

Compte rendu S.A.E R1.01

Sommaire :

1. Présentation du travail à réaliser et règles des différents jeux
2. Préparation et mise en commun des idées
3. Codage des mini-jeux
4. Codage du menu principal
5. Jeux de tests
6. Amélioration possibles

Présentation du travail à réaliser et règle des différents jeux :

1 - Présentation du travail à réaliser :

Dans cette S.A.E, nous devons créer plusieurs jeux et les intégrer à une application. Les jeux à concevoir étaient le Morpion, “Devinette”, et le jeu des Allumettes. Ces 3 jeux devaient être fonctionnels et intégrés dans un menu, et nous devons en plus créer un système pour compter les points.

2 - Les Règles :

Le Morpion : *Le morpion est un jeu de société qui se joue à deux joueurs sur une grille de 3 cases sur 3. Chaque joueur choisit un symbole (une croix ou un rond). Le but du jeu est d’aligner horizontalement, verticalement ou en diagonale 3 symboles identiques. Le premier joueur qui y parvient gagne la partie. Lors d’une partie de morpion, si aucun joueur n’arrive à aligner 3 symboles identiques et que le plateau est plein alors il y a égalité.*

Devinette : *Le joueur 1 pense à un nombre compris dans un intervalle (dans notre cas entre 0 et 100). Le joueur 2 doit alors deviner ce nombre en proposant des nombres. Pour chaque proposition, le joueur 1, connaissant le nombre va dire “plus haut” ou “plus bas” pour orienter le joueur 2 sur le nombre à trouver.*

Règles supplémentaires : Si le joueur 2 qui cherche le nombre propose plus de 5 fois un nombre sans trouver le nombre cherché alors le joueur 1 remporte la manche.

Allumettes : *On dispose d'un tas de 20 allumettes. Chaque joueur prend à tour de rôle soit 1, soit 2, ou 3 allumettes. Le joueur qui prend la dernière allumette a perdu. Pour gagner, il faut donc laisser la dernière allumette à son adversaire.*

Préparation et mise en commun des idées :

Lorsque nous avons débuté notre S.A.E, nous nous sommes concertés pour définir un cahier des charges pour éviter de nous éparpiller et réunir nos idées .

Pour commencer, on a défini comment nous voulions compter les points lors d'une victoire. Chacun a proposé un système différent mais nous avons dû trancher et nous avons choisi un système assez simple qui ajoute un point au joueur qui gagne une partie. Il n'y a pas de score globale, les scores sont comptés séparément (*ex : si joueur 1 gagne un point dans morpion il aura pas 1 point dans allumettes*) .

Ensuite, nous savons grâce à nos expériences précédentes (N.S.I), que chaque personne a une manière de réfléchir et de coder différente, donc pour éviter de devoir à chaque fois reprendre le travail de l'autre, le relire et le comprendre, nous avons décidé; nous avons décidé de séparer le travail en 2 . Rémi a codé le menu et codé le système de comptage des points, et Yaël a codé les 3 jeux.

Pour finir, nous avons défini comment nous allons lier nos fichiers entre eux et quelles valeurs ces fonctions ou programmes allaient retourner. Cette étape est primordiale car sans ça, Rémi ne pouvait pas assembler correctement son code au mien.

Codage des mini-jeux :

1 - Le morpion :

Pour créer un morpion fonctionnel, nous avons d'abord cherché à savoir comment créer le plateau et le remplir de manière algorithmique.

Pour stocker les valeurs en fonction de son emplacement on a utilisé une liste de liste d'entier .L'avantage des listes est le fait qu'elles sont ordonnées .

Voici le stockage :

```
# Représentation initiale du plateau de jeu , si case = 0 (vide).  
plateau = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

plateau[0] contiendra la première ligne

plateau[1] contiendra la seconde ligne ...

Puis on a cherché une manière de vérifier si l'un des joueurs a gagné. On a listé toutes les combinaisons possibles pour gagner en utilisant les emplacement de chaque case (1 = case en haut à gauche, et 9 = case en bas à droite).

```
# Liste des combinaisons gagnantes du jeu, chaque sous-liste représente une ligne.  
coupsGagnants = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [1, 4, 7], [2, 5, 8], [3, 6, 9], [1, 5, 9], [3, 5, 7]]  
  
# Liste des cases jouées par le joueur 1.  
joueur1Coups = []  
  
# Liste des cases jouées par le joueur 2.  
joueur2Coups = []
```

Puis nous avons rencontré un problème avec “coupsGagnants” car l’avantage des listes est le fait qu’il soit ordonné mais cela peut aussi être un inconvénient. Si un joueur a réussi à aligner 3 symboles similaires sur les cases 1,2,3, il peut l’avoir fait dans le désordre (2,3,1) et ce coup n’est pas pris en compte dans “coupsGagnants”. Nous avons 2 solutions possibles, soit ajouter tous les coups dans tous les sens possibles dans “coupsGagnants” ou créer une fonction qui vérifie dans différents ordre si cette solution est trouvée. La première solution peut marcher , mais la liste va être extrêmement longue et en plus il y a un risque d'oubli d'une possibilité.

On a donc choisi la deuxième solution et on a créé la fonction “verif_gagnant :

```
def verif_gagnant(CaseJoue: list[list[int]]) -> int:

    from Menu import couleur_affichage

    """
    Vérifie si l'un des joueurs a une combinaison gagnante.

    Paramètre:
        CaseJoue (list[list[int]]): Liste des coups joués pour chaque joueur.

    Retourne:
        int : Si renvoie 0 : pas encore de gagnant, si renvoie 1 : Gagnant , si renvoie 2 : Egalite
    """

    combinaison : list[int]
    compteur_joueur1 : int
    compteur_joueur2 : int

    for combinaison in coupsGagnants:
        # Réinitialise les compteurs pour éviter d'avoir un "faux" gagnant
        compteur_joueur1 = 0
        compteur_joueur2 = 0

        # Balayage de balayage car les listes sont ordonnées et donc il faut vérifier qu'on n'est pas les bons nombres dans le désordre
        for coup in combinaison:
            if coup in CaseJoue[0]:
                compteur_joueur1 += 1
            if coup in CaseJoue[1]:
                compteur_joueur2 += 1

        # Si l'un des compteur atteint 3 il y aura un gagnant
        if compteur_joueur1 == 3:
            return 1
        if compteur_joueur2 == 3:
            return 1

    # Si toutes les cases sont occupées et aucune combinaison gagnante, c'est une égalité
    if len(CaseJoue[0]) + len(CaseJoue[1]) == 9:
        print(couleur_affichage("magenta", "Égalité, aucun des deux joueurs n'a gagné."))
        return 2

    return 0
```

Cette fonction fonctionne en 2 étapes, elle va d'abord balayer une fois la liste et va pour chaque élément de la liste balayer le nombre d'éléments de la première liste puis la seconde liste, etc... Lorsqu'il sera dans une liste pour chaque élément il va regarder si cet élément est dans la liste des coups joués par chaque joueur, et s'il y est, alors il ajoutera 1 point à "compteur_joueur". Si "compteur_joueur" atteint 3 cela signifie qu'un des joueurs a une composition gagnante et la fonction renverra 1.

Au début, cette fonction renvoyé True ou False(Victoire ou défaite) mais elle ne prenait pas en compte s'il y avait une égalité. Nous avons donc rajouté la ligne suivante :

```
# Si toutes les cases sont occupées et aucune combinaison gagnante, c'est une égalité
if len(CaseJoue[0]) + len(CaseJoue[1]) == 9:
    print(couleur_affichage("magenta", "Égalité, aucun des deux joueurs n'a gagné."))
    return 2
```

Cette ligne regarde si toutes les cases sont occupées et si oui elle affiche le message d'égalité.

Par la suite, on a créé une fonction qui permet de demander au joueur sur quelle case il veut jouer et vérifie si cette case existe et si elle est pas occupée. Si ces 2 tests sont validés alors il ajoute le coup à la liste des coups du joueur.

```
def quijoueCoups(quijoue: int, CaseJoue: list[list[int]], joueur1: str, joueur2: str) -> list[list[int]]:
    from Menu import erreur_entree, couleur_affichage

    """
    Demande au joueur actuel de sélectionner une case pour jouer, vérifie si le coup est dans le plateau sinon redemande de jouer le coup.

    Paramètre:
        quijoue (int): Numéro du joueur actuel (1 ou 2).
        CaseJoue (list[list[int]]): liste des cases jouées par chaque joueur.
        joueur1 (str): Nom du joueur 1.
        joueur2 (str): Nom du joueur 2.

    Retourne:
        list[list[int]]: Mise à jour des coups joués par les deux joueurs.
    """
    message : str
    dernierCoup : int

    message = couleur_affichage("bleu", f"({joueur1}) : Sur quelle case voulez-vous jouer ?") if quijoue == 1 else couleur_affichage("magenta", f"({joueur2}) : Sur quelle case voulez-vous jouer ?")

    # Affiche le texte en couleur + Demande la case sur laquelle le joueur veut jouer et enfin vérifie si cette case est comprise entre 1 et 9
    dernierCoup = erreur_entree(1, 9, message, couleur_affichage("rouge", "❌ Entrée invalide. Veuillez entrer un nombre entier !"), couleur_affichage("rouge", "⚠ Nombre hors limites ! Veuillez entrer un"))

    # Vérifie que la case est libre
    while dernierCoup in CaseJoue[0] or dernierCoup in CaseJoue[1]:
        print(couleur_affichage("rouge", "⚠ Case déjà jouée, veuillez resaisir une autre case !"))
        dernierCoup = erreur_entree(1, 9, message, couleur_affichage("rouge", "❌ Entrée invalide. Veuillez entrer un nombre entier !"), couleur_affichage("rouge", "⚠ Nombre hors limites ! Veuillez entrer un"))

    # Enregistre le coup du joueur
    if quijoue == 1:
        CaseJoue[0].append(dernierCoup)
    else:
        CaseJoue[1].append(dernierCoup)

    return CaseJoue
```

Les fonctions “erreur_saisie” et “couleur_affichage” seront présentées dans menu .

Ensuite j’ai relié toutes ces fonctions dans la fonction morpion.

Allumettes :

Pour le second jeu, on a commencé par faire du pseudocode assez simple et rapide pour mettre mes idées au clair et voir comment on allait réaliser ce jeu.

```
nbr_allumettes : entier
recup : entier
quiJoue : entier

nbr_allumettes <- 20
recup <- 0
quiJoue <- 2

tant que nbr_allumettes != 0:
  si quiJoue = 1 :
    afficher "Joueur 1 : Combien d'allumettes voulez vous prendre? Il reste : "
  sinon :
    afficher "Joueur 2 : Combien d'allumettes voulez vous prendre? Il reste : "
  finsi

  Saisir recup
  Controle recup

  si nbr_allumettes - recup < 0 :
    Afficher "Il ne reste pas assez d'allumettes pour en prendre autant"
    Afficher "Il reste" , nbr_allumettes, "allumettes"
  sinon :
    nbr_allumettes <- nbr_allumettes - recup

  si quiJoue = 2 faire:
    quiJoue <- 1
  sinon :
    quiJoue <- 2
fintantque

Afficher "Joueur" , quiJoue, "a gagné la partie"
```

Après avoir fait ce pseudo code nous avons créé ce code :

```
def verification(recup : int) :
    #Contrôle si la valeur rentré est comprise entre 1 et 3
    while recup > 3 or recup < 1:
        print("On ne peut que prendre 1, 2, ou 3 allumettes par tour. Il faut réécrire le nombre d")
        recup = int(input())

def allumettes():
    #Jeu des allumettes si besoin des règles voir dans menu

    recup : int #Variable qui stocke le nombre d'allumette pris pendant une manche (recup ∈ [1,3])
    quiJoue : int #Savoir qui joue entre les deux joueurs
    nbr_allumettes : int #Variable contenant le nombre d'allumettes restante

    nbr_allumettes = 20
    quiJoue = 2

    while nbr_allumettes != 0:

        print("Joueur ",quiJoue," : Combien d'allumettes voulez vous prendre? Il reste : " , nbr_a
        recup = int(input())
        verification(recup)

        if nbr_allumettes - recup < 0 :
            #Controle s'il reste assez d'allumettes que le nombre demandé par l'utilisateur
            print("Il ne reste pas assez d'allumettes pour en prendre autant")
            print("Il reste" , nbr_allumettes, "allumettes")

        else:
            nbr_allumettes -= recup

        #Changement de joueur
        if quiJoue == 2 :
            quiJoue = 1

        else :
            quiJoue = 2

    print("Joueur" , quiJoue, "a gagné la partie")
```


Ce code fonctionnait mais il y a plusieurs problème :

- L'affichage ne mettait pas les noms des joueurs mais juste le numéro
- Si l'on rentrait une valeur différente d'un entier le programme planté
- Cette fonction ne renvoyait rien donc la synchronisation avec le menu était compliqué
- Nous avons aussi voulu améliorer l'affichage des allumettes

Pour remédier à ces problèmes on a d'abord créé la fonction "erreur_saisie" (expliqué dans menu) qui règle le problème des erreurs de saisie et vérifie si la valeur est comprise entre les limites.

```
def allumettes(joueur1: str, joueur2: str, tours: int, quiJoue : int) -> str:  
    from Menu import erreur_entree, couleur_affichage, changer_joueur
```

Puis , on a résolu le problème d'affichage des noms en important les noms depuis le menu en les mettant en paramètre de la fonction, et on

a renvoyé le nom du gagnant au menu, ce qui permet de synchroniser les données entre les deux programmes.

```
# Affiche le gagnant  
if resultatjoueur1 == tours:  
    print(couleur_affichage("vert", f"🏆 {joueur1} a gagné la partie ! Félicitations !"))  
    return joueur1  
else:  
    print(couleur_affichage("magenta", f"🏆 {joueur2} a gagné la partie ! Félicitations !"))  
    return joueur2
```

Pour finir on a ajouté l'affichage des allumettes, pour ce faire on a codé une fonction qui en fonction du nombre d'allumettes affiche un emoji d'allumette.

```
def afficher_allumettes(nbr_allumettes: int) -> None:
    from Menu import couleur_affichage
    """
    Affiche sur la console les allumettes restantes.

    Paramètre:

    - nbr_allumettes (int): Le nombre d'allumettes restant à afficher.
    """
    print("\n" + couleur_affichage("bleu", "%* " * nbr_allumettes))
    print(couleur_affichage("cyan", f"Il reste : {nbr_allumettes} allumettes.\n"))
```

Faire le pseudo code m'a permis de gagner beaucoup de temps car mon code python ressemble fortement à mon pseudo code (*voir le fichier*).

Devinette :

Ce dernier jeu, on l'a commencé de la même manière que le jeu des allumettes. On a d'abord fait un pseudo code très obsolète, mais cela m'a servi à me donner une première idée de la manière dont j'allais coder ce jeu.

```
#Le changement de joueur n'est pas pris en compte
procedure devinette(joueur1 : chaîne, joueur2 : chaîne)
début

    avec resultatjoueur1 : entier
    resultatjoueur2 : entier
    valeur1 : entier
    valeur2 : entier

    resultatjoueur1 <- 0
    resultatjoueur2 <- 0

    afficher "Joueur1 : Saisissez votre valeur"
    Saisir valeur1
    controle valeur1

    afficher "Au tour du Joueur2"

    tant que resultatjoueur1 != 3 ou resultatjoueur2 != 3 faire
        tant que valeur2 != valeur1 faire
            Afficher "Choississez un nombre Joueur2"
            Saisir valeur2
            si valeur2 < valeur1 faire
                Afficher "Trop petit"
            si valeur2 > valeur1 faire
                Afficher "Trop grand"
            sinon
                Afficher "C'est gagné"
                resultatjoueur2 <- resultatjoueur2 + 1
            finsi
        fin faire
    fin faire
```

De cette base, on a codé cette fonction :

```
def devinette() :
    resultatjoueur1 : int
    resultatjoueur2 : int
    valeur1 : int
    valeur2 : int

    resultatjoueur1 = 0
    resultatjoueur2 = 0

    valeur1 = int(getpass.getpass("Joueur1 : Saisissez votre valeur (cachée) : "))
    print("Le Joueur1 a bien saisi une valeur")

    while valeur1 > 100 or valeur1 < 0:
        print("Le Joueur1 a saisi une mauvaise valeur")
        valeur1 = int(getpass.getpass("Joueur1 : Veuillez resaisir votre valeur (cachée) : "))
        print("Le Joueur1 a bien saisi une valeur")

    print("Au tour du Joueur2")
    valeur2 = int(input("Joueur2 : Saisissez votre valeur : "))

    if valeur2 == valeur1 :
        print("C'est gagné")
        resultatjoueur2 += 1
    else :
        while valeur2 != valeur1 :
            if valeur2 < valeur1 :
                print("Trop petit")
                valeur2 = int(input("Joueur2 : Saisissez votre valeur : "))
            elif valeur2 > valeur1 :
                print("Trop grand")
                valeur2 = int(input("Joueur2 : Saisissez votre valeur : "))
            print("C'est gagné")
            resultatjoueur2 += 1

    print("Voici le resultat du joueur1 : ", resultatjoueur1)
    print("Voici le resultat du joueur2 : ", resultatjoueur2)
```

Cette fonction avait les premiers problèmes similaires à la fonction allumettes (nom, synchronisation, erreur entrée). Pour éviter la triche lorsque deux joueurs jouent ensemble, on a ajouté la bibliothèque `getpass` qui cache la valeur entrée dans la console. Dans ce prototype, lorsqu'on lance `devinette` il y a qu'une seule partie qui est jouée. Après avoir testé ce jeu nous avons remarqué qu'il était impossible de se démarquer si chaque joueur a le nombre de coups illimités; nous avons alors décidé de mettre une limite de 5 demandes et si le joueur n'a pas trouvé alors l'autre joueur remporte le point. Cette règle permet de rajouter de l'enjeu.

```
if tentative == 0 and nombre_cherche != nombre_propose:

    print("\n\033[33m🔴 Vous avez fait 5 tentatives sans trouver le nombre, c'est maintenant à l'autre joueur de jouer \n\033[0m")

    if quiJoue == 1:
        resultatjoueur2 += 1
    else:
        resultatjoueur1 += 1

else:

    print("\n\033[32m🟢 Vous avez trouvé le bon nombre, BRAVO !!!\n\033[0m")

    # Augmente les scores en fonction des parties gagnées
    if quiJoue == 1:
        resultatjoueur1 += 1
    else:
        resultatjoueur2 += 1

    print(f"\033[34mVoici le résultat de {joueur1} : \033[0m", resultatjoueur1)
    print(f"\033[35mVoici le résultat de {joueur2} : \033[0m", resultatjoueur2, "\n")

if resultatjoueur1 == tours:
    print(f"\033[32m🟢 {joueur1} a gagné la partie 🏆\033[0m")
    return joueur1
else:
    print(f"\033[32m🟢 {joueur2} a gagné la partie 🏆\033[0m")
    return joueur2
```

On a rajouté une variable tentative qui vérifie si le joueur qui cherche a fait le nombre maximal de tentatives, si oui alors il donne le point à l'autre joueur.

Codage du menu principal :

Tout d'abord, le menu sera constitué de plusieurs fonctions permettant son bon fonctionnement. Mais la fonction principale est "menu" contenant les paramètres joueur1 et joueur2 qui sont des chaînes de

caractères, définis au début du programme, correspondant aux noms des joueurs participant au programme.

Détaillons cette fonction :

Compréhension du code de la fonction menu :

Tout d'abord nous voyons qu'au début de la fonction menu il y a une boucle tant que avec la variable quitter :

```
while quitter:
```

Cette boucle “tant que” nous servira à quitter, littéralement, le menu et donc le programme. Pour faire cela, puisque la variable quitter est

égale a Vrai au début du code, la boucle tourne tout le temps jusqu’à ce qu’on l’initialise à Faux.

Ensuite, sur les lignes suivantes, dans la boucle tant que, nous utilisons la fonction proposition permettant d’afficher les items du menu.

```
while quitter:

    # Fonction qui affiche les différents jeu et fonctionnalité du menu
    proposition()
```

Dans la fonction “proposition” on utilise la fonction “couleur_affichage”.


“Couleur affichage” :

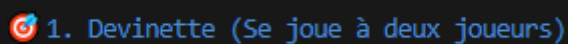
Cette fonction nous permet de choisir la couleur et le texte, entré en paramètre, ensuite si la couleur est égale a notre choix on retourne le texte de la couleur que l’on souhaite grâce à la commande “\033[..m]...\033[0m” .

```
"\033[31m{texte}\033[0m"
```

Cette commande vient du code d’échappement ANSI qui est une manière de colorer la console. Pour que la console reconnaisse la commande il faut toujours commencer par “\033[“ et finir par “m” , cela sert de balise. Ensuite les nombres derrière sert à définir comment le texte doit être affiché (ex : 1 = gras, 31 = rouge). Et pour finir on réinitialise dans la couleur de base, la

suite des affichages qui seront dans la console s'afficheront dans la couleur de base (\033[0m).

Par exemple, dans la fonction proposition, on demande d'afficher le texte “ 1. Devinette (Se joue à deux joueurs)” de la couleur bleu, ainsi dans la console on aura :



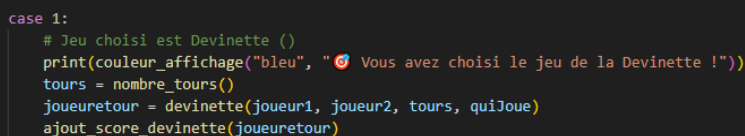
Retour dans la fonction menu :

Puis, on demande à l'utilisateur de faire un choix entre 1 et 6, grâce à la fonction erreur_entree qui nous permet de renvoyer un message d'erreur contrôlé si l'utilisateur saisit autre chose que les choix valeurs entrée en paramètre de la fonction. Dans notre cas, l'utilisateur ne peut que saisir les choix 1,2,3,4,5 et 6, sinon un message d'erreur est renvoyé, en lui proposant de refaire un choix.

Passons donc à la plus grosse structure du code de “menu”, on utilise la structure “match/case”. Pour cela, on écrit match et la variable choix, car cette dernière sert à saisir le choix de l'utilisateur.

Ensuite, puisque la structure match à besoin de cases, donc on va faire plusieurs cas :

- Cas 1 : Ce cas sert à jouer au jeu de la devinette ainsi dans le cas 1 nous aurons ces lignes de code :



```
case 1:
    # Jeu choisi est Devinette ()
    print(couleur_affichage("bleu", " 
```

- On a donc la première ligne du cas qui permet d'afficher que l'utilisateur a bien choisi le jeu de la Devinette qui sera affiché de couleur bleu,
- Ensuite on demande aux joueurs le nombre de tours qu'il souhaite.

Cette fonction permet de demander à l'utilisateur le nombre de tours qu'il souhaite jouer, puis il calcule le nombre de parties à gagner pour être sûr d'avoir battu son adversaire.

Passons aux fonctions "ajout_score_devinette", "ajout_score_allumettes" et "ajout_score_morpion", on notera que sont les mêmes fonctions, elles opèrent simplement sur des différents fichiers.

Fonctions "ajout_score" :

Dans un premier plan, on note que cette fonction prend en paramètre joueur qui est une chaîne de caractère :

```
def ajout_score_devinette(joueur : str) -> None:
```

On va commencer par une syntaxe de try and catch qui nous permet de renvoyer une erreur et d'éviter de faire planter le programme.

On voit que dans le "except", le programme nous renverra "Impossible de lire le fichier" si le programme lève une erreur.

```
except ValueError:  
  
    print("Impossible de lire le fichier")
```

Passons au contenu du try, d'abord nous faisons un test pour savoir si le fichier "Scoredevinette.txt" existe, s'il n'existe pas nous allons en

créer un autre contenant "le nom du joueur :1". On notera ici que pour chaque partie gagnée le joueur gagne un point.

```
# Si le fichier n'est pas créé alors il crée le fichier et met le joueur gagnant plus le score
if not os.path.exists("Scoredevinette.txt"):

    with open("Scoredevinette.txt", "w") as f:

        contenu = joueur + ":1\n"
        f.write(contenu)
```

Donc, ici imaginons que le fichier “Scoredevinette.txt” existe, on va donc ouvrir le fichier en lecture, pour pouvoir le lire et en écrivant le contenu du fichier dans la variable “contenu”. On referme le fichier juste après avoir fait ça.

```
# Si le fichier existe il est ouvert puis lu
with open("Scoredevinette.txt", "r") as fichier:

    contenu = fichier.read()
    fichier.close()
    liste_joueur = contenu.split("\n")
```

Ensuite nous utiliserons la fonction “split” de python qui nous permet de séparer des chaînes de caractères et les écrire dans une liste, qui pour chaque indice aura la chaîne de caractère séparé par un délimiteur choisi. Ici nous écrivons le résultat de la commande `contenu.split("\n")` dans “liste_joueur”, ce que fait cette commande, c’est séparer chaque chaîne de caractère par un saut de ligne. Par exemple si contenu contient ça :

```
Bonjour
Comment allez-vous ?|
```

Le résultat de la commande sera `liste_joueur = ["Bonjour", "Comment allez-vous ?"]`

Ensuite, on va vérifier si le gagnant est dans le contenu du fichier texte, pour cela on écrit le code suivant :


```
for i in range(len(listejoueur)):
    joueurcoupe = listejoueur[i].split(":")

    if joueurcoupe[0] == joueur:
        joueurcoupe[1] = str(int(joueurcoupe[1]) + 1)
        listejoueur[i] = ":".join(joueurcoupe)
        trouve = True

# Si le joueur n'est pas trouvé c'est que le joueur n'a pas de point encore donc on lui attribue 1
if trouve == False:
    listejoueur.append(joueur + ":1")
```

D'abord, nous devons balayer tous les joueurs du contenu, on fait donc une boucle “pour” prenant toute la longueur de la liste “listejoueur”. Après, nous découpons chaque joueur accompagné de son score en 2 dans une liste, appelé “joueurcoupe”, contenant au premier indice le nom du joueur et au deuxième indice le score du joueur.

On effectue ensuite un simple test qui teste si le nom du joueur qui a gagné est égale au nom du joueur balayé par la boucle “pour”.

- **Si il est trouvé** : On ajoute +1 au score du joueur trouvé, puis on reconstruit la liste “listejoueur” avec l’indice de i, grâce à la fonction join de python qui est de la syntaxe suivante : “délimiteur”.join(la liste que l’on souhaite). Et on change la variable “trouve” à True, pour indiquer que le joueur est bien trouvé.
- **Si il n’est pas trouvé** : on test si trouve est faux, si il ne l’est pas on ajoute à la liste “listejoueur” le nom du joueur et “:1”.

Par la suite, on reconstruit tout le contenu du fichier, grâce à la fonction join.

Et, pour finir, on réécrit le contenu de la variable “contenu” dans le fichier grâce aux lignes suivantes :

```
with open("Scoredevinette.txt", "w") as fichierécriture:

    fichierécriture.write(contenu)
    fichierécriture.close()
```

Comme il a été dit précédemment, les fonctions suivantes qui sont : “ajout_score_allumettes” et “ajout_score_morpion”, sont exactement basés sur le même fonctionnement que “ajout_score_devinette”. On va donc s’épargner l’explication de ces fonctions. Revenons sur la fonction principal soit menu. On notera encore une fois que les cas 1,2 et 3 sont les mêmes.

Retour dans la fonction menu :

Le cas 4 :

```
case 4:
    # Affichage des scores (choix 4)
    print(couleur_affichage("vert", "🏆 Vous avez choisi d'afficher les scores !"))
    afficherscore()
```

Il consiste à afficher les scores. La fonction permettant cela se nomme “afficherscore”, elle ne prend aucun paramètre et renvoie simplement le fichier score que l'utilisateur a choisis.

Afficherscore :

On commence encore une fois par un try and catch. Puis, on crée une boucle “tant que” pour que le programme tourne jusqu’à ce que l’utilisateur souhaite quitter. On utilise la fonction “erreur_entree_invalide_str” qui permet d’éviter que l’utilisateur entre une mauvaise valeur.

Ensuite, on fait un “match/case”, on adapte la variable “choixscore” en fonction du “match” qui est demandé à l’utilisateur juste avant. Les cas seront donc : “devinette”, ”allumettes”, ”morpion”, ”quitter” qui sont demandés à l'utilisateur avant le match.

Par la suite, on va donc détailler seulement le cas “devinette” et “quitter”, puisque les cas “devinette”, “allumettes” et “morpion” sont les mêmes mais avec des fichiers différents.

Cas “devinette” :

La première ligne du cas “devinette” est seulement pour afficher le format du score, de couleur violet.

Ensuite on essaye de savoir si le fichier “Scoredevinette.txt” existe :

```
if not os.path.exists("Scoredevinette.txt"):
```

- S’il n’existe pas, on crée un nouveau fichier en l’ouvrant en écriture :

```
with open("Scoredevinette.txt", "w") as f:
```

- Après l’avoir ouvert on écrit une chaîne de caractère vide à l’intérieur du fichier.
- S’il existe, et bien on n’aura pas à traiter ce cas puisque s’il existe on n’aura pas à en créer un

Après avoir effectué ces tests là on peut donc afficher les scores des joueurs, pour cela on trie d’abord les joueurs grâce à la fonction “trier_joueur”, on reviendra à cette fonction plus tard, cependant elle consiste à trier les joueurs, en fonction de leur score, du meilleur au moins bon.

Par la suite, on réouvre le fichier du score de devinette en lecture, et puis on le lit et stocke le résultat dans la variable “contenu” :

```
f = open("Scoredevinette.txt", "r")  
contenu = f.read()
```

Et ensuite, on n’a qu’à afficher la variable “contenu” avec une couleur bleu et fermer le fichier.

On rappelle donc que les cas “devinette”, “allumettes” et “morpion” sont les mêmes ont s’enlèvera donc le poids d’expliquer les autres cas.

Il nous reste un dernier cas qui est “quitter” et le cas par défaut, voici donc le cas “quitter” :

```
case "quitter":
    souhait = erreur_entree_str(["oui", "o", "non", "n"], couleur_affichage("rouge", "Souhaitez-vous quitter ? si oui=o si non=n : "))
    , "La valeur doit être un caractère ou une chaîne de caractères"]
    if souhait in ["oui", "o"]:
        print(couleur_affichage("vert", "Merci de votre utilisation"))
        quitter = False
```

La première ligne demande à l'utilisateur s'il souhaite quitter ce sous-menu. Grâce à la fonction que l'on voit, soit “erreur_entree_str”, qui est basée sur le même fonctionnement que “erreur_entree”, elle permet de vérifier que l'entrée de l'utilisateur est une chaîne valide. Revenons au cas précédent, après avoir demandé le choix de l'utilisateur, on effectue un test pour savoir si il souhaite quitter :

- S'il remplit les conditions, donc si la variable “souhait” est égale à “oui” ou “o”, alors on effectue les lignes de commandes présentes après le test
- On affiche donc de couleur verte “Merci de votre utilisation” et on met dans la variable quitter le booléen FAUX qui nous permet de quitter ce sous menu, puisque la boucle tant que vu au début de la fonction, tourne tant que la variable quitter est égale a VRAI.
- S'il ne remplit pas les conditions alors on remonte au début de la boucle tant que et on refait tourner cette boucle une fois de plus.

Cas par défaut :

Passons au cas par défaut, ce cas nous indique que l'exécution du “match” à lever une erreur, l'erreur en question est que l'utilisateur n'as pas fait le bon choix. Ainsi on affiche simplement de couleur rouge “Erreur de choix”

Trier_joueur :

Rappelez-vous, nous avons parlé précédemment de la fonction “trier_joueur”, elle consiste à trier les joueurs en fonction de leur score du meilleur au moins bon.

On notera que cette fonction prend en paramètre “nomscore” qui est une chaîne de caractère contenant soit “devinette”, soit “allumettes”, soit “morpion”. Cette variable nous servira à récupérer le nom du fichier auquel on doit trier les joueurs.

On commence d’abord par un “try and catch”, ainsi, si le programme lève une erreur, alors on affichera le message suivant : “Une erreur est survenue”

Ensuite, on ouvre le fichier en mode lecture, en faisant une concaténation avec la variable “nomscore” et “Score...txt”. On lit ensuite le contenu du fichier et on le met dans la variable contenu, puis on referme le fichier :

```
with open(f"Score{nomscore}.txt", "r") as f:  
    contenu = f.readlines()  
    f.close()
```

Lors de cette boucle, on va tester pour savoir si notre ligne contient bien le nom du joueur et son score séparé d’un “:”, on va ensuite séparer le contenu de la ligne par le délimiteur “:”, grâce à la fonction de python split. On rappelle que le format du fichier score est sous la forme “Nom:Score”, on aura donc dans la liste “joueurcoupe” en premier indice le nom du joueur et en deuxième indice le score du joueur.

Puis ensuite on ajoute simplement dans la liste “joueurs”, le nom du joueur et son score, grâce à la fonction python append :

Après, on va implémenter un tri par sélection, on va donc commencer par balayer toute la longueur de la liste “joueurs”.

Ensuite on suppose que pour chaque balayage, “i” est le joueur ayant le score max, puis, on balaye le reste des joueurs et si le score du joueur ayant, soit disant, le meilleur score est inférieur au score trouvé par la boucle effectuée ci-dessus, alors le maximum devient “j” :

```
if joueurs[j][1] > joueurs[max][1]:  
    max = j
```

Après avoir terminé cette boucle, on effectue un test pour savoir si le maximum est différent de “i” et s’il l’est cela veut dire qu’on a trouvé un élément plus grand :

```
if max != i:
```

- **Si on a trouvé un élément plus grand** : on échange la place des joueurs dans la variable “joueurs”
- **Si on a pas trouvé un élément plus grand** : On ne fait rien puisque l’élément le plus grand reste “i”

Ensuite, après avoir terminé toutes ces boucles on reconstruit le contenu avec les joueurs triés du meilleur au moins bon avec la fonction join.

Ainsi, dans la fonction join on a une boucle qui permet de créer une liste par compréhension, la voici :

```
(f"{nom}:{score}" for nom, score in joueurs
```

- Commençons par la boucle, pour chaque itération on balaye le premier et le deuxième indice du tuple de la liste joueurs,
- La chaîne de caractère permet d’écrire le nom du joueur “:” le score du joueur, récupéré grâce à la liste “joueurs”.

```
f"{nom}:{score}"
```

Après cela, on ouvre le fichier en écriture, on écrit ensuite le contenu de la variable “contenu_trie” dans le fichier

Retour dans la fonction menu :

Passons au cinquième cas,

```
case 5:  
    #Affichage des règles (choix 5)  
    print("")  
    reglejeu()
```

Ce cas nous sert à afficher les règles des jeux grâce à la fonction “reglejeu”.

reglejeu :

Cette fonction est basée sur la même structure que la fonction “menu”; elle permet d’afficher les règles des jeux, le code est basé sur le principe de “match/case” utilisé par la fonction menu.

Retour dans la fonction menu :

Puis, pour terminer avec les fonctions présente de le fichier Menu.py, on a le cas par défaut de la fonction menu, qui nous sert simplement a afficher à l’utilisateur qu’il n’a pas fait le bon choix.

Bien sûr, grâce aux fonctions “erreur_entree” et “erreur_entree_str”, le choix de l’utilisateur ne peut que bien se faire, et donc la cas par défaut n'est censé jamais être utilisé.

```
case _:  
    print(couleur_affichage("rouge", "❌ Choix invalide !!!"))
```



```
Il reste : 1 allumettes.  
yael : Combien d'allumettes voulez-vous prendre ? 3  
⚠ Pas assez d'allumettes disponibles !  
Il reste : 1 allumettes.  
yael : Combien d'allumettes voulez-vous prendre ? 2  
⚠ Pas assez d'allumettes disponibles !  
Il reste : 1 allumettes.  
yael : Combien d'allumettes voulez-vous prendre ? z  
✖ Entrée invalide. Veuillez entrer un nombre !
```

S'il ne reste qu'une ou deux allumettes l'utilisateur ne peut pas prendre 3 allumettes, car il n'en reste pas assez. Les valeurs différentes d'un nombre entier entre 1 et 3 sont refusées.

Devinettes :

```
remi : Saisissez une valeur (cachée) entre 0 et 100 :  
⚠ Erreur : Vous devez entrer un nombre entier valide.  
remi : Saisissez une valeur (cachée) entre 0 et 100 :  
⚠ Erreur : La valeur doit être comprise entre 0 et 100.  
remi : Saisissez une valeur (cachée) entre 0 et 100 :  
⚠ Erreur : La valeur doit être comprise entre 0 et 100.  
remi : Saisissez une valeur (cachée) entre 0 et 100 :
```

La valeur est caché lors de ces jeux de test :

Première valeur entrée est e

Deuxième valeur entrée est 222

Troisième valeur entrée est -1

Dernière valeur entrée est 24

```
yael : Vous avez 5 tentatives pour trouver le nombre, saisissez votre valeur : e  
⚠ La valeur doit être un nombre entier  
yael : Vous avez 5 tentatives pour trouver le nombre, saisissez votre valeur : -22  
⚠ La valeur doit être un nombre entier compris entre 0 et 100  
yael : Vous avez 5 tentatives pour trouver le nombre, saisissez votre valeur : 233  
⚠ La valeur doit être un nombre entier compris entre 0 et 100  
yael : Vous avez 5 tentatives pour trouver le nombre, saisissez votre valeur :  
⚠ La valeur doit être un nombre entier  
yael : Vous avez 5 tentatives pour trouver le nombre, saisissez votre valeur : ""  
⚠ La valeur doit être un nombre entier  
yael : Vous avez 5 tentatives pour trouver le nombre, saisissez votre valeur : "2"  
⚠ La valeur doit être un nombre entier  
yael : Vous avez 5 tentatives pour trouver le nombre, saisissez votre valeur : 2  
🔵 Trop petit ! yael, vous devez essayer une autre valeur :
```

Le nombre doit être compris entre 1 et 100

```
yael : Saisissez une valeur (cachée) entre 0 et 100 :  
yael a bien saisi une valeur  
  
Au tour de remi  
  
remi : Vous avez 5 tentatives pour trouver le nombre, saisissez votre valeur : 2  
▲ Trop petit ! remi, vous devez essayer une autre valeur : 3  
▲ Trop petit ! remi, vous devez essayer une autre valeur : 25  
▼ Trop grand ! remi vous devez essayer une autre valeur : 24
```

Le nombre cherché est 24. Le message annonçant la victoire du joueur est hors de la capture d'écran.

Si l'utilisateur fait plus de 5 tentatives alors l'utilisateur qui a proposé le nombre gagne.

```
yael : Saisissez une valeur (cachée) entre 0 et 100 :  
yael a bien saisi une valeur  
  
Au tour de remi  
  
remi : Vous avez 5 tentatives pour trouver le nombre, saisissez votre valeur : 3  
▲ Trop petit ! remi, vous devez essayer une autre valeur : 55  
▼ Trop grand ! remi vous devez essayer une autre valeur : 35  
▼ Trop grand ! remi vous devez essayer une autre valeur : 28  
▼ Trop grand ! remi vous devez essayer une autre valeur : 26  
  
■ Vous avez fait 5 tentatives sans trouver le nombre, c'est maintenant à l'autre joueur de jouer  
  
Voici le résultat de yael : 1  
Voici le résultat de remi : 0
```

La valeur cherchée était 24.

Menu :

```
Joueur 1: veuillez choisir votre pseudo :  
Le pseudo ne doit pas être vide, contenir uniquement des espaces ou être invalide.  
Joueur 1: veuillez choisir votre pseudo :  
Le pseudo ne doit pas être vide, contenir uniquement des espaces ou être invalide.  
Joueur 1: veuillez choisir votre pseudo : yael  
Joueur 2: veuillez choisir votre pseudo : yael  
Le pseudo ne doit pas être vide, contenir uniquement des espaces, ou être identique à celui du Joueur 1.  
Joueur 2: veuillez choisir votre pseudo :  
Le pseudo ne doit pas être vide, contenir uniquement des espaces, ou être identique à celui du Joueur 1.  
Joueur 2: veuillez choisir votre pseudo :  
Le pseudo ne doit pas être vide, contenir uniquement des espaces, ou être identique à celui du Joueur 1.  
Joueur 2: veuillez choisir votre pseudo : remi
```

Les joueurs ne peuvent pas entrer que des espaces ou faire qu'une entrée. Ils ne peuvent pas aussi mettre le même pseudo sinon il y aura un problème au niveau des scores.

```
-----  
1. Devinette (Se joue à deux joueurs)  
2. Allumettes (Se joue à deux joueurs)  
X 3. Morpion (Se joue à deux joueurs)  
4. Afficher les scores  
5. Afficher les règles  
6. Quitter  
  
Veuillez faire votre choix (1,2,3,4,5,6) : 7  
▲ La valeur doit être un entier compris entre 1 et 6.  
Veuillez faire votre choix (1,2,3,4,5,6) : dz  
X La valeur saisie doit être un nombre entier positif.  
Veuillez faire votre choix (1,2,3,4,5,6) : un  
X La valeur saisie doit être un nombre entier positif.  
Veuillez faire votre choix (1,2,3,4,5,6) : 1  
Veuillez faire votre choix (1,2,3,4,5,6) : 1  
Vous avez choisi le jeu de la Devinette !
```

Lorsqu'on choisit le jeu il faut rentrer une valeur entre 1 et 6 qui correspond au jeu ou à la fonctionnalité attribué.

```
➡ Vous avez choisi le jeu de la Devinette !
➡ Combien de parties souhaitez-vous jouer ? (1 à 20) : e
✗ Entrée invalide. Veuillez entrer un nombre entier !
➡ Combien de parties souhaitez-vous jouer ? (1 à 20) : 333
⚠ Nombre hors limites ! Veuillez entrer un nombre entre 1 et 20.
➡ Combien de parties souhaitez-vous jouer ? (1 à 20) : -é
✗ Entrée invalide. Veuillez entrer un nombre entier !
➡ Combien de parties souhaitez-vous jouer ? (1 à 20) : -3
⚠ Nombre hors limites ! Veuillez entrer un nombre entre 1 et 20.
➡ Combien de parties souhaitez-vous jouer ? (1 à 20) : 3
```

La valeur doit être comprise entre 1 et 20.

```
📄 Veuillez faire votre choix (1,2,3,4,5,6) : 4
📄 Vous avez choisi d'afficher les scores !
Veuillez faire votre choix (devinette,allumettes,morpion,quitter) : 2
La valeur saisie n'est pas bonne
Veuillez faire votre choix (devinette,allumettes,morpion,quitter) : allu
La valeur saisie n'est pas bonne
Veuillez faire votre choix (devinette,allumettes,morpion,quitter) : allumete
La valeur saisie n'est pas bonne
Veuillez faire votre choix (devinette,allumettes,morpion,quitter) : allumettes
Le format du score est 'Nom:Score'
yael:1
```

Pour afficher les scores, il faut écrire les noms de chaque jeu à la lettre près.

```
📄 Veuillez faire votre choix (1,2,3,4,5,6) : 5
Quel règles souhaitez vous afficher, 1 pour devinette, 2 pour allumettes, 3 pour morpion et 4 pour quitter le menu des règles : un
Vous n'avez pas rentré la bonne valeur
Quel règles souhaitez vous afficher, 1 pour devinette, 2 pour allumettes, 3 pour morpion et 4 pour quitter le menu des règles : 23
Vous n'avez pas choisi le bon chiffre
Quel règles souhaitez vous afficher, 1 pour devinette, 2 pour allumettes, 3 pour morpion et 4 pour quitter le menu des règles : 1
Voici les règles du jeu de la devinette.
Un des deux joueurs choisit un nombre entre 0 à 100 (l'affichage du nombre choisit sera caché).
Ensuite, le second joueur doit trouver le nombre choisit par le premier, il aura 5 tentatives.
Bien sur le joueur cherchant le nombre caché sera aider par des 'trop petit', 'trop grand' en fonction du nombre donné par le second joueur.
```

Pour afficher les règles, on demande de rentrer une valeur comprise entre 1 et 4.

```
➡ 6. Quitter
📄 Veuillez faire votre choix (1,2,3,4,5,6) : 6
📄 Souhaitez-vous quitter ? (oui/o, non/n) : dez
✗ Entrée invalide.
📄 Souhaitez-vous quitter ? (oui/o, non/n) : 1
✗ Entrée invalide.
📄 Souhaitez-vous quitter ? (oui/o, non/n) : o
👋 Merci de nous avoir utilisé !
```

Pour quitter, on demande de d'abord mettre la valeur 6 et ensuite, on vérifie à chaque fois s'il est sûr de vouloir quitter le jeu ou le menu des règles... Pour quitter les seules valeurs acceptées sont : o,oui,n,non.

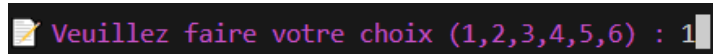
Améliorations Possibles :

- **Affichage des points** ; Nous souhaitons que, pour la seconde SAE, un autre système d’affichage des points soit mis en place, en réalité, nous aimerions qu’il ressemble à ça :

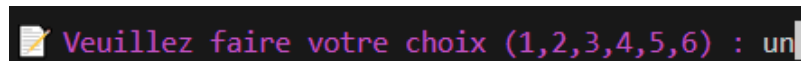
```
Partie Joueur1 - Joueur2 {  
    devinette : 1-2  
    allumettes : 0-2  
    Morpion : 1-1  
}
```

Grâce à cet affichage, les 2 joueurs pourrons reprendre la partie plus tard en choisissant le nom de leurs pseudos.

- **Fichier des Scores** : On pourra également utiliser un fichier JSON pour simplifier notre tâche puisque c’est un fichier texte, qui est mieux organisé grâce à des dictionnaires.
- **Ecrire en toutes lettres ou en chiffre** : Nous aimerions également que l’utilisateur puisse écrire les choix qui veulent faire en lettre et en chiffre, par exemple :



ou bien



- **Algorithme de tri pour la fonction “trier_joueur”** : Nous pouvons changer l’algorithme de tri de la fonction “trier_joueur”, qui est actuellement un tri par sélection, pour passer par un tri plus rapide, le “tri rapide” ou “quicksort” en anglais.