HMIN233 - Algorithmes d'exploration et de mouvement

Recherche de solutions

Suro François

Université de Montpellier
Laboratoire d'informatique, de robotique
et de microélectronique de Montpellier

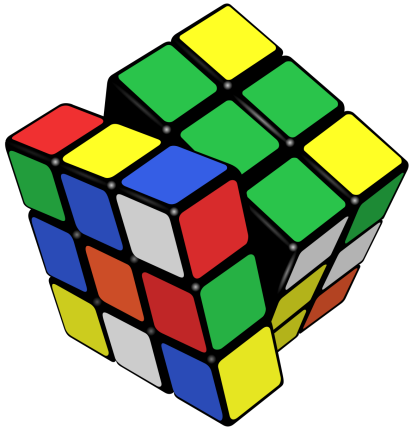
Janvier 2021

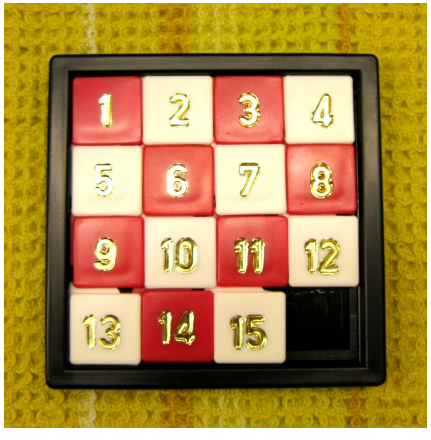
Problème ?

Comment atteindre un but à partir d'un "point de départ" ?

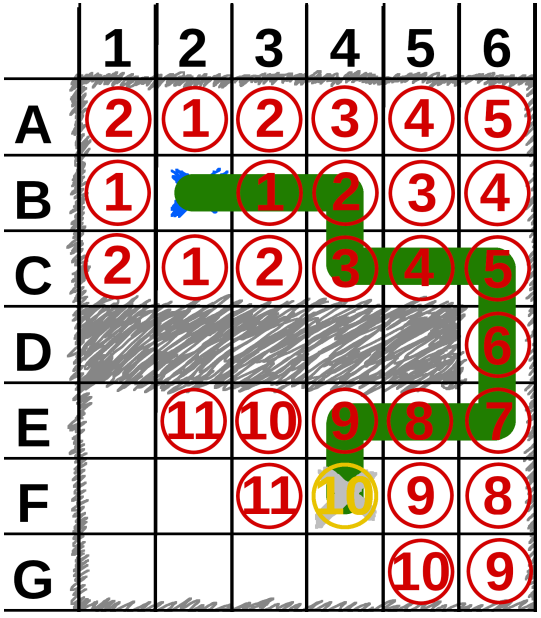
- On réalise une séquence d'actions

```
77   public static int calculerCout(int distances[][],int chemin[],int nombreVilles)
78   {
79      int precedent = 0;
80      int result = 0;
81      for(int x = 0 ; x < nombreVilles-1 ; x++)
82      {
83         result += distances[precedent][chemin[x]];
84         precedent = chemin[x];
85      }
86      result += distances[precedent][0];
87      return result;
88   }
```

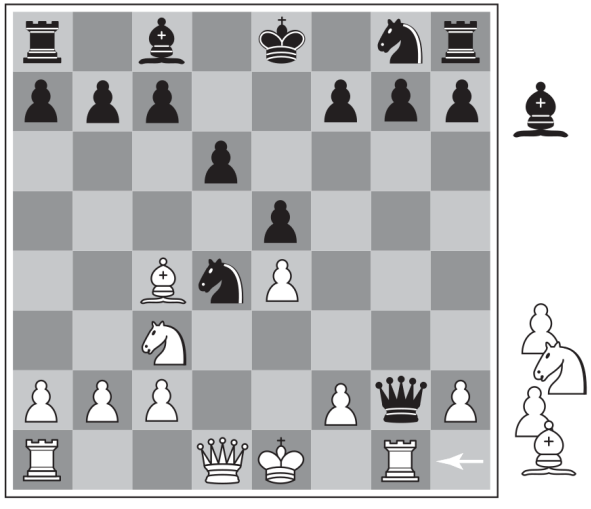


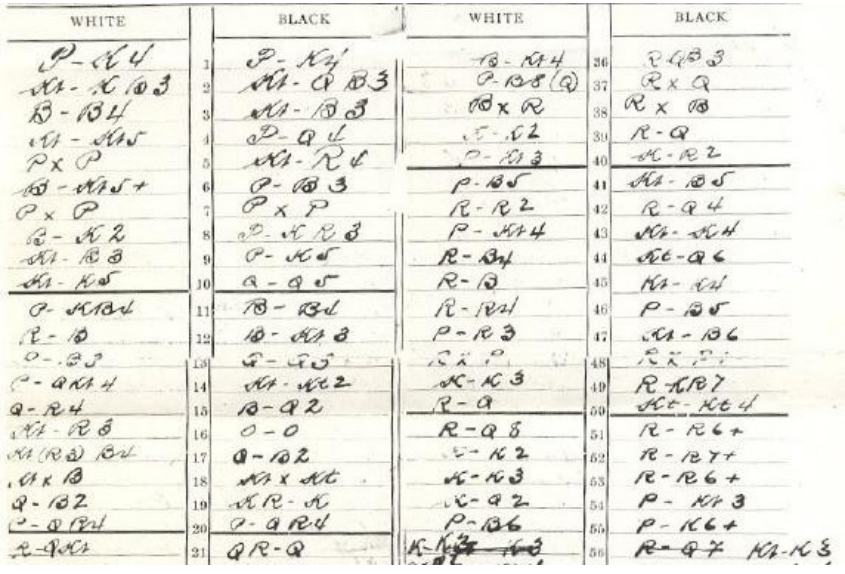


Solution



Droite-Droite-Bas-Droite-Droite-Bas-Bas-Gauche-Gauche-Bas





Recherche de solution

Éléments d'un problème:

- États (dont l'état initial et le but).
- Actions et transitions.

L'espace des états

Ces éléments définissent l'espace des états, un graphe orienté dont les noeuds sont les états et les arcs sont les actions.

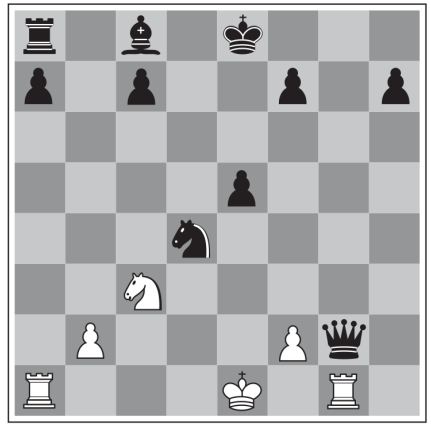
Solution

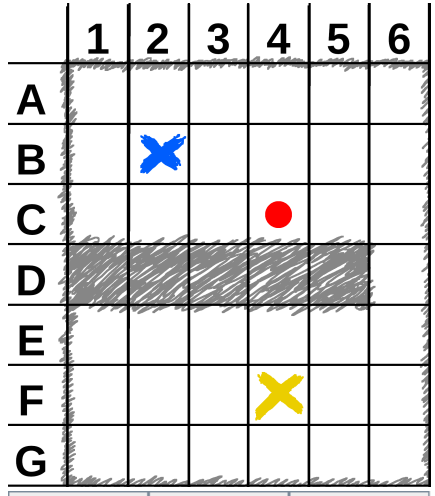
Une solution est un chemin dans l'espace des états qui va de l'état initial au but.

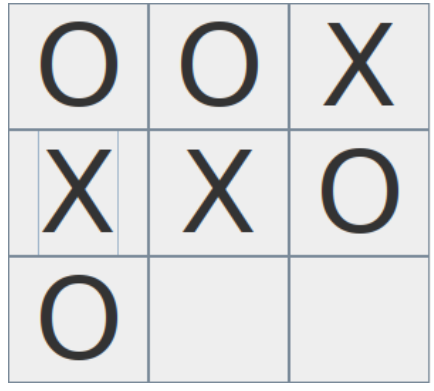
États

Une configuration possible du problème

- Position des pièces.
- Position de l'agent.
- Placement des marques.
- Étape dans une recette ...







Transitions

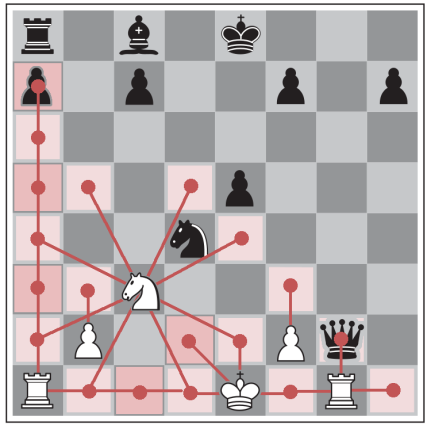
Actions

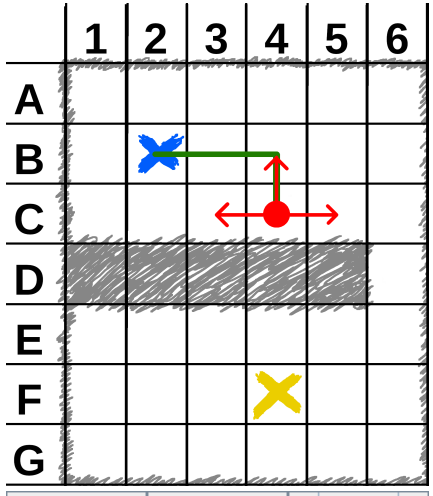
- Déplacement d'une pièce du joueur.
- Déplacement de l'agent.
- Dépôt d'une marque.
- Battre les blancs en neige ...

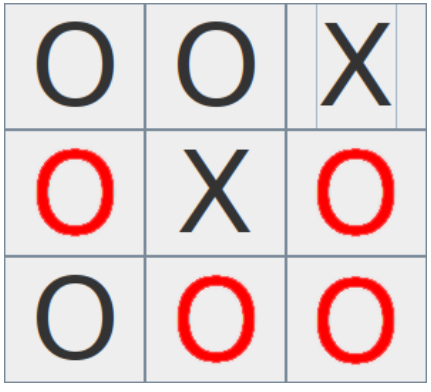
Modèle de transition

- Génère un nouvel état à partir de l'état courant et d'un action.

$$S' = T(S, A)$$







États initial/final/but

État initial

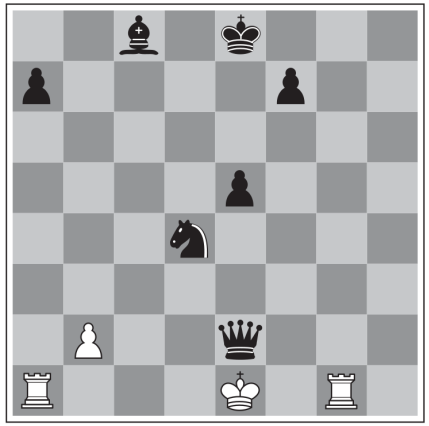
- à partir duquel on lance la recherche.

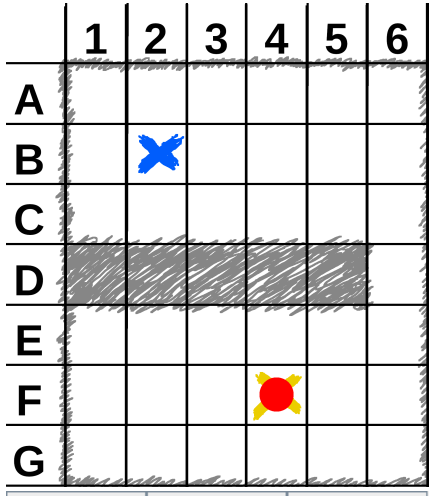
État final

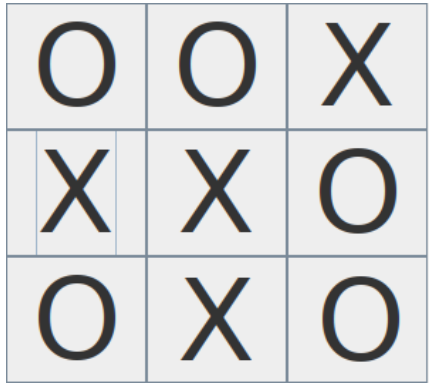
- Un ou plusieurs.

But

- Un état final particulier.
- Test du but : défini par certaines propriétés.



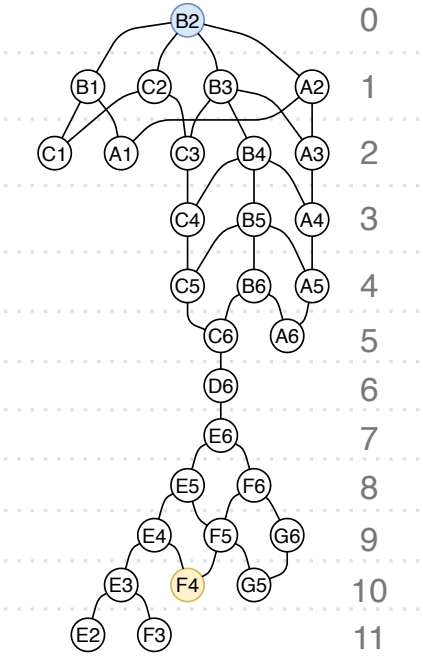




Graphe de recherche

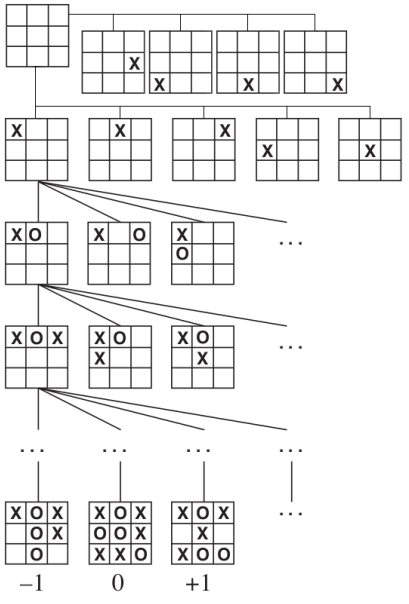
L'espace des états

- Les états sont liés les uns aux autres par des actions.
- Modélisation sous forme de graphe :
  - État : noeud
  - Action : arc
  - Transition : passer au successeur d'un noeud.



Solution

Une solution est un chemin dans ce graphe qui va de l'état initial au but.



Problèmes réels

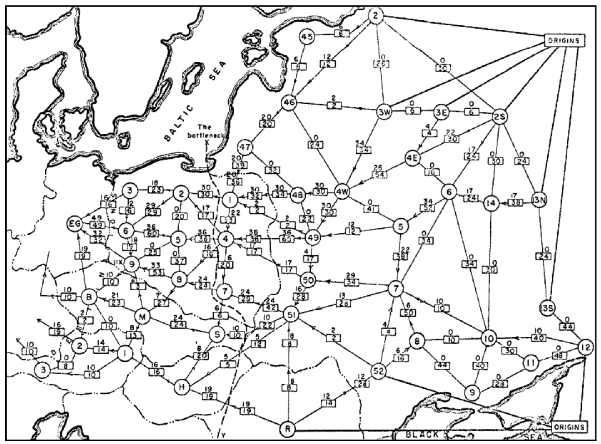
Abstraction

Retirer le maximum de détails de la représentation

- Des état
- Des actions

Une abstraction est utile si:

- elle n'empêche pas la résolution du problème
- les étapes de la solutions ne sont pas des problèmes (plus complexes) à leurs tour.



Recherche de solution

Pour rechercher on construit notre graphe de recherche en considérant un noeud à la fois.

- On applique toutes les actions valides sur le noeud courant pour générer de nouveaux noeuds.
- On sélectionne un noeud pour continuer la recherche, les autres sont conservés au cas ou notre choix ne mène pas au but.

$\rightarrow$ *On espère trouver la solution sans avoir à construire le graphe en entier ...*

Les algorithmes de recherche partagent tous cette structure. Ils ne diffèrent que par le choix du noeud suivant, la stratégie de recherche.

Recherche non-informée ("aveugle")

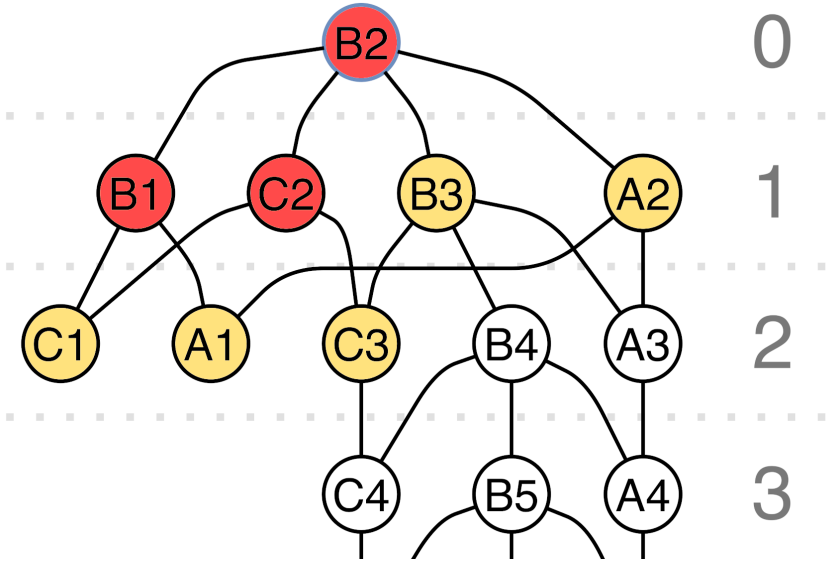
→Pas d'heuristique disponible.

Parcours de graphe

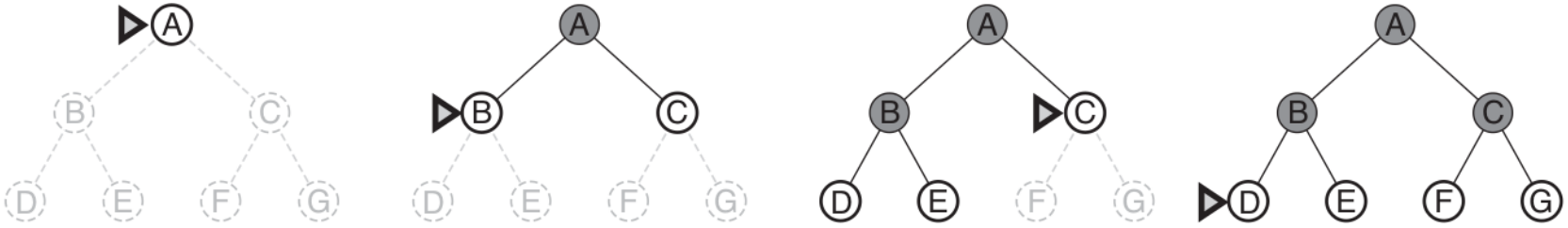
- Largeur.
- Profondeur.

Frontière

Ensemble des noeuds directement accessible à partir des noeuds visités.



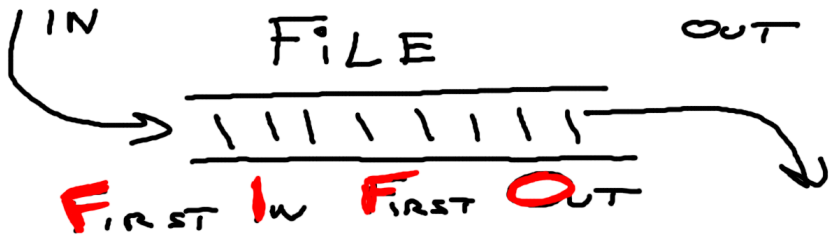
Parcours en largeur



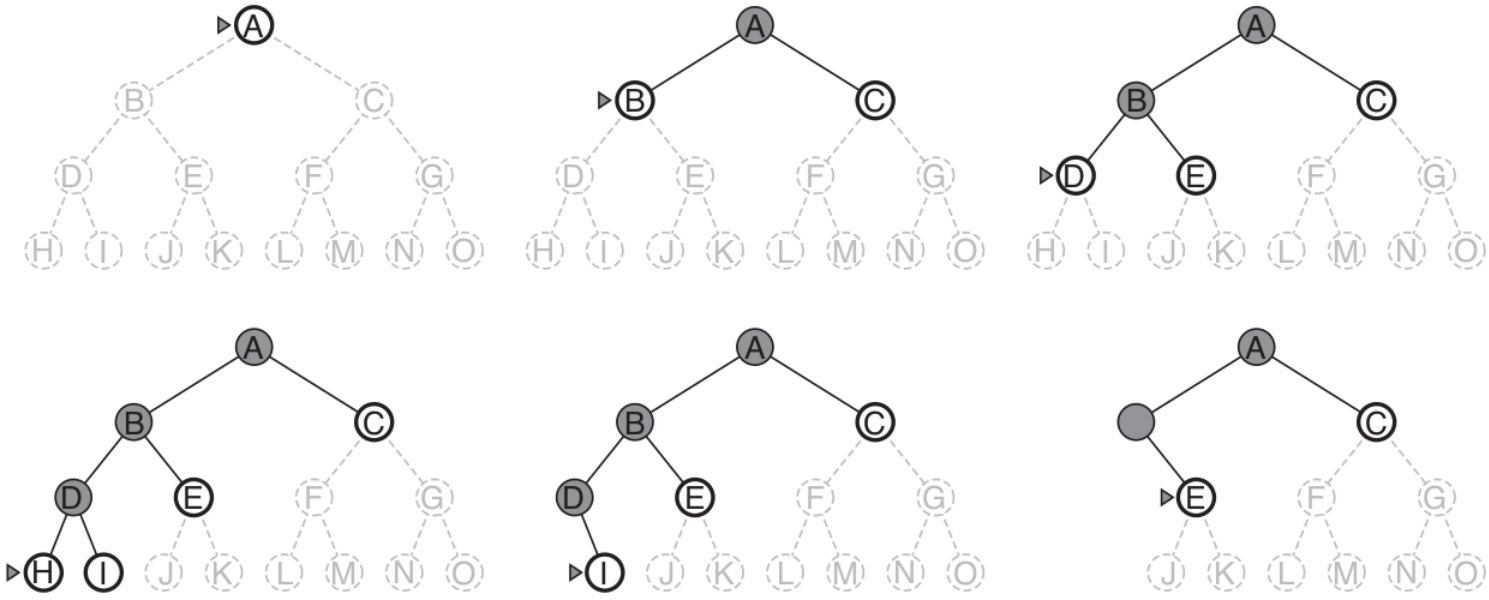
- "Inondation" à partir de l'état initial.
- Trouve l'état final le plus proche.

Implémentation :

On ajoute les noeuds à explorer à la suite de la file de recherche.



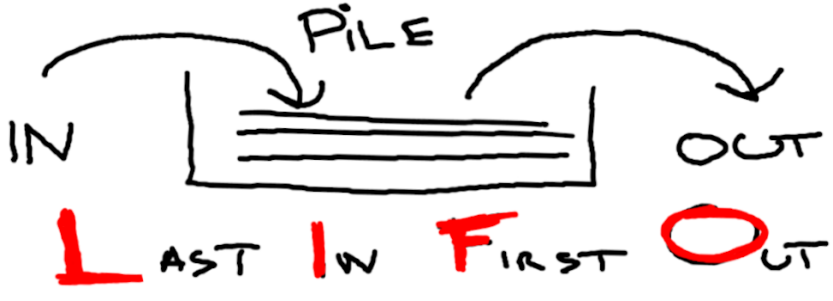
Parcours en profondeur



- "Descend" rapidement jusqu'à un état final.

Implémentation :

On ajoute les noeuds à explorer au début de la pile de recherche.

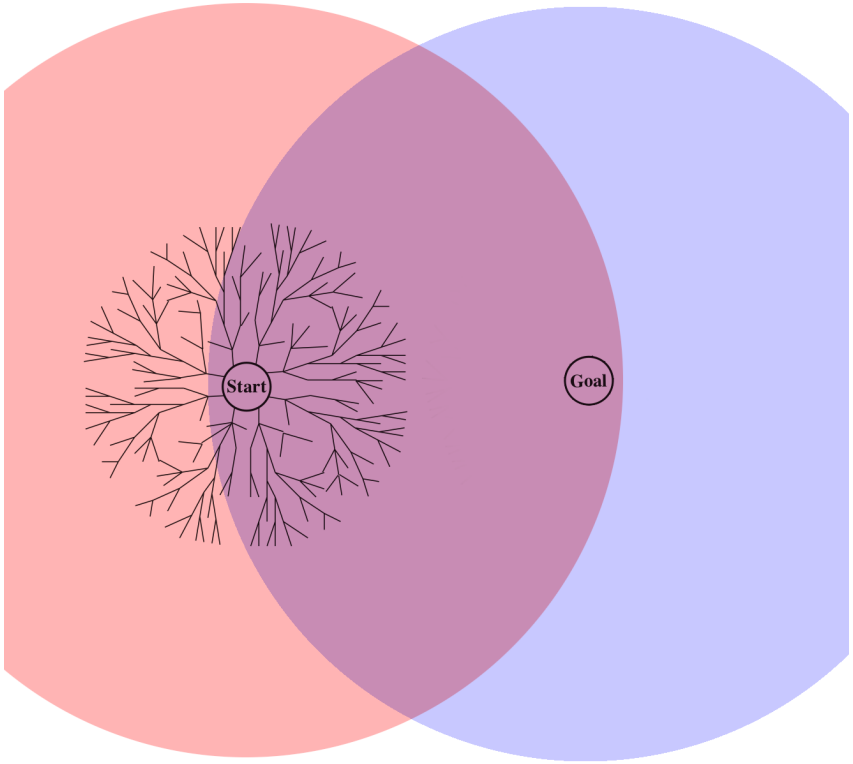


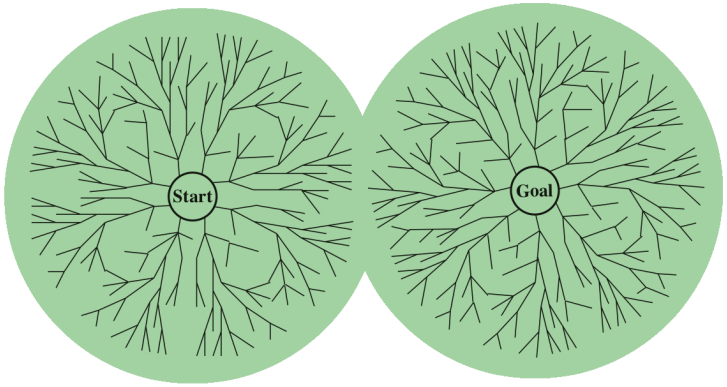
Amélioration: recherche bi-directionnelle

Deux Parcours en largeur en parallèle, un depuis l'état initial, l'autre depuis le but.

Si les frontières des deux parcours ont un noeud N en commun, alors il existe un chemin :
État initial $\rightarrow$ N $\rightarrow$ But.

Ex : Distance entre départ et arrivée $= 10$.
unidirectionel : $10\pi^2 = 985$
bidirectionel : $(5\pi^2) * 2 = 492$



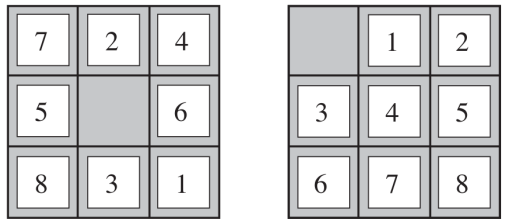


Heuristique ?

- Best-first
- A*

Le parcours du graphe de recherche reste le même, mais l'heuristique est dépendante de la nature du problème.

Exemple



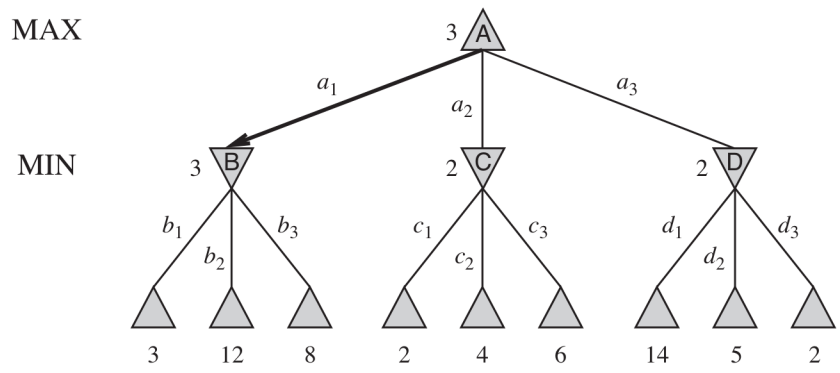
1. Nombre d'éléments hors position : 8
2. Somme des distances aux position buts de chaque éléments (distance de Manhattan): 3+1+2+2+2+3+3+2 = 18

Recherche "adversairielle"

Plusieurs joueurs

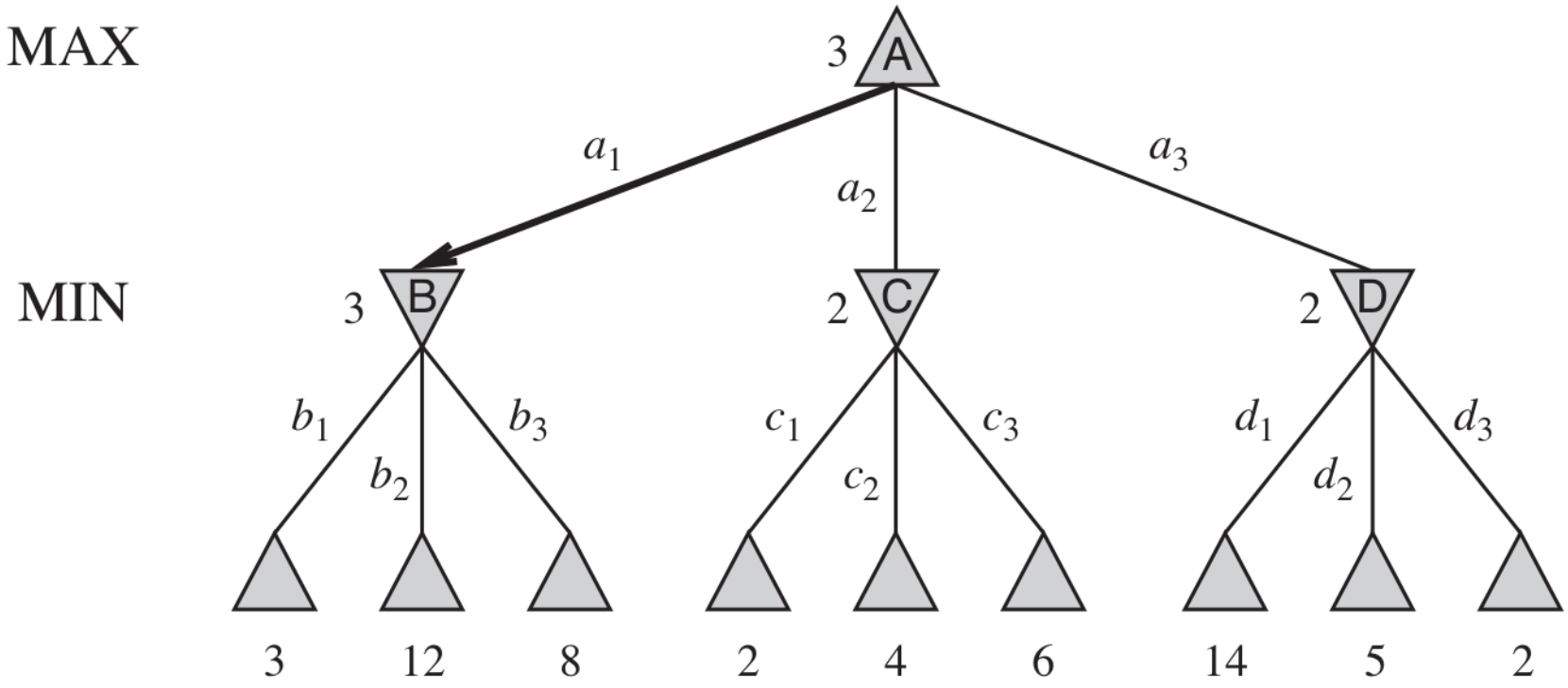
- Coups successifs.
- Contrairement à une compétition (sur un score), on cherche à contrer les coups de l'adversaire.

- Le joueur "Max" : choisit l'action qui maximise le score final.
- Le joueur "Min" : choisit l'action qui minimise le score final.



$\rightarrow$ On joue du "point de vue" Max ...

Min-max

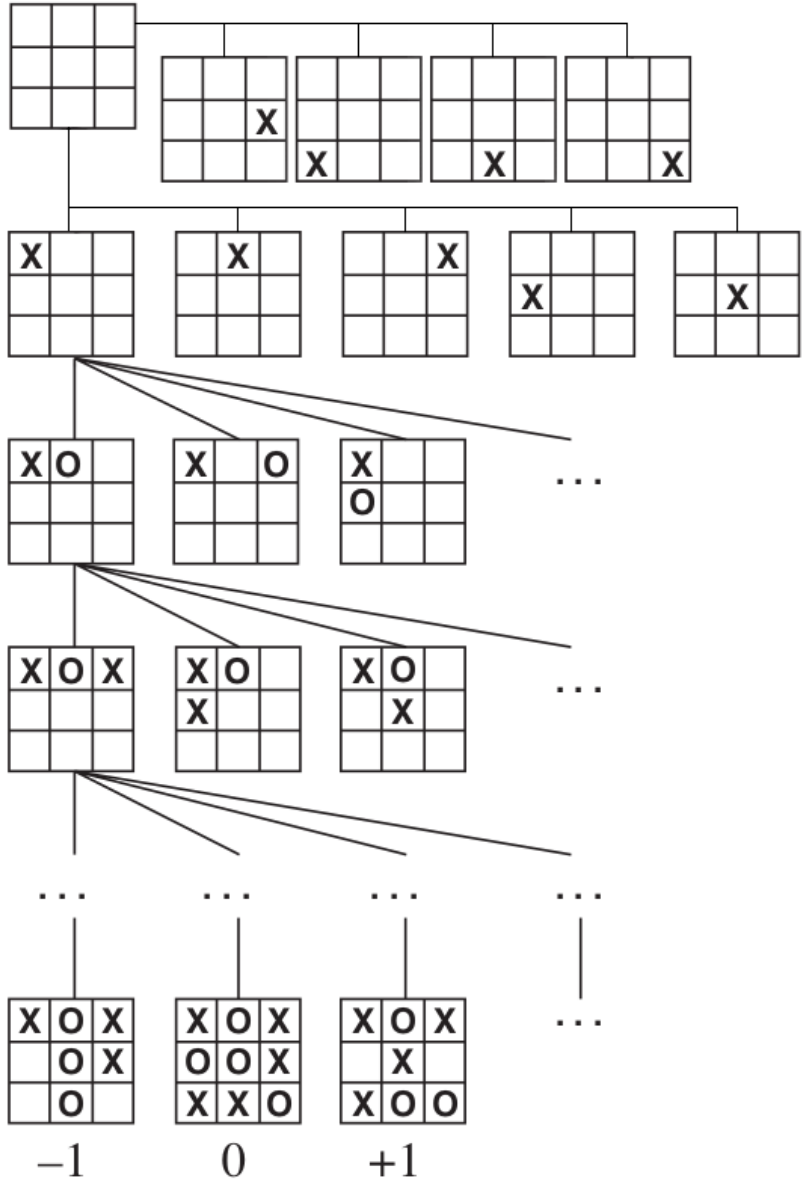


Min-max

Recherche non-informée

- On ne connaît le score que quand on arrive à l'état final.

$\rightarrow$ On assigne les scores en remontant à partir des feuilles du graphe.



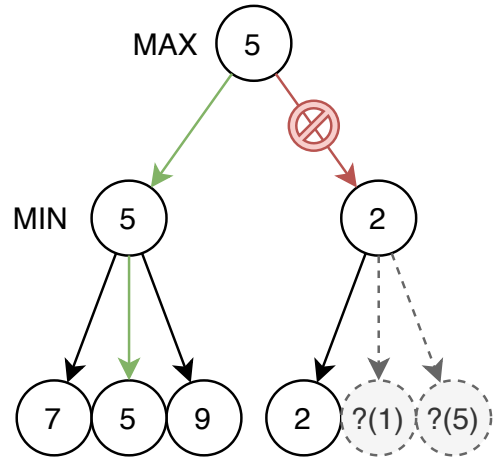
Élagage alpha-beta

Inconvénient de Min-Max

On est obligé de parcourir le graphe en entier.

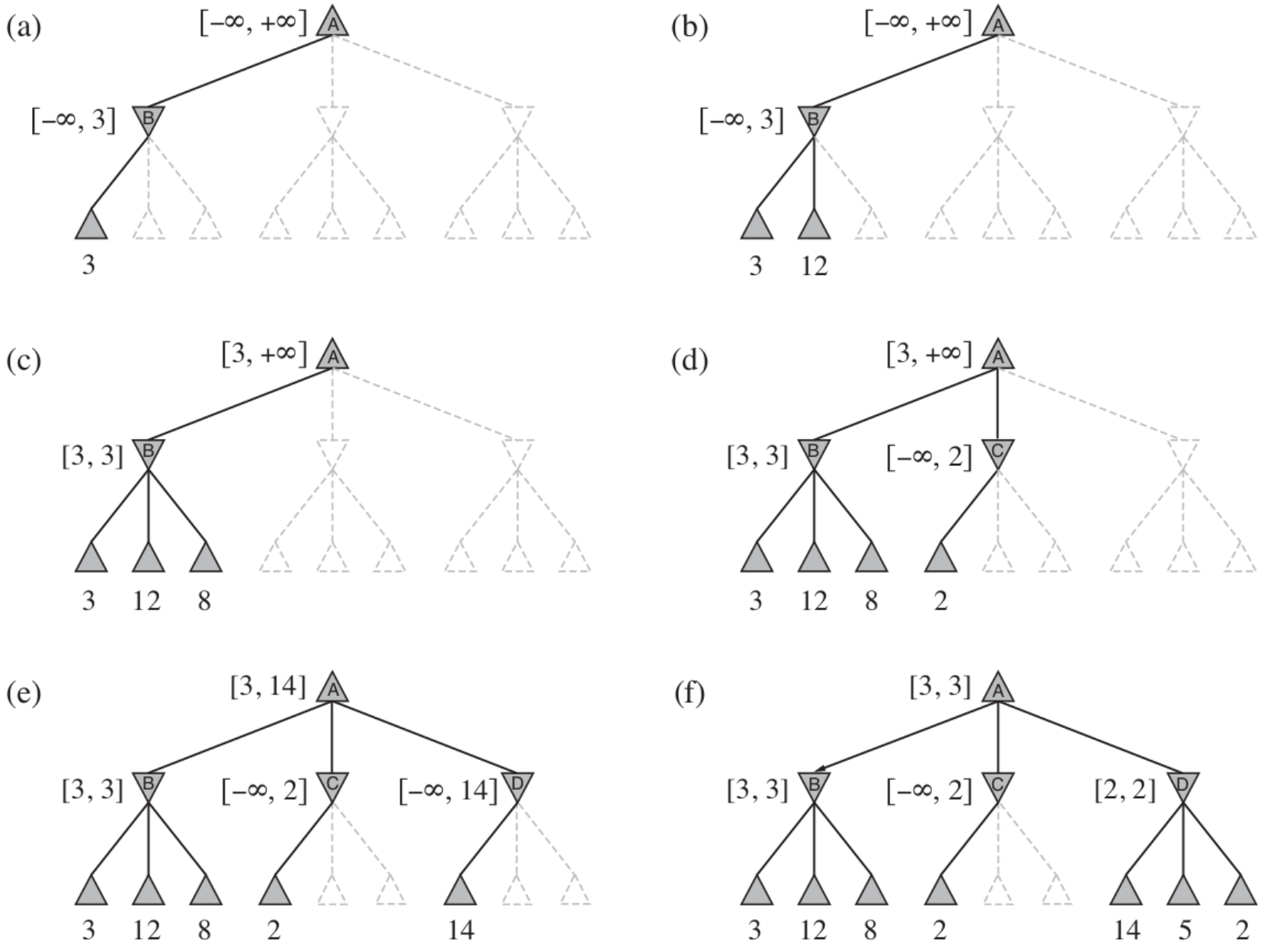
Élagage

- Si on sait déjà qu'une branche est moins avantageuse, on peut l'éliminer (élaguer).



→ On assigne les scores en remontant à partir des feuilles du graphe.

Élagage alpha-beta



Limites

Impossible de visiter tous les noeuds terminaux

- Morpion : 362 880 noeuds
- Echecs : 10^{40}

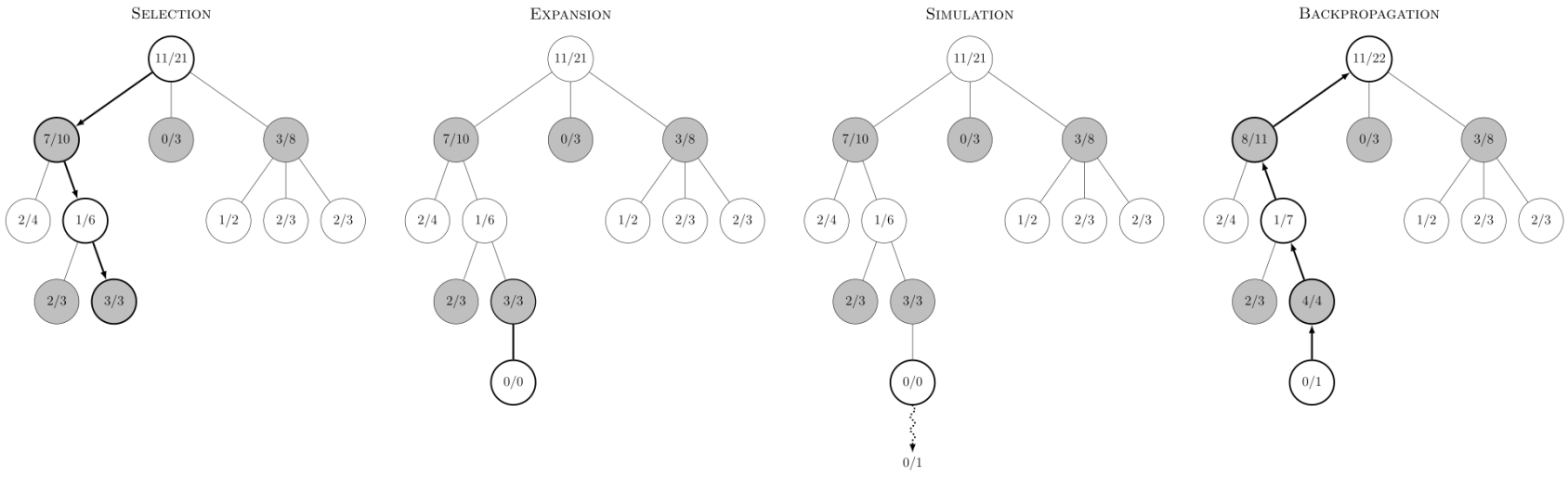
Donner un score aux noeuds intermédiaires

MinMax avec limite sur la profondeur du graphe de recherche.
Arrivé à une profondeur donnée, on attribut un score à l'état qu'on observer à l'aide d'une **fonction d'évaluation**.
Exemple: Points pour les pièces d'échecs encore en jeu.

$\rightarrow$ *idée d'objectif intermédiaire.*

Recherche arborescente Monte-Carlo (MCTS)

Pour attribuer un score au noeud intermédiaire, on joue plusieurs parties aléatoires découlant de ce noeud.



Ici la frontière n'est pas fixée à une profondeur mais évolue suivant les approches les plus prometteuses.

TP: le morpion

Java: types génériques

Il est possible d'indiquer à une classe qu'elle peut manipuler des objets dont le type n'est pas connu à l'avance :

```
class MaClasse<T,U,V>{
  T data;
  U key; ...
```

Ce(s) type(s) sont fournis au moment de l'instanciation :

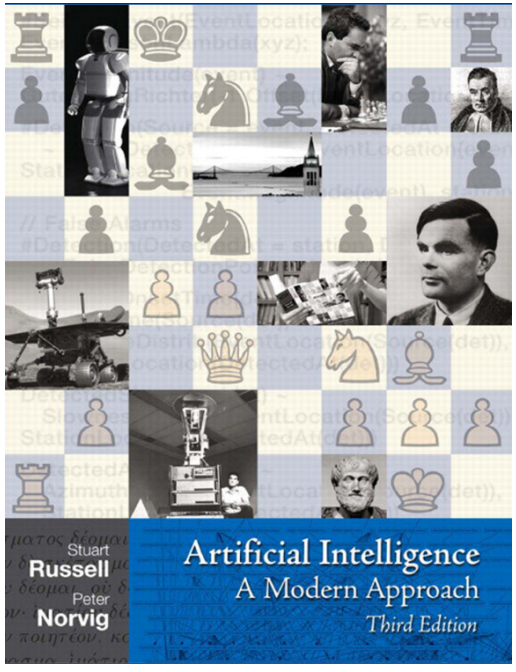
```
MaClasse<String,Integer,UneClasse> unObj =
          new MaClasse<String,Integer,UneClasse>();

MaClasse<Double,Double,Integer> unAutreObj =
          new MaClasse<Double,Double,Integer>();
```

Cela nous permettra de définir une classe *Noeud* générique pour représenter un arbre de manière complètement indépendante de son contenu (Une classe *EtatMorpion* par exemple ...)

TP: le morpion

1. Vous compléterez la classe *TreeNode* qui représente un noeud dans un arbre. Vous implémenterez les parcours en largeur et profondeur.
2. En utilisant la classe *TreeNode*, vous compléterez la classe *MinMaxAI* pour implémenter l'IA du jeu de morpion. Vous aurez besoin de créer votre propre classe pour représenter les états du jeu.
3. La classe *MinMaxAI* est prévue pour fonctionner avec une interface graphique fournie, vous testerez votre IA.

Bibliographie



Artificial Intelligence: A Modern Approach, Stuart Russell and Peter Norvig.