

# Jeu de puissance 4

17 novembre 2018

## 1 Structure du programme

Variables globales :

$N1 = 7$ ,  $N2 = 6$  qui sont la taille en x,y du tableau

T : un tableau de taille  $N1 * N2$

joueur : entier 1 ou 2 qui est le numero du prochain joueur

### 1.1 Fonction : Affiche(T)

Fonction qui sert à mettre au point le programme.

en entrée : T : tableau

en sortie écrit le contenu du tableau dans le terminal

### 1.2 Fonction $y = \text{ligne}(T, x)$

entrée : T : tableau, x : colonne

sortie : y : premiere ligne qui a une case vide, et  $y=N2$  si la colonne x est pleine

### 1.3 Fonction : Dessin\_cadre()

Pour le départ dessine les lignes verticales

### 1.4 Fonction $n = \text{evaluation}(T)$

note = evaluation(T)

— en entrée : tableau T

— en sortie : note  $\in [-1, +1]$  telle que

— note = -1 si le tableau contient 4 pièces de l'adversaire, alignées.

— sinon, note = +1 si le tableau contient 4 pièces de l'ordinateur, alignées.

— sinon, une note intermédiaire. (note =0 est possible).

**Algorithme :** parcourir le tableau et chercher des alignements de 3 ou 4 pièces identiques compter le nombre de ces alignements.

## 1.5 Fonction $y=Joue(x)$

entree :  $x$  colonne où le joueur joue

sortie : Si la colonne  $x$  n'est pas pleine, on renvoie  $y$  : ligne où sera le pion. On dessine le pion et on modifie le tableau  $T$ . On renvoie  $y = N2$  si la colonne est pleine (pas joué)

## 1.6 Fonction $on\_click(event)$

Cette fonction est appelée si on clique avec la souris dans une case.

Elle fait jouer le joueur qui a cliqué et ensuite l'ordinateur si on joue contre l'ordinateur.

## 1.7 Programme final

On appelle la fonction `Dessin_cadre()` et on indique au programme de suivre les commandes de la souris (en python) :

```
plt.connect('button_press_event', on_click) # associe la fonction aux evenements souris
plt.show() # bloque le programme ici pour gérer les evenements
```

## 2 Algorithme pour que l'ordinateur joue contre un humain (ou contre lui même)

On pose

—  $H = 4$  : profondeur de la recherche arborescente.

### 2.1 Fonction : $x, n = Note(T, h)$

$x, n = Note(T, h)$

— en entrée :

— tableau  $T$

— entier  $h = 0, 1, 2, \dots, H$  est une profondeur, signifiant que  $h$  pair : c'est à l'ordinateur de jouer,  $h$  impair : c'est à l'adversaire de jouer. Et la note sera obtenue par une exploration jusqu'à la profondeur  $H$ .

— en sortie :

—  $x$  : colonne où l'ordinateur (si  $h$  pair) ou l'adversaire (si  $h$  impair) doit jouer.

—  $n$  : note  $\in [-1, 1]$ .

### Algorithme :

- On initialise  $x_0 = -1$  et  $n_0 = 0$
- Ecrire boucle  $x = 0 \rightarrow N_1 - 1$  :
  - On pose  $y = \text{ligne}(T, x)$
  - Si la colonne  $x$  du tableau  $T$  est pleine, on passe au  $x$  suivant (commande continue). Pour cela on utilise la fonction  $\text{ligne}(T, x)$ .
  - On crée tableau :  $T' = T + \text{pièce } 1, 2$  dans la colonne  $x$  où le numéro 1 ou 2 est selon que  $h$  est impair ou pair.
  - On pose  $n = \text{evaluation}(T')$ .
  - Si  $h < H$  et  $n > -1$  et  $n < 1$  (i.e. la configuration n'est pas gagnante ni pour l'ordinateur ni pour l'adversaire) alors on appelle  $x', n = \text{Note}(T', h+1)$
  - Si  $x_0 == -1$  alors on initialise  $x_0 = x$  et  $n_0 = n$
  - Si  $h$  est pair et  $n > n_0$  alors on pose  $n_0 = n$ ,  $x_0 = x$  et si  $n = 1$  on renvoie  $x_0, n_0$
  - Si  $h$  est impair et  $n < n_0$  alors on pose  $n_0 = n$ ,  $x_0 = x$  et si  $n = -1$  on renvoie  $x_0, n_0$
- On renvoie  $x_0, n_0$

## 2.2 Fonction Test()

Pour mettre au point le programme, il peut être utile d'écrire une fonction `Test()` qui

- déclare un tableau avec certaines pièces déjà présentes. Par exemple :

```
global T
T = np.zeros([N1, N2])
T[1,0], T[1,1], T[1,2] = 2,2,2
```

- Affiche  $T$  et appelle la fonction `Note(T, 0)`

## 2.3 Conséquence

Si c'est à l'ordinateur de jouer, il doit jouer dans la colonne  $x$  donnée par  $x, n = \text{Note}(T, 0)$