiche les scores finaux des joueurs d'un jeu et détermine le ou les gagnants. Voici l'explication étape par étape de ce qu'il fait :

- 1. Initialisation des variables :**
 - gagnants: Une liste vide qui servira à stocker les noms des gagnants.
 - score_max: Valeur initiale du score maximum (pour les jeux "normaux").
 - score_proche_0: Valeur initiale pour le score le plus proche de zéro (spécial pour le jeu "allumette").
- 2. Lecture des scores des joueurs :**
 - lire_scores_par_joueur_binaire(nom_jeu): Cette fonction lit les scores des joueurs à partir d'un fichier binaire, renvoyant une liste de chaînes sous la forme nom_joueur:score.
- 3. Vérification de la disponibilité des scores :**
 - Si aucun score n'est disponible, un message d'erreur s'affiche et la fonction s'arrête.
- 4. Affichage des scores :**
 - Le code affiche les scores de chaque joueur dans un format lisible, en mettant les scores en gras (grâce aux séquences d'échappement \033[1m pour débuter le gras et \033[0m pour le terminer).
- 5. Détermination du ou des gagnants :**
 - Si le jeu est "allumette", il cherche le joueur dont le score est le plus proche de zéro.
 - Pour un autre jeu, il cherche le joueur avec le score le plus élevé.
 - En cas d'égalité, plusieurs gagnants peuvent être ajoutés à la liste.
- 6. Affichage des gagnants :**
 - Si des gagnants sont trouvés, leur(s) nom(s) et leur(s) score(s) sont affichés de manière lisible, en gras.
 - Si aucun gagnant n'a été trouvé (en cas d'erreur), un message d'erreur est affiché.
- 7. Message de fin :**
 - À la fin, un message indiquant la fin des scores est affiché, en gras également, pour marquer la fin du processus.

La fonction “reinitialiser_scores” :

```
def reinitialiser_scores_binaire(nom_jeu: str) -> None:
    """
    Fonction pour réinitialiser les scores des joueurs d'un jeu dans un fichier binaire.

    Args:
        nom_jeu (str): Nom du jeu
    Returns:
        None : None
    """

    nom_fichier: str
    nom_fichier = f'./score/{nom_jeu}.bin' # Nom du fichier binaire
    choix: int

    print(f"\033[33m Voulez-vous réinitialiser les scores pour le jeu {nom_jeu} ? \033[0m")
    choix = int(inputCustom("\033[33m 0 : Oui / 1 : Non --> \033[0m", int, "La valeur doit être un entier", 0, 1))

    if choix == 0:
        print("Réinitialisation des scores pour le jeu {nom_jeu}...")

        try:
            # Ouvrir le fichier en mode binaire et y écrire une liste vide
            with open(nom_fichier, 'wb') as fichier:
                pickle.dump([], fichier) # Écrire une liste vide pour réinitialiser les scores

            effacer_console()

            print(f'Les scores pour le jeu {nom_jeu} ont été réinitialisés. \n')
        except FileNotFoundError:
            print(f'Le fichier pour le jeu {nom_jeu} est introuvable.')

    if choix == 1:
        print("Lancement du jeu...")
```

La fonction `reinitialiser_scores_binaire` sert à réinitialiser les scores des joueurs d'un jeu en supprimant les anciens scores stockés dans un fichier binaire. Voici une explication détaillée du code :

1. Définition du fichier cible :

- La variable `nom_fichier` est définie comme étant le chemin du fichier binaire qui stocke les scores du jeu, en utilisant le nom du jeu passé en argument de la fonction (`nom_jeu`).

2. Demande de confirmation à l'utilisateur :

- La fonction affiche un message demandant à l'utilisateur s'il souhaite réinitialiser les scores pour le jeu spécifié.
- `inputCustom()` est utilisée pour lire la réponse de l'utilisateur (soit 0 pour Oui, soit 1 pour Non). Si l'entrée n'est pas correcte, un message d'erreur est montré et l'utilisateur doit entrer une valeur correcte.

3. Réinitialisation des scores si l'utilisateur choisit "Oui" (0) :

- Si l'utilisateur choisit 0 (réinitialiser les scores), le programme :
 - Affiche un message informant de la réinitialisation des scores.
 - essaie d'ouvrir le fichier binaire spécifié et y écrire une liste vide (`[]`), ce qui efface les scores stockés.
 - Si le fichier est trouvé, il est réécrit avec une liste vide.
 - En cas d'erreur de fichier (par exemple si le fichier n'existe pas), un message d'erreur est affiché.

- effacer_console() semble être une fonction qui nettoie l'affichage de la console (non définie ici, mais elle pourrait vider l'écran pour une meilleure lisibilité).
- Un message est affiché pour confirmer que les scores ont bien été réinitialisés.

4. Si l'utilisateur choisit "Non" (1) :

- Si l'utilisateur choisit 1 (ne pas réinitialiser), un message affichant "Lancement du jeu..." est montré, ce qui laisse penser que le jeu va commencer sans modifier les scores.

La fonction “afficher_scores_total” :

```
def afficher_scores_total() -> None:
    """
    Fonction pour afficher les scores finaux de tous les jeux disponibles, un par un.
    Affiche les scores dans un format clair et lisible et détermine les gagnants pour chaque jeu.

    Args:
        None

    Returns:
        None : None
    """

    # Liste des jeux disponibles (noms des jeux pour lesquels il existe des fichiers binaires de scores)
    jeux_disponibles = ['allumette', 'devinette', 'morpion', 'puissance4']

    if not jeux_disponibles:
        print("Aucun jeu n'est disponible.")
        return

    # Afficher les scores pour chaque jeu avec une séparation claire
    for nom_jeu in jeux_disponibles:
        print("=" * 50) # Séparation entre les jeux
        print(f"\033[1m\033[92mScores pour le jeu {nom_jeu}\033[0m") # Titre en vert
        print("=" * 50)
        try:
            afficher_scores_final(nom_jeu) # Fonction à définir ou à importer
        except FileNotFoundError:
            print(f"\u25bc Fichier de scores introuvable pour le jeu {nom_jeu}.")
        except Exception as e:
            print(f"\u25bc Une erreur s'est produite pour le jeu {nom_jeu} : {e}")

        # Message de fin des scores pour le jeu
        print("=" * 50)
        print(f"\033[1m\033[91mFin des scores pour le jeu {nom_jeu}\033[0m") # Message en rouge
        print("=" * 50)

    quitterjeux("main")
```

La fonction afficher_scores_total affiche les scores finaux de tous les jeux disponibles, un par un, et détermine les gagnants pour chaque jeu. Voici l'explication détaillée de ce que fait chaque partie du code :

- 1. Liste des jeux disponibles :**
 - jeux_disponibles contient une liste des jeux pour lesquels il existe des fichiers binaires de scores. Ici, les jeux sont : "allumette", "devinette", "morpion" et "puissance4".
- 2. Vérification de la disponibilité des jeux :**
 - Si jeux_disponibles est vide (aucun jeu disponible), un message indiquant qu'aucun jeu n'est disponible est affiché et la fonction s'arrête avec return.
- 3. Affichage des scores pour chaque jeu :**
 - La fonction entre dans une boucle for pour parcourir tous les jeux de la liste jeux_disponibles et afficher les scores pour chaque jeu un par un.
- 4. Affichage d'un séparateur :**
 - Un séparateur de 50 caractères (représenté par "=" * 50) est imprimé avant et après chaque jeu pour distinguer les scores de chaque jeu dans l'affichage.
- 5. Affichage du titre du jeu :**
 - Un titre avec le nom du jeu est affiché en vert, avec des codes de couleur ANSI (\033[92m pour vert et \033[0m pour réinitialiser).
- 6. Appel de la fonction afficher_scores_final :**
 - La fonction afficher_scores_final(nom_jeu) est appelée pour afficher les scores finaux et déterminer le ou les gagnants pour chaque jeu.
 - Si un fichier de scores pour le jeu n'est pas trouvé (FileNotFoundException), un message d'erreur est affiché.
 - Si une autre erreur se produit lors de l'affichage des scores, un message générique d'erreur est affiché avec le détail de l'exception.
- 7. Message de fin des scores pour chaque jeu :**
 - Après l'affichage des scores de chaque jeu, un message de fin est imprimé en rouge (\033[91m pour rouge), suivi d'un autre séparateur pour bien marquer la fin des scores pour chaque jeu.
- 8. Retour à la fonction principale :**
 - Enfin, la fonction quitterjeux("main") est appelée. Cette fonction n'est pas définie dans le code, mais on peut imaginer qu'elle permet de quitter ou de revenir à l'écran principal du programme.

Essais et exemples :

La fonction liste_joueur :

```
Vous devez d'abord rentrer les noms des joueurs :
Saisissez le nom des deux joueurs :
Entrez le nom du joueur 1 : █
```

```
Saisissez le nom des deux joueurs :
Entrez le nom du joueur 1 :
Erreur : le nom du joueur ne peut pas être vide. Veuillez réessayer.
Entrez le nom du joueur 1 : █
```

Si l'on ne saisit rien.

```
Entrez le nom du joueur 1 : Jules
Entrez le nom du joueur 2 : Alexandre
Score de 0 ajouté pour le joueur Jules dans le fichier ./score/devinette.bin.
Score de 0 ajouté pour le joueur Alexandre dans le fichier ./score/devinette.bin.
```

Après avoir saisi les 2 joueurs, on a donc l'initialisation dans le fichier binaire.

La fonction enregistrer score :

```
Bravo Jules, vous avez trouvé le nombre à deviner ! Le nombre à deviner était bien 5 !
Score de 1 ajouté pour le joueur Jules dans le fichier ./score/devinette.bin.
```

Ici après avoir trouvé le nombre on ajoute 1 au score du joueur dans le fichier correspondant.

Dans le cas d'allumette le score marche à l'envers à chaque défaite on prend -1 et c'est celui étant le plus proche de 0 sera afficher comme gagnant dans l'affichage final

```
Jules, vous avez perdu car vous avez pris la dernière allumette !
-----
Score de -1 ajouté pour le joueur Jules dans le fichier ./score/allumette.bin.
```

Les autres jeux fonctionnent de la même façon que devinette.

La fonction lire scores par joueur :

On ne peut pas vraiment la montrer car elle ne gère pas d'affichage mais elle sert à interpréter les informations dans les fichiers.

La fonction score final :

Ici pour le jeu des allumettes avec donc la proximité à 0 :

```
--- Scores finaux ---
A: -1 points
D: 0 points
Jules: -2 points
Alex: 0 points

Le(s) gagnant(s) pour le jeu des allumettes est/sont D, Alex avec 0 points (proximité à 0)!
--- Fin des scores ---
```

Ici pour les autres jeux :

```
--- Scores finaux ---
Jules: 2 points
Alexandre: 0 points
Alex: 0 points

Le(s) gagnant(s) pour le jeu devinette est/sont Jules avec 2 points!
--- Fin des scores ---
```

La fonction réinitialiser score :

```
|-----|  
Voulez-vous réinitialiser les scores pour le jeu devinette ?  
0 : Oui / 1 : Non --> |
```

Ici l'exemple est pour devinette mais cela fonctionne pour chaque jeu si l'on répond oui le fichier sera vidé et sinon on entre dans le jeu.

Si oui :

```
Les scores pour le jeu devinette ont été réinitialisés.
```

Si non :

```
Lancement du jeu...
```

La fonction scores total :

```
4 : Afficher les scores totaux.  
5 : Quitter les mini-jeux.  
Votre choix 0 / 1 / 2 / 3 / 4 / 5 : 4  
=====  
Scores pour le jeu allumette  
=====  
  
--- Scores finaux ---  
A: -3 points  
D: -2 points  
S: 0 points  
F: -1 points  
Test: 0 points  
Test2: -1 points  
Jules: -1 points  
Alex: -1 points  
1: 0 points  
2: -1 points  
  
Le(s) gagnant(s) pour le jeu des allumettes est/sont S, Test, 1 avec 0 points (proximité à 0)!  
--- Fin des scores ---  
  
=====  
Fin des scores pour le jeu allumette  
=====
```

```
=====  
Scores pour le jeu devinette  
=====  
  
--- Scores finaux ---  
A: 1 points  
D: 0 points  
T: 1 points  
G: 0 points  
Alex: 1 points  
Jules: 0 points  
L: 1 points  
M: 0 points  
  
Le(s) gagnant(s) pour le jeu devinette est/sont A, T, Alex, L avec 1 points!  
--- Fin des scores ---  
  
=====  
Fin des scores pour le jeu devinette  
=====  
  
Scores pour le jeu morpion  
=====  
Aucun score n'est disponible pour le jeu morpion.  
=====  
Fin des scores pour le jeu morpion  
=====  
  
Scores pour le jeu puissance4  
=====  
  
--- Scores finaux ---  
A: 0 points  
D: 1 points  
Jules: 0 points  
Ale: 1 points  
  
Le(s) gagnant(s) pour le jeu puissance4 est/sont D, Ale avec 1 points!  
--- Fin des scores ---  
  
=====  
Fin des scores pour le jeu puissance4  
=====
```

Cette fonction contrairement à “score_final” est accessible depuis le menu principal et affiche les scores de tous les jeux en se reposant sur “score_final” avec juste un affichage séparé.

VI : Jeu 1 : Devinette

Sa fonction principale :

```
from fonction import *
from time import sleep
from random import randint

def devinette():
    """
    Fonction pour jouer au jeu de la devinette \n
    Paramètres : None \n
    Retourne : None
    """

    intervalle: int
    nbrmystere: int
    j: int
    joueur1: str
    joueur2: str
    listej: list[str]
    gagner: bool
    gagner = False
    Sicompteur : str
    compteur : int
    compteur_max : int
    compteur = 0
    compteur_max = 0

    #Proposition réinitialisation des scores
    reinitialiser_scores("devinette")

    # Liste des joueurs
    listej = listejoueur("devinette")

    # Vérification si la liste contient bien deux joueurs
    if len(listej) < 2:
        print("Erreur : il doit y avoir exactement 2 joueurs.")
        return

    #Effacement de la console avec un sleep pour laisser le temps de lire
    sleep(3)
    effacer_console()
    nombrelignehorizontale(1, 55)
    print("\033[92m Lancement du jeu de la devinette \033[0m")
    nombrelignehorizontale(1, 55)

    # Assigner les joueurs aléatoirement
    j = randint(1, 2)

    if j == 1 :
        joueur1 = listej[0]
        joueur2 = listej[1]
    else :
        joueur1 = listej[1]
        joueur2 = listej[0]

    #Choix du compteur
    print("si vous rentrez autre chose que 'oui', le compteur sera désactivé.")
    Sicompteur = str(inputCustom("Voulez-vous choisir le nombre de tours ? (Oui/Non) : ", str, "La valeur doit être un caractère")).capitalize()
    if Sicompteur == "Oui" :
        compteur_max = int(inputCustom("Choisissez le nombre de tours : ", int, "La valeur doit être un entier", 1, 50))
    else :
        compteur_max = 1500000000000000 #Nombre très grand pour ne pas arrêter le jeu
```

Cette première partie étant l'initialisation des variables et l'assignation des joueurs et des scores.

```

#Début du jeu
intervalle = int(inputCustom(f"\{joueur1}, choisissez l'intervalle de jeu entre 1 et n : ", int, "La valeur doit être un entier", 1, 100))
nbrmystere = int(inputCustom(f"\{joueur1}, choisissez le nombre à deviner entre 1 et {intervalle} : ", int, "La valeur doit être un entier", 1, intervalle))
print("\033[F\033[K", end="")
print("\{joueur1} a choisi le nombre à deviner c'est à \{joueur2} de jouer ! \n")

while gagner==False:
    # Le joueur 2 fait une supposition
    nbrdevine = int(inputCustom(f"\{joueur2}, quel est le nombre ? ", int, "La valeur doit être un entier", 1, intervalle))
    menu("devinette")
    valeur = int(inputCustom(f"\{joueur1}, votre choix 0 / 1 / 2 : ", int, "La valeur doit être un 0, 1 ou 2", 0, 2))
    print("\033[F\033[K", end="")

    # Le joueur 1 répond
    if valeur == 0:
        print(f"Le nombre à deviner n'est pas {nbrdevine} !\033[1m\033[31m Il est plus grand\033[0m")
    elif valeur == 1:
        print(f"Le nombre à deviner n'est pas {nbrdevine} !\033[1m\033[31m Il est plus petit\033[0m")
    nomrelignehorizontale(1, 55)

    # Si le nombre n'a pas été trouvé en compteur_max tours
    if Sicompteur == "0" or Sicompteur == "o" :
        compteur += 1

    if compteur == compteur_max :
        effacer_console()
        print(f"Le nombre à deviner était {nbrmystere} !")
        print(f"Le nombre n'a pas été trouvé en {compteur_max} tours !")
        print(f"Bravo à \{joueur1} pour avoir caché le nombre !\n")
        gagner = True
        enregistrer_score("devinette", joueur1, 1)

    # Si le joueur 2 trouve le bon nombre
    if valeur == 2 and nbrdevine == nbrmystere :
        effacer_console()
        print(f"Bravo \{joueur2}, vous avez trouvé le nombre à deviner ! Le nombre à deviner était bien {nbrmystere} !")
        gagner = True
        enregistrer_score("devinette", joueur2, 1)

afficher_scores_final("devinette")
quitterjeux("devinette")

```

Cette deuxième partie correspond au jeu et à la fin du jeu.

Explication du code :

Le jeu de **devinette** est un jeu dans lequel deux joueurs s'affrontent. L'un choisit un nombre à faire deviner et l'autre joueur doit essayer de le deviner selon des paramètres modulables entre les parties.

Importation des modules nécessaires :

- Fonctions utilitaires (**fonction**).
- Gestion des délais (**sleep**) et génération de nombres aléatoires (**randint**).
- Entrées cachées via **getpass**.

Saisie du nombre mystère :

- Le premier joueur entre un nombre mystère (masqué grâce au **getpass**) dans une plage qui a lui même définie juste avant.
- Des validations assurent que le nombre saisi est un entier et se situe dans l'intervalle choisi.

Configuration initiale du jeu :

- Les scores sont réinitialisés ou non selon le choix des joueurs
- Les joueurs sont dispatchés aléatoirement dans les deux rôles du jeu
- Le joueur 1 définit l'intervalle du jeu.
- Une option permet de limiter le nombre de tours, ce qui permet au joueur qui choisit le nombre d'avoir une chance de gagner

Mécanique de jeu :

- Le joueur 2 propose un nombre.
- Le joueur 1 fournit un retour : le nombre est "plus grand", "plus petit" ou "correct".
- Si une limite de tours est activée, le jeu se termine une fois la limite atteinte.

Validation des réponses :

- Les réponses incorrectes de joueur 1 (déclarer le nombre trouvé alors qu'il ne l'est pas) sont bloquées.
- En cas d'erreur, des invitations supplémentaires permettent de corriger.

Conditions de victoire :

- Le joueur 2 gagne en trouvant le nombre mystère.
- Le joueur 1 gagne si le nombre n'est pas trouvé dans les tours impartis (ou aucune possibilité de victoire sans avoir ajouté l'option du nombres de tour)

Gestion des scores et sortie :

- Les scores sont mis à jour et affichés.
- Le jeu se termine avec une invitation à quitter.

Jeux d'essais :

```
Voulez-vous choisir le nombre de tours ? (Oui/Non) : Non  
Alex, Choisissez l'intervalle de jeu entre 1 et n : 5  
Alex, Saisissez le nombre mystère : █
```

Sélection de l'intervalle de jeu et de l'option d'activation du nombre de tours.

```
Alex, Choisissez l'intervalle de jeu entre 1 et n : 5  
Alex, Saisissez le nombre mystère :  
Le nombre doit être compris entre 1 et 5.  
Alex, Saisissez le nombre mystère : █
```

Sélection du nombre mystère (caché par le getpass)

```
Alex, Saisissez le nombre mystère :  
Alex a choisi le nombre à deviner c'est à Jules de jouer !
```

Lancement du jeu

```
Jules, quel est le nombre ? █
```

Demande au joueur 2 le nombre qu'il pense être la solution

```
Jules, quel est le nombre ? 6  
La valeur doit être comprise entre 1 et 5.
```

Vérification que le nombre choisi est bien dans l'intervalle

```
Jules, quel est le nombre ? 4  
0 : Plus grand  
1 : Plus petit  
2 : C'est gagné !  
Le nombre à deviner n'est pas 4 ! Il est plus petit
```

Choix du joueur 1 en fonction des 3 possibilités données. Ici le choix 1 est sélectionné

```
Jules, quel est le nombre ? 2  
0 : Plus grand  
1 : Plus petit  
2 : C'est gagné !  
Vous ne pouvez pas dire que le nombre est trouvé si ce n'est pas le bon nombre !  
Alex, votre choix 0 / 1 / 2 : █
```

Vérification de la véracité du choix de validation du nombre par le joueur 1

```
Jules, quel est le nombre ? 2
```

```
0 : Plus grand
```

```
1 : Plus petit
```

```
2 : C'est gagné !
```

```
Le nombre à deviner n'est pas 2 ! Il est plus grand
```

Ici le choix 2 est sélectionnée

```
Jules, quel est le nombre ? 3
```

```
0 : Plus grand
```

```
1 : Plus petit
```

```
2 : C'est gagné !
```

```
Alex, votre choix 0 / 1 / 2 : 2
```

```
Bravo Jules, vous avez trouvé le nombre à deviner ! Le nombre à deviner était bien 3 !  
Score de 1 ajouté pour le joueur Jules dans le fichier ./score/devinette.bin.
```

```
--- Scores finaux ---
```

```
A: 3 points
```

```
D: 1 points
```

```
T: 1 points
```

```
G: 0 points
```

```
Alex: 2 points
```

```
Jules: 1 points
```

```
Le(s) gagnant(s) pour le jeu devinette est/sont A avec 3 points!
```

```
--- Fin des scores ---
```

```
Voulez-vous vraiment quitter ?
```

```
0 : Oui / 1 : Non -->
```

Si les conditions sont réunis, le jeu s'arrête indiquant la victoire du joueur 2 et lui incrémentant les points dans le fichier des scores

```
Voulez-vous choisir le nombre de tours ? (Oui/Non) : Oui
```

```
Choisissez le nombre de tours : 5
```

```
Alex, Choisissez l'intervalle de jeu entre 1 et n : 5
```

```
Alex, Saisissez le nombre mystère :
```

```
Alex a choisi le nombre à deviner c'est à Jules de jouer !
```

Cas où un nombre de tour est sélectionné, choix du nombre de tour.

```
Le nombre à deviner était 3 !
Le nombre n'a pas été trouvé en 5 tours !
Bravo à L pour avoir caché le nombre !
```

```
Score de 1 ajouté pour le joueur L dans le fichier ./score/devinette.bin.
```

```
--- Scores finaux ---
```

```
A: 1 points
D: 0 points
T: 1 points
G: 0 points
Alex: 1 points
Jules: 0 points
L: 1 points
M: 0 points
```

```
Le(s) gagnant(s) pour le jeu devinette est/sont A, T, Alex, L avec 1 points!
```

```
--- Fin des scores ---
```

Cas de fin de partie où le nombre de tour est dépassé, le joueur 1 gagne et son score est incrémenter dans le fichier des scores.

VII : Jeu 2 : Allumette

Sa fonction principale :

```
from fonction import *
from time import sleep
from random import randint

def affichage_Partie_Allumette(ResteAllumette : int, joueur_actuel : str) -> None:
    """
    Fonction pour afficher le nombre d'allumettes restantes et le joueur qui doit jouer \n
    Paramètres : ResteAllumette (int), joueur_actuel (str) \n
    Retourne : None
    """

    effacer_console()
    nombrelignehorizontale(1, 55)
    print("\u033[92m Vous êtes dans une partie d'allumettes \u033[0m")
    nombrelignehorizontale(1, 55)
    print(f"\u033[92mIl reste {ResteAllumette} allumettes.\u033[0m")
    print(f"C'est au tour de \u033[0;36m{joueur_actuel}\u033[0m de jouer.")
```

```

def allumette():
    Paramètres : None \n
    Retourne : None
    """
    ResteAllumette : int
    allumetteprise : int
    choixjoueurs : int
    joueur1 : str
    joueur2 : str
    joueur_actuel : str

    ResteAllumette = 20

    # Proposition réinitialisation des scores
    reinitialiser_scores_binaire("allumette")

    # Liste des joueurs
    listej = listejoueur("allumette")

    # Vérification si la liste contient bien deux joueurs
    if len(listej) < 2:
        print("Erreur : il doit y avoir exactement 2 joueurs.")
        return

    # Effacement de la console avec un sleep pour laisser le temps de lire
    sleep(3)
    effacer_console()
    nombrelignehorizontale(1, 55)
    print("\033[92m Lancement du jeu des allumettes \033[0m")
    nombrelignehorizontale(1, 55)

    # Assigner les joueurs aléatoirement
    choixjoueurs = randint(1, 2)

    if choixjoueurs == 1 :
        joueur1 = listej[0]
        joueur2 = listej[1]
    else :
        joueur1 = listej[1]
        joueur2 = listej[0]

```

```

# Déterminer qui commence, ici joueur1 commence en se basant sur le choix joueurs qui est aléatoire
joueur_actuel = joueur1

#Début du jeu
while ResteAllumette > 0:

    # Afficher le nombre d'allumettes restantes et le joueur qui doit jouer
    affichage_Partie_Allumette(ResteAllumette, joueur_actuel)

    # Demander combien d'allumettes prendre
    allumetteprise = int(inputCustom(f"\033[0;36m{joueur_actuel}\033[0m : Combien d'allumettes voulez-vous prendre ? ", int, "La valeur doit être un entier"))

    # Vérification que le joueur ne prend pas plus d'allumettes que celles restantes
    while allumetteprise > ResteAllumette:
        affichage_Partie_Allumette(ResteAllumette, joueur_actuel)
        print("Vous ne pouvez pas prendre plus d'allumettes qu'il n'en reste !")
        allumetteprise = int(inputCustom(f"\033[0;36m{joueur_actuel}\033[0m : Combien d'allumettes voulez-vous prendre ? ", int, "La valeur doit être un entier"))

    # Vérifier si le joueur qui va jouer prend la dernière allumette
    if allumetteprise == ResteAllumette: # Si le joueur prend toutes les allumettes restantes, il perd
        effacer_console()
        print("\033[31m{joueur_actuel}, vous avez perdu car vous avez pris la dernière allumette ! \033[0m")
        nombreLigneHorizontale(1, 20)
        enregistrer_score_binaire("allumette", joueur_actuel, -1) # Enregistrer le score du perdant

    ResteAllumette = ResteAllumette - allumetteprise

    # Alterner les joueurs
    if joueur_actuel == joueur1:
        joueur_actuel = joueur2
    else:
        joueur_actuel = joueur1

afficher_scores_final("allumette")
quitterjeux("allumette")

```

Explication du code :

Le jeu des **allumettes** (ou jeu de nîme) est un jeu où deux joueurs s'affrontent pour éviter d'être le joueur à prendre la dernière allumette.

Importation des modules :

- Fonctions utilitaires (fonction).
- Gestion des délais (sleep) et tirage aléatoire pour déterminer le premier joueur (randint).

Affichage du jeu :

- La fonction `affichage_Partie_Allumette()` affiche le nombre d'allumettes restantes et le joueur dont c'est le tour.

Configuration initiale :

- Les scores sont réinitialisés ou non via une fonction utilitaire.
- Les joueurs sont définis par les utilisateurs via des insertions. Une vérification s'assure qu'il y a exactement deux participants.
- Le jeu commence avec un total de 20 allumettes.
- Le premier joueur est choisi aléatoirement.

Mécanique de jeu :

- À chaque tour, le joueur actif choisit combien d'allumettes il souhaite prendre (1 à 3).

- Une validation empêche de prendre plus d'allumettes qu'il n'en reste.
- Si un joueur prend la dernière allumette, il perd immédiatement la partie.

Alternance des joueurs :

- Le tour passe au joueur suivant après chaque prise.

Conditions de défaite :

- Si un joueur prend la dernière allumette, il perd la partie. Le score est enregistré négativement pour ce joueur.

Gestion des scores et fin de partie :

- Les scores sont mis à jour pour le joueur perdant.
- Les scores finaux sont affichés, et le jeu se termine avec une option pour quitter.

Jeux d'essais :

```
|-----|
Vous êtes dans une partie d'allumettes
|-----|
Il reste 20 allumettes.
C'est au tour de Alex de jouer.
Alex : Combien d'allumettes voulez-vous prendre ? □
```

Début de la partie et premier choix pour un joueur

```
Alex : Combien d'allumettes voulez-vous prendre ? 4
La valeur doit être comprise entre 1 et 3.
Alex : Combien d'allumettes voulez-vous prendre ? 0
La valeur doit être comprise entre 1 et 3.
Alex : Combien d'allumettes voulez-vous prendre ? □
```

Vérification du nombre d'allumette choisi par le joueur

```
Alex : Combien d'allumettes voulez-vous prendre ? 2□
```

```
Il reste 18 allumettes.
C'est au tour de Jules de jouer.
Jules : Combien d'allumettes voulez-vous prendre ? □
```

Soustraction, affichage et changement de joueur une fois la validation des allumettes retirée par le joueur précédent.

Jules : Combien d'allumettes voulez-vous prendre ? 3

Il reste 15 allumettes.

C'est au tour de Alex de jouer.

Alex : Combien d'allumettes voulez-vous prendre ?

Alex : Combien d'allumettes voulez-vous prendre ? 1

Il reste 14 allumettes.

C'est au tour de Jules de jouer.

Jules : Combien d'allumettes voulez-vous prendre ?

Il reste 1 allumette.

C'est au tour de Alex de jouer.

Alex : Combien d'allumettes voulez-vous prendre ? 1

Répétition des mêmes actions jusqu'à la dernière allumette

```
Alex, vous avez perdu car vous avez pris la dernière allumette !
```

```
|-----|  
Score de -1 ajouté pour le joueur Alex dans le fichier ./score/allumette.bin.
```

```
--- Scores finaux ---
```

```
A: -3 points
```

```
D: -2 points
```

```
S: 0 points
```

```
F: -1 points
```

```
Test: 0 points
```

```
Test2: -1 points
```

```
Jules: -1 points
```

```
Alex: -1 points
```

```
1: 0 points
```

```
2: -1 points
```

```
Le(s) gagnant(s) pour le jeu des allumettes est/sont S, Test, 1 avec 0 points (proximité à 0)!  
--- Fin des scores ---
```

Incrémantation des points en fonction du joueur perdant et affichage des scores.

VIII : Jeu 3 : Morpion

Sa fonction principale :

```
from fonction import *
from time import sleep
from random import randint

def taille_morpion() -> list[list[str]] :
    """
    Fonction pour choisir la taille du morpion
    Args:
        None : None
    Returns:
        list[list[str]]: La matrice du morpion.
    """

    mat : list[list[str]]
    i : int
    j: int
    val : str
    n : int
    mat = list([[]])
    n = int(inputCustom("De quelle taille doit être votre morpion sachant qu'il sera de dimension n*n : ", int, "La valeur doit être un entier", 3, 10))
    for i in range(0, n) :
        i = i
        ligne : list[str]
        ligne = list()
        for j in range(0, n) :
            j = j
            val = " "
            ligne.append(val)
        mat.append(ligne)

    nombrelignehorizontale(1, 55)
    print("Voici le morpion sur lequel vous allez jouer.")
    return mat

#####
#####
```



```
def jeu_morpion1(mat : list[list[str]], joueur : str) -> list[list[str]] :
    """
    Fonction pour jouer au jeu du morpion
    Args:
        mat (list[list[str]]): La matrice du morpion.
        joueur (str): Le nom du joueur.
    Returns:
        list[list[str]]: La matrice du morpion.
    """

    morpion_ligne_x : int
    morpion_colonne_x : int
    print(f'{joueur}, sélectionner la case dans laquelle vous voulez jouez (X) : ')
    morpion_ligne_x = int(inputCustom("Choisir ligne : ", int, "La valeur doit être un entier", 1, len(mat)))
    while morpion_ligne_x > len(mat) :
        morpion_ligne_x = int(inputCustom("Sélectionner la ligne dans l'intervalle de l'aire de jeu : ", int, "La valeur doit être un entier", 1, len(mat)))
    morpion_colonne_x = int(inputCustom("Choisir colonne : ", int, "La valeur doit être un entier", 1, len(mat)))
    while morpion_colonne_x > len(mat) :
        morpion_colonne_x = int(inputCustom("Sélectionner la colonne dans l'intervalle de l'aire de jeu : ", int, "La valeur doit être un entier", 1, len(mat)))
    morpion_colonne_x -= 1
    morpion_ligne_x -= 1
    if not(deja_pris(mat, morpion_ligne_x, morpion_colonne_x)) :
        mat[morpion_ligne_x][morpion_colonne_x] = "X"
    else:
        print()
        print("Case déjà prise, sélectionnez en une autre")
        jeu_morpion1(mat, joueur)
    print()
    nombrelignehorizontale(1, 55)
    print()
    return mat

#####
#####
```

```

def jeu_morpion2(mat : list[list[str]], joueur : str) -> list[list[str]] :
    """
    Fonction pour jouer au jeu du morpion
    Args:
        mat (list[list[str]]): La matrice du morpion.
        joueur (str): Le nom du joueur.
    Returns:
        list[list[str]]: La matrice du morpion.
    """

    morpion_ligne_o : int
    morpion_colonne_o : int
    print(f"{joueur}, sélectionner la case dans laquelle vous voulez jouer (0) :")
    morpion_ligne_o = int(inputCustom("Choisir ligne : ", int, "La valeur doit être un entier", 1, len(mat)))
    while morpion_ligne_o > len(mat) :
        morpion_ligne_o = int(inputCustom("Sélectionner la ligne dans l'intervalle de l'aire de jeu : ", int, "La valeur doit être un entier", 1, len(mat)))
    morpion_colonne_o = int(inputCustom("Choisir colonne : ", int, "La valeur doit être un entier", 1, len(mat)))
    while morpion_colonne_o > len(mat) :
        morpion_colonne_o = int(inputCustom("Sélectionner la colonne dans l'intervalle de l'aire de jeu : ", int, "La valeur doit être un entier", 1, len(mat)))
    morpion_colonne_o -= 1
    morpion_ligne_o -= 1
    if not(deja_pris(mat, morpion_ligne_o, morpion_colonne_o)) :
        mat[morpion_ligne_o][morpion_colonne_o] = "0"
    else:
        print()
        print("Case déjà prise, sélectionnez en une autre")
        jeu_morpion2(mat, joueur)
    print()
    nombrelignehorizontale(1, 55)
    print()
    return mat

#####
#####
```

```

def afficher_morpion(mat: list[list[str]]) -> list[list[str]]:
    Fonction pour afficher le morpion avec numérotation des lignes et colonnes (à partir de 1).
```

Args:
 | mat (list[list[str]]): La matrice du morpion.

Returns:
 | list[list[str]]: La matrice du morpion.

"""

```

GREEN = "\033[92m" # Code ANSI pour la couleur verte
RESET = "\033[0m" # Réinitialisation des couleurs
BLUE = "\033[94m" # Code ANSI pour la couleur bleue
RED = "\033[91m" # Code ANSI pour la couleur rouge
```

```

# Afficher les indices des colonnes (1 à len(mat))
print("    ", end="")
for col in range(1, len(mat) + 1):
    print(f" {GREEN}{col}{RESET} ", end="")
    if col < len(mat):
        print(" ", end="")
print("\n", end="    " + "---" * len(mat) + "\n")
```

```

# Afficher la matrice avec les indices des lignes (1 à len(mat))
for j in range(len(mat)):
    print(f" {GREEN}{j + 1}{RESET} " # Indice de ligne
    for k in range(len(mat)):
        if mat[j][k] == "X":
            print(f" {RED}{mat[j][k]}{RESET} ", end="")
        elif mat[j][k] == "O":
            print(f" {BLUE}{mat[j][k]}{RESET} ", end="")
        else:
            print(f" {mat[j][k]} ", end="")
        if k < len(mat) - 1:
            print("|", end="")
    if j < len(mat) - 1:
        print("\n", end="    " + "---|" * (len(mat) - 1) + "---\n")
print()
print()
return mat
```

```

def morpion_plein(mat : list[list[str]]) -> bool:
    """
    Fonction pour vérifier si le morpion est plein ou non

    Args:
        mat (list[list[str]]): La matrice du morpion.

    Returns:
        bool : True si le morpion est plein, False sinon
    """

    est_plein : bool
    est_plein = True
    for i in range(len(mat)) :
        for j in range(len(mat)) :
            if mat[i][j] == " " :
                est_plein = False
    return est_plein

#####
#####
```

```

def deja_pris(mat : list[list[str]], l : int, c : int) -> bool :
    """
    Fonction pour vérifier si une case est déjà prise

    Args:
        mat (list[list[str]]): La matrice du morpion.
        l (int) : La ligne
        c (int) : La colonne
    Returns:
        bool : True si la case est déjà prise, False sinon
    """

    cond : bool
    cond = False
    if mat[l][c] == "X" or mat[l][c] == "O" :
        cond = True
    return cond

#####
#####
```

```

def morpionjouer() -> None :
    cond_vic1 : bool
    cond_vic2 : bool
    cond_vic1 = False
    cond_vic2 = False
    compteur : int
    compt_j : int
    n_ligne_col : int
    n_ligne_col = 0
    compteur = 0
    compt_j = 1
    j : int
    listej : list[str]
    listej = []

    #Proposition réinitialisation des scores
    reinitialiser_scores_binaire("morpion")

    # Liste des joueurs
    listej = listejoueur("morpion")

    # Vérification si la liste contient bien deux joueurs
    if len(listej) < 2:
        print("Erreur : il doit y avoir exactement 2 joueurs.")
        return

    #Effacement de la console avec un sleep pour laisser le temps de lire
    sleep(3)
    effacer_console()
    nombrelignehorizontale(1, 55)
    print("\033[92m Lancement du jeu du morpion \033[0m")
    nombrelignehorizontale(1, 55)

    # Assigner les joueurs aleatoirement
    j = randint(1, 2)

if j == 1 :
    joueur1 = listej[0]
    joueur2 = listej[1]
else :
    joueur1 = listej[1]
    joueur2 = listej[0]

#Affichage des joueurs
print()
print(f"{joueur1} vous jouerez : X ")
print(f"{joueur2} vous jouerez : O ")
print()

#Choix de la taille du morpion
mat = taille_morpion()
afficher_morpion(mat)
nombrelignehorizontale(1, 55)

#Début du jeu
#Tant qu'il n'y a aucune condition de victoire ou que le morpion n'est pas plein
while ((cond_vic1 == False) and (cond_vic2 == False) and (not(morpion_plein(mat)))) :
    #le joueur 1 joue
    if compt_j == 1 :
        jeu_morpion1(mat, joueur1)
        afficher_morpion(mat)

```

```

#Vérifie si une colonne est remplie
while n_ligne_col < len(mat) :
    for i in range(len(mat)) :
        if mat[i][n_ligne_col] == 'X' :
            #Incrémente le compteur servant à vérifier si la colonne est remplie complètement
            compteur = compteur + 1
            if compteur == len(mat) :
                cond_vic1 = True
        #Incrémente le numéro de la colonne
        n_ligne_col = n_ligne_col + 1
    #Réinitialise le compteur pour la prochaine colonne
    compteur = 0
#Réinitialise la variable pour la prochaine vérification
n_ligne_col = 0

```

```

#Vérifie si une ligne est remplie
while n_ligne_col < len(mat) :
    for i in range(len(mat)) :
        if mat[n_ligne_col][i] == 'X' :
            #Incrémente le compteur servant à vérifier si la ligne est remplie complètement
            compteur = compteur + 1
            if compteur == len(mat) :
                cond_vic1 = True
        #Incrémente le numéro de la ligne
        n_ligne_col = n_ligne_col + 1
    #Réinitialise le compteur pour la prochaine ligne
    compteur = 0
#Réinitialise la variable pour la prochaine vérification
n_ligne_col = 0

```

```

#Vérifie si la diagonale principale est remplie
while n_ligne_col < len(mat) :
    for i in range(len(mat)) :
        if mat[i][i] == 'X' :
            #Incrémente le compteur servant à vérifier si la diagonale principale est remplie complètement
            compteur = compteur + 1
            if compteur == len(mat) :
                cond_vic1 = True
        n_ligne_col = n_ligne_col + 1
    #Réinitialise le compteur pour la prochaine diagonale principale
    compteur = 0
#Réinitialise la variable pour la prochaine vérification
n_ligne_col = 0

#Vérifie si la diagonale opposée est remplie
while n_ligne_col < len(mat) :
    for i in range(len(mat)) :
        if mat[i][-i-1] == 'X' :
            #Incrémente le compteur servant à vérifier si la diagonale opposée est remplie complètement
            compteur = compteur + 1
            if compteur == len(mat) :
                cond_vic1 = True
        n_ligne_col = n_ligne_col + 1
    #Réinitialise le compteur pour la prochaine diagonale opposée
    compteur = 0
#Réinitialise la variable pour la prochaine vérification (qui sera pour le joueur 2)
n_ligne_col = 0

#Changement de joueur
compt_j = 2

```

```

else :
    #le joueur 2 joue
    jeu_morpion2(mat, joueur2)
    afficher_morpion(mat)

    #Vérifie si une colonne est remplie
    while n_ligne_col < len(mat) :
        for i in range(len(mat)) :
            if mat[i][n_ligne_col] == '0' :
                #Incrémente le compteur servant à vérifier si la colonne est remplie complètement
                compteur = compteur + 1
                if compteur == len(mat) :
                    cond_vic2 = True
            #Incrémente le numéro de la colonne
            n_ligne_col = n_ligne_col + 1
            #Réinitialise le compteur pour la prochaine colonne
            compteur = 0
    #Réinitialise la variable pour la prochaine vérification
    n_ligne_col = 0

    #Vérifie si une ligne est remplie
    while n_ligne_col < len(mat) :
        for i in range(len(mat)) :
            if mat[n_ligne_col][i] == '0' :
                #Incrémente le compteur servant à vérifier si la ligne est remplie complètement
                compteur = compteur + 1
                if compteur == len(mat) :
                    cond_vic2 = True
            #Incrémente le numéro de la ligne
            n_ligne_col = n_ligne_col + 1
            #Réinitialise le compteur pour la prochaine ligne
            compteur = 0
    #Réinitialise la variable pour la prochaine vérification
    n_ligne_col = 0

    #Vérifie si la diagonale principale est remplie
    while n_ligne_col < len(mat) :
        for i in range(len(mat)) :
            if mat[i][i] == '0' :
                #Incrémente le compteur servant à vérifier si la diagonale principale est remplie complètement
                compteur = compteur + 1
                if compteur == len(mat) :
                    cond_vic2 = True
            n_ligne_col = n_ligne_col + 1
            #Réinitialise le compteur pour la prochaine diagonale principale
            compteur = 0
    #Réinitialise la variable pour la prochaine vérification
    n_ligne_col = 0

    #Vérifie si la diagonale opposée est remplie
    while n_ligne_col < len(mat) :
        for i in range(len(mat)) :
            if mat[i][-i-1] == '0' :
                #Incrémente le compteur servant à vérifier si la diagonale opposée est remplie complètement
                compteur = compteur + 1
                if compteur == len(mat) :
                    cond_vic2 = True
            n_ligne_col = n_ligne_col + 1
            #Réinitialise le compteur pour la prochaine diagonale opposée
            compteur = 0
    #Réinitialise la variable pour la prochaine vérification (qui sera pour le joueur 1)
    n_ligne_col = 0

    #Changement de joueur
    compt_j = 1

#Test des valeurs booléennes pour un possible gagnant
if cond_vic1 :
    print(f"{joueur1} a gagné")
    enregistrer_score_binaire("morpion", joueur1, 1)

elif cond_vic2 :
    print(f"{joueur2} a gagné")
    enregistrer_score_binaire("morpion", joueur2, 1)

#Si le morpion est plein et qu'il n'y a pas de gagnant
else :
    print("Match nul")

#Affichage de fin
afficher_scores_final("morpion")
quitterjeux("morpion")

```

Explication du code :

Fonction : taille_morpion()

```
def taille_morpion() -> list[list[str]] :
    """
    Fonction pour choisir la taille du morpion
    Args:
    | None : None
    Returns:
    | list[list[str]]: La matrice du morpion.
    """

    mat : list[list[str]]
    i : int
    j: int
    val : str
    n : int
    mat = list([[]])
    n = int(inputCustom("De quelle taille doit être votre morpion sachant qu'il sera de dimension n*n : ", int, "La valeur"))
    print()
    for i in range(0, n) :
        i = i
        ligne : list[str]
        ligne = list()
        for j in range(0, n) :
            val = " "
            ligne.append(val)
        mat.append(ligne)
    for j in range(len(mat)) :
        print(" ", end="")
        for k in range(len(mat)) :
            print(mat[j][k], end=" ")
            if k < len(mat)-1 :
                print("|", end=" ")
            if j < len(mat)-1 :
                print("\n", end="----" * len(mat) + "\n")
        print()
        print()
    nombrelignehorizontale(1, 55)
    print("Voici le morpion sur lequel vous allez jouer.")
    print()
    return mat
```

Cette fonction initialise un plateau de jeu pour le morpion avec une taille définie par l'utilisateur. Le plateau est représenté comme une matrice (liste de listes) remplie d'espaces vides (" ").

Création de la matrice vide :

- Une boucle imbriquée crée une liste vide pour chaque ligne, remplie d'espaces " ".
- Chaque ligne est ajoutée à la matrice.

Affichage du morpion :

- Le plateau est imprimé sous forme de grille, avec des séparateurs | pour les colonnes et ---- pour les lignes.

Retour de la matrice :

- La fonction retourne le plateau de jeu pour l'utiliser dans d'autres fonctions.

Fonction : jeu_morpion1(mat, joueur) retourne mat

```
def jeu_morpion1(mat : list[list[str]], joueur : str) -> list[list[str]] :
    """
    Fonction pour jouer au jeu du morpion
    Args:
        mat (list[list[str]]): La matrice du morpion.
        joueur (str): Le nom du joueur.
    Returns:
        list[list[str]]: La matrice du morpion.
    """

    morpion_ligne_x : int
    morpion_colonne_x : int
    print(f"{joueur}, sélectionner la case dans laquelle vous voulez jouez (X) : ")
    morpion_ligne_x = int(inputCustom("Choisir ligne : ", int, "La valeur doit être un entier", 1, len(mat)))
    while morpion_ligne_x > len(mat) :
        morpion_ligne_x = int(inputCustom("Sélectionner la ligne dans l'intervalle de l'aire de jeu : ", int, "La valeur doit être un entier", 1, len(mat)))
    morpion_colonne_x = int(inputCustom("Choisir colonne : ", int, "La valeur doit être un entier", 1, len(mat)))
    while morpion_colonne_x > len(mat) :
        morpion_colonne_x = int(inputCustom("Sélectionner la colonne dans l'intervalle de l'aire de jeu : ", int, "La valeur doit être un entier", 1, len(mat)))
    morpion_colonne_x -= 1
    morpion_ligne_x -= 1
    if not(deja_pris(mat, morpion_ligne_x, morpion_colonne_x)) :
        mat[morpion_ligne_x][morpion_colonne_x] = "X"
    else:
        print()
        print("Case déjà prise, sélectionnez en une autre")
        jeu_morpion1(mat, joueur)
    print()
    nombrelignehorizontale(1, 55)
    print()
    return mat
```

Permet à un joueur de jouer son tour en plaçant un "X" sur le plateau.

Saisie des coordonnées :

- L'utilisateur entre les indices de ligne et colonne où il souhaite jouer.

Vérification des coordonnées :

- Les indices doivent être dans les limites du plateau.
- On enlève 1 au choix car l'utilisateur rentre la ligne ou la colonne et non l'indice de la ligne ou de la colonne

Placement du symbole "X" :

- La fonction `deja_pris()` vérifie si la case est occupée.
- Si oui, on relance le tour pour forcer un choix valide.

Fonction : jeu_morpion2(mat, joueur) retourne mat

```
def jeu_morpion2(mat : list[list[str]], joueur : str) -> list[list[str]] :
    """
    Fonction pour jouer au jeu du morpion
    Args:
        mat (list[list[str]]): La matrice du morpion.
        joueur (str): Le nom du joueur.
    Returns:
        list[list[str]]: La matrice du morpion.
    """

    morpion_ligne_o : int
    morpion_colonne_o : int
    print(f"{joueur}, sélectionner la case dans laquelle vous voulez jouez (0) :")
    morpion_ligne_o = int(inputCustom("Choisir ligne : ", int, "La valeur doit être un entier", 1, len(mat)))
    while morpion_ligne_o > len(mat) :
        morpion_ligne_o = int(inputCustom("Sélectionner la ligne dans l'intervalle de l'aire de jeu : ", int, "La valeur doit être un entier", 1, len(mat)))
    morpion_colonne_o = int(inputCustom("Choisir colonne : ", int, "La valeur doit être un entier", 1, len(mat)))
    while morpion_colonne_o > len(mat) :
        morpion_colonne_o = int(inputCustom("Sélectionner la colonne dans l'intervalle de l'aire de jeu : ", int, "La valeur doit être un entier", 1, len(mat)))
    morpion_colonne_o -= 1
    morpion_ligne_o -= 1
    if not(deja_pris(mat, morpion_ligne_o, morpion_colonne_o)) :
        mat[morpion_ligne_o][morpion_colonne_o] = "O"
    else:
        print()
        print("Case déjà prise, sélectionnez en une autre")
        jeu_morpion2(mat, joueur)
    print()
    nombrelignehorizontale(1, 55)
    print()
    return mat
```

Elle est en tout point identique à la fonction précédente à la différence qu'elle ne se déclenche que lorsque c'est le joueur 2 qui joue et elle remplace les emplacements vides par de "O".

Fonction : afficher_morpion(mat) retourne mat

```
def afficher_morpion(mat: list[list[str]]) -> list[list[str]]:  
    Fonction pour afficher le morpion avec numérotation des lignes et colonnes (à partir de 1).  
  
    Args:  
        mat (list[list[str]]): La matrice du morpion.  
  
    Returns:  
        list[list[str]]: La matrice du morpion.  
    """  
    GREEN = "\033[92m" # Code ANSI pour la couleur verte  
    RESET = "\033[0m" # Réinitialisation des couleurs  
    BLUE = "\033[94m" # Code ANSI pour la couleur bleue  
    RED = "\033[91m" # Code ANSI pour la couleur rouge  
  
    # Afficher les indices des colonnes (1 à len(mat))  
    print(" ", end="")  
    for col in range(1, len(mat) + 1):  
        print(f" {GREEN}{col}{RESET} ", end="")  
        if col < len(mat):  
            print(" ", end="")  
    print("\n", end=" " + "--- " * len(mat) + "\n")  
  
    # Afficher la matrice avec les indices des lignes (1 à len(mat))  
    for j in range(len(mat)):  
        print(f" {GREEN}{j + 1}{RESET} ", end="") # Indice de ligne  
        for k in range(len(mat)):  
            if mat[j][k] == "X":  
                print(f" {RED}{mat[j][k]}{RESET} ", end="")  
            elif mat[j][k] == "O":  
                print(f" {BLUE}{mat[j][k]}{RESET} ", end="")  
            else:  
                print(f" {mat[j][k]} ", end="")  
            if k < len(mat) - 1:  
                print("|", end="")  
        if j < len(mat) - 1:  
            print("\n", end=" " + "---|" * (len(mat) - 1) + "---\n")  
    print()  
    print()  
    return mat
```

Affiche le plateau sous forme de grille à tout moment du jeu.

Elle fonctionne de la même manière que pour la création du plateau dans la fonction taille_morpion().

Fonction : morpion_ plein(mat) retourne un booléen

```
def morpion_ plein(mat : list[list[str]]) -> bool:  
    """  
        Fonction pour vérifier si le morpion est plein ou non  
  
    Args:  
        mat (list[list[str]]): La matrice du morpion.  
  
    Returns:  
        bool : True si le morpion est plein, False sinon  
    """  
  
    est_ plein : bool  
    est_ plein = True  
    while est_ plein :  
        for i in range(len(mat)) :  
            for j in range(len(mat)) :  
                if mat[i][j] == " " :  
                    est_ plein = False  
    return est_ plein
```

Vérifie si le plateau est plein (plus de cases vides).

Initialisation :

- Suppose que le plateau est plein.

Vérification :

- Parcourt toutes les cases. Si une case est vide, est_ plein prend la valeur False et indiquera à la fonction appelante que le plateau est complet.

Fonction : deja_ pris(mat, l, c) retourne un booléen

```

def déjà_pris(mat : list[list[str]], l : int, c : int) -> bool :
    """
    Fonction pour vérifier si une case est déjà prise

    Args:
        mat (list[list[str]]): La matrice du morpion.
        l (int) : La ligne
        c (int) : La colonne
    Returns:
        bool : True si la case est déjà prise, False sinon
    """

    cond : bool
    cond = False
    if mat[l][c] == "X" or mat[l][c] == "O" :
        cond = True
    return cond

```

Vérifie si la case sélectionnée par le joueur est déjà occupée.

Vérification :

- Retourne True si la case contient "X" ou "O", ce qui entraîne un réamorçage du tour de jeu (le joueur rejoue son coup).
 - Sinon, la fonction renvoie False indiquant que la case est vide et qu'il est possible d'y placer un symbole

Fonction jeu_morpion() :

```

def morpionjouer() -> None :
    cond_vic1 : bool
    cond_vic2 : bool
    cond_vic1 = False
    cond_vic2 = False
    compteur : int
    compt_j : int
    n_ligne_col : int
    n_ligne_col = 0
    compteur = 0
    compt_j = 1
    j : int
    listej : list[str]
    listej = []

    # Proposition réinitialisation des scores
    reinitialiser_scores_binaire("morpion")

    # Liste des joueurs
    listej = listejoueur("morpion")

    # Vérification si la liste contient bien deux joueurs
    if len(listej) < 2:
        print("Erreur : il doit y avoir exactement 2 joueurs.")
        return

    # Effacement de la console avec un sleep pour laisser le temps de lire
    sleep(3)
    effacer_console()
    nombrelignehorizontale(1, 55)
    print("\033[92m Lancement du jeu du morpion \033[0m")
    nombrelignehorizontale(1, 55)

    # Assigner les joueurs aléatoirement
    j = randint(1, 2)

```

```

if j == 1 :
    joueur1 = listej[0]
    joueur2 = listej[1]
else :
    joueur1 = listej[1]
    joueur2 = listej[0]

#Affichage des joueurs
print()
print(f"{joueur1} vous jouerez : X ")
print(f"{joueur2} vous jouerez : O ")
print()

#Choix de la taille du morpion
mat = taille_morpion()
afficher_morpion(mat)
nombrelignehorizontale(1, 55)

#Début du jeu
#Tant qu'il n'y a aucune condition de victoire ou que le morpion n'est pas plein
while ((cond_vic1 == False) and (cond_vic2 == False) and (not(morpion_ plein(mat))) :
    #le joueur 1 joue
    if compt_j == 1 :
        jeu_morpion1(mat, joueur1)
        afficher_morpion(mat)

```

```

#Vérifie si une colonne est remplie
while n_ligne_col < len(mat) :
    for i in range(len(mat)) :
        if mat[i][n_ligne_col] == 'X' :
            #Incrémente le compteur servant à vérifier si la colonne est remplie complètement
            compteur = compteur + 1
            if compteur == len(mat) :
                cond_vic1 = True
    #Incrémente le numéro de la colonne
    n_ligne_col = n_ligne_col + 1
    #Réinitialise le compteur pour la prochaine colonne
    compteur = 0
#Réinitialise la variable pour la prochaine vérification
n_ligne_col = 0

#Vérifie si une ligne est remplie
while n_ligne_col < len(mat) :
    for i in range(len(mat)) :
        if mat[n_ligne_col][i] == 'X' :
            #Incrémente le compteur servant à vérifier si la ligne est remplie complètement
            compteur = compteur + 1
            if compteur == len(mat) :
                cond_vic1 = True
    #Incrémente le numéro de la ligne
    n_ligne_col = n_ligne_col + 1
    #Réinitialise le compteur pour la prochaine ligne
    compteur = 0
#Réinitialise la variable pour la prochaine vérification
n_ligne_col = 0

```

```

#Vérifie si la diagonale principale est remplie
while n_ligne_col < len(mat) :
    for i in range(len(mat)) :
        if mat[i][i] == 'X' :
            #Incrémente le compteur servant à vérifier si la diagonale principale est remplie complètement
            compteur = compteur + 1
            if compteur == len(mat) :
                cond_vic1 = True
    n_ligne_col = n_ligne_col + 1
    #Réinitialise le compteur pour la prochaine diagonale principale
    compteur = 0
#Réinitialise la variable pour la prochaine vérification
n_ligne_col = 0

#Vérifie si la diagonale opposée est remplie
while n_ligne_col < len(mat) :
    for i in range(len(mat)) :
        if mat[i][-i-1] == 'X' :
            #Incrémente le compteur servant à vérifier si la diagonale opposée est remplie complètement
            compteur = compteur + 1
            if compteur == len(mat) :
                cond_vic1 = True
    n_ligne_col = n_ligne_col + 1
    #Réinitialise le compteur pour la prochaine diagonale opposée
    compteur = 0
#Réinitialise la variable pour la prochaine vérification(qui sera pour le joueur 2)
n_ligne_col = 0

#Changement de joueur
compt_j = 2

else :
    #le joueur 2 joue
    jeu_morpion2(mat, joueur2)
    afficher_morpion(mat)

#Vérifie si une colonne est remplie
while n_ligne_col < len(mat) :
    for i in range(len(mat)) :
        if mat[i][n_ligne_col] == 'O' :
            #Incrémente le compteur servant à vérifier si la colonne est remplie complètement
            compteur = compteur + 1
            if compteur == len(mat) :
                cond_vic2 = True
    #Incrémente le numéro de la colonne
    n_ligne_col = n_ligne_col + 1
    #Réinitialise le compteur pour la prochaine colonne
    compteur = 0
#Réinitialise la variable pour la prochaine vérification
n_ligne_col = 0

#Vérifie si une ligne est remplie
while n_ligne_col < len(mat) :
    for i in range(len(mat)) :
        if mat[n_ligne_col][i] == 'O' :
            #Incrémente le compteur servant à vérifier si la ligne est remplie complètement
            compteur = compteur + 1
            if compteur == len(mat) :
                cond_vic2 = True
    #Incrémente le numéro de la ligne
    n_ligne_col = n_ligne_col + 1
    #Réinitialise le compteur pour la prochaine ligne
    compteur = 0
#Réinitialise la variable pour la prochaine vérification
n_ligne_col = 0

```

```

#Vérifie si la diagonale principale est remplie
while n_ligne_col < len(mat) :
    for i in range(len(mat)) :
        if mat[i][i] == '0' :
            #Incrémente le compteur servant à vérifier si la diagonale principale est remplie complètement
            compteur = compteur + 1
            if compteur == len(mat) :
                cond_vic2 = True
    n_ligne_col = n_ligne_col + 1
    #Réinitialise le compteur pour la prochaine diagonale principale
    compteur = 0
#Réinitialise la variable pour la prochaine vérification
n_ligne_col = 0

#Vérifie si la diagonale opposée est remplie
while n_ligne_col < len(mat) :
    for i in range(len(mat)) :
        if mat[i][-i-1] == '0' :
            #Incrémente le compteur servant à vérifier si la diagonale opposée est remplie complètement
            compteur = compteur + 1
            if compteur == len(mat) :
                cond_vic2 = True
    n_ligne_col = n_ligne_col + 1
    #Réinitialise le compteur pour la prochaine diagonale opposée
    compteur = 0
#Réinitialise la variable pour la prochaine vérification(qui sera pour le joueur 1)
n_ligne_col = 0

#Changement de joueur
compt_j = 1

```

```

#Test des valeurs booléennes pour un possible gagnant
if cond_vic1 :
    print(f"{joueur1} a gagné")
    enregistrer_score_binaire("morpion", joueur1, 1)

elif cond_vic2 :
    print(f"{joueur2} a gagné")
    enregistrer_score_binaire("morpion", joueur2, 1)

#Si le morpion est plein et qu'il n'y a pas de gagnant
else :
    print("Match nul")

#Affichage de fin
afficher_scores_final("morpion")
quitterjeux("morpion")

```

Jeux d'essais :

Jules vous jouerez : X
Alex vous jouerez : O

De quelle taille doit être votre morpion sachant qu'il sera de dimension n*n : 3

Voici le morpion sur lequel vous allez jouer.

1	2	3
1		
2		
3		

Affichage du morpion de la taille incrémenter par les joueurs et tirage aléatoire du signe et par conséquent de l'ordre de jeu.

```
De quelle taille doit être votre morpion sachant qu'il sera de dimension n*n : 6
Voici le morpion sur lequel vous allez jouer.
  1  2  3  4  5  6
  -----|-----|-----|-----|-----|-----|
1 |       |       |       |       |       |
2 |       |       |       |       |       |
3 |       |       |       |       |       |
4 |       |       |       |       |       |
5 |       |       |       |       |       |
6 |       |       |       |       |       |
  -----|-----|-----|-----|-----|
```

Affichage d'un morpion de 6*6 de taille

```
De quelle taille doit être votre morpion sachant qu'il sera de dimension n*n : 12
La valeur doit être comprise entre 3 et 10.
De quelle taille doit être votre morpion sachant qu'il sera de dimension n*n : 10
```

Limite de taille de création d'une grille de morpion

```
Jules, sélectionner la case dans laquelle vous voulez jouer (X) :
Choisir ligne :
La valeur doit être un entier
Choisir ligne : 4
La valeur doit être comprise entre 1 et 3.
Choisir ligne : []
```

Demande et vérification des entrées de valeur des joueurs.

```
Choisir ligne : 1
Choisir colonne : 2
```

```
  1  2  3
  -----|-----|-----|
1 |   X  |       |
2 |       |       |
3 |       |       |
```

Affichage de la grille avec le signe rajouter.

```
Alex, sélectionner la case dans laquelle vous voulez jouez (0) :  
Choisir ligne : []
```

```
Alex, sélectionner la case dans laquelle vous voulez jouez (0) :  
Choisir ligne : 1  
Choisir colonne : 2
```

Case déjà prise, sélectionnez en une autre

```
Alex, sélectionner la case dans laquelle vous voulez jouez (0) :  
Choisir ligne : []
```

Même principe pour le second joueur

1	2	3
1	0	X
2		
3		

```
Jules, sélectionner la case dans laquelle vous voulez jouez (X) :  
Choisir ligne : []
```

Incrémantation du deuxième signe dans la grille.

Cas de victoires :

1	2	3
1	X	X
2	0	0
3		

Alex a gagné

Score de 1 ajouté pour le joueur Alex dans le fichier ./score/morpion.bin.

--- Scores finaux ---

Alex: 1 points

Jules: 0 points

Le(s) gagnant(s) pour le jeu morpion est/sont Alex avec 1 points!

--- Fin des scores ---

```
|-----|  
 1   2   3  
-----  
1 X | 0 |  
-----|-----|  
2 0 | 0 |  
-----|-----|  
3 X | X | X  
  
Jules a gagné  
Score de 1 ajouté pour le joueur Jules dans le fichier ./score/morpion.bin.  
  
--- Scores finaux ---  
Alex: 0 points  
Jules: 1 points  
  
Le(s) gagnant(s) pour le jeu morpion est/sont Jules avec 1 points!  
--- Fin des scores ---
```

```
|-----|  
 1   2   3  
-----  
1 0 | 0 |  
-----|-----|  
2 X | X | X  
-----|-----|  
3   |   |  
  
Alex a gagné  
Score de 1 ajouté pour le joueur Alex dans le fichier ./score/morpion.bin.  
  
--- Scores finaux ---  
Alex: 1 points  
Jules: 0 points  
  
Le(s) gagnant(s) pour le jeu morpion est/sont Alex avec 1 points!  
--- Fin des scores ---
```

```
|-----|  
| 1   2   3 |  
|--- --- ---|  
| 1 X | 0 | |
|---|---|---|
| 2 X | 0 |  
|---|---|---|  
| 3 X | |
```

Alex a gagné
Score de 1 ajouté pour le joueur Alex dans le fichier ./score/morpion.bin.

--- Scores finaux ---

Alex: 1 points
Jules: 0 points

Le(s) gagnant(s) pour le jeu morpion est/sont **Alex** avec 1 points!
--- Fin des scores ---

```
|-----|  
| 1   2   3 |  
|--- --- ---|  
| 1 0 | X | |
|---|---|---|
| 2 0 | X |  
|---|---|---|  
| 3   | X |
```

Jules a gagné
Score de 1 ajouté pour le joueur Jules dans le fichier ./score/morpion.bin.

--- Scores finaux ---

Alex: 0 points
Jules: 1 points

Le(s) gagnant(s) pour le jeu morpion est/sont **Jules** avec 1 points!
--- Fin des scores ---

Voulez-vous vraiment quitter ?
0 : Oui / 1 : Non -->

```
|-----|  
1 2 3  
--- --- ---  
1 0 |   | X  
---|---|---  
2 0 | 0 | X  
---|---|---  
3   | X | X
```

Jules a gagné
Score de 1 ajouté pour le joueur Jules dans le fichier ./score/morpion.bin.

--- Scores finaux ---
Alex: 0 points
Jules: 1 points

Le(s) gagnant(s) pour le jeu morpion est/sont Jules avec 1 points!

--- Fin des scores ---

```
|-----|  
1 2 3  
--- --- ---  
1 X | 0 |  
---|---|---  
2 0 | X |  
---|---|---  
3   |   | X
```

Alex a gagné
Score de 1 ajouté pour le joueur Alex dans le fichier ./score/morpion.bin.

--- Scores finaux ---
Alex: 1 points
Jules: 0 points

Le(s) gagnant(s) pour le jeu morpion est/sont Alex avec 1 points!

--- Fin des scores ---

```

  1  2  3
  - - -
1 | 0 | 0 | X
  - - - | -
2 |   | X |
  - - - | -
3 | X |   |

Alex a gagné
Score de 1 ajouté pour le joueur Alex dans le fichier ./score/morpion.bin.

--- Scores finaux ---
Alex: 1 points
Jules: 0 points

Le(s) gagnant(s) pour le jeu morpion est/sont Alex avec 1 points!
--- Fin des scores ---

```

L'ensemble des possibilités pour X (et donc pour O car les codes sont identiques) sont fonctionnels

```

  1  2  3
  - - -
1 | X | X | X
  - - - | -
2 | X | 0 | 0
  - - - | -
3 | X | 0 | 0

Jules a gagné
Score de 1 ajouté pour le joueur Jules dans le fichier ./score/morpion.bin.

--- Scores finaux ---
Alex: 0 points
Jules: 1 points

Le(s) gagnant(s) pour le jeu morpion est/sont Jules avec 1 points!
--- Fin des scores ---

```

Aucun problème avec un double morpion, le jeu n'attribue bien qu'un seul point au joueur gagnant.

Le cas du match nul.

```
|-----|  
| 1   2   3 |  
|-----|  
| 1   0 | X | 0 |  
|-----|  
| 2   0 | X | X |  
|-----|  
| 3   X | 0 | X |  
  
Match nul  
  
--- Scores finaux ---  
Alex: 0 points  
Jules: 0 points  
  
Le(s) gagnant(s) pour le jeu morpion est/sont Alex, Jules avec 0 points!  
--- Fin des scores ---
```

```
|-----|  
| 1   2   3   4   5   6 |  
|-----|  
| 1   X | X | X | X | X | X |  
|-----|  
| 2   0 | 0 | 0 | 0 | 0 | |
|---|---|---|---|---|---|
| 3   | | | | | |  
|-----|  
| 4   | | | | | |  
|-----|  
| 5   | | | | | |  
|-----|  
| 6   | | | | | |  
  
* a gagné  
Score de 1 ajouté pour le joueur * dans le fichier ./score/morpion.bin.  
  
--- Scores finaux ---  
*: 1 points  
N: 0 points  
  
Le(s) gagnant(s) pour le jeu morpion est/sont * avec 1 points!  
--- Fin des scores ---
```

Cas Bis où pour une grille de plus grande taille.

IX : Jeu 4 : Puissance 4

Sa fonction principale :

```
def plateau(grille: list[list[str]]) -> None:
    """
    Affiche le plateau de jeu avec des lignes et colonnes numérotées.
    Chaque case est reliée à une grille.
    :param grille: Liste de listes représentant le plateau de jeu
    """
    effacer_console()
    print("    1  2  3  4  5  6  7") # En-tête des colonnes
    print(" " + " " + " " * 7) # Ligne supérieure

    for i, ligne in enumerate(grille): # Parcourt chaque ligne de la grille
        print(f"{i + 1} | " + "|".join(f" {case} " for case in ligne) + "|") # Affiche les cases
        print(" " + "---+" * 7) # Ligne de séparation
```

```
def verif_plein (grille: list[list[str]]) -> bool:
    """
    Vérifie si la grille est pleine.
    Args:
        grille (list[list[str]]): Grille de jeu.
    Returns:
        bool : True si la grille est pleine, False sinon.
    """

    for ligne in grille: # Parcourt chaque ligne de la grille
        if " " in ligne: # S'il reste une case vide
            return False # La grille n'est pas pleine
    return True # La grille est pleine
```

```
def verif_victoire(grille: list[list[str]], joueur: str) -> bool:
    """
    Vérifie si un joueur a gagné.
    Args:
        grille (list[list[str]]): Grille de jeu.
        joueur (str): Joueur actuel.
    Returns:
        bool : True si un joueur a gagné, False sinon.
    """

    for i in range(6): # Parcourt les lignes
        for j in range(7): # Parcourt les colonnes
            if grille[i][j] == joueur: # Si la case contient le jeton du joueur
                if j + 3 < 7 and grille[i][j + 1] == grille[i][j + 2] == grille[i][j + 3] == joueur: # Vérifie l'horizontale
                    return True
                if i + 3 < 6:
                    if grille[i + 1][j] == grille[i + 2][j] == grille[i + 3][j] == joueur: # Vérifie la verticale
                        return True
                    if j + 3 < 7 and grille[i + 1][j + 1] == grille[i + 2][j + 2] == grille[i + 3][j + 3] == joueur: # Vérifie la diagonale droite
                        return True
                    if j - 3 >= 0 and grille[i + 1][j - 1] == grille[i + 2][j - 2] == grille[i + 3][j - 3] == joueur: # Vérifie la diagonale gauche
                        return True
    return False
```

```

def colonne_pleine(grille: list[list[str]], colonne: int) -> bool:
    """
    Vérifie si une colonne est pleine.
    Args:
        grille (list[list[str]]): Grille de jeu.
        colonne (int): Colonne choisie par le joueur.
    Returns:
        bool : True si la colonne est pleine, False sinon.
    """
    if " " in grille[0][colonne]:
        return False
    else:
        return True

```

```

def occurrencejouer(grille: list[list[str]], joueur: str, colonne: int) -> None:
    """
    Joue un coup.
    Args:
        grille (list[list[str]]): Grille de jeu.
        joueur (str): Joueur actuel.
        colonne (int): Colonne choisie par le joueur.
    Returns:
        None : None
    """
    case_placee = False # Variable pour savoir si un jeton a été placé

    for i in range(5, -1, -1): # Parcourt les lignes de la colonne choisie
        if grille[i][colonne] == " " and not case_placee: # Si la case est vide et qu'un jeton n'a pas encore été placé
            grille[i][colonne] = joueur # Place le jeton du joueur
            case_placee = True # On marque qu'un jeton a été placé

    # La boucle se termine sans le besoin de `break`, car `case_placee` gère l'arrêt du placement.

```

```

#Initialisation du jeu
def puissance4() -> None:
    """
    Jeu Puissance 4.

    Args:
        None : None

    Returns:
        None : None
    """

    grille = [[" " for _ in range(7)] for _ in range(6)] # Grille de jeu
    victoire = False # Indique si une victoire a eu lieu
    plein = False # Indique si la grille est pleine

    # Réinitialisation des scores
    reinitialiser_scores("puissance4")

    # Initialisation des joueurs
    listej = listejoueur("puissance4") # Récupère la liste des joueurs
    if len(listej) < 2:
        print("Erreur : il doit y avoir exactement 2 joueurs.")
        return

    # Attribution aléatoire des joueurs aux signes
    if randint(1, 2) == 1:
        joueur1 = listej[0]
        joueur2 = listej[1]
        signe1 = "\033[33m\033[0m" # Couleur jaune 🟨
        signe2 = "\033[31m\033[0m" # Couleur rouge 🟥
    else:
        joueur1 = listej[1]
        joueur2 = listej[0]
        signe1 = "\033[31m\033[0m" # Couleur rouge 🟥
        signe2 = "\033[33m\033[0m" # Couleur jaune 🟨

    print(f"{joueur1} jouera avec les {signe1} et {joueur2} jouera avec les {signe2}")
    sleep(3)
    effacer_console()

```

```

# Alternance des joueurs / Commence avec le joueur 1 sachant que l'attribution a été faite aléatoirement
joueur_actuel = joueur1
signe_actuel = signe1

# Boucle principale du jeu
while not victoire and not plein:
    plateau(grille) # Affiche la grille
    colonne = int(inputCustom(f"(joueur_actuel) ({signe_actuel}), choisissez une colonne entre 1 et 7 : ",int,"La valeur doit être un entier",1, 7)) - 1 # Demande au joueur de choisir une colonne

    # Vérifie si la case est prise
    while colonne_pleine(grille, colonne):
        print("La colonne est pleine. Veuillez en choisir une autre.")
        colonne = int(inputCustom(f"(joueur_actuel) ({signe_actuel}), choisissez une colonne entre 1 et 7 : ",int,"La valeur doit être un entier",1, 7)) - 1

    # Joue le coup
    occurencejoue(grille, signe_actuel, colonne)

    # Vérifie la victoire ou si la grille est pleine
    victoire = verif_victoire(grille, signe_actuel)
    plein = verif_plein(grille)

    # Affiche le résultat si le jeu est terminé
    if victoire:
        plateau(grille)
        print(f"Félicitations {(joueur_actuel)} ({signe_actuel}) ! Vous avez gagné ! 🎉")
        enregistrer_score("puissance4", joueur_actuel, 1)

    elif plein:
        plateau(grille)
        print("Match nul ! La grille est pleine.")

    # Change de joueur
    if joueur_actuel == joueur1:
        joueur_actuel, signe_actuel = joueur2, signe2
    else:
        joueur_actuel, signe_actuel = joueur1, signe1

```

```

joueur_actuel, signe_actuel = joueur1, signe1

#Affichage final
afficher_scores_final("puissance4")
quitterjeux("puissance4")

```

Explication du code :

Le jeu **Puissance 4** est un jeu classique où deux joueurs s'affrontent pour aligner quatre jetons de leur couleur dans une grille 6x7.

Fonctions principales

1. **Plateau de jeu (plateau) :**
 - Affiche la grille actuelle avec les numéros des lignes et des colonnes.
 - Chaque case montre le jeton du joueur ou est vide (" ").
 2. **Vérification si la grille est pleine (verif_plein) :**
 - Parcourt la grille pour vérifier s'il reste des cases vides.
 - Retourne **True** si toutes les cases sont occupées.
 3. **Vérification de victoire (verif_victoire) :**
 - Vérifie si un joueur a aligné 4 jetons horizontalement, verticalement ou en diagonale.
 - Retourne **True** si un joueur remplit l'une des conditions de victoire.
 4. **Vérification de colonne pleine (colonne_pleine) :**
 - Vérifie si une colonne choisie par un joueur est déjà remplie.
 5. **Placement d'un jeton (occurrencejouer) :**
 - Place un jeton dans la colonne choisie par un joueur en trouvant la case vide la plus basse.
-

Mécanique du jeu (puissance4)

1. **Initialisation du jeu :**
 - Une grille 6x7 est créée avec des cases vides (" ").
 - Les scores sont réinitialisés, et les joueurs sont attribués aléatoirement à des jetons colorés (● ou ○).
2. **Déroulement du jeu :**
 - Le jeu alterne entre les deux joueurs.
 - À chaque tour :
 - Le joueur choisit une colonne (validation incluse si pleine).
 - Le jeton est placé dans la colonne.
 - Le programme vérifie une victoire ou un match nul.

- Si un joueur gagne, le score est mis à jour et le jeu s'arrête.
3. **Gestion de la fin de partie :**
- En cas de victoire, le plateau affiche l'alignement gagnant.
 - Si la grille est pleine sans vainqueur, un message de match nul est affiché.
4. **Affichage des scores et sortie :**
- Une fois le jeu terminé, les scores des joueurs sont affichés.
 - Une option de quitter le jeu est proposée.

Jeux d'essais :

```

Lancement du jeu du puissance 4
Voulez-vous réinitialiser les scores pour le jeu puissance4 ?
0 : Oui / 1 : Non --> 1
Lancement du jeu...
Vous devez d'abord rentrer les noms des joueurs :
Saisissez le nom des deux joueurs :
Entrez le nom du joueur 1 : alex
Entrez le nom du joueur 2 : jules
Score de 0 ajouté pour le joueur Alex dans le fichier ./score/puissance4.bin.
Score de 0 ajouté pour le joueur Jules dans le fichier ./score/puissance4.bin.
Jules jouera avec les ■ et Alex jouera avec les □

```

Attribution des carrés de couleur, aléatoirement, à chaque joueur.

Vérification de tous les cas de victoires :

```

    1 2 3 4 5 6 7
+ + + + + + +
1 | | | | | | |
+-----+
2 | | | | | | |
+-----+
3 | ■ | | | | | |
+-----+
4 | ■ | ■ | | | | |
+-----+
5 | ■ | ■ | | | | |
+-----+
6 | ■ | ■ | | | | |
+-----+
Félicitations Jules (■) ! Vous avez gagné ! 🎉
Score de 1 ajouté pour le joueur Jules dans le fichier ./score/puissance4.bin.

--- Scores finaux ---
A: 0 points
D: 1 points
Jules: 1 points
Ale: 1 points
Alex: 0 points

Le(s) gagnant(s) pour le jeu puissance4 est/sont D, Jules, Ale avec 1 points!
--- Fin des scores ---

```

```

    1 2 3 4 5 6 7
+ + + + + + +
1 | | | | | | |
+-----+
2 | | | | | | |
+-----+
3 | | | | | | |
+-----+
4 | | | | | | |
+-----+
5 | ■ | ■ | ■ | | | |
+-----+
6 | ■ | ■ | ■ | ■ | |
+-----+
Félicitations Alex (■) ! Vous avez gagné ! 🎉
Score de 1 ajouté pour le joueur Alex dans le fichier ./score/puissance4.bin.

--- Scores finaux ---
A: 0 points
D: 1 points
Jules: 1 points
Ale: 1 points
Alex: 1 points

Le(s) gagnant(s) pour le jeu puissance4 est/sont D, Jules, Ale, Alex avec 1 points!
--- Fin des scores ---

```

```

    1   2   3   4   5   6   7
+ + + + + + +
1 | | | | | | |
+-----+
2 | | | | | | |
+-----+
3 | | | | | ■ | | |
+-----+
4 | | | | ■ | ■ | | |
+-----+
5 | | ■ | ■ | ■ | | |
+-----+
6 | ■ | ■ | ■ | ■ | ■ | ■ | ■ |
+-----+
Félicitations Jules (■) ! Vous avez gagné ! 🎉
Score de 1 ajouté pour le joueur Jules dans le fichier ./score/puissance4.bin.

--- Scores finaux ---
A: 0 points
D: 1 points
Jules: 2 points
Ale: 1 points
Alex: 1 points

Le(s) gagnant(s) pour le jeu puissance4 est/sont Jules avec 2 points!
--- Fin des scores ---

```

```

    1   2   3   4   5   6   7
+ + + + + + +
1 | | | | | | |
+-----+
2 | | | | | | |
+-----+
3 | | | | | ■ | |
+-----+
4 | | | | | ■ | ■ | ■ | |
+-----+
5 | | | | | ■ | ■ | ■ | |
+-----+
6 | | | | | ■ | ■ | ■ | ■ | ■ |
+-----+
Félicitations Alex (■) ! Vous avez gagné ! 🎉
Score de 1 ajouté pour le joueur Alex dans le fichier ./score/puissance4.bin.

--- Scores finaux ---
A: 0 points
D: 1 points
Jules: 2 points
Ale: 1 points
Alex: 2 points

Le(s) gagnant(s) pour le jeu puissance4 est/sont Jules, Alex avec 2 points!
--- Fin des scores ---

```

	1	2	3	4	5	6	7
1	+	+	+	+	+	+	+
2	■						
3	■						
4	■						
5	■						
6	■						

Jules (■), choisissez une colonne entre 1 et 7 : 1

La colonne est pleine. Veuillez en choisir une autre.
Jules (■), choisissez une colonne entre 1 et 7 :

Si une colonne est pleine, il n'est pas possible de plus la remplir

	1	2	3	4	5	6	7
1	+	+	+	+	+	+	+
2	■						
3	■						
4	■						
5	■						
6	■	■	■	■	■	■	■

Jules (■), choisissez une colonne entre 1 et 7 : 8
La valeur doit être comprise entre 1 et 7.
Jules (■), choisissez une colonne entre 1 et 7 :

Vérification de l'index du carré entré.

	1	2	3	4	5	6	7	
1	+	+	+	+	+	+	+	+
2	+	+	+	+	+	+	+	+
3	+	+	+	+	+	+	+	+
4	+	+	+	+	+	+	+	+
5	+	+	+	+	+	+	+	+
6	+	+	+	+	+	+	+	+

Match nul ! La grille est pleine.

--- Scores finaux ---

A: 0 points

D: 1 points

Jules: 3 points

Ale: 1 points

Alex: 2 points

Le(s) gagnant(s) pour le jeu puissance4 est/sont **Jules** avec 3 points!

--- Fin des scores ---

Cas match nul, aucun score ajouté.