Rapport S1.01

Implémentation d’un besoin client

Delhomel Mattéo
Pinaud Louis

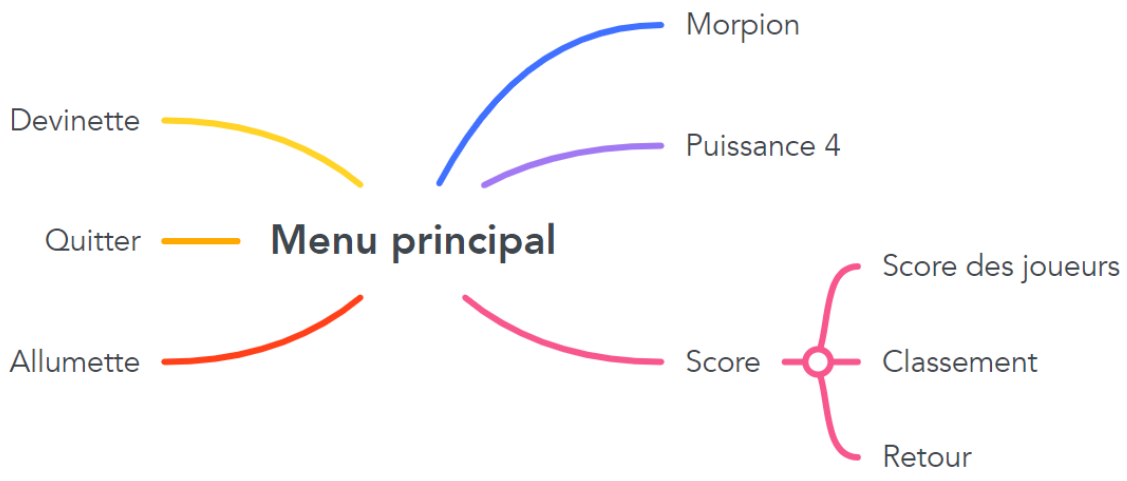
Table des matières

Delhomel Mattéo
Pinaud Louis

Phase 1

Mise en place du squelette du programme

Dans l'application il y a un menu principal qui permet d'accéder facilement aux fonctionnalités proposées. Nous avons schématisé le squelette du programme à l'aide d'une pieuvre pour mieux comprendre son architecture.



Outils supplémentaires

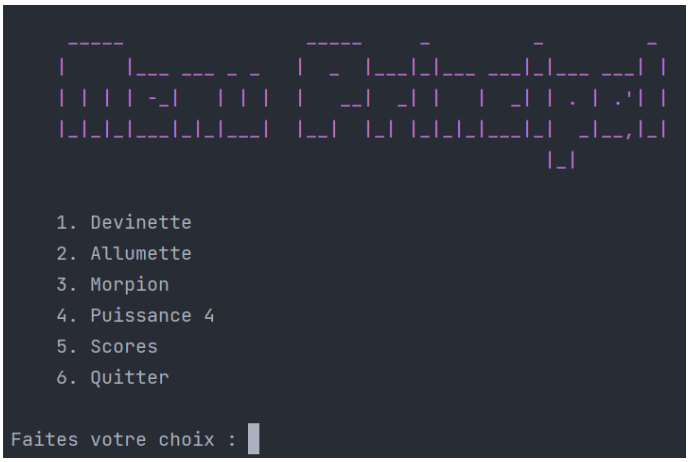
Afin de travailler de manière organisée et en parallèle, nous avons créé un répertoire sur GitHub contenant notre SAE. Le versionning ainsi rendu disponible nous a grandement aidé lors de l'ajout de nouvelles fonctionnalités.

Définition et implémentation de l'interface utilisateur

Pour interagir avec l'utilisateur, il fallait utiliser une méthode simple et compréhensible de tous, mais aussi facile à mettre en œuvre.

Pour faire un choix dans les menus, il suffit de choisir le numéro associé à l'action souhaitée.

Les menus secondaires sont construits de la même manière.


1. Devinette
2. Allumette
3. Morpion
4. Puissance 4
5. Scores
6. Quitter
Faites votre choix :

Développement de chaque jeu

Jeu Devinette

Règle du jeu

Le joueur 1 choisit un nombre entre 1 et une limite à décider. Le joueur 2 doit deviner ce nombre. A chacune de ses propositions, le joueur 1 répond :

- Trop petit
- Trop grand
- Égale

Analyse des règles

Dans ce jeu il faut comprendre qu'il y a un joueur qui indique la plage de nombre dans laquelle l'autre joueur va devoir chercher le nombre à deviner.

Jeu Allumette

Règle du jeu

On dispose d'un tas de 20 allumettes. Chaque joueur à tour de rôle peut en prélever 1, 2 ou 3. Le perdant est celui qui pioche la dernière allumette.

Morpion

Règle du jeu

Chaque joueur pose sa marque (X ou O) à tour de rôle dans les cases d'un plateau de 3x3. Le premier joueur à aligner 3 marques gagne.

Puissance 4

Règle du jeu

Les 2 joueurs posent leurs pions tour à tour dans une grille de 6 par 7 cases. Les pions tombent systématiquement en bas de la grille. Le premier qui aligne 4 de ses pions gagne la partie.

Phase 2

Définition et choix des scores

Gestion des scores

Pour gérer les scores de la manières la plus simple, et que ce soit clair pour l'utilisateur nous avons fait le choix suivant :

- 1 victoire 1 point
- 1 défaite 0 point

Ce système est utilisé pour l'intégralité des jeux.

Gestion des joueurs

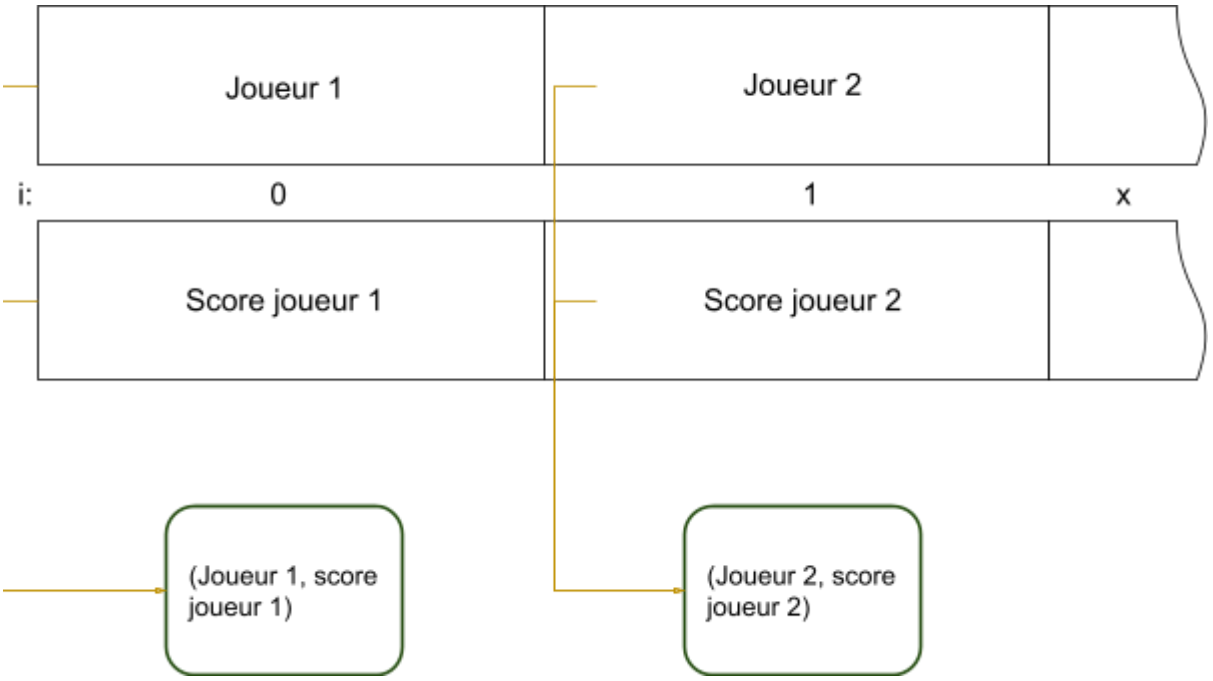
Pour manipuler les joueurs nous avons créé le type "Joueur" avec la structure suivante :

```
Type Joueur = {
    pseudo : chaine
    scoreDevinette : entier
    scroreAllumettes : entier
    scoreMorpion : entier
    scroeP4 : entier
}
```

(Note : il s'agit bien de "scoreP4")

Tri des scores

Lors de la récupération du classement dans le fichier de sauvegarde, on crée une **liste** contenant les joueurs et une autre le score des joueurs dans un jeu donné. Ces deux listes suivent le même ordre de lecture ce qui permet d'avoir, pour un joueur stocké en index i de la première liste, son score en index i de la seconde.
Pour trier facilement les joueurs, nous les avons organisés sous forme de **tuples** contenant le joueur et son score dans un jeu *(voir schéma ci-dessous)*. Nous avons ensuite utilisé une fonction de tri, pour ranger les joueurs par leur score puis l'afficher en commençant par le meilleur. Etant donné que cette association se fait pour un jeu précis, il suffit de la répéter pour chaque jeu pour obtenir un classement trié par jeu, du meilleur joueur au moins bon.



Association d'un joueur au score d'un jeu donné en vue du tri.

Système de sauvegarde

Format

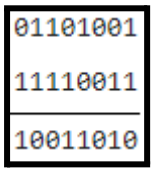
Nous avons fait le choix d'utiliser un format particulier pour stocker les éléments à sauvegarder, plutôt que de stocker directement les objets dans un fichier binaire. Cela apportait plusieurs avantages, notamment au niveau du développement, car il rendait l'accès aux données facile pour effectuer des vérifications. Nous avons choisi d'utiliser le JSON, qui a l'avantage de permettre le stockage d'objets et de listes de manière formalisée tout en étant inclus dans un module de python prévu à sa gestion (module json).

```
{
    "players": {
        "a": {
            "allumette": 0,
            "devinette": 0,
            "morpion": 1,
            "puissance 4": 1
        },
        "b": {
            "allumette": 1,
            "devinette": 0,
            "morpion": 0,
            "puissance 4": 0
        }
    }
}
```

Exemple de deux objets json correspondant à des joueurs, identifiés par leurs pseudos ("a" et "b") contenant eux-même une propriété par jeu associé à la valeur de leur score.

Prévention de triche

Plutôt que de stocker notre chaîne formalisée en json sous forme binaire, ce qui aurait eu l'avantage de la rendre relativement illisible mais comme inconvénient de devoir récupérer la chaine caractère par caractère, nous avons opté pour une solution permettant d'écrire d'abord le json formalisé entier dans un fichier, puis de rendre celui-ci illisible. Nous utilisons donc une méthode de chiffrement simple et réversible permettant par l'intermédiaire d'un mot de passe, de rapidement passer d'un fichier lisible à un autre illisible.



La méthode de chiffrement (XOR) repose sur l'utilisation du OU exclusif pour comparer un octet à chiffrer (première ligne) et un octet de chiffrement (clé ou mot de passe). Le résultat de l'opération est inscrit dans un fichier binaire.

Jeux d’essai et algo

Devinette

Jeux d’essai

```
matteo entrez la borne superieure à 1 : 0
matteo entrez la borne superieure à 1 : 10
```

```
Bienvenue dans le jeu de la devinette

Entrez le nombre que louis doit trouver entre 0 et 10 :

louis devine le nombre :
```

Le nombre à deviner ne s’affiche pas mais s'écrit bien.

```
louis devine le nombre : 3

matteo à vous de jouer
3 est :
1. Plus grand
2. Plus petit
3. Egal
matteo faites votre choix : 1
3 est plus grand
matteo n'essaye pas de tricher !
3 est plus petit

louis devine le nombre :
```

La triche est impossible même si le joueur qui choisit le nombre met une réponse fausse, un algorithme anti-triche prend le relais pour ce tour.

```
Bien joué louis vous avez trouvé le nombre en 2 coups

matteo, à vous de deviner le nombre que louis a choisi

Entrez le nombre que matteo doit trouver entre 0 et 10 :
```

Lorsque la réponse est la bonne, c’est à l’autre joueur de choisir un nombre.

Les mêmes étapes que précédemment se répètent une fois. Le joueur ayant trouvé le nombre en moins de coups que l'autre remporte le match. Si les deux joueurs ont le même nombre de coups, il y a égalité et aucun point n'est marqué.

Algorithme

Voici un extrait de la fonction qui effectue une manche entière de Devinette. Celle-ci est appelée deux fois dans la fonction principale de Devinette.

```
tant que choix != 3 faire
    proposition <- saisieInt( #proposition par le joueur
        "\n"+joueurCherche.pseudo + " devine le nombre : ", "Erreur de saisie")
    comptJoueur <- comptJoueur + 1

    afficher("\n"+joueurChoisit.pseudo, "à vous de jouer")
    afficher(proposition, "est :")
    afficher("1. Plus grand")
    afficher("2. Plus petit")
    afficher("3. Egal")

    choix <- saisieInt(joueurChoisit.pseudo +
                       " faites votre choix : ", "Erreur de saisie")

    Selon choix faire
        cas 1: afficher(proposition, "est plus grand")
        cas 2: afficher(proposition, "est plus petit")
    FinFaire

    si choix = 1 and proposition < nbATrouver alors
        afficher(joueurChoisit.pseudo, "n'essaye pas de tricher !")
        afficher(proposition, "est plus petit")
    FinSi
    si choix = 2 and proposition > nbATrouver alors
        afficher(joueurChoisit.pseudo, "n'essaye pas de tricher !")
        afficher(proposition, "est plus grand")
    FinSi
    si choix != 3 and proposition = nbATrouver alors
        afficher(joueurChoisit.pseudo, "n'essaye pas de tricher !")
        choix <- 3
    FinSi
    si choix = 3 and proposition != nbATrouver alors
        afficher("Fait attention", joueurChoisit.pseudo)
    FinSi
FinFaire
```

Allumettes

Jeux d'essai

```
Bienvenue dans le jeu des allumettes

 | | | | | | | | | | | | | | | | | | | |

louis, retirez entre 1 et 3 allumettes :
```

Lors de l'entrée dans le jeu, le premier joueur est invité à retirer entre 1 et 3 allumettes.

```
Bienvenue dans le jeu des allumettes

 | | | | | | | | | | | | | | | | | | | |

louis, retirez entre 1 et 3 allumettes : a
Erreur de saisie
louis, retirez entre 1 et 3 allumettes : ^X
Erreur de saisie
louis, retirez entre 1 et 3 allumettes :
```

En cas d'erreur de saisie, un message d'erreur apparaît et on refait une demande au joueur.

```
louis, retirez entre 1 et 3 allumettes : 35
Nombre invalide, il doit être compris entre 1 et 3 inclus
louis, retirez entre 1 et 3 allumettes : -12
Nombre invalide, il doit être compris entre 1 et 3 inclus
louis, retirez entre 1 et 3 allumettes : _
```

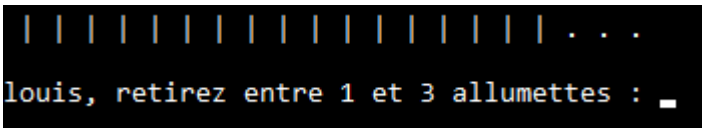
En cas de saisie d'un nombre au-delà des bornes acceptées, on affiche une erreur différente et on refait une demande au joueur.

```
Bienvenue dans le jeu des allumettes

 | | | | | | | | | | | | | | | | | | | |

louis, retirez entre 1 et 3 allumettes : 1
```

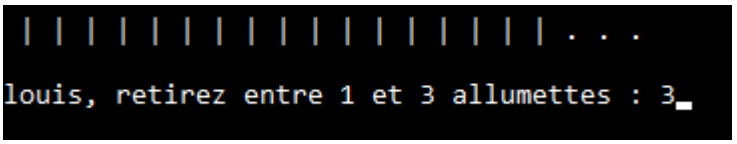
```
 | | | | | | | | | | | | | | | | | | | .

matteo, retirez entre 1 et 3 allumettes : _
```

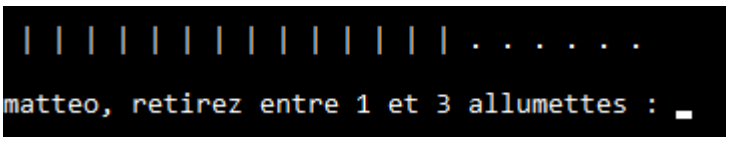
Le joueur retire une allumette, le tour passe au joueur suivant

```
 | | | | | | | | | | | | | | | | | | | .

matteo, retirez entre 1 et 3 allumettes : 2
```

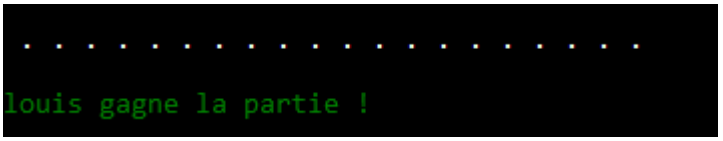


Le joueur retire deux allumettes, le tour passe au joueur suivant





Le joueur retire trois allumettes, le tour passe au joueur suivant



OU

Le joueur qui prend la dernière allumette perd la partie, on affiche un message à l'autre joueur (fonctionne peu importe le nombre d'allumettes pris étant donné que la dernière est de toute façon prise).

Algorithme

```
Fonction affichageAllumette(nbrAllumette: entier, allumetteRestantes: entier) -> chaine:
Début
    show : chaine
    show <- ""

    Pour (i allant de 0 à nbrAllumette) Faire
        Si (i < allumetteRestantes):
            show <- show + " |"
        Sinon:
            show <- show + " ."
        FinSi
    FinFaire

    retourne show
Fin
```

Algorithme de construction de la chaîne des allumettes

Morpion

Jeux d'essai

```
  0 1 2
0 - - -
1 - - -
2 - - -
matteo entrez la ligne : 1
matteo entrez la colonne : 1
```

Pour placer un pion sur le plateau du Morpion il suffit de choisir la ligne puis la colonne.

```
  0 1 2
0 - - -
1 - X -
2 - - -
louis entrez la ligne : 0
louis entrez la colonne : 0
```

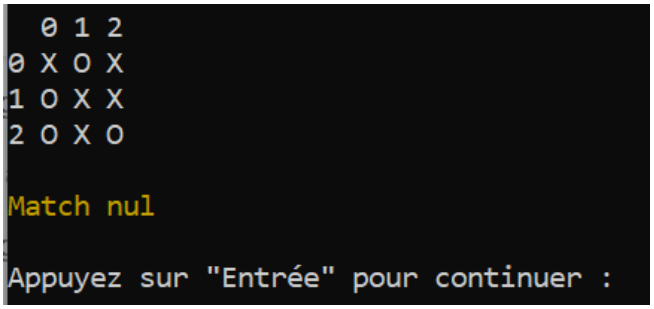
On remarque que le pion se place à l'endroit souhaité.

```
  0 1 2
0 O X -
1 - X -
2 O - -
matteo entrez la ligne : 2
matteo entrez la colonne : 1
  0 1 2
0 O X -
1 - X -
2 O X -

matteo a gagné !

Appuyez sur "Entrée" pour continuer :
```

Lorsqu'il y a un gagnant, la partie s'arrête et le programme attend que l'utilisateur appuie sur la touche "Entrée".

```
  0 1 2
0 X O X
1 O X -
2 - - -
louis entrez la ligne : 1
louis entrez la colonne : 0
Case déjà prise
louis entrez la ligne :
```

Quand un joueur essaye de poser son pion sur une case déjà occupée, un message lui est envoyé et le joueur doit recommencer sa saisie.



Lorsque toutes les cases du plateau sont remplies et qu'il n'y a aucune ligne de complète, la partie s'arrête et il y a match nul.

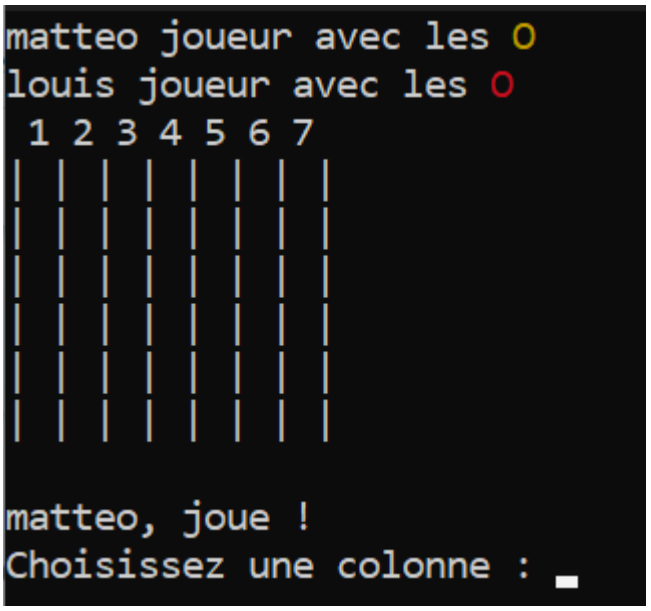
Algorithme

Pour représenter algorithmiquement le plateau du Morpion nous avons utilisé une matrice de dimension 3, 3. Ceci permet de bien représenter le plateau du Morpion.

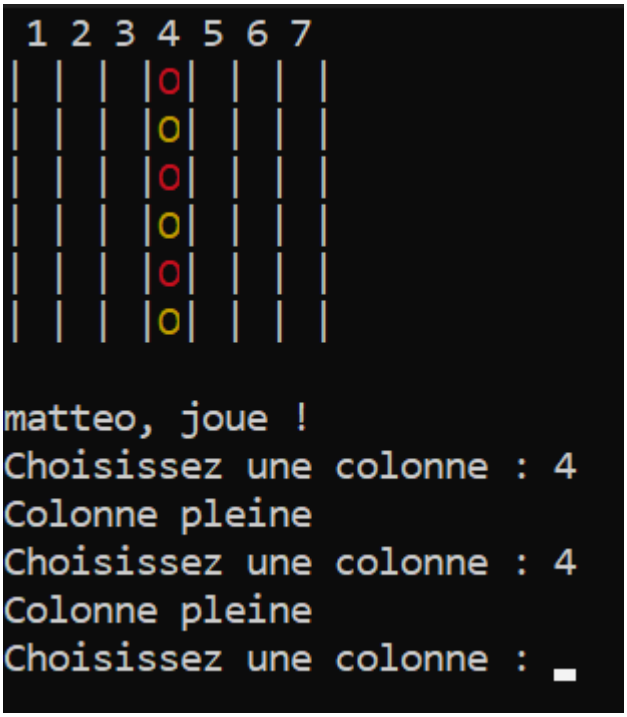
```
plateau[3][3] : chaine

plateau <- [["-", "-", "-"], ["-", "-", "-"], ["-", "-", "-"]]
```

Puissance 4

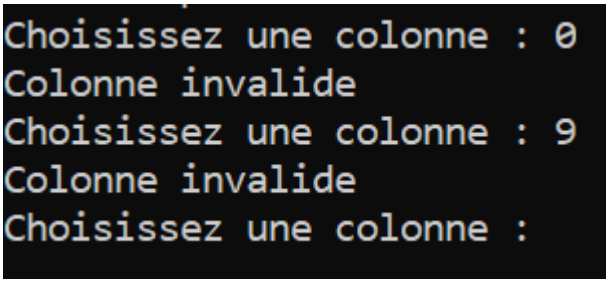
Jeux d'essaie



Pour jouer les joueurs ont juste à choisir la colonne ou il veulent placer leur pion.

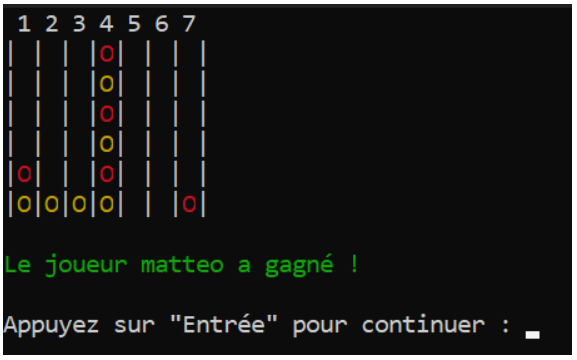


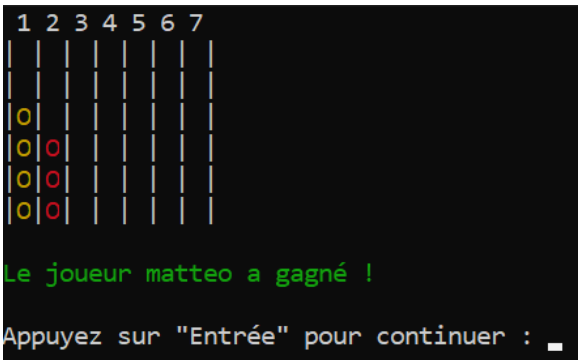
Quand une colonne est pleine, on ne peut plus placer de pion dedans et le joueur doit en choisir une autre.

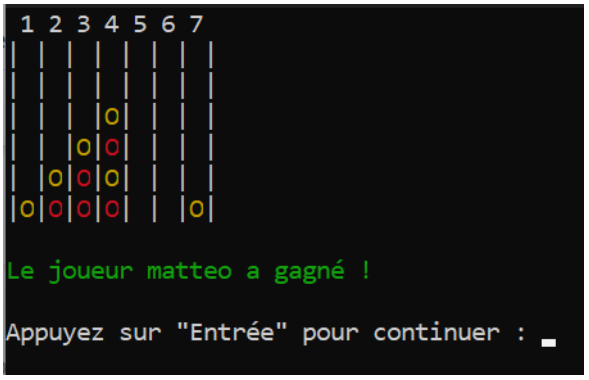


On ne peut pas non plus saisir des colonnes invalides c'est à dire inférieures à 1 ou supérieures à 7.

Détection des lignes :







Algorithme

Cette fonction parcourt chaque ligne, colonne et chaque diagonale ou il est possible d'aligner au moins quatre pions, et renvoie vrai si une ligne de quatre pions de la même couleur est formée.

```
fonction pionAligne(plateau[6][7] : Chaine, pion : Chaine) retourne bool
    """Vérifie si un pion est aligné

    Args:
        plateau[6][7] (chaine): Plateau de Jeu
        pion (chaine): Chaine affichée en tant que pion

    Returns:
        booleen: Vrai si le pion est aligné
    """
    # Vérification horizontale
    pour j allant de 0 à 6 faire
        pour i allant de 0 à 4 faire
            Si plateau[j][i] = pion et plateau[j][i + 1] = pion et plateau[j][i + 2] = pion et plateau[j][i + 3] = pion alors
                retourne vrai
            finsi
        finFaire
    finFaire

    # Vérification verticale
    pour j allant de 0 à 7 faire
        pour i allant de 0 à 3 faire
            Si plateau[i][j] = pion et plateau[i + 1][j] = pion et plateau[i + 2][j] = pion \
                    et plateau[i + 3][j] = pion alors
                retourne vrai
            finsi
        finFaire
    finFaire

    # Vérifications diagonales
    pour i allant de 0 à 3 faire
        pour j allant de 0 à 4 faire
            Si plateau[i][j] = pion et plateau[i + 1][j + 1] = pion et plateau[i + 2][j + 2] = pion et \
                    plateau[i + 3][j + 3] = pion alors
                retourne vrai
            finsi
        finFaire
    finFaire

    pour i allant de 0 à 3 faire
        pour j allant de 3 à 7 faire
            Si plateau[i][j] = pion et plateau[i + 1][j - 1] = pion et plateau[i + 2][j - 2] = pion et \
                    plateau[i + 3][j - 3] = pion alors
                retourne vrai
            finsi
        finFaire
    finFaire

    retourne faux
Fin pionAligne
```