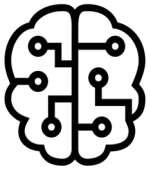
L’intelligence d’un morpion



1. Complexification du jeu

But de cette partie : Augmenter la taille du plateau afin complexifier le jeu et développer les solutions gagnantes de l’IA.

Analyse de l’algorithme d’IA

L’algorithme utilisé pour modéliser l’IA du morpion teste systématiquement l'ensemble des solution du jeu et pondère les coups gagnants.

Cet algorithme entre dans la catégorie des algorithmes de backtracking. Lorsqu’une solution est trouvée, la méthode revient sur les affectations qui ont été faites précédemment (d'où le nom de retour sur trace) afin d’en explorer de nouvelles.

L’algorithme de backtracking est très utilisé pour répondre à des problèmes de recherche et d’optimisation. Le principe de l’algorithme est simple : construire dynamiquement une solution au problème en faisant croître des parties de solution, tant que ces parties sont suffisamment prometteuses.

Q1. **Indiquer** le type de parcours d’arbre effectué par cet algorithme.

Parcours en profondeur postfixe

La recherche des meilleurs solutions impose à notre algorithme de parcourir l’ensemble des coups possibles. Augmenter la taille du plateau aura pour conséquence d’augmenter le temps de calcul. La question qui se pose est de savoir si notre algorithme se terminera en un temps acceptable lorsque le plateau deviendra grand.

Complexité du problème

Pour rappel, la théorie de la complexité étudie la quantité de ressources (temps, espace mémoire, etc.) dont a besoin un algorithme pour résoudre un problème algorithmique. Dans notre cas la complexité de notre algorithme dépend directement du nombre de cases sur le plateau.

Q2. **Justifier** que cet algorithme est de complexité factorielle **O(n!)**

Au premier coup, 8 coups sont possibles puis 7 pour chacun des 8 coups initiaux. Puis 6, puis 5....

On est en présence d'une fonction factorielle. La complexité est bien O(n!) avec n représentant le nombre de cases.

Q3. **Calculer** le nombre cas que l'algorithme devra étudier si le plateau possède 9 (3x3) puis 16 (4x4) puis 25 (5x5) cases.

Taille plateau	Nombre solutions (approximation)
9 cases	9 ! = 362880
16 cases	16 ! = 20922789888000
25 cases	25 ! = 15511210043330985984000000

On considère le temps nécessaire à la recherche d'une solution à 50 opérations machines.

Q4. **Estimer** le temps de calcul qu'il faudra à un processeur 1GHz pour terminer l'algorithme dans les trois cas précédents (9, 16, 25 cases).

Taille plateau	Temps
9 cases	18ms
16 cases	1046139 s
25 cases	7.75e+17s = 24592862194 années

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

def calcul_temps(n : int) :
    return factorial(n) * 50 * 1*10**(-9)
```

Q5. **Conclure** sur la possibilité d'augmenter la taille du plateau en conservant cet algorithme.

Cet algorithme ne terminera pas dans un temps raisonnable au-delà de 9 cases.

Amélioration de l'algorithme

Une première piste d'amélioration de performance de notre algorithme consiste à ne pas explorer une solution qui l'a déjà été (figure 3). Pour cela, il convient de mémoriser au fur et à mesure, les différents coups avec leur score.

Cette technique, dite technique de **mémoïsation**, réduit le temps de calcul (coût en temps) au détriment de l'occupation mémoire (coût spatial).

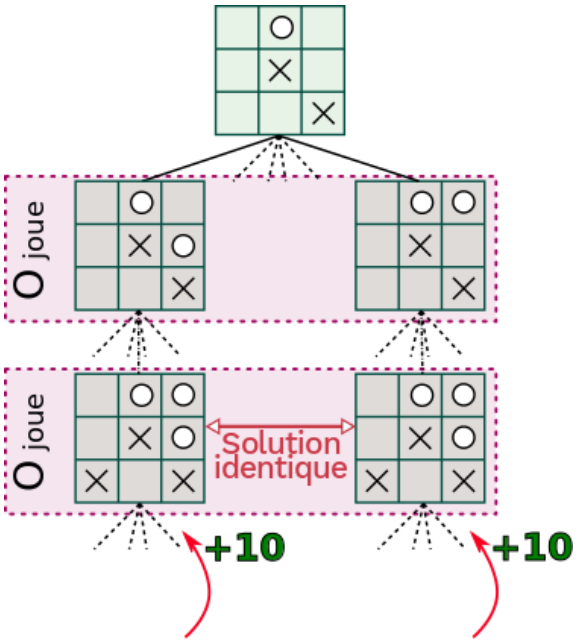


Figure 1: Exemple de situation identique

Implémentation de de la mémoïsation

L'ensemble des solutions / score sera stocké dans un dictionnaire, les états du plateau constituant les clés de ce dictionnaire et les scores seront les valeurs associées.

Les objets listes ne pouvant pas constituer une clé valide (objet non hashable), une chaîne de caractère composée des caractères de la liste séparés par des virgules servira de clé. Par exemple la solution de la figure 3 sera stockée de la manière suivante :

```
memo → { … , '.,o,o,.,x,o,x,.,x' : 10 , … }
```

La concaténation des caractères de la liste séparés par des virgules peut être obtenu avec l'instruction :

```
>>> ','.join(['.','o','o','.','x','o','x','.','x'])
'.,o,o,.,x,o,x,.,x'
```

Le chemin inverse est obtenu avec la méthode split() :

```
>>>'.,o,o,.,x,o,x,.,x'.split(',')
['.','o','o','.','x','o','x','.','x']
```

La méthode minmax_memo() suivante implémente cette mémoïsation.

```
def minmax_memo(self,joueur):
      scoreBranches = []
      for coup in self.coupsRestants():
            score=self.evaluerCoup(joueur,coup)
            if score==None:
                  ##### Test si la cas est dans le dico de memoisation
                  if ','.join(self.getPlateau()) in self.memo :
                        #### Retourne le score sans tester les sous arbres
                        score = self.memo[','.join(self.getPlateau())] Q9

                        #### Efface le coup joué
                        self.jouer('.', coup)
                        return score

            score,_=self.minmax_memo(self.adversaire(joueur) )

            scoreBranches.append((score,coup))
            self.jouer('.', coup) # efface le coup joué

      if joueur=='x' :
            score = max(scoreBranches)
            ###### Ajout de la memorisation du score si 'x' joue
            self.memo[','.join(self.getPlateau())] = score  Q10
            ######
            return score
      else :
            score = min(scoreBranches)
            ###### Ajout de la memorisation du score si 'o' joue
            self.memo[','.join(self.getPlateau())] = score  Q10
            ######

            return score
```

Q6. **Compléter** la zone repérée *Q9* afin de renvoyer le score si l'état du plateau est dans le dictionnaire self.memo

Q7. **Compléter** la zone repérée *Q10* afin de mémoriser l'état du plateau et le score dans la variable self.memo

Evaluation des performances de la mémoïsation

Le répertoire .\donnee\NSI\activite24\performances contient la classe MorpionIA et une bibliothèque d'évaluation des performance des algorithmes de backtracking avec et sans mémoïsation (runtest.py). Les fonctions suivantes sont disponibles :

➔ runtest.compare_performances() : Compare les performances en temps des deux algorithmes pour un certain nombre de coups restants sur un plateau de 9 cases

➔ runtest.performance_IA_memo() : Mesure les performances en temps et espace de l'algorithme de backtracking avec mémoïsation.

Q8. **Ouvrir** et **exécuter** le fichier MorpionIA.py puis **exécuter** les commandes de test

```
**************************************************************************
******************** Rapport de performance ******************************
**************************************************************************
Conditions : |Plateau 9 cases
             |Nombres de coups restants variant de 3 à 9 (plateau vide)

--------------------------------------------------------------------------
Nombre de coups |temps execution sans memo  |temps execution avec memo |
--------------------------------------------------------------------------
     3          |6.580352783203125e-05      |6.771087646484375e-05     |
     4          |0.00021147727966308594     |0.00019168853759765625    |
     5          |0.001959085464477539       |0.0011172294616699219     |
     6          |0.011432170867919922       |0.0029227733612060547     |
     7          |0.06593871116638184        |0.00916147232055664       |
     8          |0.3553142547607422         |0.025458097457885742      |
     9          |3.9537580013275146         |0.08855223655700684       |
```

```
**************************************************************************
******************** Rapport de performance ******************************
**************************************************************************
Conditions : |Plateau 9 et 12 cases
             |Algorithme de backtracking avec memoisation

--------------------------------------------------------------------------
Nombre de cases |temps execution           |taille du dictionnaire    |
--------------------------------------------------------------------------
     9          |0.09772729873657227       |1713                      |
     12          |3.3351640701293945        |27483                      |
```

Q9. **Analyser** les performances affichées et conclure en décrivant les avantages / inconvénients de la mémoïsation. **Préciser** les limites des deux algorithmes quant à leur capacité à supporter l'agrandissement du plateau.

La mémoïsation permet de réduire de façon importante le temps d'exécution de l'algorithme pour un plateau 9 cases (88ms contre 3,95s). Sans cette technique l'algorithme de backtracking simple est inefficace pour un plateau supérieur à 9 cases.

La mémoïsation demande malgré tout un grand espace mémoire pour stocker les coups. Entre un plateau 9 et 12 cases l'espace mémoire requis est multiplié par 16. Bien que cette technique permettrais d'augmenter la taille du plateau de 9 à 12 cases, cela reste insuffisant au-delà.

La complexité de cet algorithme reste en O(n!) et ne permet pas d'appliquer cette technique à un plateau de 16 cases et plus.