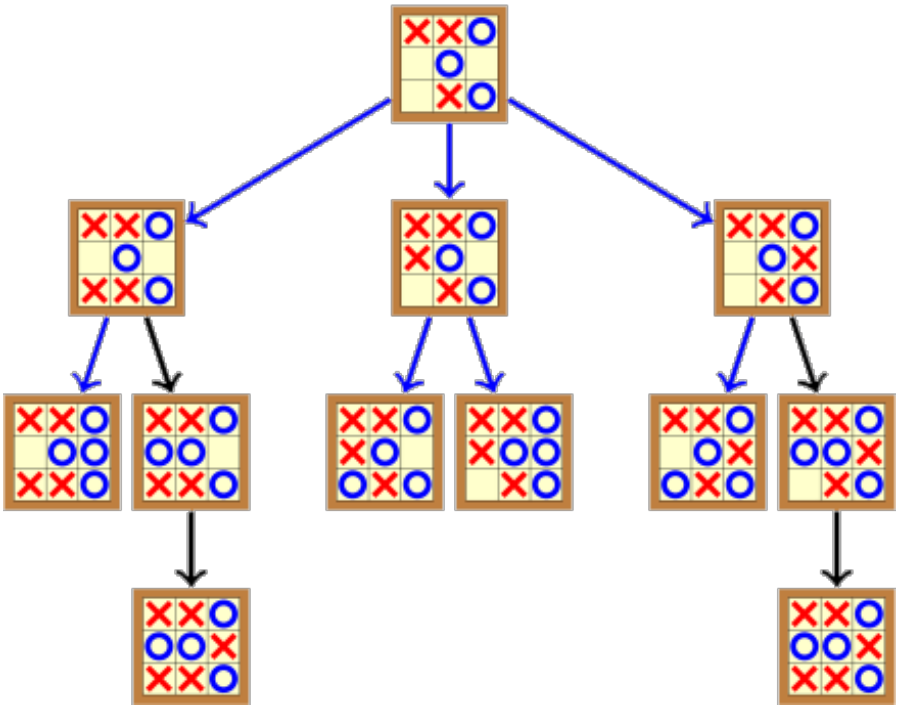
Année 2015-2016
Prépa CPP-2A
Matthieu ZIMMER
Jean-Philippe MANGEOT

Projet Othello
TP4 : Quelques pistes pour votre IA

Recherche dans un arbre

Le but de cette technique est de calculer les coups futurs qui vont être joués. Pour cela, il va falloir les stocker dans une structure arborescente. En prédisant les prochains coups, on cherche à prendre l'avantage sur l'adversaire en modélisant toutes les possibilités.



Python ne propose pas d'arbre, il faudra utiliser un tuple pour représenter un nœud.
Nœud(Nœud : pere, Liste<Noeud> : fils, Tuple<Damier, Entier> : data, entier : profondeur)

La racine initiale de l'arbre s'initialise de cette façon :

Remarquez que la liste des fils est vide au départ et qu'il n'y a pas de père pour la racine.

```
pere = ()
fils = []
data = (initialiserDamier(), joueur)
profondeur = 0
racine = (pere, fils, data, profondeur)
print(racine)
((),
[],
([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 2, 1, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 2, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
 1),
0)
```

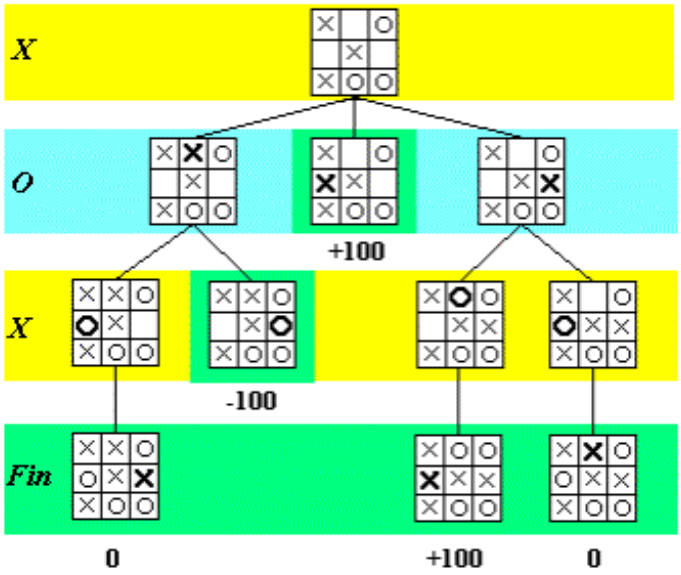
Pour développer la suite des nœuds, voici un exemple de recherche en profondeur :

```
def rechercheProfondeur(noeud, profondeur_max):
    #noeud[1]: liste des fils
    #noeud[2][0]: damier du père
    #noeud[2][1]: tour du joueur du père
    #noeud[3] : profondeur
    if(noeud[3] == profondeur_max):
        return

    for nouveau_damier in damiers_possibles(noeud[2][0], noeud[2][1]):
        nouveau_noeud=(noeud, [], (nouveau_damier, 3-noeud[2][1]), noeud[3] + 1)

        #développement du nouveau noeud
        rechercheProfondeur(nouveau_noeud, profondeur_max)

        #ajout du noeud fils dans la liste du père
        noeud[1].append(nouveau_noeud)
```

Après un appel à cette fonction, la racine va se peupler :

```
rechercheProfondeur(racine, 1)
print(racine)

((),
 [((...),
   [],
   ([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 1, 1, 0, 0, 0, 0],
     [0, 0, 0, 0, 1, 2, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
    2),
   1),
  ((...),
   [],
   ([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
     [0, 0, 0, 0, 1, 2, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
    2),
   1),
  ((...),
   [],
   ([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 2, 1, 0, 0, 0, 0],
     [0, 0, 0, 0, 1, 1, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
    2),
   1),
  ((...),
```

Une simple recherche en profondeur ne suffira pas pour pouvoir faire jouer votre intelligence artificielle de la bonne manière. Il faut modéliser le comportement de l'adversaire en supposant qu'il joue le meilleur coup possible (de son point de vue) grâce à l'**algorithme min-max**.



Étant donné que le nombre de coup joués est trop important par rapport au Tic Tac Toe, l'utilisation d'une heuristique sera obligatoire. Elle associera à un damier une estimation de son score. Plus elle sera précise et meilleures seront les décisions.

Pistes supplémentaires :

- Tout ce qui a été calculé avant de jouer une partie vous permettra de gagner du temps pendant la partie. Ainsi vous pouvez pré-calculer les meilleurs coups de plusieurs damiers (proche d'une fin de partie).
- Paralléliser l'exploration des nœuds
- Elaguer les branchements à l'aide de l'algorithme **alpha-beta**
- Utiliser une heuristique plus précise :

500	-150	30	10	10	30	-150	500
-150	-250	0	0	0	0	-250	-150
30	0	1	2	2	1	0	30
10	0	2	16	16	2	0	10
10	0	2	16	16	2	0	10
30	0	1	2	2	1	0	30
-150	-250	0	0	0	0	-250	-150
500	-150	30	10	10	30	-150	500

exemple d'heuristique en 8x8

- **pour les plus courageux**

Année 2015-2016
Prépa CPP-2A
Matthieu ZIMMER
Jean-Philippe MANGEOT

Apprentissage statistique

Le but de cette méthode est d'apprendre un modèle mathématique qui permette de caractériser un damier.

Modèle :

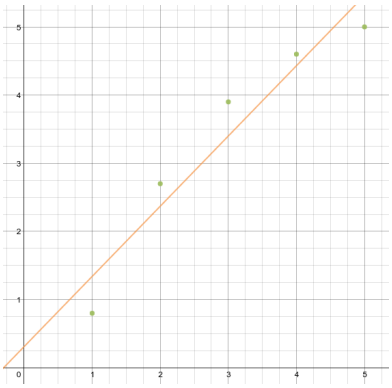
$$V_\theta : Damier \rightarrow \mathbb{R}$$

La forme du modèle est prédéfinie mais ajustable à travers les paramètres θ.

Exemple simple : Si le modèle est définie comme l'ensemble des fonctions affines en dimension 1

$$V_\theta(x) = \theta_0 x + \theta_1$$

Et qu'on tente d'approximer des données (x1, y1), …,(xn, yn), on trouverait ce résultat :



Un modèle linéaire ne pourra pas approximer assez précisément cette fonction pour un damier. De plus l'entrée n'est pas en dimension 1, mais 100 (nombre de case).
Une possibilité est d'utiliser un réseau de neurone :

$$V_\theta(x) = \sum_{i=1}^{h} \left(\theta_i \times \psi\left[\sum_{j=1}^{100} (\theta_{j,i} \times x_j) \right] \right)$$

h:nombre de neurone dans la couche cachée
ψ : fonction sigmoïde

Pour apprendre les poids θ, il est possible d'utiliser l'algorithme de back-propagation (descente de gradient).

Année 2015-2016
Prépa CPP-2A
Matthieu ZIMMER
Jean-Philippe MANGEOT

Critère : Apprentissage par renforcement

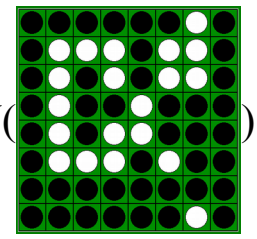
On cherche à optimiser les meilleurs θ du critère suivant :

$$V_\theta(damier) = R(damier) + \gamma \max_{prochain\_damier} V_\theta(prochain\_damier)$$

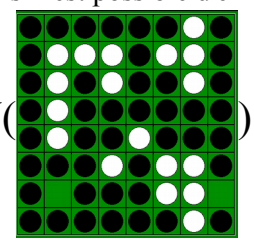
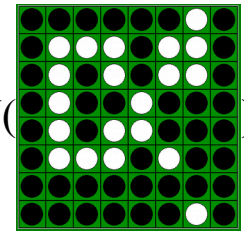
Dans le cas présent, on fixera γ = 0.95 (centaine de décision). Plus γ est grand, plus le futur est important. La fonction R s'appelle fonction de récompense, c'est celle qu'on cherche à maximiser.

$$R(damier) = \begin{cases} 1 & \text{si } gagnant \\ -1 & \text{si } perdant \\ 0 & \text{sinon} \end{cases}$$

En partant des damiers finaux, on peut apprendre les premières valeurs de V.

V() = -1

Puis il est possible d'en déduire la suite :

V() = 0 + γ V() = -γ = -0.95

On renforce petit à petit la base de connaissance en jouant de nombreuses parties.

Pistes supplémentaires :

- Plutôt que de générer des parties totalement aléatoires, il est plus efficace de faire jouer son IA contre elle même. Ainsi elles générerons des parties bien plus intéressantes.