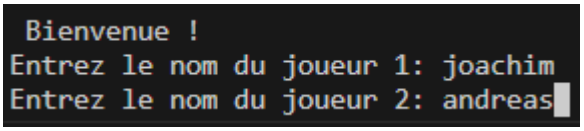
Fevre Joachim | Rey Malissein Andréas

Compte Rendu

Sae R1.01

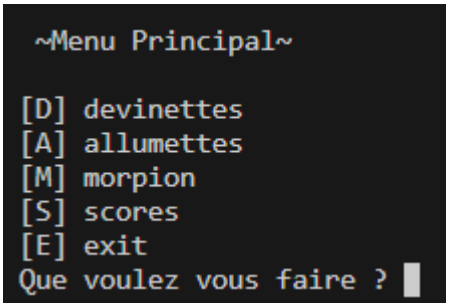
I - Présentation de l’application

Arrivée dans l’application:



Le programme demande d’entrer les noms des joueurs 1 et 2.

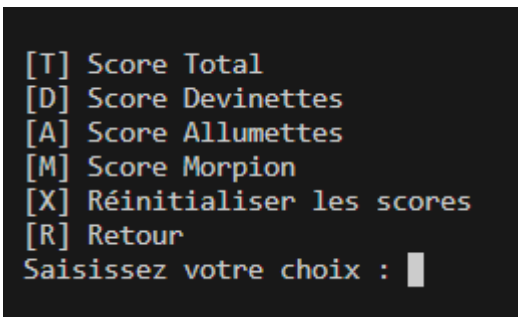
Menu principal



Le menu principal a un affichage et une utilisation simple.
Il affiche tous les choix possibles, les différents jeux, l’affichage des scores ainsi que la possibilité de quitter l’application.

Il suffit juste de saisir la lettre entre crochet pour pouvoir lancer ce qui est écrit à côté.

Menu score

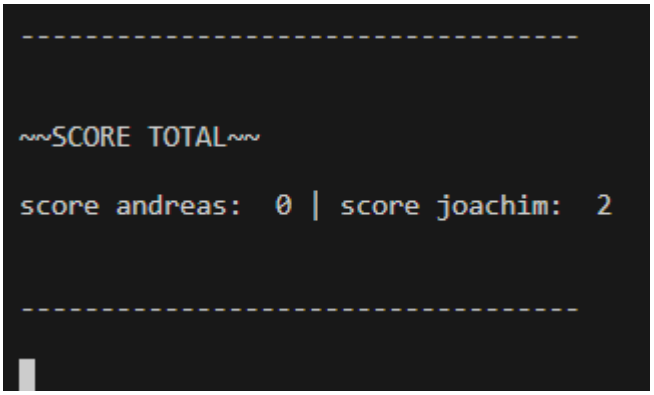


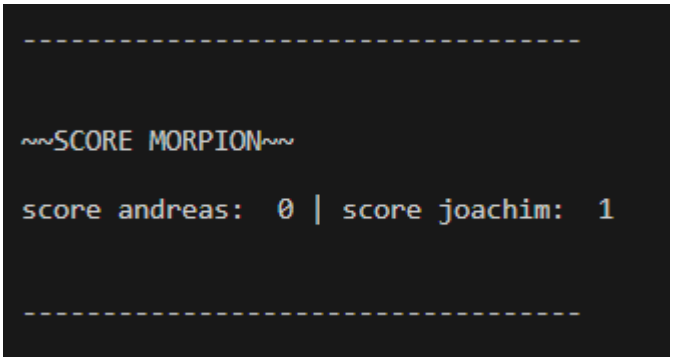
Le menu score a le même style d’affichage que le menu principal.

Il affiche tous les choix possibles, les différents scores ainsi que la possibilité de revenir au menu principal.

Il suffit de saisir la lettre qui se trouve entre les crochets pour lancer ce qui est écrit à côté.

Exemple de score:

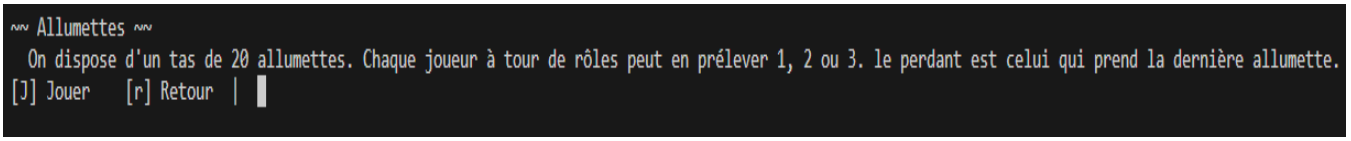




Les différents jeux:

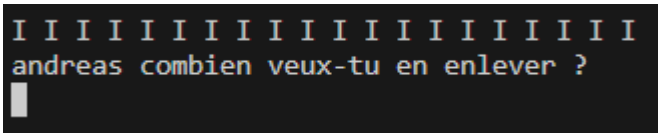
Jeu des allumettes

Le menu

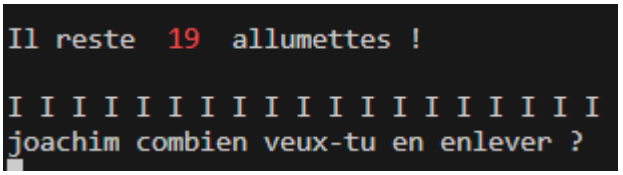


En saisissant A dans le menu principal, le programme affiche le menu du jeu des allumettes. Dans ce menu, il y a l'affichage des règles ainsi que la possibilité de lancer le jeu en saisissant un autre caractère que 'r' ou 'R' ou de retourner dans le menu principal en saisissant R.

Comment jouer ?

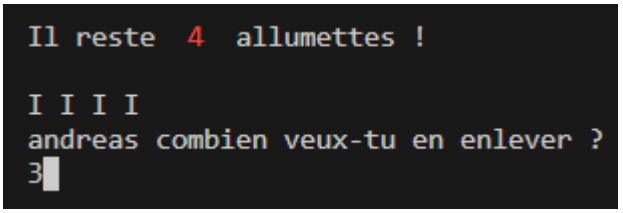


Le jeu se présente ainsi. Il y a l'affichage des bâtonnets en haut avec une question qui demande à un utilisateur combien de bâtonnets il veut enlever.



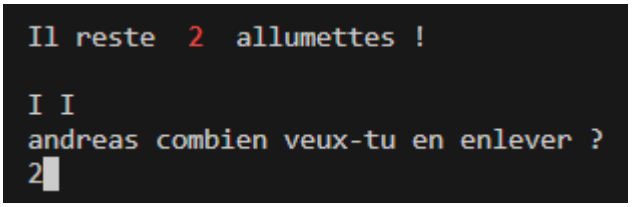
Lorsque l'utilisateur a saisi combien il veut en enlever, le programme dit combien il en reste, dans ce cas, l'utilisateur en a enlever 1 donc il en reste 19. Il y a aussi un affichage des bâtonnets restants Et la question de combien l'autre utilisateur veut en enlever.

Cas de victoire:



Il reste 4 allumettes et c'est au tour d'andreas de jouer. Si il enlève 3 bâtonnets, il a alors gagné la partie et un message s'affiche:

Dans cet autre cas, andreas doit enlever 1 allumette pour gagner, mais il en enlève 2 donc c'est joachim qui gagne.



```
joachim a gagné !
```

Après ce message, il suffit d'appuyer sur la touche entrée pour revenir au menu principal.

Jeu des devinettes

Le menu

```
~~ Devinettes ~~
  Un joueur choisit un nombre entre 1 et 100.
 L'autre doit deviner ce nombre: à chacune de ses propositions.
Le premier joueur répond '+' si c'est trop petit, '-' si trop grand et '=' si c'est gagné
[J] Jouer    [r] Retour  |
```

En saisissant D dans le menu principal, le programme affiche le menu du jeu des devinettes. Dans ce menu, il y a l'affichage des règles ainsi que la possibilité de lancer le jeu en saisissant un autre caractère que R ou de retourner dans le menu principal en saisissant R.

Comment jouer ?

```
a commence à faire deviner !
a, choisissez un nombre entre 1 et 100 :
g, devinez un nombre entre 1 et 100 :
```

Dans ce cas, Joachim commence à faire deviner un nombre, ce nombre est invisible pour qu'Andréas ne puisse pas voir ce nombre, ici Joachim a saisi 50.

```
g, devinez un nombre entre 1 et 100 : 25


nombre = 25
a, Est-ce +, -, ou = ? +


C'est plus.
g, devinez un nombre entre 1 et 100 :
```

Andreas a saisi le nombre 25, Joachim doit alors dire si son nombre est + grand, - grand ou =. Dans ce cas il est plus grand donc il saisit +. Le programme renvoie le message “C’est plus.”.
C’est au tour d’Andréas d’encore essayer de trouver.

```
g, devinez un nombre entre 1 et 100 : 75


nombre = 75
a, Est-ce +, -, ou = ? -


C'est moins.
g, devinez un nombre entre 1 et 100 :
```

Andréas a saisi le nombre 75, Joachim saisis alors -. Le programme renvoie alors le programme “C’est moins.”.
C’est au tour d’Andréas d’encore essayer de trouver.

Cas de victoire:

```
g, devinez un nombre entre 1 et 100 : 50


nombre = 50
a, Est-ce +, -, ou = ? =


Bravo, vous avez trouvé en 3 coups !


à g de faire deviner !
```

Andréas saisit le nombre choisis par Joachim, Joachim saisis alors ‘=’ et le programme affiche un message disant le nombre de coups testés pour trouver le nombre.

A la fin de la partie, il y a le score d’afficher avec par exemple 5 point à l’adversaire si on met 5 coups à trouver le nombre

Dans ce cas, les 2 ont trouvé en 1 coup.

```
andreas à gagné 1pts | joachim à gagné 1pts
```

Jeu du morpion

Le menu

```
~~ Morpion ~~
 Chaque joueur pose sa marque (un O ou un X) à tour de rôle dans les cases d'une grille de 3x3.
 Le premier qui aligne 3 marques a gagné.
[J] Jouer   [r] Retour  |
```

En saisissant M dans le menu principal, le programme affiche le menu du jeu du Morpion. Dans ce menu, il y a l'affichage des règles ainsi que la possibilité de lancer le jeu en saisissant un autre caractère que R ou de retourner dans le menu principal en saisissant R.

Comment jouer ?

```
joachim est O, andreas est X
7 | 8 | 9
---------
4 | 5 | 6
---------
1 | 2 | 3

à : joachim
entrer une case (numéroté de comme un pavé tactile) :
```

Le programme affiche qui joue les O et qui joue les X. Il y a aussi l'affichage du morpion avec les numéros à taper pour choisir la case.

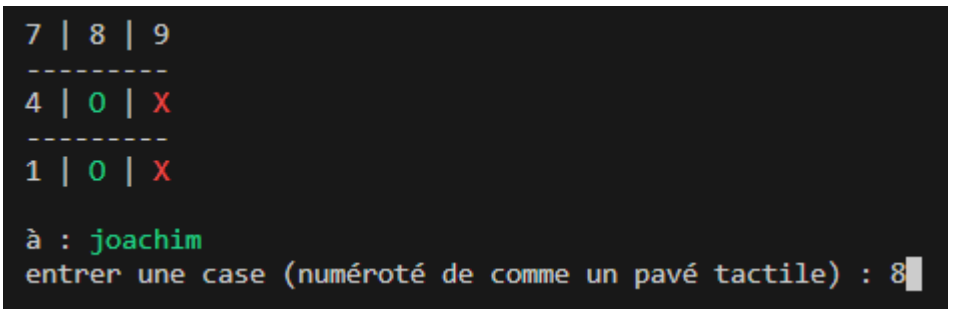
En couleur, il y a le nom de qui doit jouer, puis la demande de case a choisir.

```
entrer une case (numéroté de comme un pavé tactile) : 2
```

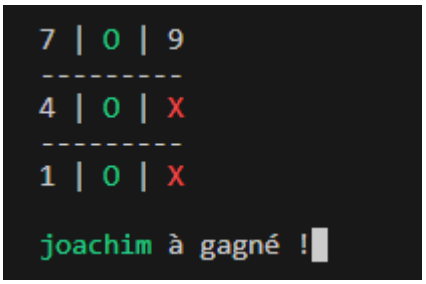
```
7 | 8 | 9
---------
4 | 5 | 6
---------
1 | O | 3

à :  andreas
entrer une case (numéroté de comme un pavé tactile) :
```

Lorsque joachim rentre 2, l'affichage se met a jour, et c'est au tour d'Andréas.

Cas de victoire:



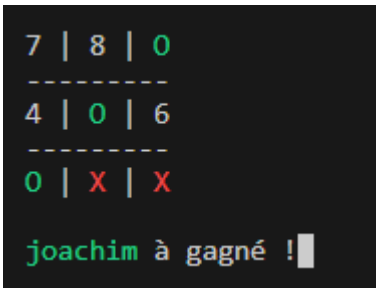
Joachim a juste à saisir 8 pour gagner.



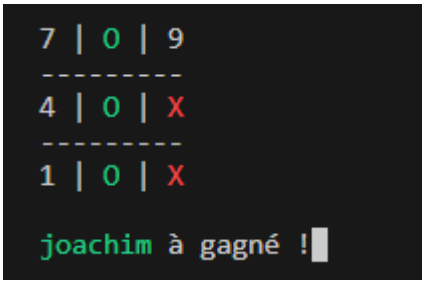
Il a saisi 8, le programme affiche le morpion et une phrase avec le gagnant.

Il y a 3 possibilité pour gagner:

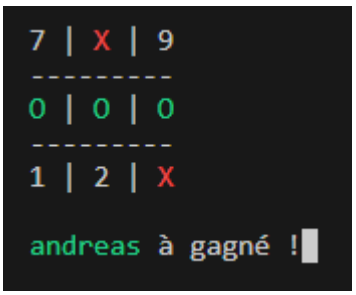
la diagonale:



la colonne:



la ligne:



II - Explication du programme

Organisation des fichiers

On a 6 fichiers:

main.py, contenant le code pour le menue et qui appelle les fonctions des autres fichiers pour lancer les jeux et gérer les scores.

verif.py, contenant des utilitaires utilisables partout.

score.py, contenant les fonctions servant à gérer le score et les sauvegardes

devinettes.py, contenant les fonctions servant au jeu des devinettes

allumettes.py, contenant les fonctions servant au jeu des allumettes

morpion.py, contenant les fonctions servant au jeu du morpion

Utilitaires

Nous avons un fichier dans lequel sont posées des fonctions que l'on utilise partout.

De temps en temps, on voudra effacer le terminal pour ne pas que l'utilisateur puisse remonter dans tous les prints affichés par le programme.
Pour celà, nous utilisons la méthode system du module os permettant d'exécuter une commande du terminal.
Voici la ligne permettant celà:

```
os.system('clear || cls')
```

"clear" étant pour un système linux, "cls" pour windows.

Voici la class bcolors, nous permettant de mettre des couleurs dans le terminal:

```
35  class bcolors:
36      GREEN = "\033[92m"  # GREEN
37      YELLOW = "\033[93m"  # YELLOW
38      RED = "\033[91m"  # RED
39      RESET = "\033[0m"  # RESET COLOR
40      HIDE = "\033[8m" #hide color
```

C’est simplement une batterie de couleur. On a du vert, du jaune, du rouge, du blanc (la couleur de base du terminal), et hide, qui est une couleur invisible nous permettant de ne pas montrer ce qui est tapé dans un input par exemple.
Pour l’utiliser, on concatène les str contenue dans cette classe aux str que l’on veut afficher.
Par exemple:

```
print(bcolors.GREEN + "ceci s'affichera en vert")
print(bcolors.RED + "ceci s'affichera en rouge")
print(bcolors.HIDE + "ceci ne s'affichera pas" + bcolors.RESET + 'Mais ceci en blanc.')
```

```
ceci s'affichera en vert
ceci s'affichera en rouge
                        Mais ceci en blanc.
```

De la même manière pour les inputs, les bcolors concaténé après le str concernent ce qui est entré par l’utilisateur. Par exemple:

```
input(bcolors.GREEN + "ceci s'affichera en vert" + bcolors.RESET)
input(bcolors.RED + "ceci s'affichera en rouge" + bcolors.GREEN)
input(bcolors.YELLOW + "ceci s'affichera en jaune" + bcolors.HIDE)
input(bcolors.HIDE + "ceci ne s'affichera pas" + bcolors.RESET)
```

```
ceci s'affichera en vertinput
ceci s'affichera en rougeinput
ceci s'affichera en jaune
                        input
```

Dans le programme, on utilise à plusieur reprise des input auquel on ne reprend pas ce qui est renvoyé par la fonction, ne servant que pour créer une pause et demander d’appuyer sur entrer pour continuer. C’est une question de lisibilité.
Ainsi, pour éviter que l’utilisateur n’affiche plus de choses sur le terminal, on utilise un bcolors.HIDE sur ses inputs.

Ensuite nous avons la fonction verif qui permet de vérifier que les inputs de l'utilisateur (qu’ils soit convertible en nom et dans les bornes).
Elle renvoie un str correspondant soit à “pass” s’il n’y a pas d'erreur, ou alors l'erreur à renvoyer.

Devinettes

Le fichier devinettes est composé comme suit:
order() Une fonction renvoyant l’ordre entre deux nombres donnée (+, - ou =)
saisi_n() Une fonction pour saisir le nombre à deviner
manche() Une fonction pour faire une manche, prenant le nom du poseur et du devineur.
Elle est composée d’une boucle qui tourne jusqu'à ce que le devineur trouve le nombre.
Elle fonctionne ainsi :

- On fait entrer un essai par le devineur
- On ajoute 1 au nombres de coups
- On fait entrer la réponse du poseur

- On vérifie le résultat afin de l'afficher

Elle renvoie le nombre de points gagnés par le poseur (correspondant au nombre de tour prit par le devineur pour trouver son nombre).

Jouer() Enfin, la fonction principale.

Celle-ci prend le nom des joueurs puis lance consécutivement une partie avec le joueur 1 en tant que poseur, puis une deuxième en inversant les rôles. Elle renvoie un tuple contenant les points des joueurs comme celà : (pts_j1, pts_j2).

Allumettes

jouer( ) permet de choisir quel est le joueur qui commence à jouer aléatoirement. Initialise le compteur de tour à 1. Il prend +1 à chaque tour effectué. Un tour correspond à ce qu'un joueur enlève entre 1 et 3 allumettes.

Au fur et à mesure de la partie, le nombre d'allumettes diminue en fonction du choix des joueurs.

Tant que le nombre d'allumettes est supérieur à 1, le jeu se joue tour à tour.

- Si le compteur de tour est pair, c'est au joueur 1 de jouer
- Si le compteur de tour est impair, c'est au joueur 2 de jouer

Le programme permet alors de faire jouer à chaque fois le joueur 1 et le joueur 2 à tour de rôle.

Si le nombre d'allumettes est égal à 0, le programme regarde quel joueur à gagner grâce à un calcul mathématiques. Il va alors afficher un message avec le nom du vainqueur puis retourner le joueur gagnant pour pouvoir mettre à jour les scores.

debut_partie( ) La partie est d'abord initialisée par la fonction qui va initialiser la liste des 20 allumettes. Cette fonction va donc retourner la liste des allumettes.

afficher_allumette( ) La liste des allumettes restantes est sans les guillemets.

tour( ) Prend en paramètre les joueurs et la liste des allumettes restantes. Le programme demande combien d'allumettes le joueur veut enlever. Le programme vérifie alors si la réponse se trouve entre 1 et 3. Si la vérification n'est pas concluante, il renvoie un message d'erreur grâce à la fonction **vérif( )** , sinon il demande à l'autre joueur, combien d'allumettes il veut enlever. La fonction supprime le nombre d'allumettes saisis pour l'affichage et retourne enfin la liste des allumettes restantes.

Morpion

Le morpion fonctionne sur une grille de 3 par 3. Pour la matérialiser dans le code on utilise une liste de liste de str, un tableau comme ceci:

```
tab : list[list[str]] = [
    ['7','8','9'],
    ['4','5','6'],
    ['1','2','3']
]
```

Elle est initiée avec des chiffres de 1 à 9 positionnés comme sur un pavé numérique.
Au fur et à mesure de la partie, ces chiffres seront remplacés par des O verts et des X rouges.

Le fichier morpion.py est donc composé des fonctions suivantes:

afficher_board() Une fonction pour afficher la grille

find() Une fonction pour savoir si un élément se trouve dans la grille, elle permet de savoir si une case à été prise ou non (si l'élément '1' à été pris, alors il n'est plus dans le tableau). La fonction renvoie l'index de l'élément afin de trouver quel élément du tableau il faut modifier lorsqu'on va mettre un signe dans la grille.

saisi_case() Permet de saisir le numéro d'une case et de le renvoyer.

change_tour() Permet de changer le tour (elle renvoie X ou O en fonction de si l'argument est O ou X).

alone_in_tab() Permet de savoir si une liste n'est composé que d'un élément donné (elle est utile pour vérifier le gagnant)

tab_complet() Permet de savoir si la grille est complète ou non.

win() Permet de renvoyer s'il y a un gagnant ou non.
Elle prend dans des listes, les positions gagnantes et regarde si ces listes ne sont composées que de O ou de X.

initialisation_tour() Permet de désigner au hasard qui sera le joueur O (qui joue en premier) et qui sera le joueur X.

jouer() C'est la fonction principale qui prend le nom des joueurs, lance le jeu, et renvoie le nom du gagnant.
Elle fonctionne comme cela:
Elle initialise la grille et l'ordre des joueurs,
Fait tourner la boucle principale du jeu qui tourne tant que la fonction win() ne trouve pas de vainqueur, et tant que la grille n'est pas complète:

- afficher la grille
- fait entrer une case
- si cette case n'est pas prise, met dans cette case le signe du tour
- change le tour

On vérifie une dernière fois la fonction win() pour savoir si la boucle s'est arrêtée dans le cas d'une égalité ou d'un vainqueur.

Scores et sauvegardes

Le système de score est basé sur une structure de Profil.
Un profil est donc composé comme suit:

```
4   class Profil:
5       nom : str
6       total : int
7       devinettes : int
8       allumettes : int
9       morpion : int
```

Un nom ainsi que les scores qui lui sont associés.

Dans le programme, on a aussi une liste de profil créé nommé all_profil, ainsi qu'une structure de données comprenant les 2 profils courant, les 2 profiles s'étant connecté au lancement du programme:

```
11  class CurrentProfil():
12      j1 : Profil
13      j2 : Profil
```

Dans le menu des scores, on affichera les scores des profils de cette structure.
L'instance de CurrentProfil est nommée c_profil dans le programme.

Ainsi, au début du programme principale, les joueurs entrent leurs noms afin de se connecter à leurs profile:

```
89      print(bcolors.RESET + ' Bienvenue ! ')
90
91      #Entré des joueurs
92      print("Joueur 1")
93      joueur1 = entrer_nom()
94      c_profil.j1 = search_profil(joueur1, all_profil)
95      if not profil_exist(joueur1, all_profil):
96          all_profil.append(c_profil.j1)
97
98      print('Joueur 2')
99      joueur2 = entrer_nom()
100     c_profil.j2 = search_profil(joueur2, all_profil)
101     if not profil_exist(joueur2, all_profil):
102         all_profil.append(c_profil.j2)
```

Ce bout de code permet de

- faire saisir le nom du joueur 1
- le rechercher dans la liste all_profil
- s'il existe le mettre dans la structure current profil à la place du joueur 1
- sinon on créer un profil à la place
- on recommence pour le joueur 2

Nous avons donc besoin des fonctions suivantes.

profil_exist( ) Renvoie un booléen en fonction de l'existence d'un profil avec un nom correspondant à celui en entré.
Si la liste est vide, on est sûr que le profil n'y est pas et on renvoie False directement.
Dans le cas contraire, on doit parcourir la liste jusqu'à trouver un nom correspondant pour renvoyer True. Ou False si on parcourt toute la liste et que rien n'a été trouvé.

search_profil( ) Ressemble beaucoup à **profil_exist( )**, mais au lieu de renvoyer un booléen, renvoie un profil provenant de la liste s'il y est, ou alors un profil vierge fraîchement créé.

Si la liste est vide, alors on peut être sûr que le profil n'existe pas, et qu'il faut le créer.
Sinon, on doit parcourir cette liste. Le profil à renvoyer est par défaut un profil vide avec un nom vide. Une fois la boucle parcourue, si le profil est toujours vide, cela veut dire qu'il n'est pas dans la liste est qu'il faut le créer. La boucle fonctionne de la même manière que la fonction précédente.

create_profil( ) Permet de créer un profil vierge avec un nom donné:
Elle initialise simplement les scores à 0 d'un profil de nom 'nom' pour le renvoyer.

update_profil( ) Permet de mettre à jour les scores totaux des joueurs 1 et 2 contenu dans les CurrentProfil de manière rapide

Nous avons maintenant toutes les clés pour gérer le système de score interne au programme.
Cependant, en l'état, il nous est impossible de les conserver une fois le programme fermer.

C'est pourquoi nous devons sauvegarder la liste des profils all_profil dans un fichier binaire que l'on a nommé 'save.txt'.
Pour celà, nous avons 2 fonction:

save( ) Permet de sauvegarder les profils dans un fichier binaire 'save.txt'. Pour cela, le fichier 'save.txt est ouvert en mode lecture (read). Cette fonction utilise le module pickle afin de transformer la liste en données binaire et de la sauvegarder dans le fichier.
La sauvegarde est automatiquement faite à la fermeture du programme

load( ) Permet de charger le fichier 'save.txt' lors du lancement du programme. Il permet de retourner la liste de profils (liste_profil). Il va essayer d'ouvrir le fichier 'save.txt'.
S' il réussi à l'ouvrir: il va récupérer tout ce qui se trouve dans le fichier et le mettre dans la variable liste_profil.
Si il ne le trouve pas: il va renvoie liste_profil comme étant une liste vide
Il retourne enfin la variable liste_profil.

Grâce à cette structure de données, la seule chose que nous avons besoin d'enregistrer est la liste des profils all_profils.

Main

Le fichier principale contient
menu_principale() Pour afficher les options du menu principale et renvoyer le choix de l'utilisateur
menu_score() Pour afficher les options du menu score et renvoyer le choix de l'utilisateur
entrer_nom() Permet de saisir un nom. Après avoir récupéré le nom entré par le joueur, il vérifie qu'il n'est pas vide (un profil à nom vide nous servira dans les fonctions suivantes)
separator() Permet d'afficher un séparateur.

Maintenant, entrons dans le programme principale:

Au lancement du programme, les joueurs entrent successivement leurs noms afin de se connecter à leurs profils s'il existent.
On utilise donc les fonctions entrer_nom(), search_profil(), et profil_exist() afin de mettre les profils de la liste sélectionnée dans la structure CurrentProfil.
Ensuite on affiche le menu principal avec la fonction adéquate, puis nous avons la boucle du menu principale qui permet de lancer les différents jeux, et d'aller sur le menu score, et de quitter le programme.

Lancer un jeu:
Avant de lancer un jeu, on affiche les règles et demande la confirmation du joueur avant de le lancer.

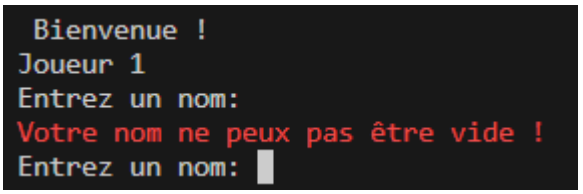
Récupération des scores:
on modifie les valeurs de la structure CurrentProfil en fonction du résultat donné par la fonction jouer() du jeu lancé.

Quitter:
Avant de quitter le programme, nous devons mettre à jour le total des score de CurrentProfil, puis on sauvegarde.

III - Batterie de tests

Menu principal

Pour commencer le programme, il nous demande notre nom et il est impossible de saisir un nom vide.



La condition ‘else’ du menu principal permet d’afficher un message d’erreur si l’utilisateur ne saisit pas une des lettres correspondante aux choix possibles dans le menu.

C’est la même chose pour le menu du score, il est possible de saisir une lettre qui n’est pas proposée par le programme mais ce-dernier renverra un message d’erreur.

```
[T] Score Total
[D] Score Devinettes
[A] Score Allumettes
[M] Score Morpion
[X] Réinitialiser les scores
[R] Retour
Saisissez votre choix : z
```

```
commande inexistante !
```

Dans chaque début de jeu, lorsque les règles sont affichées, il est encore possible de retourner dans le menu principal en saisissant ‘r’ ou ‘R’.

```
~~ Morpion ~~
 Chaque joueur pose sa marque (un O ou un X) à tour de rôle dans les cases d'une grille de 3x3.
 Le premier qui aligne 3 marques a gagné.
[J] Jouer   [r] Retour  |  r
```

```
 ~Menu Principal~

[D] devinettes
[A] allumettes
[M] morpion
[S] scores
[E] exit
Que voulez vous faire ?
```

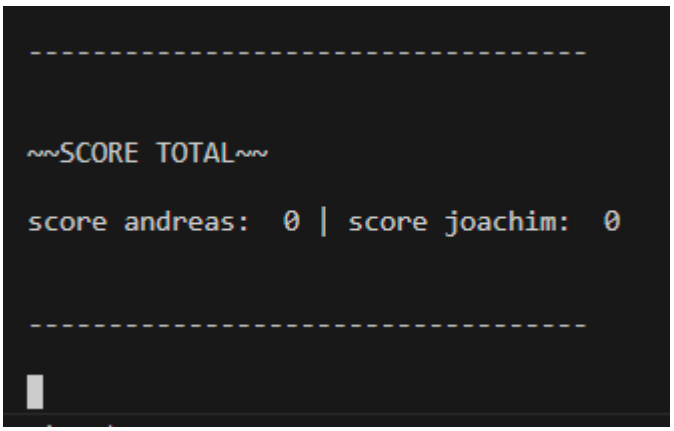
```
 ~Menu Principal~

[D] devinettes
[A] allumettes
[M] morpion
[S] scores
[E] exit
Que voulez vous faire ? e
```

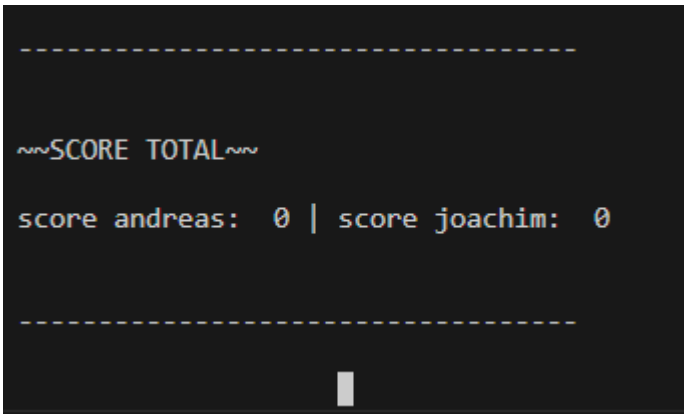
```
Au revoir !
```

Lorsque l’utilisateur saisit ‘e’ dans le menu principal, le programme renvoie un message et le programme s’arrête. La casse n’est pas vérifiée, c’est-à-dire que l’utilisateur peut saisir ‘e’ ou ‘E’, le programme comprendra.

Menu score

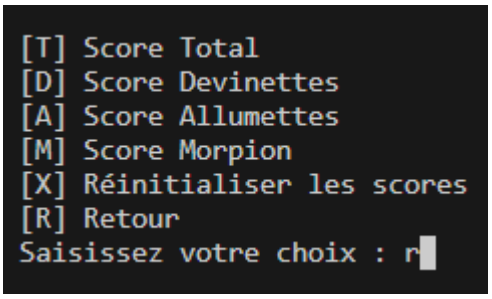
Dans n’importe quelle partie du menu, lorsque c’est juste un affichage et que l’utilisateur n’a pas besoin de saisir quoi que ce soit, les caractères saisis seront invisibles.

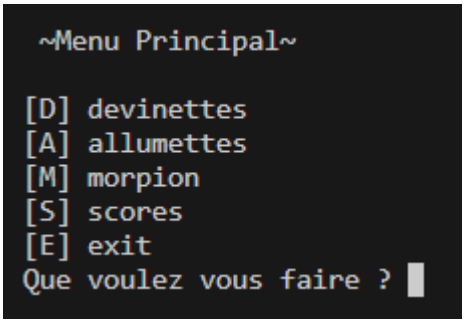




Cela est possible grâce à bcolors et à la couleur HIDE:

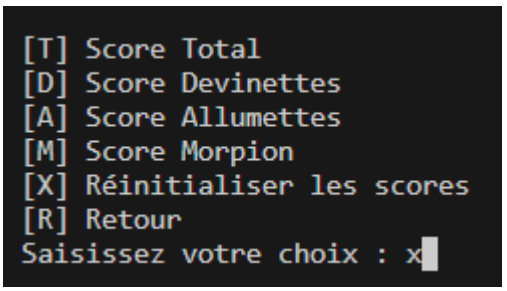
```
{bcolors.HIDE}
```

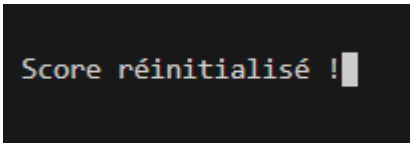




Si, dans le menu score, l'utilisateur saisit 'r' ou 'R', le programme est fait pour que l'on retourne dans le menu principal.

Réinitialiser le score:





Lorsque l'utilisateur saisit le caractère 'x', le fichier 'save.txt' est supprimé et les CurrentProfil sont aussi réinitialisés. Le programme renvoie alors un message pour avertir l'utilisateur.

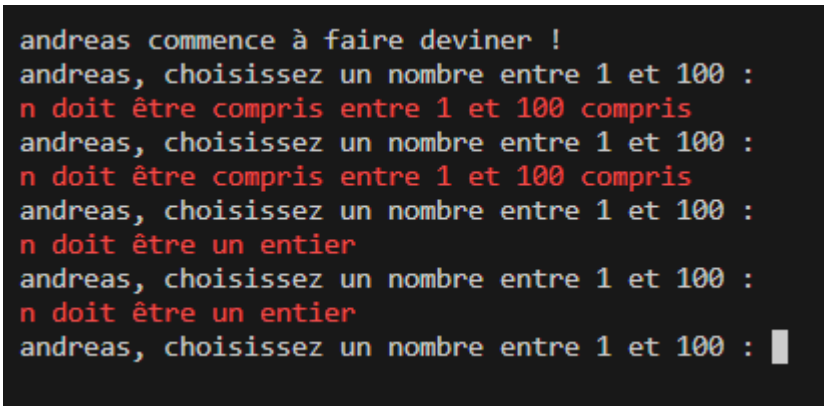
Jeu des allumettes

On peut voir que le joueur doit forcément choisir un entier entre 1 et 3 compris pour pouvoir jouer.
En effet, le programme renvoie un message d'erreur si le joueur saisie:

- un entier supérieur à 3
- un entier inférieur à 1
- un caractère autre qu'un chiffre comme une lettre (e dans l'exemple)*
- la touche entrée
- un float

Jeu des devinettes

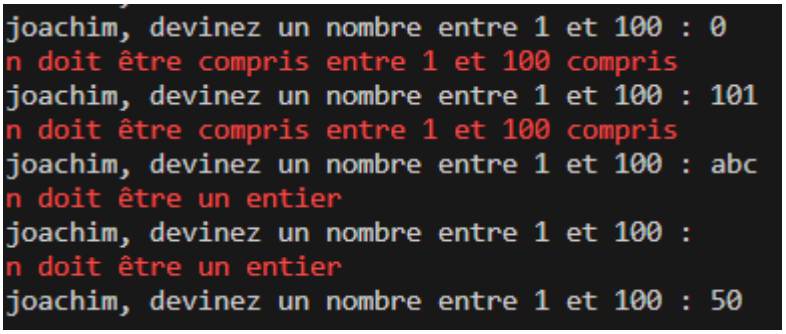
Dans cet exemple, le nombre qu'Andréas a choisis est 50 et celui de Joachim est 25.



On peut voir que le joueur doit forcément choisir un entier entre 1 et 100 compris pour pouvoir le faire deviner.
En effet, le programme renvoie un message d'erreur si le joueur saisie:

- un entier supérieur à 100
- un entier inférieur à 1
- un caractère autre qu'un chiffre comme une chaîne de caractère (ex: abc)
- la touche entrée
- un float

C'est pareil pour le joueur qui doit deviner, le nombre saisi doit être un entier entre 1 et 100 inclus sinon le programme renvoie un message d'erreur.



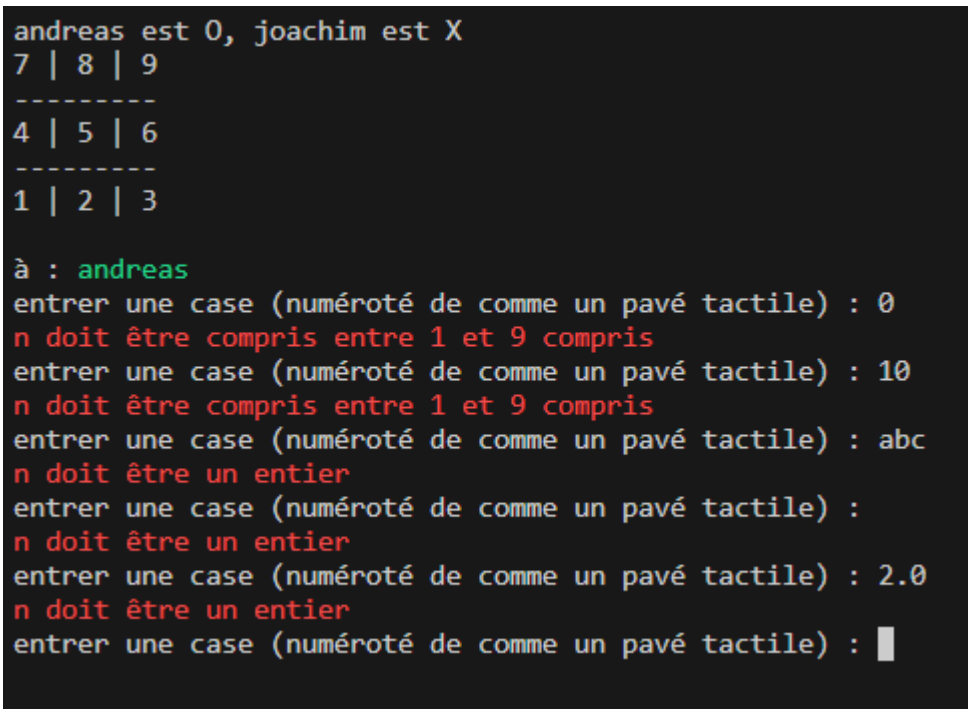
Si joachim qui doit trouver 50, saisit 50, et qu'Andréas dit que le nombre est plus petit ou plus grand, le programme renverra un message d'erreur comme c'est faux car il aurait dû saisir =.

```
nombre = 50
andreas, Est-ce +, -, ou = ? -
C'est faux
andreas, Est-ce +, -, ou = ? +
C'est faux
andreas, Est-ce +, -, ou = ? =


Bravo, vous avez trouvé en 1 coups !
```

C’est exactement pareil lorsque l’autre joueur doit deviner:

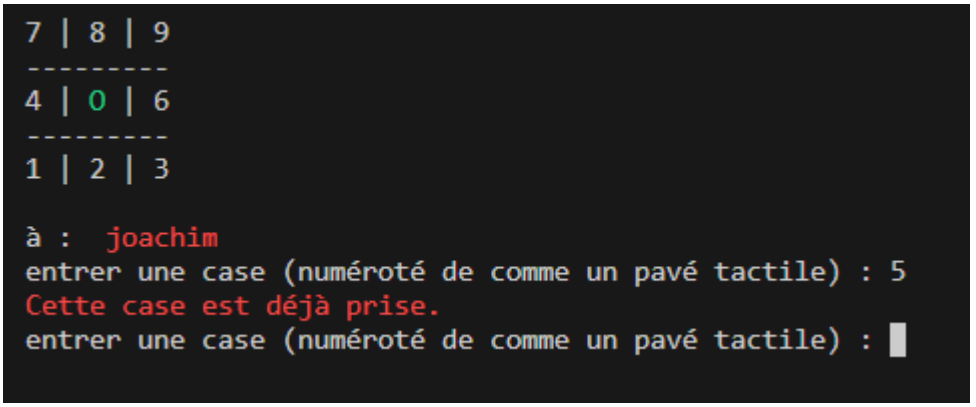
```
à joachim de faire deviner !
joachim, choisissez un nombre entre 1 et 100 :
andreas, devinez un nombre entre 1 et 100 : 0
n doit être compris entre 1 et 100 compris
andreas, devinez un nombre entre 1 et 100 : 101
n doit être compris entre 1 et 100 compris
andreas, devinez un nombre entre 1 et 100 : abc
n doit être un entier
andreas, devinez un nombre entre 1 et 100 :
n doit être un entier
andreas, devinez un nombre entre 1 et 100 : 25


nombre = 25
joachim, Est-ce +, -, ou = ? -
C'est faux
joachim, Est-ce +, -, ou = ? +
C'est faux
joachim, Est-ce +, -, ou = ? =


Bravo, vous avez trouvé en 1 coups !
```

Jeu du morpion



On peut voir que le joueur doit forcément choisir un entier entre 1 et 9 compris pour pouvoir le faire deviner.
En effet, le programme renvoie un message d'erreur si le joueur saisie:

- un entier supérieur à 9
- un entier inférieur à 1
- un caractère autre qu'un chiffre comme une chaîne de caractère (ex: abc)
- la touche entrée
- un float



Si le joueur d'après saisi une case déjà prise, le programme renverra un message d'erreur et demandera de saisir un autre chiffre.