RAPPORT SAE 1.02

Par : GILLOT Yann

SOMMAIRE

1.A : Présentation du sujet

Le principe de cette SAE est de créer deux robots programmés en Python permettant de jouer à une partie de Puissance 4. Le premier fera des actions de manière aléatoire, et sera considéré comme « naïf », tandis que le second sera une version qui essayera de choisir les meilleurs coups, que ce soit pour gagner ou pour éviter de perdre, et sera considéré comme « intelligent ».

1.B : Présentations des objectifs

L'objectif est d'effectuer des recherches afin de découvrir les différents algorithmes possibles afin de résoudre efficacement une partie de Puissance 4 ainsi que de savoir lequel serait le plus efficace ou encore le plus rapide.

2. : Présentation des attendus

Les attendus nécessaires sont les suivants :

- Deux fichiers pythons, un pour chaque type de bot demandé, documenté avec des commentaires et des docstrings, permettant touts les deux de retourner les colonnes dans lesquels ils souhaitent jouer pendant une partie de Puissance 4.
- La fonction principale qui devra renvoyer ce choix devra être nommée 'choix(p)' et ou p est un dictionnaire (c.f sujet).
- Un compte rendu (ce fichier).

Le rendu de cette SAE consiste en trois fichiers :

- Le compte rendu (ce fichier)
- Le fichier 'botnaif.py', contenant le bot « naïf ».
- Le fichier 'botintelligent.py', contenant le bot « intelligent ».

3.A : Benchmark des solutions (description des méthodes auxquelles vous avez pensées ou que vous avez trouvées)

Au cours de mes recherches, j'ai trouvé plusieurs méthodes afin qu'un robot puisse résoudre une partie de Puissance 4, en voici certaines que j'ai retenu car pouvant être intéressantes à utiliser :

- L'algorithme aléatoire (random)
- L'algorithme Minimax
- L'algorithme par poids
- L'algorithme de recherche d'arbre Monte Carlo

3.B : Benchmark des solutions (comparaison entre les méthodes)

L'algorithme aléatoire (random) : C'est un algorithme qui effectue des actions de manière aléatoire, sans suivre de stratégie particulière. Il peut être utile pour explorer un grand espace de recherche ou pour simuler des scénarios divers.

L'algorithme Minimax : C'est un algorithme utilisé dans les jeux à deux joueurs pour déterminer le meilleur coup à jouer. Il explore l'arbre des possibilités en supposant que l'adversaire joue de manière optimale, et choisit le coup qui maximise le gain du joueur ou minimise la perte dans le pire des cas.

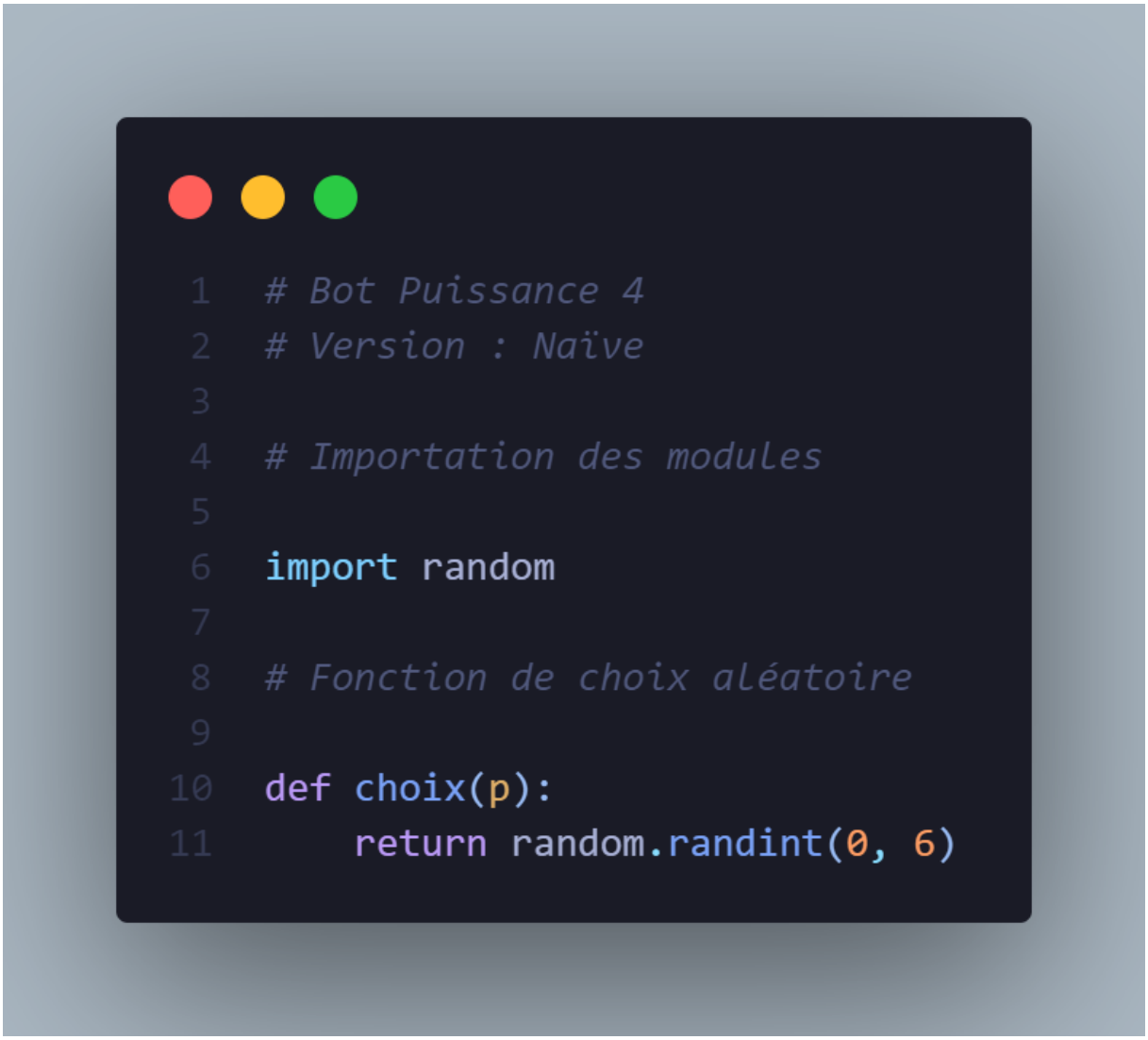
L'algorithme par poids : Cet algorithme assigne des poids aux différentes options ou actions possibles, puis utilise ces poids pour prendre des décisions. Les poids peuvent représenter l'importance relative des options ou des critères à considérer.

L'algorithme de recherche d'arbre Monte Carlo : Cet algorithme utilise des méthodes d'échantillonnage aléatoire pour évaluer les différentes branches d'un arbre de décision. Il peut être utilisé pour estimer les probabilités de

succès dans des situations complexes, notamment dans les jeux ou les problèmes de planification.

4.A : Explication de votre solution (Description des méthodes implantées)

- Au sein du fichier ‘botnaif.py’, correspondant comme dit dans son nom au bot « naïf », il utilise l’algorithme aléatoire (ou random) afin d’effectuer son coup.



```python
# Bot Puissance 4
# Version : Naïve

# Importation des modules

import random

# Fonction de choix aléatoire

def choix(p):
    return random.randint(0, 6)
```

- Au sein du fichier ‘botintelligent.py’, correspondant comme dit dans son nom au bot « intelligent », il utilise un algorithme moins puissant

que celui des poids mais permet de contrer l'adversaire lorsqu'il peut effectuer un puissance 4, et le fait quand lui le peut. Sinon, il effectue un coup aléatoire.

```python
import copy
import random

def jouables(grille: list) -> list:
    """
    Renvoie une liste des colonnes jouables dans la grille.

    Arguments:
        grille (list): La grille de jeu représentée sous forme de liste.

    Retourne:
        list: Une liste des colonnes jouables dans la grille.
    """
    colonnes = []

    for i in range(7):
        if grille[5][i] == 0:
            colonnes.append(i)

    return colonnes

def jouer(grille: list, colonne: int, joueur: int) -> list:
    """
    Effectue un coup dans la grille de jeu.

    Arguments:
        grille (list): La grille de jeu.
        colonne (int): La colonne dans laquelle effectuer le coup.
        joueur (int): Le joueur effectuant le coup.

    Retourne:
        list: La grille mise à jour après le coup.
    """
    for i in range(6):
        if grille[i][colonne] == 0:
            grille[i][colonne] = joueur
            return grille

    return grille

def gagne(grille:dict, joueur:int)-> bool:
    """
    Vérifie si le joueur a gagné dans la grille donnée.

    Arguments:
        grille (dict): La grille de jeu représentée sous forme de dictionnaire.
        joueur (int): Le joueur dont on vérifie s'il a gagné (1 ou 2).

    Retourne:
        bool: True si le joueur a gagné, False sinon.
    """
    # Vérification des lignes :
    for i in range(6):
        for j in range(4):
            if grille[i][j] == grille[i][j + 1] == grille[i][j + 2] == grille[i][j + 3] == joueur:
                return True

    # Vérification des colonnes :
    for i in range(7):
        for j in range(3):
            if grille[j][i] == grille[j + 1][i] == grille[j + 2][i] == grille[j + 3][i] == joueur:
                return True

    # Vérification des diagonales en 2 parties :
    for i in range(3):
        for j in range(4):
            if grille[i][j] == grille[i + 1][j + 1] == grille[i + 2][j + 2] == grille[i + 3][j + 3] == joueur:
                return True

    for i in range(3):
        for j in range(3, 7):
            if grille[i][j] == grille[i + 1][j - 1] == grille[i + 2][j - 2] == grille[i + 3][j - 3] == joueur:
                return True

    return False
```

```python
def troispionsplaces(grille, joueur):
    """
    Vérifie si le joueur a placé trois pions consécutifs dans la grille.

    Arguments:
        grille (list): La grille de jeu représentée par une liste de listes.
        joueur (int): Le joueur dont on vérifie les pions (1 ou 2).

    Retourne:
        bool: True si le joueur a placé trois pions consécutifs, False sinon.
    """
    # Vérification des lignes
    for i in range(6):
        for j in range(4):
            if grille[i][j] == grille[i][j + 1] == grille[i][j + 2] == joueur and grille[i][j + 3] == 0:
                return True

    # Vérification des colonnes
    for i in range(7):
        for j in range(3):
            if grille[j][i] == grille[j + 1][i] == grille[j + 2][i] == joueur and grille[j + 3][i] == 0:
                return True

    # Vérification des diagonales
    for i in range(3):
        for j in range(4):
            if grille[i][j] == grille[i + 1][j + 1] == grille[i + 2][j + 2] == joueur and grille[i + 3][j + 3] == 0:
                return True

    for i in range(3):
        for j in range(3, 7):
            if grille[i][j] == grille[i + 1][j - 1] == grille[i + 2][j - 2] == joueur and grille[i + 3][j - 3] == 0:
                return True

    return False

def simuler(grille:dict, joueur:int)-> dict:
    """
    Effectue une simulation de jeu en utilisant une copie de la grille donnée et le joueur donné.
    Sélectionne le meilleur coup parmi les coups potentiels en évaluant différentes conditions.

    Arguments:
        grille (dict): La grille de jeu représentée par un dictionnaire.
        joueur (int): Le joueur actuel (1 ou 2).

    Retourne:
        dict: Le meilleur coup à jouer.
    """
    grille_copie = copy.deepcopy(grille)

    coups_potentiels = jouables(grille)
    meilleur_coup = random.choice(coups_potentiels)

    for coup in coups_potentiels:
        grille_copie = jouer(grille_copie, coup, joueur)

        jouer(grille_copie, coup, joueur)

        if gagne(grille_copie, joueur):
            meilleur_coup = coup
        elif gagne(grille_copie, joueur%2+1):
            meilleur_coup = coup
        elif troispionsplaces(grille_copie, joueur):
            meilleur_coup = coup
        elif troispionsplaces(grille_copie, joueur%2+1):
            meilleur_coup = coup

    return meilleur_coup

def choix(partie: dict)-> int:
    grille = partie["grille"]
    tourDe = partie["tourDe"]
    dernierCoup = partie["dernierCoup"]

    if dernierCoup is None:
        return 3

    return simuler(grille, tourDe)
```

4.B : Explication de votre solution (Adaptation aux structures de données imposées par le sujet)

La principale adaptation de ce sujet est le programme principal, qui doit de ce fait être nommé ‘choix(p)’, et qui doit obligatoirement retourner la valeur dans laquelle le bot dois jouer.

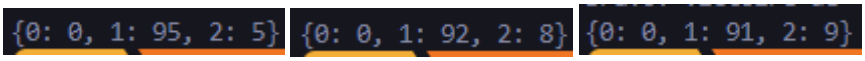
4.C : Explication de votre solution (Choix des tests - nature des tests, résultats attendus et obtenus).

Les tests se sont principalement basés sur un principe de pourcentage de victoire, ou de ce fait, des légères modifications ont été effectués sur le moteur du puissance 4 afin de pouvoir obtenir le nombre de victoires/défaites.

4.D : Explication de votre solution (Comparaison des performances et stratégies).

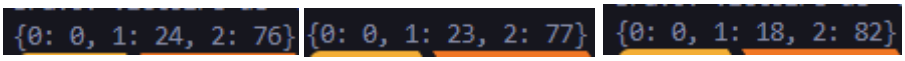
Selon les performances entre mes deux codes, soit celui du naïf et celui de l’intelligent, j’obtiens en moyenne 90% de victoire pour le programme intelligent.

```python
def testerGame():
    #mode = input("mode affichage : 1=>console 2=>graphique")
    mode = "1"
    if mode == "2" :
        foncAffichage=afficherPartieGraphique
        pygame.init()

        fenetre = pygame.display.set_mode([900,800])
        fenetre.fill([255,255,255])
        pygame.display.update()
    else:
        fenetre = None
        foncAffichage = afficherPartie
    game = creationPartie()
    afficherPartie(game,fenetre)
    ok = True
    while(ok) :

        ok = jouerUnTour(game,foncAffichage,fenetre)
    print("fin de partie")
    if game["gagnant"]!=0 :
        print("Bravo! Victoire de ",game[game["gagnant"]]["nom"])
        return game["gagnant"]
    else :
        print("C'est une égalité")
        return 0
    if mode=="2":
        quitter = False
        while not quitter:
            for event in pygame.event.get():
                if event.type==pygame.KEYDOWN:
                    quitter = True

                if event.type == pygame.QUIT:
                        quitter = True
        pygame.quit()

res = {
    0:0,
    1:0,
    2:0
}

for i in range(100):
    res[testerGame()]+=1

print(res)

# testerGame()
```



(Le bot 1 est l'intelligent, le 2 le naif)

Dans le cas inverse, le taux de victoire descend hélas à 75% de victoire.



5.A : Analyse critique du travail (Points atteints et non atteints).

Les points atteints sont dans un premier temps que les deux bots sont fonctionnels, il savent par eux même jouer à une partie de Puissance 4 et on peut calculer les pourcentages de parties gagnées.

De plus, il à la capacité d'esquiver un puissance 4 à un ennemi, et d'effectuer un puissance 4 lorsque l'occasion se présente.

Les points non atteints sont multiples, dans un premier temps, l'algorithme utilisé n'est pas celui que je souhaitait, car j'aurais voulu implémenter un algorithme Minimax, mais il se trouve que mes ambissions étaient trop grandes.

Dans un second temps, le bot est beaucoup moins efficace que prévu, est je pense qu'avec un peu plus de temps, le bot aurai pu être bien plus efficace.

5.B : Analyse critique du travail (Analyse sur l'organisation du travail).

Mon travail de recherche m'a pris une grande majorité du temps prévu en SAE, mais il m'a permis de pouvoir découvrir les meilleurs algorithmes pour jouer à des jeux de type tour par tour (Ex : Puissance 4, Echecs, Dames,…).

Mon travail en terme de développement aurai pu être meilleur, pour être franc, je suis personnellement déçu par mon rendu, pricipalement à cause du manque de temps et mon incapacité à répartir mon travail d'une bonne manière.

6. : Bilan global du projet

J'ai appris beaucoup grâce à cette SAE, j'en suis malgré tout peu fier, mais cela restera malgré tout une expérience enrichissante pour moi, et comme dit le dicton :

« Il ne faut jamais arrêter d'apprendre »

7. : Bibliographie

https://fr.wikipedia.org/wiki/Algorithme_minimax

https://fr.wikipedia.org/wiki/Recherche_arborescente_Monte-Carlo

https://fr.wikipedia.org/wiki/%C3%89lagage_alpha-b%C3%AAta

http://connect4.ist.tugraz.at:8080/ (pour savoir comment fonctionnait l'un des algorithmes, et pour comparer comment fonctionnait mon algorithme face à un des meilleurs possibles)