Introduction à l'intelligence artificielle

Historique

History of artificial intelligence

Bible : argile + souffle [http://saintebible.com/genesis/2-7.htm]

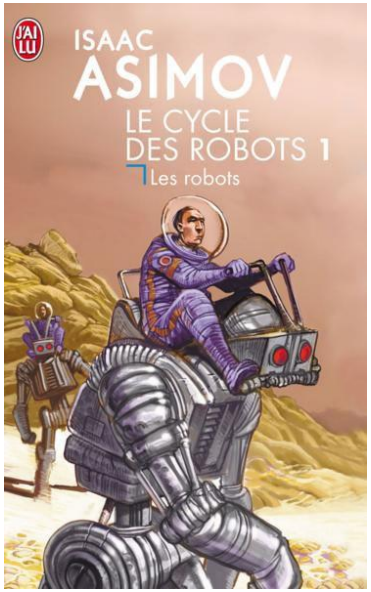
Mythologie grecque : Pygmalion et Galatée [https://fr.wikipedia.org/wiki/Pygmalion_et_Galat%C3%A9e]

Automates de la civilisation arabe

1818 : *Frankenstein, or the modern Prometheus* de Mary Shelley

1920 : Karel Capek utilise le mot "robot" dans R. U. R. (Rossum Universal Robots)

1950 : *I, robot* d'Asimov



Historique (suite)

1950 : article de Turing *Computing machinery and intelligence*

The imitation game

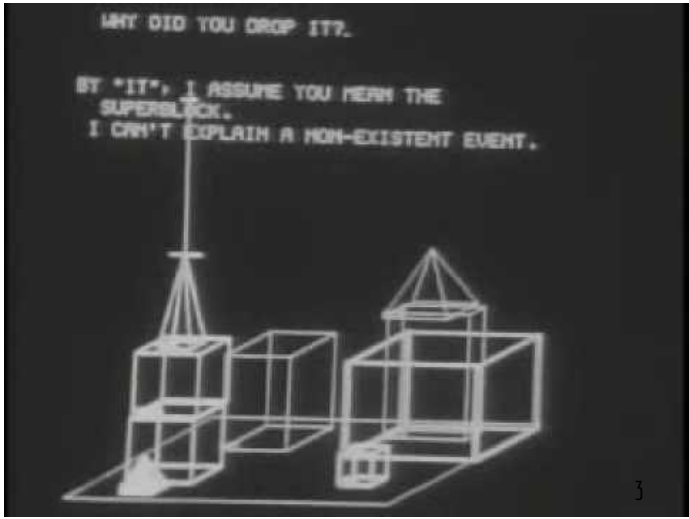
I propose to consider the question, "Can machines think ?"

1956 : conférence de Dartmouth

- Le terme IA est inventé
- Problèmes initiaux :
  - théorie des jeux
  - traduction automatique
  - McCarthy, Minsky, Papert, Simon, Newell

1960s : Terry Winograd SHRDLU (monde de blocs)

1970s : Systèmes experts : Mycin, Dendral



Historique (suite)

1988 : Puissance 4 résolu par Victor Allis et James Allen (indépendamment)
1997 : Deep Blue bat Kasparov
2002 : invention d’Arimaa, conçu pour être compliqué pour un ordinateur

2007 : Schaeffer (Games Alberta) résout le jeu de dames (Chinook)

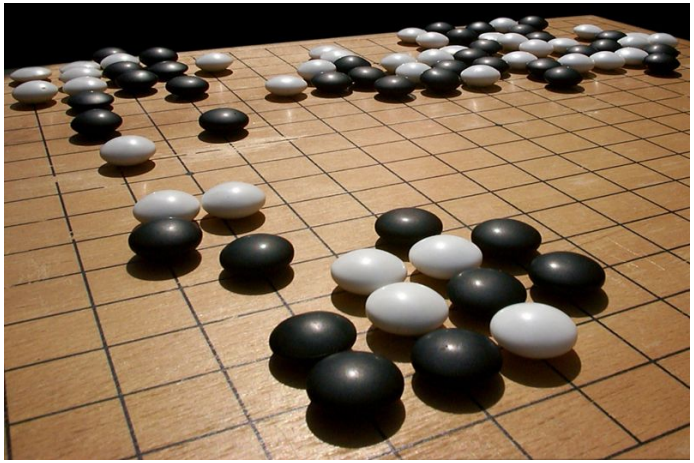
http://cristal.univ-lille.fr/~jdelahay/pls/2008/165.pdf

One jump ahead

Historique (fin)

2010s : Watson bat Ken Jennings (TED talk) à Jeopardy (Connaissances générales + jeux de mots)
2015 : Le champion d'Arimaa est un ordinateur

Actualités



2015 : Alphago (google Deepmind) bat Fan Hui 5-0
2016 : Alphago - Lee Sedol 4-1
2016 : Google annonce la traduction automatique quasi parfaite GNMT
2017 : Alphago bat Ke Jie 3-0
2017 : Alphago Zero apprend à jouer sans expertise humaine
2017 : Humans are still better than AI at Starcraft - for now
2017 : Apple : iphone X, reconnaissance de visage par réseau de neurones
2018 : Uber : premier accident mortel impliquant une voiture autonome
2018 : FDA autorise le diagnostic autonome de la rétinopathie
2019 : Une IA de Starcraft bat les meilleurs joueurs mondiaux
2020 : Alphafold (une IA de Deepmind) replie les protéines
2020 : Une IA du MIT reconnaît le COVID

Modélisation d’un problème en espace d’états

Un espace d'états est un graphe dont les noeuds sont des configurations du problème. Un problème est alors équivalent à la recherche d'un chemin dans ce graphe.

Méthodes classiques (*brute force*) :

- profondeur d'abord (*depth-first*)
  avec détection des cycles, avec profondeur bornée

- largeur d'abord (*breadth-first*)

- profondeur itérative (*iterative deepening*)

Modélisation d’un problème en un espace d’états

Robotique (Bloc) : trouver une succession d'étapes (un plan) pour qu'un robot réarrange une pile de blocs. Le robot ne peut bouger qu'une pile de blocs à la fois. Un bloc ne peut être déplacé que s'il est au sommet d'une pile. Il peut alors être déplacé sur la table (nouvelle pile), ou sur une autre pile. L'objectif est de trouver une séquence de déplacements entre la situation initiale et une situation finale.

Jeu (Taquin) : résoudre un puzzle de 8 cases (jeu à un joueur). Un puzzle de 8 cases est constitué de 8 tuiles glissantes numérotées de 1 à 8, placées dans un tableau 3*3. Une des cases est donc vide, et chaque tuile adjacente à cette case vide peut y glisser, laissant une nouvelle case vide. La situation finale correspond à un arrangement particulier des tuiles dans le tableau.

Modélisation informatique

Bloc

structure de données : liste de piles

voisinage : déplacement d'un sommet de pile

Taquin

structure de données : liste ordonnée de coordonnées (vide,1,...,n)

voisinage: permutation de la case vide avec une case voisine
(distance de Manhattan = 1)

Exercices

Soit un espace de solution sous forme d'un arbre d'arité *b* (ou facteur de branchement *branching factor*) dont la solution se trouve à une profondeur *d*.
- Dans le cas particulier où *b*=2 et *d*=3, combien de noeuds sont générés dans le pire des cas par le parcours en largeur et par le parcours en profondeur ? Soit *N(b,d)* le nombre de noeuds générés par le parcours en profondeur dans le cas général. Trouver une expression récursive exprimant *N(b,d)* en fonction de *N(b,d-1)*.
- Comparer le nombre de noeuds visités par les algorithmes en profondeur d'abord et en largeur d'abord dans le meilleur et le pire des cas.

Parcours en largeur, algorithme de Dijkstra, A*

Chacun de ces algorithmes est une généralisation du précédent.

L'algorithme de Dijkstra permet de prendre en compte des arêtes de poids différents (alors que pour le parcours en largeur, elles ont toutes le même).

A* permet de prendre en compte une évaluation de la distance qui reste à parcourir alors que l'algorithme de Dijkstra ne prend en compte que la distance jusqu'au sommet considéré.

A* est un algorithme de *branch and bound*.

Algorithme A* (*best-first*)

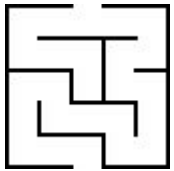
```
// fonctions spécifiques à chaque problème
sol(E) // retourne true si E est un état solution au problème
succ(E)// retourne l'ensemble des successeurs de E
k(E,F) // retourne le coût de E à F
h(E)   // retourne la valeur de l'heuristique en E
//fonctions génériques
astar()
        FileAVoir := [E0]
        ListeVus := []
        g(E0) := 0
        E := E0
        tant que !(fileVide(FileAVoir)) et !sol(E) faire
                supprimer(FileAVoir)
                inserer(ListeVus,E)
                pour tout F := succ(E) faire
                        si !(appartient(FileAVoir,F) et appartient(ListeVus,F)) ou g(F)>g(E)+k(E,F) alors
                                g(F) := g(E)+k(E,F); f(F) := g(E)+h(F) pere(F) := E
                                insererTrier(FileAVoir,F)
                si !(fileVide(FileAVoir)) alors E := premier(FileAVoir)
        si sol(E) retourner chemin(E0,E) sinon retourner []
chemin(E,F)
        ListeRes := [F]
        tant que pere(F)!=E    faire
                ajouterTete(ListeRes,E)
        retourner ajouterTete(ListeRes,E)
```

A*

Propriétés :

Une heuristique est admissible si elle sous-évalue toujours la distance restante (elle est toujours trop optimiste) et qu'elle est monotone sur un chemin. Avec une telle heuristique, on trouve toujours le chemin optimal (le plus court).

Exercice : Utiliser l'algorithme A^* avec
la distance de Manhattan pour sortir par le bas
en rentrant dans le labyrinthe depuis le haut.
labyrinthe généré par mazegenerator



Exercices (suite)

Appliquer l'algorithme A* sur le taquin suivant

Configuration initiale

	1	3
4	2	5
7	8	6

Configuration finale

1	2	3
4	5	6
7	8	

1. En utilisant la distance de hamming comme heuristique (nombre de cases mal placées)
2. En utilisant la distance de manhattan comme heuristique (somme des distances des cases entre leur positions et leurs positions finales)

Computerphile

Dijkstra’s algorithm

A* (A star) search algorithm

Maze solving

Jeux à deux joueurs

Extension de la recherche dans un espace d'états avec des graphes alternés.

Deux situations successives sont deux types différents.

Selon le type de situation, la stratégie de recherche diffère.

Deux approches dans ce cours :

* MiniMax (algorithme déterministe basé sur une heuristique 1970-2010)

* Deep reinforcement learning (algorithmes d’apprentissage 2010-...)

MiniMax

Alternative aux algorithmes exhaustifs lorsque les chemins sont potentiellement infinis (ou trop longs)

* Se fixer une profondeur maximum (nombre de coups à évaluer)
* Construire un arbre de jeu alterné jusqu'à cette profondeur
* Évaluer la qualité des feuilles (distance à la victoire du joueur qui commence)
* Remonter la valeur vers les noeuds pères
    * comme un Max des fils si joueur A (celui qui commence)
    * comme un Min des fils sinon
* Utiliser l'arbre de jeu pour établir la stratégie du joueur A

Algorithme MiniMax

```
// fonctions spécifiques à chaque problème


sol(E)  // retourne true si E est un état solution au problème
succ(E) // retourne l'ensemble des successeurs de E
h(E)    // retourne la valeur de l'heuristique en E
```

Algorithme MiniMax

```
// fonctions génériques

minimax(E)
    si sol(E) alors
        Res := h(E)
    sinon
        Succ := succ(E)
        N := taille(Succ)
        Res := minimax(Succ(1))
        pour I de 2 a N faire
            si noeud_max(E) alors
                Res := max(Res,minimax(Succ(I))
            sinon
                Res := min(Res,minimax(Succ(I))
        retourner Res
```

Algorithme AlphaBeta

```
// fonctions spécifiques à chaque problème
sol(E)  // retourne true si E est un état solution au problème
succ(E) // retourne l'ensemble des successeurs de E
h(E)    // retourne la valeur de l'heuristique en E


// fonctions génériques
noeud_max(E) // retourne true si E est un noeud max
```

Algorithme AlphaBeta

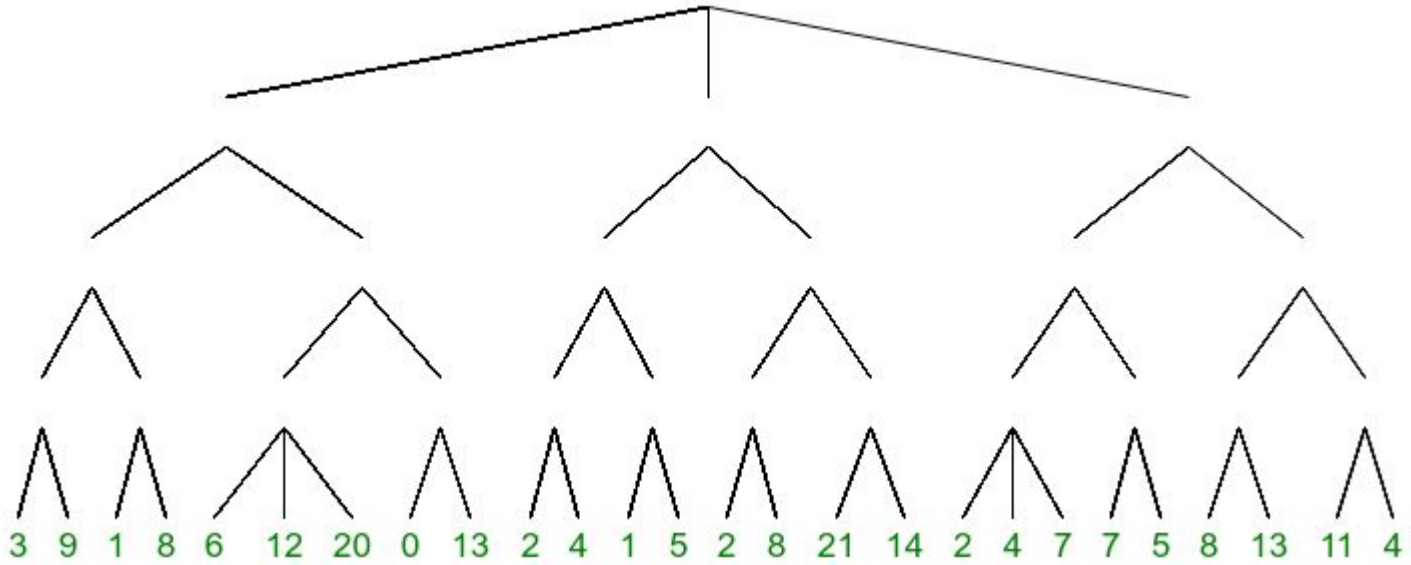
```
alpha_beta(E,Alpha,Beta)
    si sol(E) alors
        score(E) := h(E)
```

Algorithme AlphaBeta

```
sinon
    I := 1; Fin := false; Succ := succ(E); N := taille(Succ)
    tant que I <= N et !Fin faire
        si noeud_max(E) alors
            A := alpha_beta(Succ(i),Alpha,Beta)
            si A > Alpha alors Alpha := A
            si Alpha >= Beta alors Res := Beta; Fin := true
            sinon si I = N alors Res := Alpha
                      sinon B := alpha_beta(Succ(i),Alpha,Beta)
            si B < Beta alors Beta := B
            si Alpha >= Beta alors Res := Alpha; Fin := true
                        sinon si I = N alors Res := Beta
        I := I+1
retourner Res
```

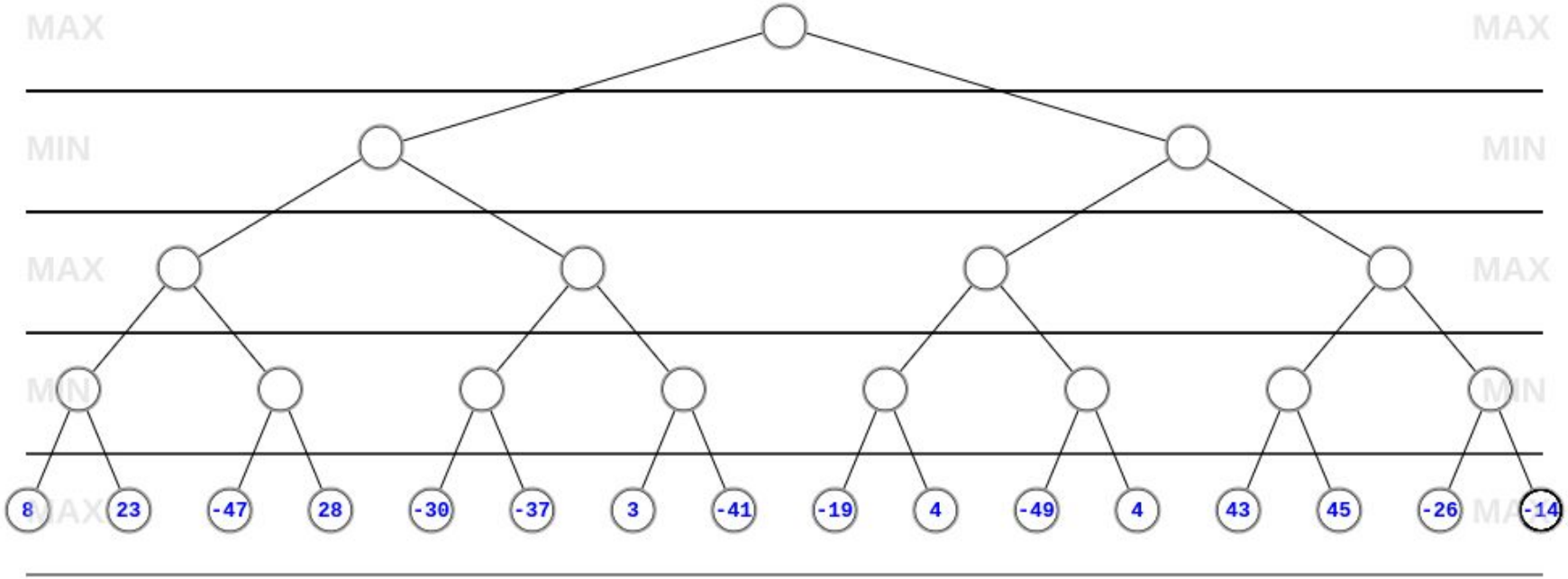
Exercice 1

Appliquer les algorithmes Minimax et Alpha-Beta sur l’arbre de jeux :



Exercice 2

Appliquer l’algorithme Alpha-Beta sur l’arbre de jeux :



Application

application en ligne pour vous entraîner :

- http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html