Bastide Rémi
Betton Yaël

Compte rendu S.A.E S1.02

Sommaire :

Contexte :

L'objectif de cette S.A.E est d'améliorer le code de la S.A.E S1.01 en ajoutant des robots de différentes difficultés, avec différents modes d'affrontement, mais aussi d' optimiser notre ancien code. Nous vous avons mis à disposition notre premier compte rendu dans le fichier zip, si vous souhaitez revoir les règles des jeux et la structure de notre code d'origine …

Codage des mini-jeux :

Dans cette partie nous allons voir comment nous avons réalisé les jeux Devinettes, Morpion, et Allumettes.

Devinette :

Logique suivie par le robot en fonction du niveau :

- Niveau 1 : Le robot va proposer un nombre aléatoire entre les deux bornes. Au début du jeu ces bornes sont initialisées à 0 et 100, puis au fur et à mesure des propositions du robot c'est bornes vont réduire . Pour rajouter de l'enjeu, nous avons décidé de donner 7 tentatives au robot.

  Exemple :

  Borne = [0,100]
  Première proposition : 46 -> Trop petit
  Borne = [46,100]
  Deuxième proposition : 78 -> Trop grand
  Borne = [46,78]
  …

```
return randint(minimum, maximum)
```

- Niveau 2 : Le second robot va utiliser la méthode dichotomique pour trouver le nombre. La dichotomie est une technique qui consiste à prendre le chiffre du milieu (dans une liste de nombre triés par ordre croissant) et de vérifier s'il est trop petit où trop grand. Ensuite on continuera cette opération en mettant à jour les bornes à chaque coup jusqu'à trouver le bon nombre. De plus, pour rajouter une part d'aléatoire, si le robot n'a pas trouvé avant la dernière tentative, il proposera un nombre aléatoire compris entre les deux bornes pour son dernier coup. Ce robot aura 5 tentatives pour trouver le résultat.

  Exemple :

  Borne = [0,100]

Première proposition : 50 -> Trop grand
Borne = [0,50]
Deuxième proposition : 25 -> Trop petit
Borne = [25,50]
…

```
if tentative != 1 :
    milieu = (maximum+minimum)/ 2
else :
    milieu = randint(minimum,maximum)
return int(milieu)
```

- Niveau 3 : Pour finir, ce robot va suivre la même logique que le robot de niveau 2, sauf qu'il aura 7 tentatives ce qui fait qu'il sera impossible à battre si le robot recherche un nombre entre 0 et 100. Cette méthode est optimale car si nous divisons successivement 100 par 2 au bout de la 7ème fois le résultat sera 1.

```
milieu = (maximum+minimum)/ 2
return int(milieu)
```

Dans le cas où le robot doit choisir le nombre recherché, il choisira un nombre aléatoire entre 0 et 100. Nous avons décidé de ne pas donner de stratégie au niveau de ce choix pour éviter de faciliter la victoire du joueur s'il comprend celle-ci.

Implémentation des robots dans le code d'origine :

Après avoir expliqué la logique que chaque robot va suivre, nous allons les rajouter au code créé dans la S.A.E S1.01. Pour ce faire, nous avons créé une fonction qui gère tous les choix des robots vu précédemment.

```
def diff_niveau(minimum : int, maximum : int, tentative : int, niveau : int) -> int|None:
```

Ensuite, la fonction robot va permettre d'afficher ce que le robot joue et prendre certaines décisions comme par exemple le nombre de tentatives en fonction du niveau, ou le choix des nouvelles bornes a appliqués.

```
if nombre_propose < nombre_cherche:

    minimum = max(minimum, nombre_propose)

    print(couleur_affichage("jaune", f"Le nombre proposé est {nombre_propose}"))
    print(couleur_affichage("vert", "🔼 Trop petit !"))

    nombre_propose = diff_niveau(minimum + 1, maximum - 1, tentative, niveau)

elif nombre_propose > nombre_cherche:

    maximum = min(maximum, nombre_propose)

    print(couleur_affichage("jaune", f"Le nombre proposé est {nombre_propose}"))
    print(couleur_affichage("vert", "🔽 Trop grand !"))

    nombre_propose = diff_niveau(minimum + 1, maximum - 1, tentative, niveau)
```

Maintenant, il restera juste à implémenter ce code dans la fonction "devinette" , qui dans le cas où un des deux joueur à le mot robot dans son nom, il exécutera cette fonction pour faire jouer le robot de difficulté demandée .

```
# Vérifie si un des deux joueurs est un robot
if contient_robot(joueur1) and quiJoue == 1:
    nombre_propose = -1
    joueur_eq_bot = True

elif contient_robot(joueur2) and quiJoue == 2:
    nombre_propose = -1
    joueur_eq_bot = True

else:
    joueur_eq_bot = False
    nombre_propose = erreur_entree(0, 100, couleur_affichage("vert",f"{joueur1 if quiJoue == 1 else joueur2} : Vous avez {tentative + 1} tentatives pour trouver
```

```
if joueur_eq_bot == True :

    nombre_propose,nombre_cherche,tentative = robot(niveau,tentative,nombre_cherche,joueur1,joueur2,quiJoue)
```

La vitesse de l'algorithme est quasiment instantanée.
Pour le niveau 1, il a gagné 10 fois avant que l'on atteigne 25 victoires.
Pour le niveau 2, il a gagné 22 fois avant que l'on atteigne 25 victoires.
Pour le niveau 3, il a gagné 25 fois et nous n'avions que 15 victoires.
L'expérience utilisateur peut être frustrante si l'on joue le robot 3 ou 1 car d'un côté le robot est mauvais mais de l'autre il est impossible à battre. Le robot de niveau 2 est une bonne alternative pour avoir une part de challenge et un part de victoire.

Allumettes :

Logique suivie par le robot en fonction du niveau :

- Niveau 1 : Le premier robot va choisir aléatoirement le nombre d'allumettes à choisir entre 1 et 3.

```
return randint(1,nbr_allumettes)
```

- Niveau 2 : Le robot de niveau 2 va suivre la même logique que le premier robot sauf que lorsque le nombre d'allumettes est inférieur à 4 il joue de manière à laisser qu'une seule allumettes à son adversaire.

```
if nbr_allumettes > 4 :
    return randint(1,3)

else:

    if nbr_allumettes == 1 :
        return 1
    else :
        return nbr_allumettes - 1
```

- Niveau 3 : Dans ce jeu, il n'existe pas de stratégie permettant de gagner à tous les coups, mais une méthode existe pour maximiser les chances de victoire. Pour cela, le robot doit toujours essayer de laisser à son adversaire un nombre d'allumettes correspondant à 4n-1 allumettes , il y a certaines situations où le robot ne pourra pas suivre ce schéma. Pour finir lorsqu'il arrivera à la fin, il aura souvent la possibilité de laisser qu'une seule allumette à son adversaire.

```
recup = 1
if nbr_allumettes > 4:

    #Verifie si c'est un multiple de 4
    while (nbr_allumettes - recup) % 4 != 1 and recup <= 2:
        print(nbr_allumettes - recup % 4)
        recup += 1

else:

        if nbr_allumettes <= 4 and nbr_allumettes != 1:
            recup = nbr_allumettes - 1

return recup
```

Implémentation des robots dans le code d'origine :

Si le robot ne peut pas prendre un multiple de 4n - 1, il prendra alors 3 allumettes.

Nous avons ensuite créé une fonction qui, en fonction du niveau choisi, exécute la logique correspondante parmi les trois niveaux. De plus, après que le robot ait joué, un temps d'attente est affiché pour simuler une réflexion de sa part, suivi d'un message indiquant le nombre d'allumettes qu'il a décidé de prendre.

```
print(couleur_affichage("jaune",f"Le {joueur1 if quiJoue ==1 else joueur2} réfléchi ..."))
time.sleep(1.5)
print(couleur_affichage("jaune",f"Le {joueur1 if quiJoue == 1 else joueur2} a pris {recup} allumette(s)"))
return recup
```

Pour finir, il nous restera juste à l'ajouter dans notre ancienne version d'allumettes, en vérifiant si le joueur qui joue est un robot :

```
    if quiJoue == 1:
        if contient_robot(joueur1) == False:
            recup = erreur_entree(1, 3,couleur_affichage("vert", f"{joueur1} : Combien d'allumettes voulez-vous prendre ? "),couleur_affichage("rouge", "❌ Entrée invalide. Veuillez entrer un nombre !"),cou
        else:
            recup = robot(nbr_allumettes,niveau,joueur1,joueur2,quiJoue)

    else:
        if contient_robot(joueur2) == False:
            recup = erreur_entree(1, 3,couleur_affichage("magenta", f"{joueur2} : Combien d'allumettes voulez-vous prendre ? "),couleur_affichage("rouge", "❌ Entrée invalide. Veuillez entrer un nombre !"),
        else:
            recup = robot(nbr_allumettes,niveau,joueur1,joueur2,quiJoue)

nbr_allumettes -= recup
```

La vitesse de l'algorithme est quasiment instantané sur ce jeu.
Pour le niveau 1, il est assez simple de le battre mais ça arrive qu'il gagne certaines fois.
Pour le niveau 2, il gagne assez souvent grâce à son algorithme
Pour le niveau 3, il gagne presque à chaque fois, on peut le contrer que lorsqu'on commence à jouer.
Du point de vue d'un utilisateur, le niveau 2 et 3 sont intéressants car ils sont complexes et donc c'est assez dur de gagner contre ces robots. Le niveau 1, est facile à battre donc c'est moins intéressant de l'affronter.

Morpion :

Le jeu morpion a été le jeu a posé le plus de problème au niveau de l'ajout des robots surtout sur le niveau 3.

Logique suivie par le robot en fonction du niveau :

- Niveau 1 : Le robot de niveau 1 va jouer ses coups aléatoirement parmi les coups pas encore joués. Pour ce faire, il va vérifier que le coup choisi aléatoirement n'est pas déjà présent dans la liste "CaseJoue".

```
dernierCoup : int

dernierCoup = randint(1,9)
while dernierCoup in CaseJoue[0] or dernierCoup in CaseJoue[1]:
    dernierCoup = randint(1,9)

return dernierCoup
```

- Niveau 2 : Ensuite dans le niveau 2, le robot jouera aléatoirement sauf si ce robot a la possibilité de gagner. A chaque coup, il vérifiera d'abord s'il peut gagner; si oui il jouera sur la bonne case, sinon il jouera aléatoirement.

```
# Vérifie si le robot peut gagner
case = (verif_gagnant(CaseJoue,quiJoue) - 10)
if case >= 0:
    return case

# Sinon, joue aléatoirement
case = randint(1, 9)
while case in CaseJoue[0] or case in CaseJoue[1]:
    case = randint(1, 9)
return case
```

Dans cette nouvelle version de morpion, nous avons modifié "verif_gagnant" pour qu'il puisse satisfaire nos nouvelles exigences. Dans notre ancienne version la fonction renvoie 1 pour signaler qu'il y a un gagnant mais ne précisait pas lequel, maintenant la fonction renvoie soit un ou deux en fonction du gagnant, et si un joueur est proche de gagner il retourne la case + 10. Nous ajoutons 10 à la valeur de la case pour éviter de faire bugger les valeurs, car si la case est 1 nous ne pourrons pas déterminer si c'est la case 1 ou si le joueur1 a gagné. Pour déterminer la case sur laquelle joué ou si un des deux joueurs a gagné nous faisons ces tests :

```python
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 vérifi
    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

    if compteur_joueur1 == 2 and quiJoue == 1:
        # Cherche case manquante pour trouver la combinaison
        for coup in combinaison:
            if coup not in CaseJoue[0] and coup not in CaseJoue[1]:
                return coup + 10

    # Si joueur 2 proche de gagner il cherche la case qui manque pour gagner
    if compteur_joueur2 == 2 and quiJoue == 2:
        for coup in combinaison:
            if coup not in CaseJoue[0] and coup not in CaseJoue[1]:
                return coup + 10
```

- Niveau 3 : Le niveau 3 est le niveau imbattable, un joueur ou un robot ne peut pas le battre. Pour que ce robot soit imbattable, nous utilisons la méthode “minimax”.

La méthode “minimax” : Cette méthode consiste a essayé tous les coups possibles et à chaque fois on attribue la valeur 100 si l’ordinateur gagne, -100 si l’adversaire gagne et 0 s’il y a égalité de plus nous optimisons la manière de gagner en ajoutant le paramètre de profondeur (ça sera expliqué plus tard).

- La fonction “coup_ordi” :

Pour tester tous les coups possibles, on balaye le plateau et on vérifie que la case est libre :

```python
for largeur in range(3):
    for longueur in range(3):

        if plateau[largeur][longueur] == 0:
```

Ensuite on simule le coup donc on attribue la valeur du joueur correspondantes sur cette case :

```python
case = largeur * 3 + longueur + 1
if quiJoue == 1:
    CaseJoue[0].append(case)
    plateau[largeur][longueur] = 1
else:
    CaseJoue[1].append(case)
    plateau[largeur][longueur] = 2
```

Puis, nous exécuterons la fonction “minimax” qui renverra le meilleur score possible en fonction des différents coups de l’adversaire (La fonction minimax sera expliquée ensuite). Après, avoir simulé et récupéré le meilleur score de ce coup, ce coup simulé sera supprimé et nous vérifierons ensuite si ce coup a un meilleur score que les précédents :

```
score = minimax(plateau, 0, False, quiJoue, CaseJoue,9)

if quiJoue == 1:
    CaseJoue[0].pop()
else:
    CaseJoue[1].pop()

plateau[largeur][longueur] = 0


if score > meilleur_score:
    meilleur_score = score
    meilleurs_coups = [case]
elif score == meilleur_score:
    meilleurs_coups.append(case)
```

Ensuite pour éviter que les parties soient toutes similaires et redondantes, on a ajouté “random.choice” qui va choisir aléatoirement parmi les meilleurs coups, un coup a renvoyé.

```
# Choisir un coup aléatoire parmi les meilleurs
return random.choice(meilleurs_coups)
```

- La fonction “Minimax” :

Cette fonction est basée sur la récursivité, elle va balayer toutes les possibilités du tableau et va essayer de maximiser le score du robot, et si c’est au tour de l’adversaire cette fonction va essayer de minimiser le score de l’adversaire.

Dans cette fonction la condition d’arrêt se fera soit lorsqu’un des deux joueurs à gagner, soit lorsque le plateau est plein et qu’il y a égalité, ou sinon lorsque la profondeur est supérieur à la profondeur maximale entrée en paramètre. Ces conditions sont obligatoires pour éviter que le programme tourne en boucle et ne s'arrête jamais :

```
# Condition de fin de recursivité
if verif_gagnant_ordi(CaseJoue,quiJoue) == quiJoue:
    return 100 - profondeur

elif verif_gagnant_ordi(CaseJoue,quiJoue) == changer_joueur(quiJoue):
    return -100 + profondeur

elif len(CaseJoue[0]) + len(CaseJoue[1]) == 9:
    return 0

elif profondeur >= profondeur_max:
    return 0
```

Ensuite comme nous l’avons expliqué précédemment, le robot va essayer de maximiser son score, pour ce faire si c’est au tour du robot le paramètre booléen “maximiser” sera évalué à True sinon il sera évalué à “False”. Après être rentré dans une des deux possibilités il va rechercher le meilleur coup :

```python
if maximiser :

    meilleur_score = -1000

    for largeur in range(3):
        for longueur in range(3):

            if plateau[largeur][longueur] == 0:

                case = largeur * 3 + longueur + 1
                CaseJoue[0 if quiJoue == 1 else 1].append(case)
                if quiJoue == 1 :
                    plateau[largeur][longueur] = 1
                else :
                    plateau[largeur][longueur] = 2

                score = minimax(plateau, profondeur + 1, False, quiJoue, CaseJoue, profondeur_max)

                CaseJoue[0 if quiJoue == 1 else 1].pop()

                plateau[largeur][longueur] = 0

                # Mettre à jour le meilleur score
                if score > meilleur_score :
                    meilleur_score = score

    return meilleur_score
```

On attribue le score de -1000 à “meilleur_score” pour être sur qu’il sera modifié ensuite et pour éviter toutes les erreurs; à l’inverse si c’est à l’adversaire de jouer et que “maximiser == False” alors “meilleur_score” sera à 1000 et on recherchera plus petit score entre “meilleur_score” et “score” pour essayer de minimiser le score adverse. A chaque coup, on ajoute 1 à “profondeur” car pour arriver à une des conditions de fin nous faisons un coup de plus à chaque fois, et notre but est d'arriver à gagner le plus vite possible. Le paramètre “profondeur_max” peut servir à raccourcir le temps de calcul car pour tous les coups mais particulièrement lors du premier coup, le robot fera énormément de calcul.

Nous pouvons optimiser la vitesse de calcul du premier coup si le robot est de niveau 3. Si nous limitons la profondeur maximum à 3 au lieu de 9, l’ordinateur s'arrêtera plus tôt. Pour montrer cela, nous avons regardé le nombre de passage dans la fonction “minimax”.

```
Robot2_Difficle doit jouer
Le Robot2_Difficle réfléchi ...
549945

1 | 2 | 3
--+---+--
4 | O | 6
--+---+--
7 | 8 | 9
```

Et voici si l’on modifie 9 par 3 pour la profondeur maximum

```
Robot1_Difficle doit jouer
Le Robot1_Difficle réfléchi ...
3609

1 | 2 | 3
--+---+--
4 | 5 | 6
--+---+--
7 | 8 | O
```

Nous avons décidé de ne pas mettre 3 car le robot ne sera pas optimal.

Implémentation des robots dans le code d’origine :

Pour implémenter nos robots, nous n'avons pas modifié la fonction "morpion" mais la fonction "quiJoueCoups". Nous vérifions s'il y a des robots dans les joueurs avec ces vérifications :

```
if contient_robot(joueur1) == False and quiJoue == 1:
```

S'il n'y a pas de robot alors le jeu se fait comme la version précédente, sinon il regarde de quelle niveau est le robot et exécute en fonction la bonne fonction.

```
else:
    print(couleur_affichage("jaune",f"Le {joueur1 if quiJoue ==1 else joueur2} réfléchi ..."))
    time.sleep(1.5)
    if niveau == 1:
        dernierCoup = niv1(CaseJoue)

    elif niveau == 2:
        dernierCoup = niv2(CaseJoue,quiJoue)

    else :
        dernierCoup = niv3(CaseJoue,quiJoue)
```

Pour les deux premiers niveaux l'algorithme est rapide mais pour le niveau 3 il prend environ 3 secondes à s'exécuter.
Pour le niveau 1, ce robot perd la plupart du temps.
Pour le niveau 2, il a gagnera pas beaucoup car il ne bloque pas la victoire de son adversaire, mais si l'adversaire fait une erreur d'attention il peut perdre.
Pour le niveau 3, il gagnera 25 fois, et on ne pourra pas le battre.

Codage du menu principal :

Par rapport à notre dernière SAE, nous avons décidé d'ajouter plusieurs fonctions qui nous permettront d'ajouter les fonctionnalités demandées. Tout d'abord voici les 8 fonctions ajoutés :

- contient_robot
- pileouface
- conversion_chiffrelettre
- proposition_difficulte
- proposition_robot_joueur
- test_niveau
- adaptation
- testfichiers

Contient_robot :

Cette fonction nous permet de renvoyer vrai ou faux si la variable "texte" (chaîne de caractères) prise en paramètre, contient "Robot" dans la chaîne de caractère.

```python
def contient_robot(texte: str) -> bool:
    """
    Vérifie si le texte contient le mot 'robot' ou une partie de mot contenant 'robot'.

    Paramètre :

        - texte (str) : La chaîne de caractères à vérifier.

    Retourne :

        - bool : True si 'robot' est présent, False sinon.

    """

    return "Robot" in texte.lower()
```

pileouface :

pileouface est une fonction reprenant le célèbre jeu du pile ou face qui permet de partager par le hasard deux joueurs. Pour cela on utilisera la bibliothèque random qui nous permet de choisir un chiffre aléatoire entre un minimum et un maximum.

La fonction prend en paramètre le nom du joueur 1 et du joueur 2, elle possède aussi 3 variables :

- pileouface : qui nous permettra de savoir sur quel côté tombe la pièce
- choixjoueur1 : qui est le choix du côté de la pièce par le joueur 1
- tmp : qui nous servira de variable temporaire pour échanger les joueurs de place

On notera avant tout que le côté pile est égale à 1 et face à 2
Nous commencerons par demander quel est le côté de la pièce que le joueur souhaite choisir. Si le joueur 1 est un robot, il choisira aléatoirement entre 1 et 2.

```python
if contient_robot(joueur1):
    choixjoueur1 = int(randint(1,2))
    if choixjoueur1 == 1:
        print(" ")
        print(couleur_affichage("cyan",f"{joueur1} a choisi pile !"))
    else:
        print(" ")
        print(couleur_affichage("cyan",f"{joueur1} a choisi face !"))
else:
    choixjoueur1 = erreur_entree(1,2,couleur_affichage("cyan","Saississer 1 pour pile ou 2 pour face : "),
                                 couleur_affichage("rouge", "❌ La valeur saisie doit être un nombre entier positif."),
                                 couleur_affichage("rouge", "⚠️  La valeur doit être un entier compris entre 1 et 2."))
```

Ensuite nous lancerons la pièce, simplement par un un randint entre 1 et 2.

On affichera le côté de la pièce qui est retombée.

```
if pileouface == 1:
    print(" ")
    print(couleur_affichage("bleu","La pièce est retomber sur pile !"))
else:
    print(" ")
    print(couleur_affichage("bleu","La pièce est retomber sur face !"))
```

Puis nous testerons si le choix du joueur 1 est égale à “pileouface”, si c’est le cas le joueur 1 à la main, sinon nous échangerons de place les joueurs, puis nous renverrons un tuple du nom du joueur 1, du joueur 2, et la valeur 1 qui nous permettra de savoir qui joue.

```
if pileouface == choixjoueur1:
    print(" ")
    print(couleur_affichage("cyan",f"{joueur1} à la main !"))
else:
    print(" ")
    print(couleur_affichage("cyan",f"{joueur2} à la main !"))
    tmp = joueur1
    joueur1 = joueur2
    joueur2 = tmp


print(" ")
print(couleur_affichage("magenta",f"Le joueur 1 est : {joueur1}"))
print(couleur_affichage("magenta",f"Le joueur 2 est : {joueur2}"))

return joueur1,joueur2,1
```

conversion_chiffrelettre :

Cette fonction permet de convertir un nombre écrit en lettre en un nombre écrit en chiffre. Elle prend en paramètre un “choix” de l’utilisateur et le “message” de demande, qui sont tous les deux des chaînes de caractères.
Il y a plusieurs variables, deux d’entre elles sont des dictionnaires contenant pour chaque clé une chaîne de caractères et pour valeurs un entier. Ces deux variables nous permettent de contenir les nombres de 1 à 100 en lettre ou bien entier. Pourquoi 100 ? Cette valeur correspond à la valeur maximale que notre programme demande (dans le cas de devinette). Voici un exemple de “dictionnairenombrelettre” et “dictionnairenombrelettreentier” :

```
dictionnairenombrelettre = {
"un": 1,
"deux": 2,
"trois": 3,
"quatre": 4,
"cinq": 5,
"six": 6,
"sept": 7,
"huit": 8,
"neuf": 9,
"dix": 10,
"onze": 11,
"douze": 12,
"treize": 13,
"quatorze": 14,
"quinze": 15,
"seize": 16,
"dix-sept": 17,
"dix-huit": 18,
"dix-neuf": 19,
"vingt": 20,
"vingt-et-un": 21,
"vingt-deux": 22,
"vingt-trois": 23,
```

```
dictionnairenombrelettreentier = {
    "1": 1,
"2": 2,
"3": 3,
"4": 4,
"5": 5,
"6": 6,
"7": 7,
"8": 8,
"9": 9,
"10": 10,
"11": 11,
"12": 12,
"13": 13,
"14": 14,
"15": 15,
"16": 16,
"17": 17,
"18": 18,
"19": 19,
"20": 20,
"21": 21,
"22": 22,
"23": 23,
```

Nous créerons une boucle “tant que” qui tourne tant que la variable “tourne” est vrai ainsi nous l’initialiserons à “True”.
Lors de cette boucle nous balaierons les deux dictionnaires avec deux boucles “pour”, qui pour chaque itérations testerons si la valeur de l’itération est égale au “choix” demander par l’utilisateur si c’est le cas on mettra le contenu du “choix” de l’utilisateur dans “nombre”. Puis après ces boucles, nous referons un test pour savoir si “nombre” est différent de “choix”, si c’est le cas nous afficherons un message d’erreur, puis nous redemanderons à l’utilisateur de refaire son choix et nous changerons le contenu de “tourne” à “True”. Sinon nous changerons le contenu de “tourne” à False.

```
while tourne:

    for chiffres in dictionnairenombrelettre:
        if chiffres == choix:
            nombre = choix

    for chiffres in dictionnairenombrelettreentier:
        if chiffres == choix:
            nombre = choix

    if nombre != choix:
        print(couleur_affichage("rouge","❌ Vous n'avez pas saisie la bonne valeur, recommencez ! "))
        choix = str(input(message))
        tourne = True
    else:
        tourne = False
```

Ensuite nous testerons si “nombre” appartient à “dictionnairenombrelettre”, si oui on retournera la valeur de “dictionnairenombrelettre[nombre]” sinon “dictionnairenombrelettreentier[nombre]” :

```
    if nombre in dictionnairenombrelettre :
        return int(dictionnairenombrelettre[nombre])
    else:
        return int(dictionnairenombrelettreentier[nombre])
```

propostion_difficulte et propostion_robot_joueur :

Ces deux fonctions permettent d'afficher et de proposer soit la difficulté du robot "propostion_difficulte" soit le mode de jeu "propostion_robot_joueur". Elles ne prennent aucun paramètres et renvoient un entier. Leur structure est composée majoritairement de "print".

```
print(couleur_affichage("cyan","Il existe 3 niveaux de difficulté"))
print("")
print(couleur_affichage("cyan","    - 1. Facile (Si vous gagnez cela vous rapportera moins de points)"))
print(couleur_affichage("cyan","    - 2. Moyen (Si vous gagnez cela vous rapportera plus de points que le niveau facile)"))
print(couleur_affichage("cyan","    - 3. Difficile (Si vous gagnez cela vous rapportera plus de points) \n"))

entree_utilisateur = erreur_entree(1,3,couleur_affichage("cyan","➡️  Quelle difficulté choisissez vous? : "),
                                   couleur_affichage("rouge","❌ La valeur entrée doit être un entier positif"),
                                   couleur_affichage("rouge","❌ La valeur doit être comprise entre 1 et 3"))
```

```
print(couleur_affichage("cyan","Les différents modes de jeu : \n"))
print(couleur_affichage("cyan","    - 1 : Joueur contre Joueur"))
print(couleur_affichage("cyan","    - 2 : Joueur contre Robot"))
print(couleur_affichage("cyan","    - 3 : Robot contre Robot\n"))

choix = erreur_entree(1,3,couleur_affichage("cyan","➡️  Quelle est votre choix (1,2 ou 3) : "),
couleur_affichage("rouge","❌  Il faut soit rentrer 1 pour Joueur/Joueur, 2 pour Joueur/Robot ou 3 pour Robot/Robot"),
couleur_affichage("rouge","❌  Vous avez dépassé la limite il faut soit rentrer un ou trois"))

return choix
```

adaptation :

Cette fonction permet de demander à l'utilisateur le nom du joueur 1, du joueur 2, ainsi que la difficulté des ou du robot en fonction de son choix pris en paramètre.
Si le choix est égal à 1, on demandera donc le nom du joueur 1 puis le nom du joueur 2, et on initialisera le niveau des robots à 0.

```
if choix == 1:
    joueur1 = erreur_entree_invalide_str(choix_invalides=["", " "],
    message=couleur_affichage("cyan","Joueur 1: veuillez choisir votre pseudo : "),
    message_erreur = couleur_affichage("rouge","Le pseudo ne doit pas être vide, contenir uniquement des espaces, contenir 'bot' dans son pseudo ou être invalide.")).strip()
    joueur2 = erreur_entree_invalide_str(choix_invalides=["", " ", joueur1],message=couleur_affichage("jaune","Joueur 2: veuillez choisir votre pseudo : "),
    message_erreur=couleur_affichage("rouge","Le pseudo ne doit pas être vide, contenir uniquement des espaces, contenir 'bot' dans son pseudo ou être identique à celui du Joue
    niveau = 0
```

Aussi, on notera qu'on enlèvera les espaces lors de la demande des pseudos des joueurs, pour une question de meilleur affichage.

Si le choix est égal à 2, on demandera le nom du joueur 1 uniquement, puis on demandera de choisir les difficultés du robot à l'utilisateur, qu'on stockera dans "niveau", et on nommera le joueur 2 avec "Robot_" et sa difficulté tester avec la fonction "test_niveau".

```
elif choix == 2:
    joueur1 = erreur_entree_invalide_str(choix_invalides=["", " "],message=couleur_affichage("cyan","Joueur 1: veuillez choisir votre pseudo : "),message_erreur = couleur_affich
    niveau = proposition_difficulte()
    joueur2 = "Robot_" + test_niveau(niveau)
```

Sinon, on demandera à l'utilisateur de choisir la difficulté des robots, puis on nommera le joueur 1 et le joueur 2, qui seront 2 robots, en fonction de leur niveau.

```
else:
    print(couleur_affichage("cyan","Choississer la difficulté des Robots : "))
    niveau = proposition_difficulte()
    joueur1 = "Robot1_" + test_niveau(niveau)
    joueur2 = "Robot2_" + test_niveau(niveau)
```

Pour finir, nous renverrons le nom du joueur 1, du joueur 2 et le niveau de difficulté des ou du robot.

```
return joueur1,joueur2,niveau
```

testfichiers :

Cette fonction permet de tester si les fichiers des jeux sont présents dans le même dossier que “menu.py”, donc “JeuMorpion.py”,”JeuAllumettes.py” et “JeuDevinette.py”. Elle ne prend aucun paramètre et renvoie un booléen.
On déclare donc une variable “fichierimport” qui est une liste de chaîne de caractères contenant le noms des fichiers contenant le code des jeux. Puis nous balaierons, grâce une boucle “pour” cette liste et pour chaque itérations nous testerons grâce au bloc “try” si nous pouvons importer dynamiquement, grâce à la bibliothèque “importlib”, un des fichiers de la liste “fichierimport”, si nous ne pouvons pour un seul des cas, nous renverrons “False” grâce au bloc “except ModuleNotFoundError”. Si aucune erreur est levée nous retournerons simplement “True”.

Par la suite nous avons décidé de modifier plusieurs éléments du code “menu.py” :

- Ajout de 2 sous-menu dans le menu principal
- Ajout des règles des difficulté des robots
- Modification de l’ajout des scores et de son affichage

Ajout de 2 sous-menu dans le menu principal :

Tout d’abord, nous avons décidé d’ajouter 2 sous menu dans le menu principal, le premier étant la modification du mode de jeu et le deuxième étant la modification de la difficulté des ou du robot.
Pour le premier, nous demanderons à l’utilisateur le mode de jeu qu’il souhaite jouer, ensuite nous nous servirons de la fonction adaptation pour changer le nom du joueur 1, du joueur 2 et le niveau de difficultés des robots, puis en fonction du choix du mode de jeu nous afficherons les changements, c’est-à-dire le mode de jeu, la difficulté du ou des robots, et le nom des joueurs.

```
case 6:
    # Modifier le mode de jeu
    choix_mode = proposition_robot_joueur()
    joueur1,joueur2,niveau = adaptation(choix_mode)
    print(couleur_affichage("bleu","Mode de jeu changé en : "))
    if choix_mode == 1:
        print(" ")
        print(couleur_affichage("bleu","Joueur contre Joueur"))
        print(" ")
        print(couleur_affichage("bleu","Les joueurs sont : "))
        print(couleur_affichage("bleu",f"           -{joueur1} : joueur 1"))
        print(couleur_affichage("bleu",f"           -{joueur2} : joueur 2"))
    elif choix_mode == 2:
        print(" ")
        print(couleur_affichage("bleu","Joueur contre Robot"))
        print(" ")
        print(couleur_affichage("bleu","Le joueur est : "))
        print(couleur_affichage("bleu",f"           -{joueur1} : joueur 1"))
        print(" ")
        print(couleur_affichage("bleu","Et le robot est de difficulté : "))
        print(couleur_affichage("bleu",f"           -{joueur2} : joueur 2"))
    else:
        print(" ")
        print(couleur_affichage("bleu","Robot contre Robot"))
        print(" ")
        print(couleur_affichage("bleu","Les robots sont de difficultés : "))
        print(" ")
        print(couleur_affichage("bleu",f"           -{joueur1} : joueur 1"))
        print(couleur_affichage("bleu",f"           -{joueur2} : joueur 2"))
```

Pour le deuxième :

On testera si le choix du mode est joueur contre joueur, si c'est le cas nous afficherons un message d'erreur puisque l'utilisateur ne joue contre aucun robot.
Sinon, nous proposerons à l'utilisateur de changer le niveau de difficulté puis nous afficherons le niveau de difficulté choisi.

```
case 7:
    # Modifier la difficulté
    if choix_mode == 1:
        print(couleur_affichage("rouge","❌ Impossible de changez de difficulté !"))
        print(couleur_affichage("rouge","❌ Vous ne jouez contre aucun robot !"))
    else:
        niveau = proposition_difficulte()
        print(" ")
        print(couleur_affichage("bleu",f"Niveau de difficulté changé en : {niveau}"))
```

Bien sûr, nous avons modifier l'affichage de la fonction "propostion" pour faire afficher ces choix présents.

Ajout des règles des difficulté des robots :

Nous avons également ajouté un sous menu dans le menu des règles, qui nous permet d'afficher le nombre de points remporté par l'utilisateur en fonction de la difficulté du robot.

```
case 4:
    print(" ")
    print(couleur_affichage("cyan","Voici le nombre de points remportés en fonction de la difficulté : "))
    print(" ")
    print(couleur_affichage("cyan","    - La difficulté facile vous rapporte 1 point"))
    print(couleur_affichage("cyan","    - La difficulté moyen vous rapporte 2 point"))
    print(couleur_affichage("cyan","    - La difficulté difficile vous rapporte 5 point"))
    print(" ")
```

Modification de l'ajout des scores et de son affichage :

Tout d'abord, nous avons revu l'ajout des scores de façon à ce que l'ajout se fasse dans un fichier json. Pourquoi ? Le fichier est json est bien plus simple à manipuler puisqu'il reprend la structure d'un dictionnaire, son affichage est également plus agréable et il répondait à nos attentes car nous avons décidé d'ajouter nos scores de façon à ce qu'il soit plus lisible :

```
Partie joueur1-joueur2 {
        devinette : 0-0
        allumettes : 1-1
        morpion : 2-2
}
```

Notre fonction "ajout_score" prendra donc en paramètre "chemin_fichier" une chaîne de caractère qui sera le nom du fichier json, cela permettra aux futurs développeur de pouvoir utiliser plusieurs fichiers json pour une meilleure gestion d'affichage, on aura également le nom du joueur 1, du joueur 2, du gagnant et du jeu en paramètre, ce seront tous des chaînes de caractères, on prendra en paramètre aussi le niveau des robots qui seront des entiers. La fonction ne renverra rien.

Ensuite, nous déclarons 5 variables :

- cle_partie (str) : ce sera la clé des parties du dictionnaires
- score_j1 (int) : le score du joueur 1
- score_j2 (int) : le score du joueur 2
- f : un objet de TextIO
- scoredesjoueurs (liste[str]) : le score des deux joueurs sous le format (scorej1-scorej2)
- data (dict[str,dict[str,str]]) : C'est le contenu du fichier json

La structure du code est donc composée de 2 blocs, un bloc "try" et un bloc "except ValueError". Dans le cas du bloc "except ValueError", on affichera un message d'erreur à l'utilisateur en lui demandant de supprimer le fichier.

```
except ValueError:
    print(couleur_affichage("rouge","Une erreur est survenue"))
    print(couleur_affichage("rouge","Le fichier est surement corrompu, supprimez le et le problème sera résolu"))
```

Ensuite, dans le bloc “try”, on effectuera un test pour savoir si le joueur 1 et le joueur 2 ne sont pas des robots, grâce à la fonction “contient_robot” vu précédemment. Si ce n’est pas le cas nous n’ajouterons pas les scores puisque leurs scores ne servent à rien.

```
else:
    print(" ")
    print(couleur_affichage("magenta","Nous n'ajouterons pas les scores puisque les deux joueurs sont des robots !"))
```

Si c’est le cas, nous testerons si le fichier correspondant au fichier renseigné par le développeur existe ou non, si c’est le cas nous le créerons en créant un dictionnaire vide puis en ouvrant le fichier en écriture et en écrivant le contenu de data dans le fichier. Sinon, on stockera simplement le contenu du fichier json dans la variable data.

```
if not os.path.exists(chemin_fichier):
    data = {}
    with open(chemin_fichier, 'w') as f:
        json.dump(data, f, indent=4)
else:
    with open(chemin_fichier, 'r') as f:
        data = json.load(f)
```

Par la suite, nous créerons une clé de la partie des 2 joueurs en la stockant dans la variable “cle_partie” puis nous effectuerons deux tests, le premier qui nous permet de savoir si la partie entre les deux joueurs n’existe pas, si c’est le cas nous la créons. Le deuxième tests nous permet de savoir si le score du “jeu” n'existe pas, si c’est le cas nous allons l’initialiser.

```
cle_partie = f"{joueur1}-{joueur2}"

if cle_partie not in data:
    data[cle_partie] = {}

if jeu not in data[cle_partie]:
    data[cle_partie][jeu] = "0-0"
```

Nous stockerons ensuite le contenu des scores en le récupérant grâce aux clés “cle_partie” et “jeu” dans la variable “score” puis nous stockerons les scores des 2 joueurs dans une liste qui pour chaque indice contiendra le score d’un joueur. Puis nous les stockerons dans “score_j1” et “score_j2”.
On ajoutera les points en fonction du gagnant et du niveau de difficulté si c’est un robot. Puis nous modifierons les scores et nous écraserons le contenu du fichier json avec notre contenu stocké dans la variable “data”

```
scoredesjoueurs = score.split("-")

score_j1 = int(scoredesjoueurs[0])
score_j2 = int(scoredesjoueurs[1])

if niveau == 3:
    if gagnant == joueur1:
        score_j1 += 5
    else:
        score_j2 += 5
elif niveau == 2:
    if gagnant == joueur1:
        score_j1 += 2
    else:
        score_j2 += 2
else:
    if gagnant == joueur1:
        score_j1 += 1
    else:
        score_j2 += 1


data[cle_partie][jeu] = f"{score_j1}-{score_j2}"

with open(chemin_fichier, 'w') as f:
    json.dump(data, f, indent=4)
    print(couleur_affichage("magenta","Les scores ont bien étaient modifiés !"))
```

Passons à l'affichage des scores, la fonction "afficherscore" prend en paramètre le nom du joueur 1 et le nom du joueur 2 et elle ne renvoie rien.

Elle possède 12 variables, dont 5 qui reprennent la structure d'un menu. Cette fonction utilise 2 blocs principaux, "try" et "except ValueError". Le bloc "except ValueError" permet d'afficher à l'utilisateur qu'aucun fichier n'est disponible et qu'il faut en créer un en lançant une partie.

```
except ValueError:
    print(couleur_affichage("rouge","Aucun fichier disponible"))
    print(couleur_affichage("rouge","Veuillez en créer un en lancant une partie \n"))
```

Le bloc "try" utilise une boucle "tant que" qui tourne tant que la variable "quitter" est Vrai. On demande ensuite le choix de l'utilisateur donc, soit d'afficher le score, soit de quitter. On utilise ensuite une structure de "match and case" ou pour le cas où l'utilisateur saisie de quitter, on lui demande s' il veut réellement quitteri.

```
case 2:

    souhait = erreur_entree_str(["o", "oui", "non", "n"], couleur_affichage("magenta", "  Souhaitez-vous quitter ? (oui/o, non/n) : "),couleur_affichage("rouge", "❌ Entrée i

    if souhait in ["o", "oui"]:
        print(couleur_affichage("jaune", "Vous avez quitter le sous menu !"))
        quitter = False
```

Puis on a le cas par défaut qui affiche que l'utilisateur à saisi un choix invalide.

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

Ensuite, si l'utilisateur saisit le cas 1 on demande dans un premier temps si il souhaite afficher le score de la partie en cours. Si il répond oui on affiche simplement le nom des deux joueurs liés d'un tiret du 6 "-" puis on affiche les scores.

```
with open("Scores.json", 'r') as f:
    data = json.load(f)

if affichage in ["oui","o"]:
    print(" ")
    cle_partie = f"{joueur1}-{joueur2}"
    print(couleur_affichage("cyan",cle_partie))
    parties = str(data[cle_partie])
    print(couleur_affichage("cyan",parties))
```

Si il répond non, on va trier les scores dans l'ordre alphabétique, qu'on stockera dans "data_triees", grâce aux fonctions python "dict" et "sorted". Puis on les affichera dans l'ordre grâce à une boucle "pour". Une fois l'affichage terminé on demandera à l'utilisateur d'appuyer sur entrée pour signaler que l'affichage des scores entiers est terminé.

```
print(" ")
print(couleur_affichage("vert","Voici tous les scores triés par ordre alphabétique : "))
print(" ")

data_triees = dict(sorted(data.items()))

for parties in data_triees:
    print(couleur_affichage("cyan",parties))
    print(couleur_affichage("cyan",f"{data_triees[parties]}\n"))


input(couleur_affichage("magenta","Appuyer sur entrée . . ."))
```

Ensuite, nous proposerons à l'utilisateur s'il souhaite afficher une partie en particulier, s'il répond oui nous lui demanderons le nom d'un des joueurs de la partie. Puis nous balaierons les parties des joueurs triées par ordre alphabétique et nous afficherons la partie seulement si le nom du joueur donné par l'utilisateur est présent. Grâce à la variable "trouve" nous pourrons effectuer un test pour savoir si au moins une partie à été trouvée, sinon on affichera qu'aucune partie n'est trouvée.

```
choixpartie = erreur_entree_str(["o", "oui", "non", "n"], couleur_affichage("magenta", "🔍 Souhaitez-vous chercher une partie en particulier ? (oui/o, non/n) : "),coul

if choixpartie in ["o","oui"]:
    joueur1 = str(input(couleur_affichage("magenta","Quel est le nom d'un des joueurs de la partie ? : ")))
    print(" ")

    for parties in data_triees:
        if joueur1 in parties:
            trouve = True
            print(couleur_affichage("cyan",parties))
            print(couleur_affichage("cyan",f"{data_triees[parties]}\n"))

    if not trouve:
        print(couleur_affichage("rouge","Aucunes parties n'a été trouvées !"))
```

Par la suite, nous demanderons si l'utilisateur veut afficher une partie en particulier, si oui il faudra qu'il donne le nom du second joueur de la partie et on affichera donc la partie.

```
choixpartie = erreur_entree_str(["o", "oui", "non", "n"], couleur_affichage("magenta", "  Souhaitez-vous afficher une partie en particulier ?

if choixpartie in ["oui","o"]:

    joueur2 = str(input(couleur_affichage("magenta","Quel est le nom du second joueur de la partie ? : ")))

    cle_partie1 = f"{joueur1}-{joueur2}"
    cle_partie2 = f"{joueur2}-{joueur1}"

    if cle_partie1 in data_triees:
        print(couleur_affichage("cyan", cle_partie1))
        print(couleur_affichage("cyan", f"{data_triees[cle_partie1]}\n"))
    elif cle_partie2 in data_triees:
        print(couleur_affichage("cyan", cle_partie2))
        print(couleur_affichage("cyan", f"{data_triees[cle_partie2]}\n"))
    else:
        print(couleur_affichage("rouge", "La partie entre ces deux joueurs n'existe pas."))
```

Jeux de test :

Nous avons décidé de ne pas remettre les jeux de tests entre Joueur contre Joueur car ils sont dans le compte rendu S1.01 en lien dans le fichier zip.

Devinette

Les erreurs de saisie sont gérées de la même manière qu'au premier compte rendu Pour tous les jeux de tests la valeur caché sera 24, si nous faisons deviner le nombre.

Joueur contre Robot :

- Niveau 1 :

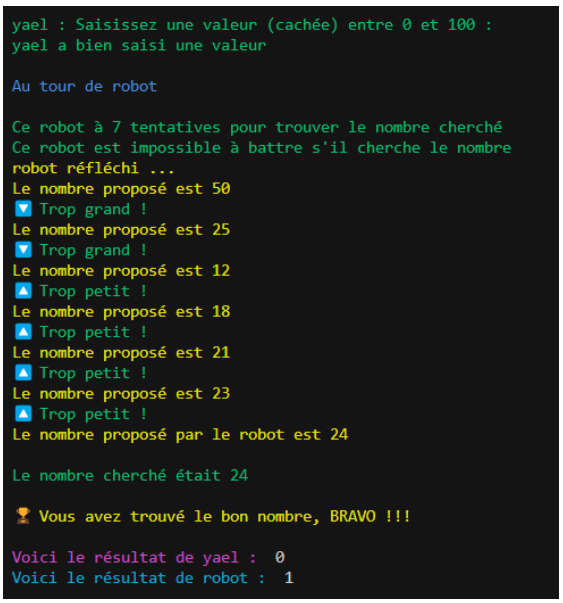
```
Robot_Facile : Saisissez une valeur (cachée) entre 0 et 100 :
Robot_Facile a bien saisi une valeur

Au tour de yael

yael : Vous avez 7 tentatives pour trouver le nombre, saisissez votre valeur : quinze
🔽 Trop grand ! yael vous devez essayer une autre valeur : """
❌ Vous n'avez pas saisie la bonne valeur, recommencez !
🔽 Trop grand ! yael vous devez essayer une autre valeur : fzfz
❌ Vous n'avez pas saisie la bonne valeur, recommencez !
🔽 Trop grand ! yael vous devez essayer une autre valeur : 45
🔽 Trop grand ! yael vous devez essayer une autre valeur : douze
🔽 Trop grand ! yael vous devez essayer une autre valeur : cinq
🔽 Trop grand ! yael vous devez essayer une autre valeur : trois
🔽 Trop grand ! yael vous devez essayer une autre valeur : un

Le nombre cherché était 1

🏆 Vous avez trouvé le bon nombre, BRAVO !!!
```

```
yael : Saisissez une valeur (cachée) entre 0 et 100 :
yael a bien saisi une valeur

Au tour de robot

Ce robot est facile à battre donc au lieu d'avoir 5 tentatives, il en a 7.
robot réfléchi ...
Le nombre proposé est 45
🔽 Trop grand !
Le nombre proposé est 2
🔼 Trop petit !
Le nombre proposé est 3
🔼 Trop petit !
Le nombre proposé est 43
🔽 Trop grand !
Le nombre proposé est 20
🔼 Trop petit !
Le nombre proposé est 41
🔽 Trop grand !
Le nombre proposé par le robot est 26

Le nombre cherché était 24

📋 Vous avez fait trop tentatives sans trouver le nombre, c'est maintenant à l'autre joueur de jouer

Voici le résultat de yael :  1
Voici le résultat de robot :  0
```

- Niveau 2 :

```
yael : Saisissez une valeur (cachée) entre 0 et 100 :
yael a bien saisi une valeur

Au tour de robot

robot réfléchi ...
Le nombre proposé est 50
🔽 Trop grand !
Le nombre proposé est 25
🔽 Trop grand !
Le nombre proposé est 12
🔼 Trop petit !
Le nombre proposé est 18
🔼 Trop petit !
Le nombre proposé par le robot est 23

Le nombre cherché était 24

📋 Vous avez fait trop tentatives sans trouver le nombre, c'est maintenant à l'autre joueur de jouer

Voici le résultat de yael :  1
Voici le résultat de robot :  0
```

- Niveau 3 :

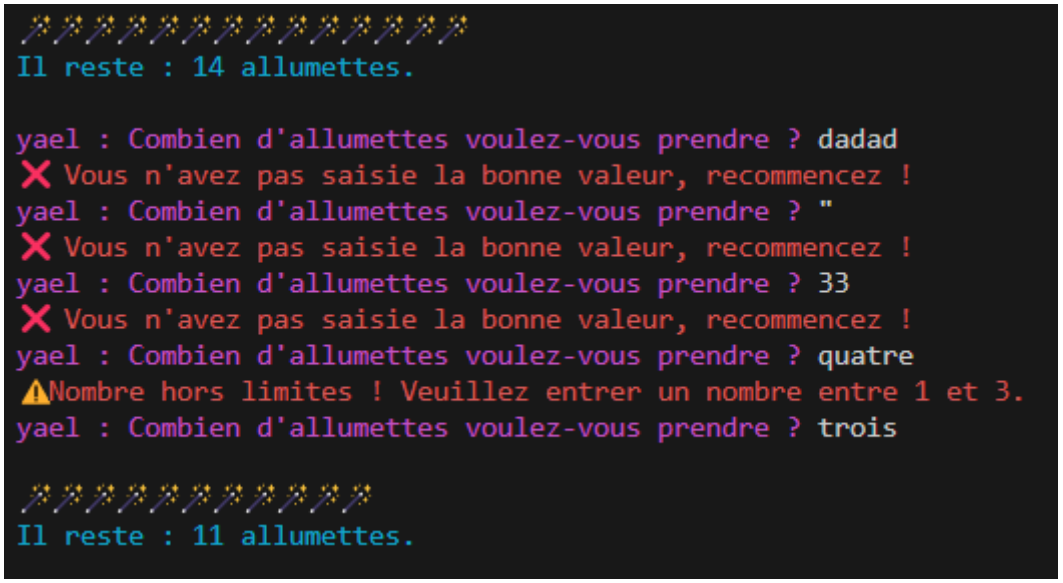


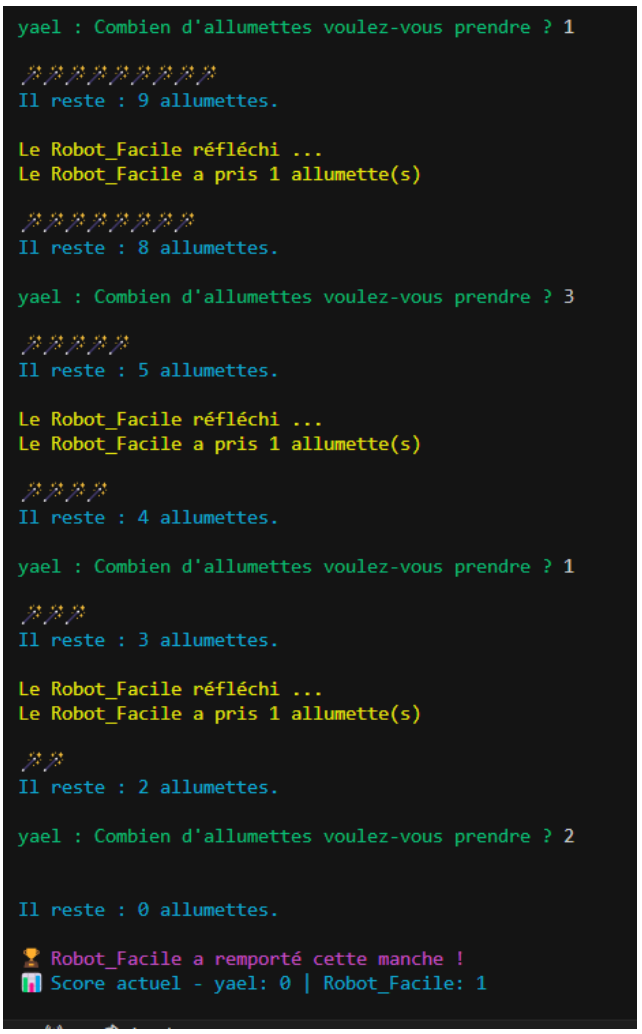
Les parties Robot contre Robot sont similaires à celle ici présente, juste le robot qui propose le nombre choisi aléatoirement entre 0 et 100.

Allumettes

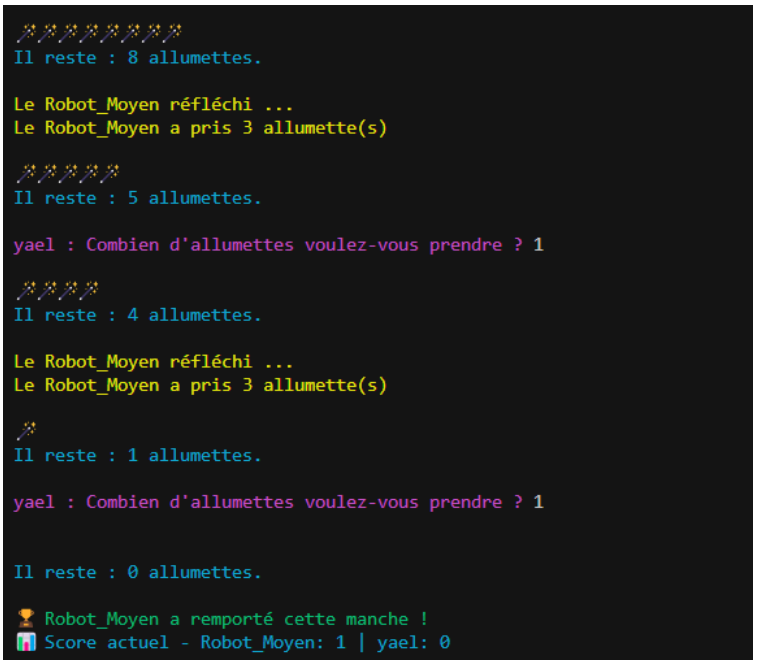
Joueur contre Robot :

- Niveau 1 :

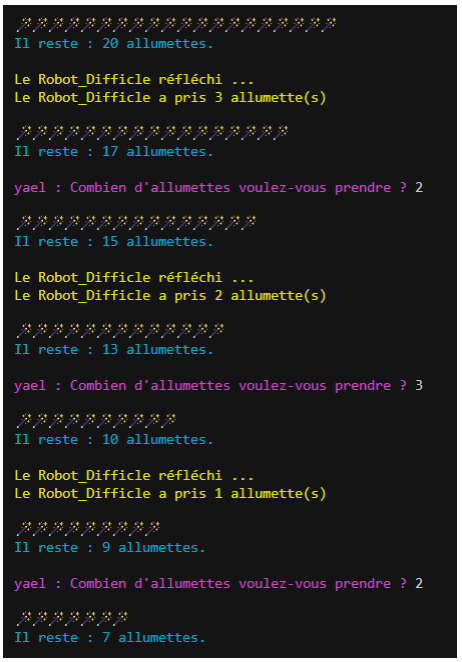


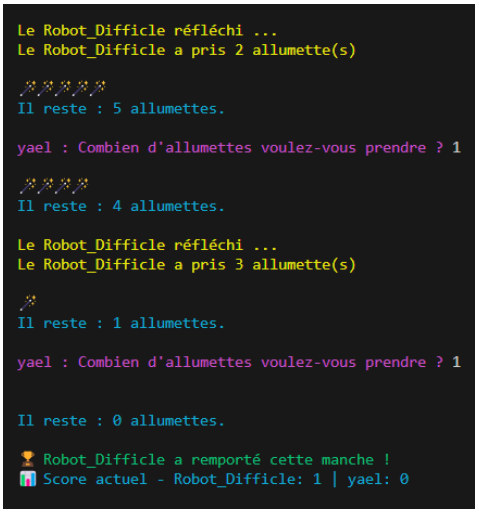

yael : Combien d'allumettes voulez-vous prendre ? 1
Il reste : 9 allumettes.
Le Robot_Facile réfléchi ...
Le Robot_Facile a pris 1 allumette(s)
Il reste : 8 allumettes.
yael : Combien d'allumettes voulez-vous prendre ? 3
Il reste : 5 allumettes.
Le Robot_Facile réfléchi ...
Le Robot_Facile a pris 1 allumette(s)
Il reste : 4 allumettes.
yael : Combien d'allumettes voulez-vous prendre ? 1
Il reste : 3 allumettes.
Le Robot_Facile réfléchi ...
Le Robot_Facile a pris 1 allumette(s)
Il reste : 2 allumettes.
yael : Combien d'allumettes voulez-vous prendre ? 2
Il reste : 0 allumettes.
Robot_Facile a remporté cette manche !
Score actuel - yael: 0 | Robot_Facile: 1

- Niveau 2 :

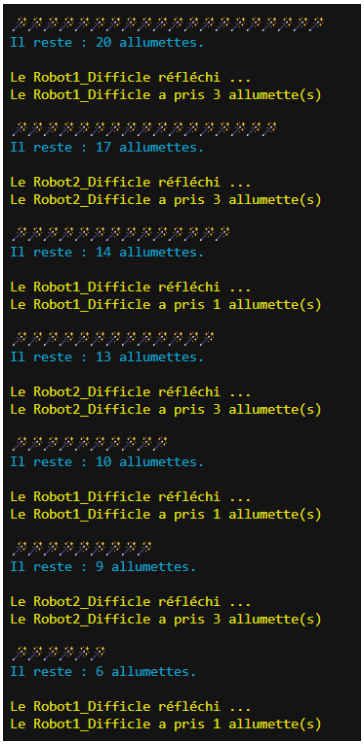

Il reste : 8 allumettes.
Le Robot_Moyen réfléchi ...
Le Robot_Moyen a pris 3 allumette(s)
Il reste : 5 allumettes.
yael : Combien d'allumettes voulez-vous prendre ? 1
Il reste : 4 allumettes.
Le Robot_Moyen réfléchi ...
Le Robot_Moyen a pris 3 allumette(s)
Il reste : 1 allumettes.
yael : Combien d'allumettes voulez-vous prendre ? 1
Il reste : 0 allumettes.
Robot_Moyen a remporté cette manche !
Score actuel - Robot_Moyen: 1 | yael: 0

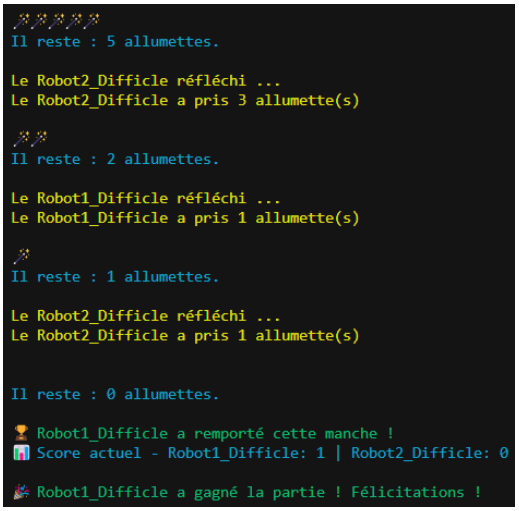
- Niveau 3 :





Robot contre Robot (niveau 3):

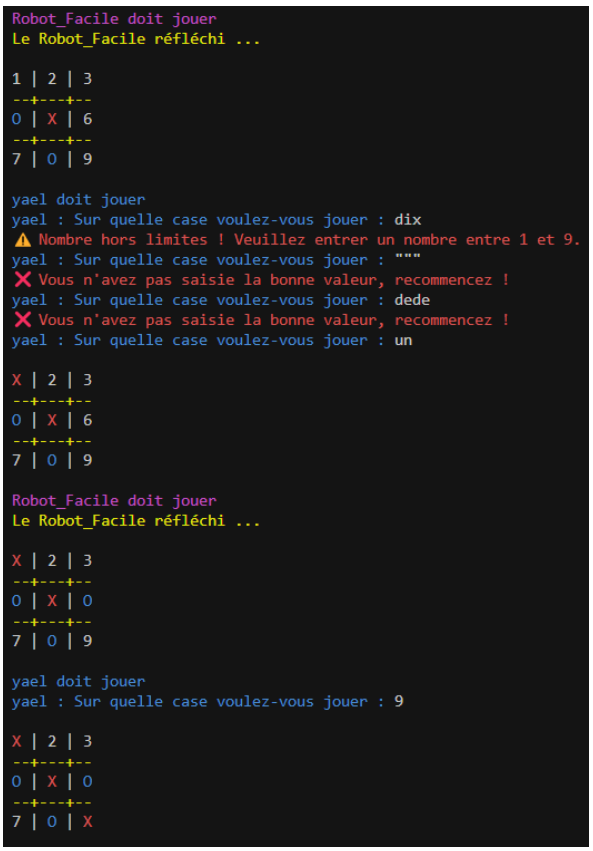




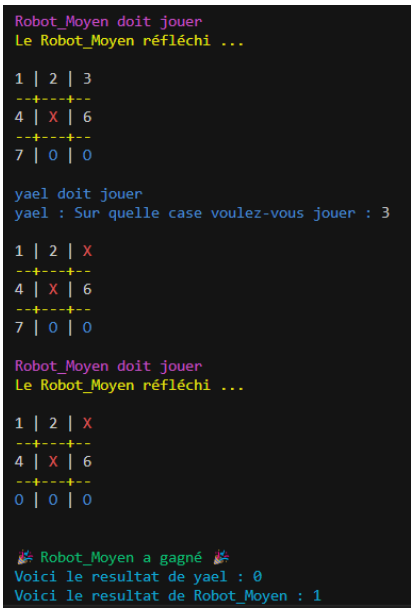
Morpion :

Joueur contre Robot :

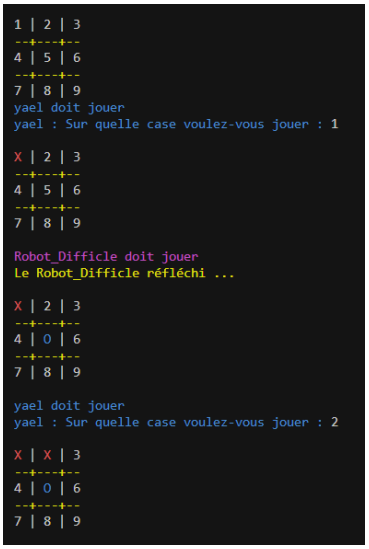
- Niveau 1 :

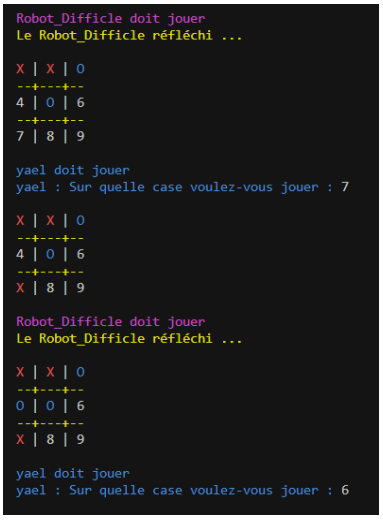

Robot_Facile doit jouer
Le Robot_Facile réfléchi ...
1 | 2 | 3
--+---+--
0 | X | 6
--+---+--
7 | 0 | 9
yael doit jouer
yael : Sur quelle case voulez-vous jouer : dix
⚠️ Nombre hors limites ! Veuillez entrer un nombre entre 1 et 9.
yael : Sur quelle case voulez-vous jouer : """
❌ Vous n'avez pas saisie la bonne valeur, recommencez !
yael : Sur quelle case voulez-vous jouer : dede
❌ Vous n'avez pas saisie la bonne valeur, recommencez !
yael : Sur quelle case voulez-vous jouer : un
X | 2 | 3
--+---+--
0 | X | 6
--+---+--
7 | 0 | 9
Robot_Facile doit jouer
Le Robot_Facile réfléchi ...
X | 2 | 3
--+---+--
0 | X | 0
--+---+--
7 | 0 | 9
yael doit jouer
yael : Sur quelle case voulez-vous jouer : 9
X | 2 | 3
--+---+--
0 | X | 0
--+---+--
7 | 0 | X

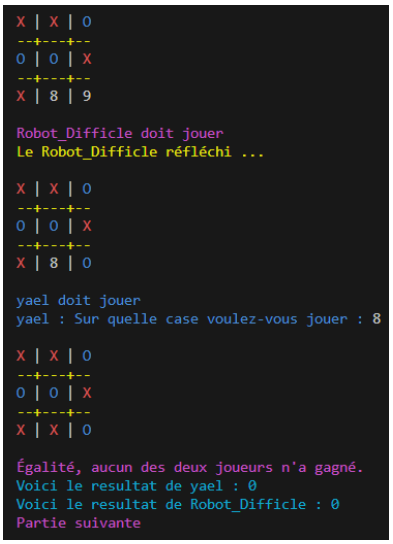
- Niveau 2 :


Robot_Moyen doit jouer
Le Robot_Moyen réfléchi ...
1 | 2 | 3
--+---+--
4 | X | 6
--+---+--
7 | 0 | 0
yael doit jouer
yael : Sur quelle case voulez-vous jouer : 3
1 | 2 | X
--+---+--
4 | X | 6
--+---+--
7 | 0 | 0
Robot_Moyen doit jouer
Le Robot_Moyen réfléchi ...
1 | 2 | X
--+---+--
4 | X | 6
--+---+--
0 | 0 | 0
🎉 Robot_Moyen a gagné 🎉
Voici le resultat de yael : 0
Voici le resultat de Robot_Moyen : 1

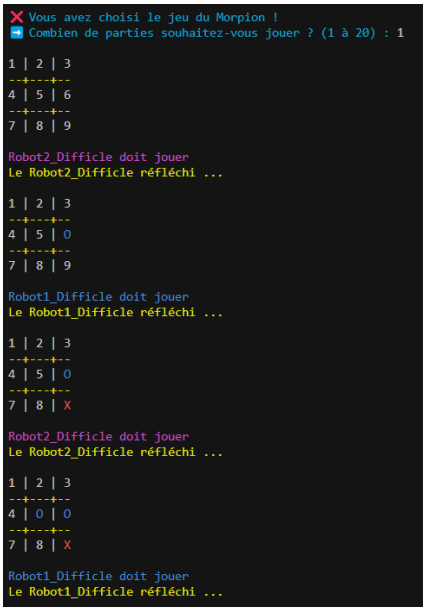
- Niveau 3 :

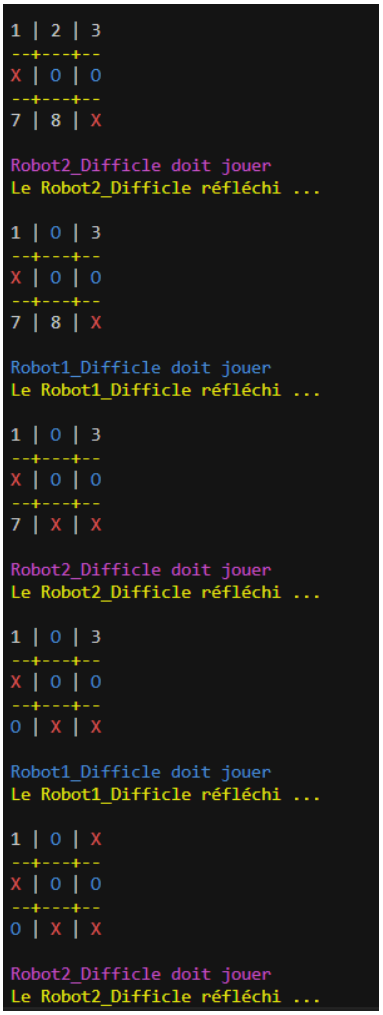

1 | 2 | 3
4 | 5 | 6
7 | 8 | 9
yael doit jouer
yael : Sur quelle case voulez-vous jouer : 1
X | 2 | 3
4 | 5 | 6
7 | 8 | 9
Robot_Difficle doit jouer
Le Robot_Difficle réfléchi ...
X | 2 | 3
4 | O | 6
7 | 8 | 9
yael doit jouer
yael : Sur quelle case voulez-vous jouer : 2
X | X | 3
4 | O | 6
7 | 8 | 9

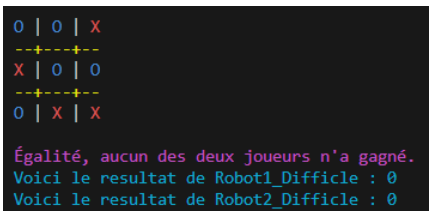

Robot_Difficle doit jouer
Le Robot_Difficle réfléchi ...
X | X | O
4 | O | 6
7 | 8 | 9
yael doit jouer
yael : Sur quelle case voulez-vous jouer : 7
X | X | O
4 | O | 6
X | 8 | 9
Robot_Difficle doit jouer
Le Robot_Difficle réfléchi ...
X | X | O
O | O | 6
X | 8 | 9
yael doit jouer
yael : Sur quelle case voulez-vous jouer : 6


X | X | O
O | O | X
X | 8 | 9
Robot_Difficle doit jouer
Le Robot_Difficle réfléchi ...
X | X | O
O | O | X
X | 8 | O
yael doit jouer
yael : Sur quelle case voulez-vous jouer : 8
X | X | O
O | O | X
X | X | O
Égalité, aucun des deux joueurs n'a gagné.
Voici le resultat de yael : 0
Voici le resultat de Robot_Difficle : 0
Partie suivante

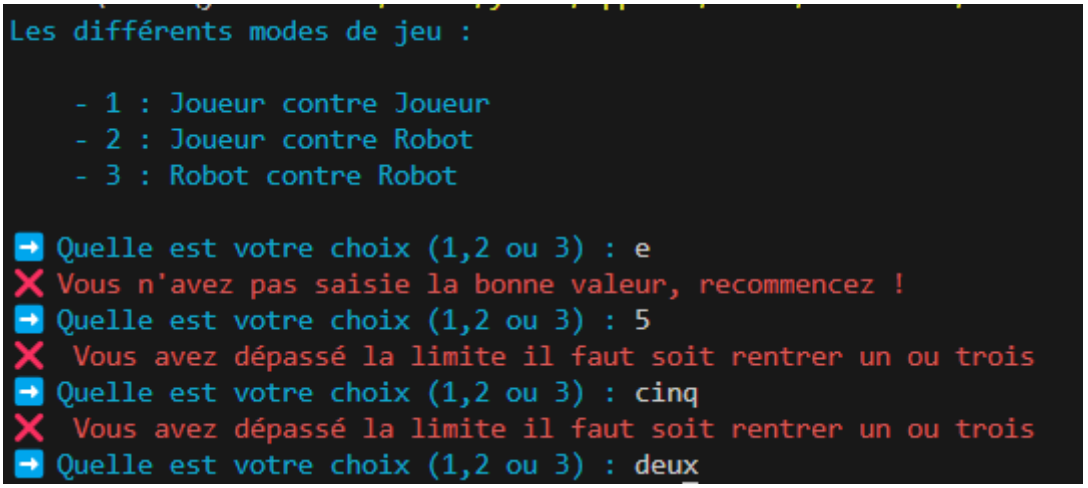
Robot contre Robot (niveau 3):

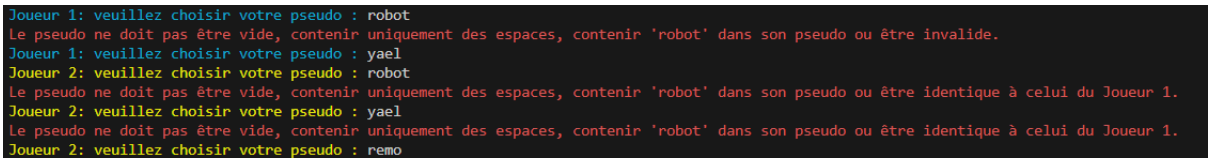


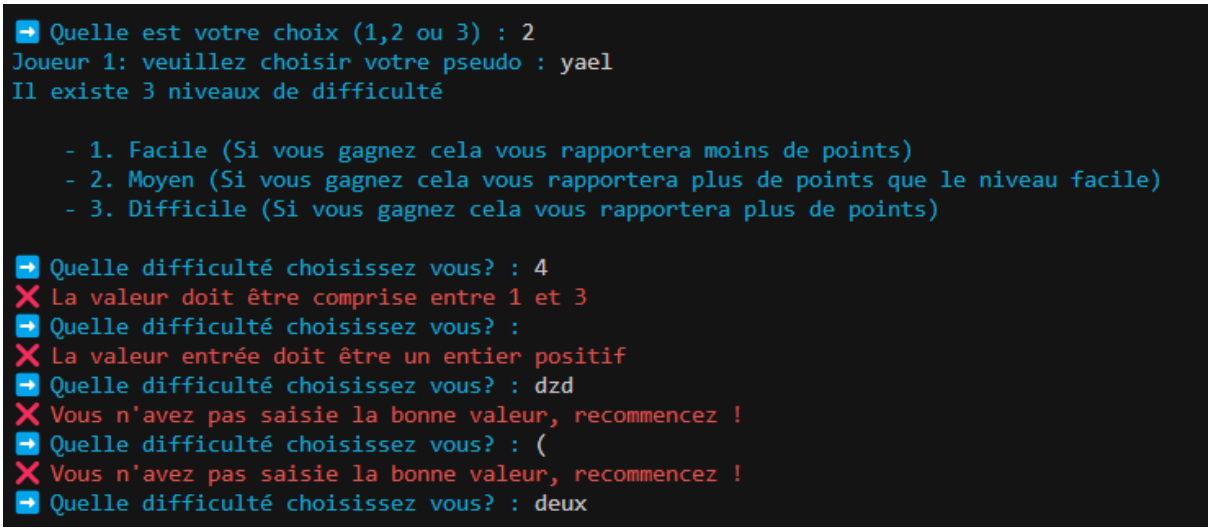


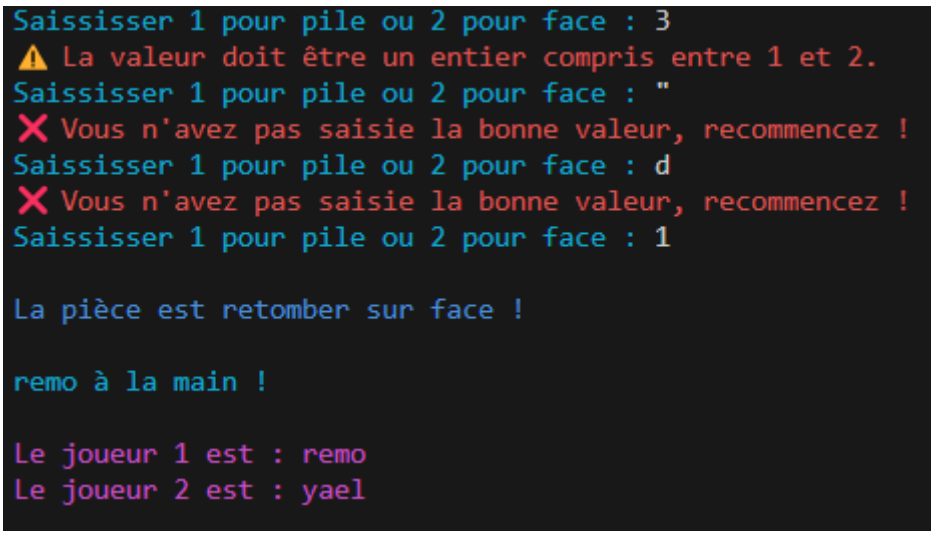

O | O | X
--+---+--
X | O | O
--+---+--
O | X | X
Égalité, aucun des deux joueurs n'a gagné.
Voici le resultat de Robot1_Difficle : 0
Voici le resultat de Robot2_Difficle : 0

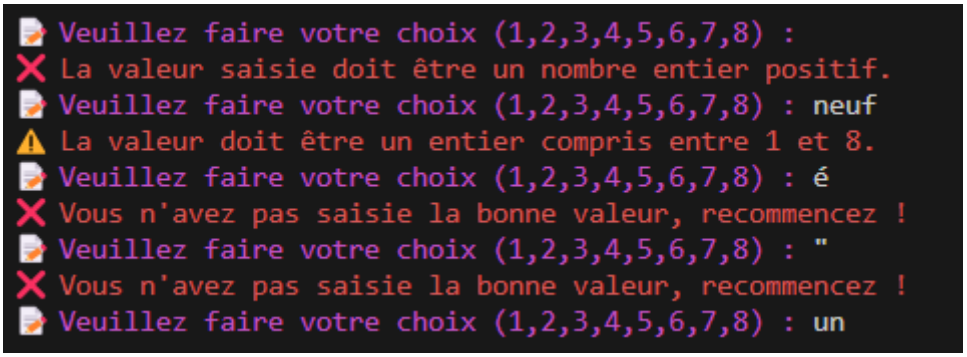
Menu :

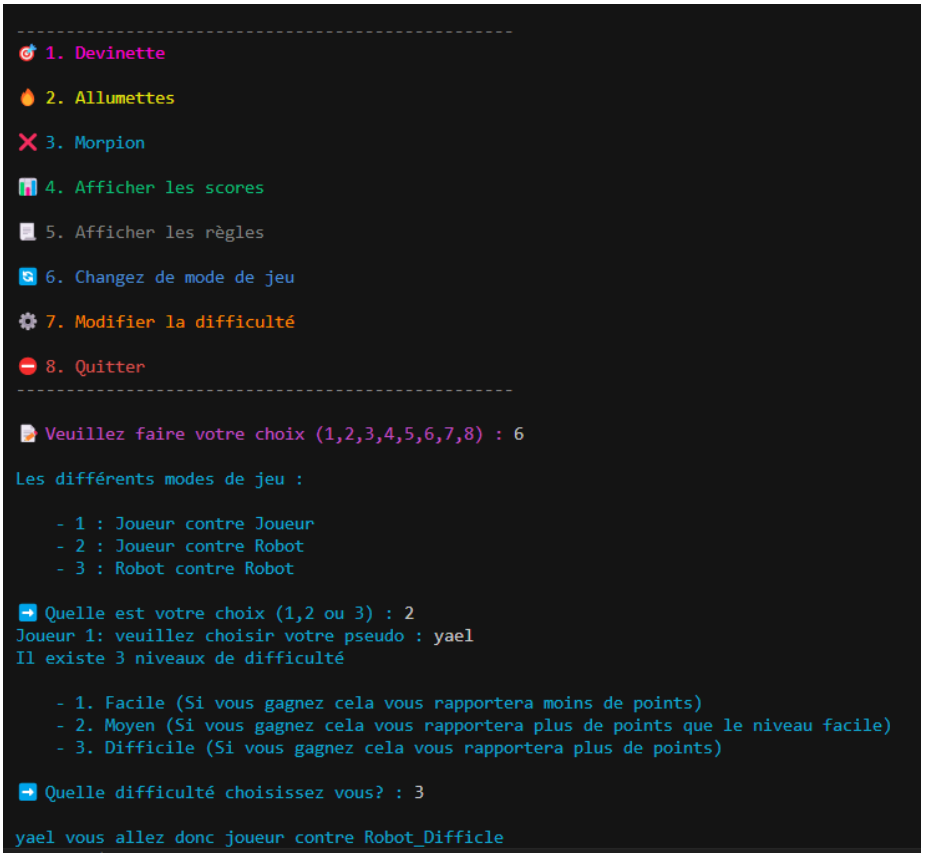

Les différents modes de jeu :
- 1 : Joueur contre Joueur
- 2 : Joueur contre Robot
- 3 : Robot contre Robot
➡ Quelle est votre choix (1,2 ou 3) : e
❌ Vous n'avez pas saisie la bonne valeur, recommencez !
➡ Quelle est votre choix (1,2 ou 3) : 5
❌ Vous avez dépassé la limite il faut soit rentrer un ou trois
➡ Quelle est votre choix (1,2 ou 3) : cinq
❌ Vous avez dépassé la limite il faut soit rentrer un ou trois
➡ Quelle est votre choix (1,2 ou 3) : deux


Joueur 1: veuillez choisir votre pseudo : robot
Le pseudo ne doit pas être vide, contenir uniquement des espaces, contenir 'robot' dans son pseudo ou être invalide.
Joueur 1: veuillez choisir votre pseudo : yael
Joueur 2: veuillez choisir votre pseudo : robot
Le pseudo ne doit pas être vide, contenir uniquement des espaces, contenir 'robot' dans son pseudo ou être identique à celui du Joueur 1.
Joueur 2: veuillez choisir votre pseudo : yael
Le pseudo ne doit pas être vide, contenir uniquement des espaces, contenir 'robot' dans son pseudo ou être identique à celui du Joueur 1.
Joueur 2: veuillez choisir votre pseudo : remo


➡ Quelle est votre choix (1,2 ou 3) : 2
Joueur 1: veuillez choisir votre pseudo : yael
Il existe 3 niveaux de difficulté
- 1. Facile (Si vous gagnez cela vous rapportera moins de points)
- 2. Moyen (Si vous gagnez cela vous rapportera plus de points que le niveau facile)
- 3. Difficile (Si vous gagnez cela vous rapportera plus de points)
➡ Quelle difficulté choisissez vous? : 4
❌ La valeur doit être comprise entre 1 et 3
➡ Quelle difficulté choisissez vous? :
❌ La valeur entrée doit être un entier positif
➡ Quelle difficulté choisissez vous? : dzd
❌ Vous n'avez pas saisie la bonne valeur, recommencez !
➡ Quelle difficulté choisissez vous? : (
❌ Vous n'avez pas saisie la bonne valeur, recommencez !
➡ Quelle difficulté choisissez vous? : deux


Saississer 1 pour pile ou 2 pour face : 3
⚠️ La valeur doit être un entier compris entre 1 et 2.
Saississer 1 pour pile ou 2 pour face : "
❌ Vous n'avez pas saisie la bonne valeur, recommencez !
Saississer 1 pour pile ou 2 pour face : d
❌ Vous n'avez pas saisie la bonne valeur, recommencez !
Saississer 1 pour pile ou 2 pour face : 1
La pièce est retomber sur face !
remo à la main !
Le joueur 1 est : remo
Le joueur 2 est : yael


Veuillez faire votre choix (1,2,3,4,5,6,7,8) :
❌ La valeur saisie doit être un nombre entier positif.
Veuillez faire votre choix (1,2,3,4,5,6,7,8) : neuf
⚠️ La valeur doit être un entier compris entre 1 et 8.
Veuillez faire votre choix (1,2,3,4,5,6,7,8) : é
❌ Vous n'avez pas saisie la bonne valeur, recommencez !
Veuillez faire votre choix (1,2,3,4,5,6,7,8) : "
❌ Vous n'avez pas saisie la bonne valeur, recommencez !
Veuillez faire votre choix (1,2,3,4,5,6,7,8) : un


1. Devinette
2. Allumettes
3. Morpion
4. Afficher les scores
5. Afficher les règles
6. Changez de mode de jeu
7. Modifier la difficulté
8. Quitter
Veuillez faire votre choix (1,2,3,4,5,6,7,8) : 6
Les différents modes de jeu :
- 1 : Joueur contre Joueur
- 2 : Joueur contre Robot
- 3 : Robot contre Robot
Quelle est votre choix (1,2 ou 3) : 2
Joueur 1: veuillez choisir votre pseudo : yael
Il existe 3 niveaux de difficulté
- 1. Facile (Si vous gagnez cela vous rapportera moins de points)
- 2. Moyen (Si vous gagnez cela vous rapportera plus de points que le niveau facile)
- 3. Difficile (Si vous gagnez cela vous rapportera plus de points)
Quelle difficulté choisissez vous? : 3
yael vous allez donc joueur contre Robot_Difficle

```
yael vous allez donc joueur contre Robot_Moyen
Mode de jeu changé en :

Joueur contre Robot

Le joueur est : yael (joueur 1)

Et le robot est de difficulté : Robot_Moyen (joueur 2)
```

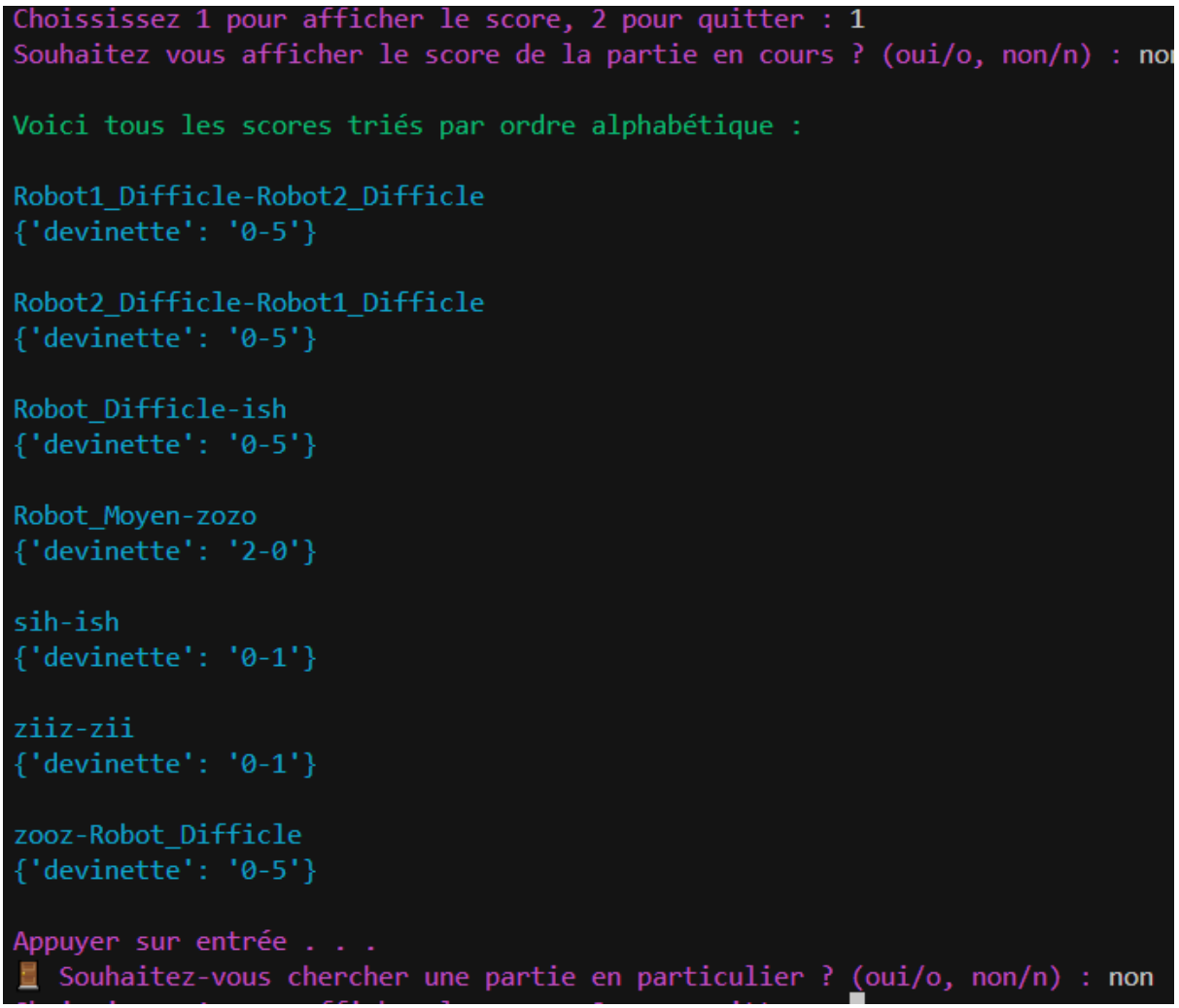
```
📝 Veuillez faire votre choix (1,2,3,4,5,6,7,8) : 7

❌ Impossible de changez de difficulté !
❌ Vous ne jouez contre aucun robot !
```

```
📝 Veuillez faire votre choix (1,2,3,4,5,6,7,8) : 7

Il existe 3 niveaux de difficulté

    - 1. Facile (Si vous gagnez cela vous rapportera moins de points)
    - 2. Moyen (Si vous gagnez cela vous rapportera plus de points que le niveau facile)
    - 3. Difficile (Si vous gagnez cela vous rapportera plus de points)

➡️ Quelle difficulté choisissez vous? : 1

Niveau de difficulté changé en : 1

Robot1_Difficle a choisi pile !

La pièce est retomber sur face !

Robot2_Difficle à la main !

Le joueur 1 est : Robot2_Difficle
Le joueur 2 est : Robot1_Difficle
```

```
📝 Veuillez faire votre choix (1,2,3,4,5,6,7,8) : 5


Quel règles souhaitez vous afficher, 1 pour devinette, 2 pour allumettes, 3 pour morpion, 4 pour afficher le nombre de points remporté par difficulté et 5 pour qui
tter 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.

Quel règles souhaitez vous afficher, 1 pour devinette, 2 pour allumettes, 3 pour morpion, 4 pour afficher le nombre de points remporté par difficulté et 5 pour qui
tter le menu des règles : 2

Voici les règles du jeu des allumettes.

On dispose d'un tas de 20 allumettes.
Chaque joueur à tour de rôle en prélève 1,2 ou 3.
Le perdant est celui que prend la dernière allumettes.

Quel règles souhaitez vous afficher, 1 pour devinette, 2 pour allumettes, 3 pour morpion, 4 pour afficher le nombre de points remporté par difficulté et 5 pour quitter le menu des règles : 3

Voici les règles du jeu du morpion.

Ce jeu se déroule sur une grille de 3x3.
Chaque joueur à tour de rôle pose sa marque, soit 'o' soit 'x'.
Le gagnant est celui qui aligne 3 marques.

Quel règles souhaitez vous afficher, 1 pour devinette, 2 pour allumettes, 3 pour morpion, 4 pour afficher le nombre de points remporté par difficulté et 5 pour quitter le menu des règles : 4

Voici le nombre de points remportés en fonction de la difficulté :

    - La difficulté facile vous rapporte 1 point
    - La difficulté moyen vous rapporte 2 point
    - La difficulté difficile vous rapporte 5 point
```

```
📝 Veuillez faire votre choix (1,2,3,4,5,6,7,8) : 4

📊 Vous avez choisi d'afficher les scores !

Choississez 1 pour afficher le score, 2 pour quitter : 1
Souhaitez vous afficher le score de la partie en cours ? (oui/o, non/n) : oui

Robot1_Difficle-Robot2_Difficle
{'devinette': '0-5'}
```

```
Choississez 1 pour afficher le score, 2 pour quitter : 1
Souhaitez vous afficher le score de la partie en cours ? (oui/o, non/n) : non

Voici tous les scores triés par ordre alphabétique :

Robot1_Difficle-Robot2_Difficle
{'devinette': '0-5'}

Robot2_Difficle-Robot1_Difficle
{'devinette': '0-5'}

Robot_Difficle-ish
{'devinette': '0-5'}

Robot_Moyen-zozo
{'devinette': '2-0'}

sih-ish
{'devinette': '0-1'}

ziiz-zii
{'devinette': '0-1'}

zooz-Robot_Difficle
{'devinette': '0-5'}

Appuyer sur entrée . . .
🚪 Souhaitez-vous chercher une partie en particulier ? (oui/o, non/n) : oui
Quel est le nom d'un des joueurs de la partie ? : ziiz

ziiz-zii
{'devinette': '0-1'}

🚪 Souhaitez-vous afficher une partie en particulier ? (oui/o, non/n) : oui
Quel est le nom du second joueur de la partie ? : zii
ziiz-zii
{'devinette': '0-1'}
```


Choisissez 1 pour afficher le score, 2 pour quitter : 1
Souhaitez vous afficher le score de la partie en cours ? (oui/o, non/n) : no
Voici tous les scores triés par ordre alphabétique :
Robot1_Difficle-Robot2_Difficle
{'devinette': '0-5'}
Robot2_Difficle-Robot1_Difficle
{'devinette': '0-5'}
Robot_Difficle-ish
{'devinette': '0-5'}
Robot_Moyen-zozo
{'devinette': '2-0'}
sih-ish
{'devinette': '0-1'}
ziiz-zii
{'devinette': '0-1'}
zooz-Robot_Difficle
{'devinette': '0-5'}
Appuyer sur entrée . . .
Souhaitez-vous chercher une partie en particulier ? (oui/o, non/n) : non