SAé 1.02
Appréhender et construire des algorithmes

Nous devons, pour cette SAé, créer un jeu de puissance 4. Nous allons alors permettre à l'utilisateur de jouer contre un autre humain ou contre une machine qui peut être plus ou moins intelligente.

Table des matières

Implémentation

Nous avons pour cette Saé, implémenter notre Puissance 4 à notre menu de mini-jeux de la SAé 1.01 :

```
print("Mini-jeux :")
print()
print(bcolors.LightMagenta,"1 - ",bcolors.OKGREEN,"Allumettes")
print(bcolors.LightMagenta,"2 - ",bcolors.OKGREEN,"Morpion")
print(bcolors.LightMagenta,"3 - ",bcolors.OKGREEN,"Devinette")
print(bcolors.LightMagenta,"4 - ",bcolors.OKGREEN,"Puissance 4")
print(bcolors.LightMagenta,"5 - ",bcolors.OKGREEN,"Retour")
print()

choix = int(input("Choisissez une option : "+bcolors.OKYELLOW))
```

Il y a alors la possibilité de choisir le mini-jeu de Puissance 4.

Puis, nous avons relié la première procédure du Puissance 4 au programme principal :

```
elif choix == 4:
    Puissance4.ChoixP4(listJ[choix1-1],listJ[choix2-1])
```

Finalement, il ne faut pas oublier d'importer le Puissance 4 pour que le lien fonctionne.

```
import Puissance4
```

La difficulté du Puissance 4 se fera en fonction de l'intelligence du robot. Si un robot est intelligent, la difficulté est élevée, à l'inverse, si il est normal, la difficulté est plus basse.

```
choix = 0
while choix != 1 and choix != 2:
    print(bcolors.LightMagenta,"1 - ",bcolors.OKGREEN,"Le Robot est basique")
    print(bcolors.LightMagenta,"2 - ",bcolors.OKGREEN,"Le Robot est intelligent")

    choix = int(input("Choisissez une option : "+bcolors.OKYELLOW))

    if choix == 1:
        listJ[-1].mode = "Normal"
    elif choix == 2:
        listJ[-1].mode = "Intelligent"
    else:
        print(bcolors.OKRED+"Erreur.")
```

Puissance 4 :

Rappel de l'énoncé :

Nous devons pour ce projet créer un jeu de puissance 4.
Celui-ci doit être jouable entre 2 humains ou avec des machines qui doivent être alors plus ou moins intelligentes.

Analyse du travail à faire :

Pour gagner dans ce jeu, il faut aligner 4 de ses pions avant son adversaire.

Nous devons dans ce mini-jeu faire attention à plusieurs choses. Dans un premier temps, quand un joueur va déposer un pion, celui-ci doit tomber jusqu'à atterrir tout en bas ou sur un autre pion. Il faut donc faire un système pour que le pion tombe. De plus, si une colonne est déjà pleine, on ne peut pas mettre un pion en plus.

Pour vérifier un puissance 4, il faut faire attention à bien prendre en compte, les lignes, les colonnes mais aussi les diagonales. En effet, il y a de nombreux cas où 4 pions peuvent être alignés et même si les lignes et colonnes peuvent paraître logiques, les diagonales peuvent être plus cachées.

Nous allons alors, pour ce travail, réaliser 8 procédures, qui posséderont chacune un rôle bien défini :

- La première procédure "ChoixP4" permet de choisir le mode de jeu adéquat en fonction des joueurs. Alors, si il y a des deux humains ou deux robots ou un humain et un robot, la procédure redirige l'utilisateur dans la procédure de jeu adéquate. Pour ceci, il compare la typologie des joueurs.

En fonction du choix du mode de jeu (Humain vs humain ou humain vs robot ou robot vs robot), plusieurs procédures s'enchaînent :
Nous allons voir dans un premier temps le cas où les 2 joueurs sont des humains :

- La première procédure "GrilleP4" permet de créer la grille où va se jouer la partie. Il s'agit d'une grille comportant 6 rangées et 7 colonnes. Alors, la procédure crée une matrice de 14 par 14 car il faut placer les limiteurs de cases entre. Elle l'affiche ensuite avec une boucle de boucle, en effet chaque ligne est affichée une par une.

- La seconde procédure “Puissance4” est la procédure principale de placement du jeu, c’est elle qui permet aux joueurs de placer leur pion, de vérifier qu’il n’y ait pas une colonne pleine ou même d’afficher la grille si la partie n’est pas terminée.

- La troisième et dernière procédure “Verif”du jeu en “humain contre humain” est la procédure qui vérifie l’alignement des pions. En effet c’est cette procédure qui va vérifier si 4 pions sont alignés ou non. Cette procédure vérifie donc pour chaque ligne à l’aide d’un compteur (qui saute 2 par 2 car il ne faut pas vérifier les lignes de délimitation). Il s’agit d’une vérification brute, on vérifie pour toutes les cases sa ligne, sa colonne et ses diagonales, dans le but que tout soit pris en compte.
  On a décidé dans le but que l’utilisateur sache où se trouve la ligne de 4, d’afficher en vert les pions constituant celle-ci. On utilise alors le print avec `\33[1;32;42m` qui permet de donner de la couleur.

```
if fin :
    if grille[compteur][compteur1]==' r ' :
        print('Victoire de', j1.nom, '(rouge). Il gagne 1 point !')
        j1.score=j1.score+1
    else :
        print('Victoire de', j2.nom, '(jaune). Il gagne 1 point !')
        j2.score=j2.score+1
```

  Cette procédure permet finalement de mettre à jour le score des joueurs, le gagnant remporte 1 point.

Nous pouvons désormais voir le cas où il, dans l’affrontement y a un humain et un robot ou deux robots :

- Dans cette partie, les procédure “Puissance4”, “GrilleP4” et “VerifGrille” sont presques les mêmes que dans le mode “Humain contre Humain” à quelques détails près.
  Dans un premier temps, la procédure “GrilleP4” reçoit une petite partie de en plus lui permettant de savoir si un robot est intelligent (grâce au mode qu’on lui accorde).
  De plus la procédure “Puissance4” permet alors en fonction du mode de jeu, de demander ou non au joueur son nombre. En effet, si il est humain, il doit lui demander sinon, non !
  Finalement, la procédure “Verif” reste exactement la même étant donné qu’il n’y a pas de différence dans la vérification entre les machines et les humains.

- Cependant, pour cette partie, on a besoin d'une nouvelle procédure "JouerBot" qui permet de faire jouer la machine intelligement.
  En effet, on lui permet de contrer le joueur adverse si celui-ci est sur le point de gagner (en mettant un pion à l'emplacement) ou de mettre le dernier pion d'une ligne si la machine est sur le point de gagner (pour finir la partie).
  Pour faire ceci, on vérifie de manière brute si un des deux cas est sur le point de se produire, ceci avec des boucles "if" qui vérifient les cases susceptibles d'être intéressantes une par une. La procédure "JouerBot" permet alors de vérifier la plupart des cas échéant, et de dire au joueur machine si un des deux cas est vérifié, puis de placer le pion à l'emplacement adéquat pour gagner ou contrer, sinon, la procédure lui propose de placer le pion au hasard.
  Néanmoins si aucun de ces deux cas n'a lieu, il mettra un pion dans le jeu au hasard, il ne réfléchit donc pas à ses futures actions dans les détails.
  De plus, en fonction de son intelligence, il y a une part de chance pour qu'il se trompe ou pas. En effet, plus le robot est intelligent, moins il a de chance de se tromper. (mais il peut toujours se tromper, en effet, un robot intelligent a une QI de 90, alors qu'un robot normal a un QI de 50. On choisit au hasard un nombre entre 0 et 100, si ce nombre est plus élevé que le QI du robot, celui-ci se trompe.

Algorithme :

Pour gérer la difficulté du Bot, on se sert de l'intelligence du robot. Si le robot est intelligent, il s'agit d'une difficulté difficile (2) sinon c'est une difficulté normale (2).
Si la difficulté choisie par l'utilisateur n'est pas conforme, on impose comme difficulté par défaut la difficulté moyenne. Une fois dans la fonction de jeu, en fonction de la difficulté, on donne une valeur à une variable 'chance' : 50 pour facile et 100 pour difficile.
Une autre variable appelée 'proba' va prendre une valeur entre 1 et 100 à chaque tour. Si la variable 'proba' est supérieure à la variable 'chance', le Bot jouera au hasard, sinon il joue comme il est programmé pour. On a donc une chance sur deux de jouer au hasard en mode Facile et 10% pour le mode difficile.
Par exemple, pour le Puissance 4, on a :

En fonction de l'intelligence il y a une difficulté :

si difficulté = 1 alors
proba <- 50
sinon
si difficulté =2 alors
proba <- 90
sinon
Afficher 'Erreur, difficulté Normal choisi'
proba <- 50

Puissance(grille, menu, score, difficulté)
Procédure Puissance(grille[14][14]:chaine,menu: entier, score[10][5]: chaine,difficulté : entier)
Début
...
Si difficulté = 1 alors
Chance<- 50
Sinon
si difficulté = 2 alors
Chance 90
Sinon Afficher (« Erreur, difficulté par défaut : Moyen) A la ligne Chance 50
Fin si
...

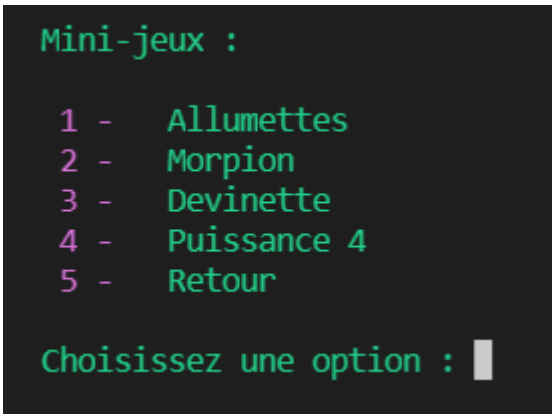
Proba <-random.randint(1,100)
si proba > chance alors
    colonne <-random.choice(CaseLibre)
 ...
Sinon Colonne <-JouerBot(caseLibre, grille)
Fin si
On répète ce processus pour chaque jeu afin d'avoir un choix de difficulté.

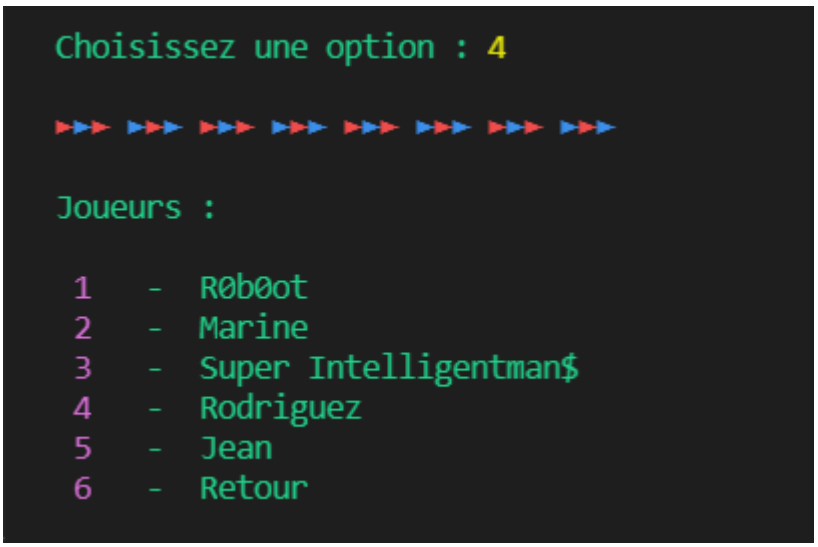
Programme :

Le code Python sera lui entièrement envoyé dans un dossier zip à part.

Jeux d'essais :

Dans un premier temps, nous vérifions que le Puissance 4 a bien été implémenté aux mini-jeux :





Nous pouvons voir qu'on nous propose bien de jouer au Puissance 4. De plus quand on ouvre le mini-jeu, il y a bien le choix des joueurs, puis quand les joueurs sont choisis. C'est bien le Puissance 4 qui se lance !

Dans le mode Humain contre humain, le jeu fonctionne bien :

L'alignement se fait bien et les deux joueurs peuvent donner la colonne dans laquelle ils veulent jouer.
Les victoires sont bien reconnues.

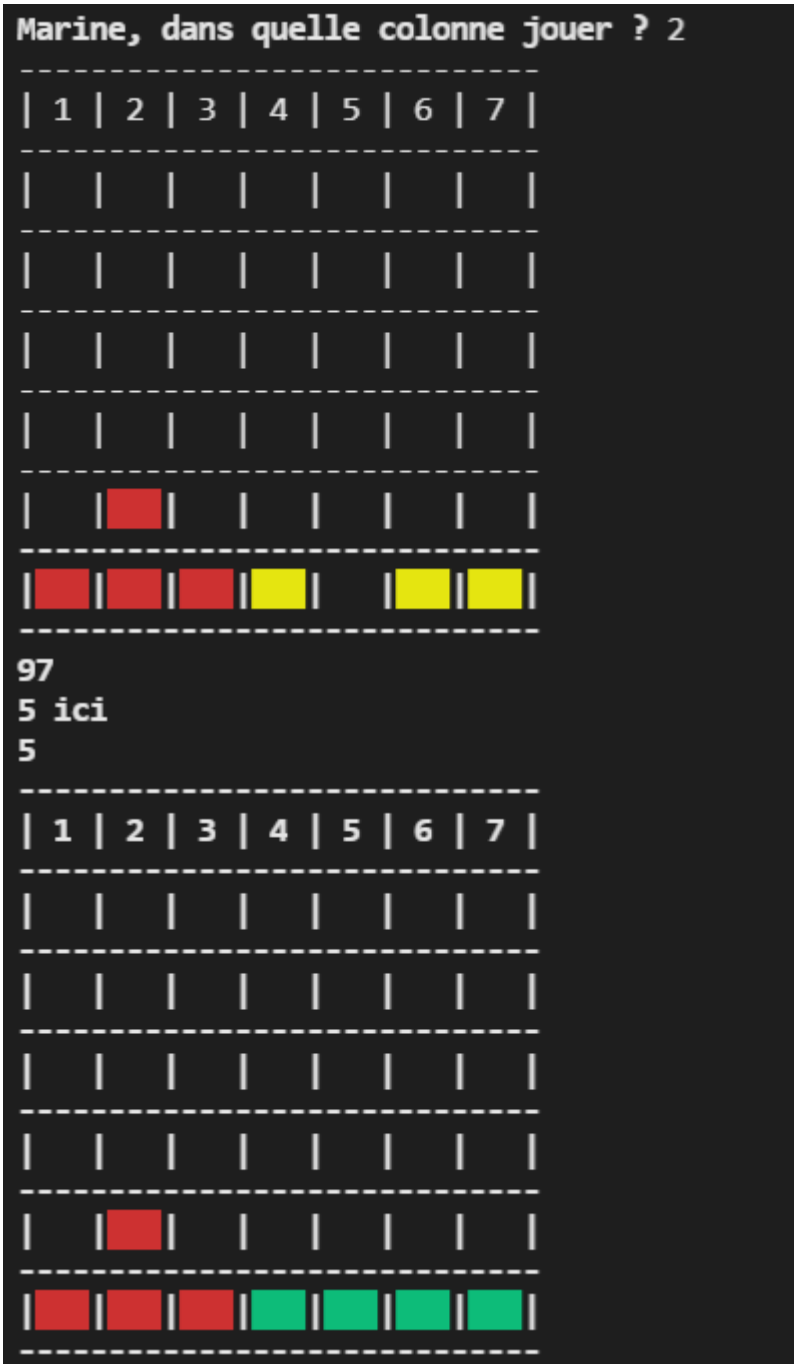
Si la colonne est pleine, il y a un message d'erreur et le joueur rejoue !


Jean,dans quelle colonne voulez-vous jouer ? 1
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
Marine,dans quelle colonne voulez-vous jouer ? 1
Erreur, colonne pleine.
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
Marine,dans quelle colonne voulez-vous jouer ?

Nous avons mis pour les jeux d'essais des petits messages qui permettent de savoir où-est ce que le joueur machine met son pion et si c'est intentionnelle ou pas (Il y a écrit "ici" ou "hasard").
Nous pouvons voir ici que la machine décide bien de bloquer intentionnellement le joueur

Ici, nous pouvons voir que le joueur machine a bien vu qu'il fallait mettre un pion dans la colonne 4 mais il ne l'a pas fait car il s'est "trompé" (on peut savoir cela grâce au message "pas de chance")

Finalement, nous pouvons voir que si le joueur machine a la possibilité de gagner, il joue bien à la bonne case pour mettre fin à la partie.



L'intelligence de la machine dépend donc de la probabilité que le joueur machine ait joué de façon intelligente ou au hasard (plus son "QI" est élevé, moins il a de chance de tomber sur un nombre au hasard entre 0 et 100 plus élevé, donc moins il a de chance de jouer mal).

Comparaison dans le mini-jeu "Devinette" :

Nous avons, dans le but de comparer plusieurs algorithmes, décidé de tester le mini-jeu devinette intelligent et pas intelligent.
Le joueur machine normal propose un nombre au hasard et réduit les possibilités petit à petit.
Alors que le joueur machine intelligent utilise un système de recherche dichotomique. En effet, la première fourchette de nombre est entre 0 et le nombre limite, le joueur machine divise à chaque fois la fourchette en deux, dans le but d'être efficace.
Nous avons alors décidé, dans un premier temps, de faire plusieurs parties entre la machine intelligente et celle normale pour voir qui gagne combien de fois.
Puis, nous avons calculé une moyenne des temps pour les deux machines pour voir la différence :

Nous avons alors lancé 10 parties :

```
9  -  Machine_Intelligente  -  Robot  -  Intelligent
1  -  Machine_Basique  -  Robot  -  Normal
```

Sur les 10 matchs, la machine intelligente a gagné 9 fois, la machine basique 1 fois. Le ratio de victoire de la machine intelligente est donc de 90% en moyenne, contre la machine normale

De plus, après avoir récupéré les temps des différentes parties nous avons calculé les moyennes :

```
machine intelligente : 9+9+7.5+10.5+9+9+7.5+10.5+6.5+9 moy=8.75sec
machine basique : 18+15+21+21+21+42+30+30+6+27 moy=23.1sec
```

Nous pouvons donc voir qu'en moyenne la machine intelligente trouve le nombre en 8.75 secondes tandis que la machine basique met 23.1 secondes. On peut alors dire qu'en moyenne la machine intelligente est 2.5 fois plus rapide que la machine basique.