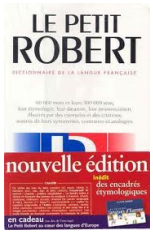
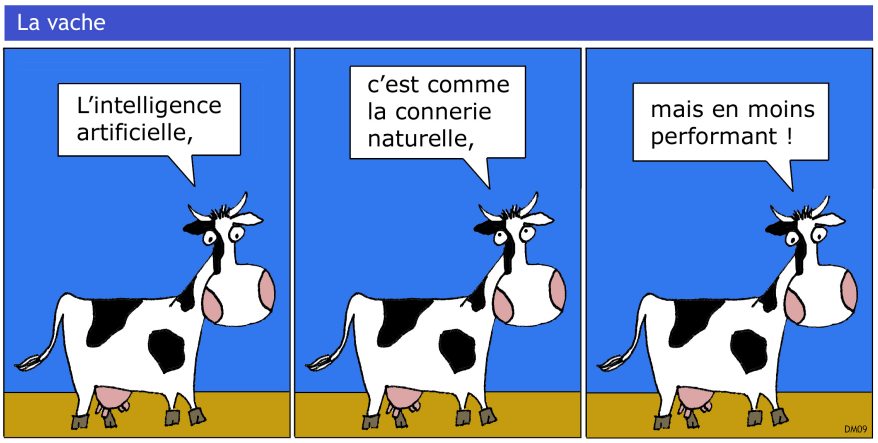
Chapitre 10

Intelligence artificielle et jeux



Intelligence artificielle : Partie de l'informatique qui a pour but la simulation de facultés cognitives afin de suppléer l'être humain pour assurer des fonctions dont on convient, dans un contexte donné, qu'elles requièrent de l'intelligence. Langages : Lisp, Prolog. => Reconnaissance de formes et de la parole, simulation, jeu, conduite de robots, apprentissage.



L'intelligence artificielle (IA) comprend plusieurs domaines :

- le dialogue automatique : se faire comprendre d'un ordinateur en lui parlant ;
- la traduction automatique, si possible en temps réel ou très légèrement différé ;
- le raisonnement automatique (systèmes experts) ;
- l'apprentissage automatique ;
- la reconnaissance de formes, des visages et la vision en général ;
- l'intégration automatique d'informations provenant de sources hétérogènes ;
- l'aide aux diagnostics ;
- l'aide à la décision ;
- la résolution de problèmes complexes, tels que les problèmes d'allocation de ressources ;
- l'assistance par des machines dans les tâches dangereuses, ou demandant une grande précision.

L'intelligence artificielle est utilisée (ou intervient) dans une variété de domaines tels que :

- la banque, avec des systèmes experts d'évaluation de risque lié à l'octroi d'un crédit ;
- le militaire, avec les systèmes autonomes tels que les drones ;
- les jeux, domaine qui va plus particulièrement nous intéresser ici ;
- la médecine, avec les systèmes experts d'aide au diagnostic ;
- la logistique, au travers d'approches heuristiques de type résolution de problème de satisfaction de contraintes.

10.1. Différents types de jeux

Un jeu est un terrain d'expérimentation idéal pour l'intelligence artificielle (IA). Les règles simples et peu nombreuses, les situations bien définies modélisent un mode simplifié mais quand même intéressant. Les jeu bien connu « Les Sim's » en est un bon exemple. Pour l'ordinateur, les choix sont réduits, les configurations faciles à analyser, les conséquences des choix sont calculables. Pensons à des jeux comme les échecs, les dames, mais aussi des jeux où le hasard intervient comme le backgammon, le poker, où l'ordinateur aura à calculer des probabilités.

Le terme d' « intelligence artificielle » est très pompeux. Certains parlent en ce domaine plutôt de force brute. On verra en effet qu'il s'agira uniquement de calculs et d'exploration d'arbre de jeu. Il est vrai que l'ordinateur donne *l'impression* d'une certaine intelligence[1] quand il joue contre un humain. L'auteur se souvient encore du choc qu'il a ressenti quand, dans les années 80, il a joué (et perdu) pour la première fois aux échecs contre une machine. Comment cet objet sans vie faisait-il ? C'est ce que nous allons étudier dans ce chapitre.

10.1.1. Jeux vidéo

C'est dans les jeux vidéo que l'intelligence artificielle s'est le plus popularisée, et c'est aussi un des domaines où elle se développe rapidement. Celle-ci bénéficie en effet des progrès de l'informatique, avec par exemple les cartes graphiques dédiées qui déchargent le processeur principal des tâches graphiques. Le processeur principal peut désormais être utilisé pour développer des systèmes d'IA plus perfectionnés.

MMOG : massively multiplayer online game

Par exemple, l'intelligence artificielle peut être utilisée pour piloter des bots (c'est-à-dire les personnages artificiels) évoluant dans les MMOGs ou les mondes virtuels, mais on peut aussi citer son utilisation dans des jeux de simulation, ou pour animer des personnages artificiels.

Dans le domaine du jeu vidéo, l'IA caractérise toute prise de décision d'un personnage (ou d'un groupe) géré par le jeu, et contraint par l'intérêt ludique. Jusqu'à la fin des années 1990, l'IA dans les jeux vidéo (plus particulièrement dans les jeux en temps réel) a été délaissée par rapport au rendu visuel et sonore. L'évolution vers des univers toujours plus réalistes, leur peuplement par des personnages aux comportements crédibles devient une problématique importante.

Avec les jeux en réseau, le besoin d'IA a tout d'abord été négligé, mais, particulièrement avec l'apparition des jeux massivement multijoueur, et la présence d'un nombre très important de joueurs humains se confrontant à des personnages non joueur (PNJ), ces derniers doivent s'adapter à des situations qui ne peuvent être prévues. Actuellement ces types de jeux intéressent particulièrement des chercheurs en IA.

10.1.2. Jeux de réflexion

Un ordinateur peut simuler la prudence, la malice, voire l'intelligence. Tel est le cas lorsqu'un programme permet à un ordinateur de rivaliser avec un être humain dans un jeu de stratégie qui, à l'évidence, mobilisera la réflexion de ce dernier.

1979 : Le champion du monde battu au Backgammon

Le premier logiciel de valeur, BKG 9.8, a été conçu par Hans **Berliner** dans les années 1970 sur un DEC PDP-10 pour expérimenter l'évaluation des positions des jeux de tablier. Les versions précédentes de BKG n'étaient pas en mesure de battre de manière répétitive des joueurs débutants, mais **Berliner** remarqua que ses erreurs critiques se situaient toujours dans des phases de transition de la partie. Il appliqua les principes de la logique floue pour améliorer son jeu au cours de ces phases, et en juillet 1979, BKG 9.8 devint suffisamment fort pour jouer contre le champion mondial Luigi **Villa**. Il gagna la rencontre 7-1, devenant le premier logiciel à battre un champion du monde dans un jeu de tablier. Berliner constata cependant que la victoire était due en grande partie à la chance du fait d'un plus grand nombre de résultats favorables obtenus par le logiciel.

Dans les années 1980, les informaticiens obtinrent plus de succès avec une approche basée sur l'utilisation de réseaux de neurones artificiels. *TD-Gammon*, développé par Gerald **Tesauro** chez IBM, fut le premier de ces logiciels à atteindre un niveau proche de celui d'un joueur expérimenté.

1 Encore faudrait-il définir ce qu'est l'intelligence. Donnez votre définition et discutez-en en groupe.

Son réseau de neurones était entraîné par auto-apprentissage. Selon Bill **Robertie** et Kit **Woolsey**, deux grands joueurs de backgammon, le jeu de TD-Gammon était alors à la hauteur, et même au-dessus, de celui des meilleurs joueurs du monde. Woolsey déclara ainsi « Il n'y a aucun doute dans mon esprit que son analyse des positions est de beaucoup supérieure à la mienne. »

Les recherches basées sur les réseaux artificiels de neurones ont abouti à la génération de trois logiciels commerciaux modernes, *Jellyfish*, *Snowie* et e*Xtreme Gammon*, ainsi qu'au partagiciel *BGBlitz* et au logiciel libre *GNU Backgammon*. La force de ces logiciels repose sur des mois d'entraînement de leurs réseaux de neurones sans lesquels ils ne pourraient pas dépasser le niveau d'un joueur novice. La phase de sortie des dames est généralement traitée par les logiciels à partir d'une base de données – obtenue par ordinateur – contenant toutes les positions possibles des dames au moment de la sortie.

1997 : Deep Blue bat Garry Kasparov aux Échecs

Voir [2] pour un historique plus complet.

Depuis les années 1990, les ordinateurs sont capables de battre le champion de monde d'échecs. Mais l'histoire des programmes d'échecs est longue. Il aura fallu un demi-siècle pour passer d'un programme connaissant les règles du jeu à un ordinateur capable de battre le champion du monde.

- 1948, le livre *Cybernétique* de Norbert **Wiener** décrit comment un programme d'échecs peut être développé en utilisant une profondeur minimale de recherche avec une fonction d'évaluation.
- 1951, Alan **Turing** développe sur le papier le premier programme capable de jouer une partie d'échecs complète.
- 1958, les premiers programmes qui sont capables de jouer une partie complète sont créés, le premier par Alex **Bernstein** et l'autre par des programmeurs russes sur mainframe BESM.
- 1966-1967, le premier match entre programmes d'échecs voit le jour. Le programme *Kaissa* de l'Institut de physique théorique et expérimentale de Moscou triomphe de *Kotok-McCarthy* de l'université Stanford. Les coups étaient échangés par télégraphe et le match a duré 9 mois.
- 1967, *Max Hack 6* de Richard **Greenblatt** devient le premier programme à gagner contre une personne en tournoi.
- 1974, *Kaissa* devient le premier champion du monde des ordinateurs.
- 1977, le premier jeu d'échecs électronique, *Chess Challenger*, est commercialisé. Et devient la même année le premier ordinateur à remporter un tournoi d'échecs majeur.
- 1985, *HiTech* réalise une performance Elo de 2530, c'est le premier programme à atteindre le classement de 2400 (niveau d'un maître international).

-
- 1994, *Fritz 3* gagne une partie de blitz contre le champion du monde de l'époque, Garry **Kasparov** et ils terminent *ex æquo*. Kasparov prend sa revanche dans le départage : 4-1. Cette même année à Londres, *Chess Genius* bat Garry Kasparov en partie semi-rapide (1.5-0.5) avec un Pentium 100 MHz.
- 1997, *Deep Blue* bat Garry **Kasparov**. (2 victoires, 3 nulles et 1 défaite).
- 2005, *Hydra* gagne face à Michael **Adams** par 5 victoires et une nulle contre 0 victoire.
- 2006, *Deep Fritz* gagne face à Vladimir **Kramnik** par 2 victoires et 4 nulles contre 0 victoire.
- 2017, *AlphaZero*, une variante d'*AlphaGo* (voir paragraphe consacré au go) qui pratique l'apprentissage par renforcement, n'a mis que quatre heures en partant des règles de base pour vaincre *Stockfish*, le meilleur programme de jeux d'échecs du moment.

1997 : au tour d'Othello

Dans ce jeu, le but est de retourner les pions adverses en les encerclant. En 1997, quelques mois après la victoire de *DeepBlue* sur **Kasparov**, le logiciel *Logistello* bat le champion du monde d'Othello, Takeshi **Murakami**. Depuis, comme pour les échecs, les programmes ont évolué et battent facilement les humains, mais l'intelligence artificielle n'a toujours pas réussi à « résoudre » parfaitement ce jeu. Nous reviendrons sur ce jeu au § 10.5.

2002 : l'awalé entièrement résolu

L'awalé est un jeu créé en Afrique où les joueurs doivent déplacer des jetons dans 12 récipients, l'un après l'autre, dans le but de capturer les pions adverses. En 2002, des chercheurs néerlandais ont réussi à résoudre le jeu en calculant les quelques 889 milliards de positions possibles, ce qui a pris plus de 51 heures de calcul à l'algorithme.

2008 : Chinook imbattable aux Dames

Des scientifiques canadiens ont mis au point un programme d'ordinateur impossible à battre au jeu de dames. Une formidable avancée dans le domaine de l'intelligence artificielle. Jonathan **Schaeffer**, détenteur de la chaire de sciences informatiques à l'université d'Alberta (Canada), aidé par d'autres informaticiens de cet établissement, se sont acharnés durant 18 ans à programmer les quelque 500 milliards de milliards de combinaisons possibles au jeu de dames, un grand classique du genre répandu dans le monde entier.

Et le résultat est là : *Chinook*, puisque c'est le nom du logiciel, s'avère impossible à battre. Au pire, il conduira une partie jusqu'à une impasse débouchant sur la nullité, et confronté à un autre ordinateur utilisant le même programme, ne produira que des parties nulles.

Pour mettre au point son programme, **Schaeffer** a mobilisé environ 50 ordinateurs quotidiennement depuis 1989, parfois jusqu'à 200 dans les moments critiques, et a fait appel à plusieurs joueurs professionnels.

À l'origine, *Chinook* avait été élaboré pour participer au Championnat du Monde de Dames. Perdant en finale en 1992, il l'a remporté deux ans plus tard en devenant ainsi le premier logiciel à obtenir un titre mondial dans un jeu de compétition. Mais estimant alors que les ordinateurs de nouvelle génération devraient permettre de créer un programme infaillible, **Schaeffer** se remettait au travail en 2001 pour arriver au résultat actuel.

Il est possible de... perdre contre Chinook sur le site http://webdocs.cs.ualberta.ca/~chinook

2015 : Texas Hold'Em limit

Le Poker est un peu particulier. En 2015, des chercheurs ont créé un programme qui est presque sûr de gagner chaque partie de poker. Mais pas n'importe lequel : le « Texas Hold'Em limit » à deux joueurs. A l'instar du jeu de go, le programme apprend de chaque partie jouée et affine sa stratégie. Pour le no-limit à plusieurs joueurs (la version du poker la plus jouée), on est encore très loin d'une intelligence artificielle capable de vaincre les meilleurs joueurs du monde. Les possibilités sont tellement énormes (comment analyser une relance de 10, 100 ou 10.000 dollars ?) qu'il est difficile pour un ordinateur de maîtriser ce jeu. Pour l'instant.

2016 : AlphaGo bat l'un des meilleurs joueurs du monde

Son apparente simplicité semble faire du go un candidat idéal à l'exploration informatique. Mais des difficultés considérables ne tardent pas à surgir. La taille du goban détermine une combinatoire qui dépasse de très loin les possibilités de calcul des ordinateurs (la taille très approximative de l'arbre des possibilités du jeu de go est environ de 10^{600}, le nombre 361!/100! des différentes parties de plus de 260 coups). Cette difficulté est amplifiée par d'autres caractéristiques du jeu : la nature de la condition de victoire, le placement virtuellement illimité de chaque pierre, la nature non locale de la règle du *ko*, le haut niveau de reconnaissance de formes exigé. Pour ces raisons, certains chercheurs en intelligence artificielle considèrent le go comme un meilleur test que les échecs.

En décembre 2017, **DeepMind** dévoile *AlphaZero*, une variante d'*AlphaGo*, qui reprend le principe de l'apprentissage autodidacte par renforcement dans une approche moins spécialisée. En disposant pour seule base des règles des jeux d'échecs, de go et de shogi (variante japonaise des échecs), cette IA est parvenue à atteindre un « niveau de jeu surhumain » et à battre les meilleurs programmes existant dans ces trois disciplines, et cela en moins de 24 heures !

À partir de 2006, la programmation du jeu de go a fait des progrès importants notamment grâce à la méthode de Monte-Carlo (on explore seulement certaines parties de l'arbre de jeu, avec une probabilité donnée). Les programmes parviennent désormais à égaler des joueurs de haut niveau sur un goban de taille 9x9 ou à des handicaps de 6 à 9 pierres sur un goban de taille 19x19. En 2009, les meilleurs programmes sont parvenus (en parties rapides) à obtenir un niveau de 1er dan amateur sur le serveur KGS.

Samedi 12 mars 2016, **Lee Se-Dol**, qui domine le jeu de go depuis une décennie, a perdu son match face à un ordinateur. Le programme, *AlphaGo*, a remporté sa troisième victoire consécutive dans une série de cinq parties à Séoul. Le programme a été développé par **DeepMind**, une start-up rachetée par **Google** spécialisée dans le « deep learning », une méthode permettant aux algorithmes d'apprendre par eux-mêmes pour résoudre un problème. On utilise cette technique notamment pour la reconnaissance d'image ou vocale ; elle a permis à Google de battre le maître mondial du jeu de go. Un exploit que l'on pensait impossible avant longtemps.

10.1.3. Jeux de connaissance

2011 : un ordinateur bat deux champions au Jeopardy

Du nom du fondateur d'IBM, Thomas **Watson**

En remportant deux manches sur trois, *Watson* a gagné 1 million de dollars au jeu Jeopardy (un jeu où on donne la réponse et où il faut deviner la question).

L'écran était installé entre les deux joueurs humains et l'ordinateur, commandé par un opérateur, devait répondre aux questions de culture générale posées par l'animateur. Watson étant sourd et muet, l'opérateur tapait les questions sur le clavier et annonçait ses réponses. Ses adversaires humains n'étaient pas les premiers venus : la machine a combattu en effet les deux plus brillants compétiteurs de l'histoire de ce jeu, revenus sur le plateau pour ce match du siècle.

Avec ses 15 To (téraoctets) de mémoire vive, ses 2.880 processeurs Power 7, Watson n'a rien d'un micro. « S'ils tournaient sur un micro-ordinateur de bureau, les logiciels mettraient 2 heures pour répondre à une question » affirme-t-on chez IBM. Comme ses adversaires, Watson n'avait pas accès à Internet mais avait tout de même un avantage certain : IBM avoue que Watson disposait d'une antisèche équivalent à 200 millions de pages.

10.2. Jeu avec stratégie gagnante : Marienbad

Ce jeu de société combinatoire abstrait, dont il existe plusieurs variantes, se joue avec des graines, des dominos, des jetons, des cartes, des allumettes, ... Son origine est probablement très ancienne. Il appartient à la famille plus large des jeux de Nim.

Le **jeu de Marienbad** a été popularisé par le film d'Alain **Resnais**, *L'année dernière à Marienbad*, en 1961, au point d'en prendre le nom. Dans ce film, le héros gagne parties sur parties. Il prononce cette phrase à la portée symbolique : *Je peux perdre, mais je gagne toujours...*

Dans la version du film, il y a quatre rangées, avec respectivement 1, 3, 5, 7 allumettes et celui qui prend la dernière allumette perd. À chaque tour, le joueur prend le nombre d'allumettes qu'il veut, au moins une et dans une même rangée.

Stratégie gagnante

La méthode repose sur le système binaire. La position de départ précisée par le dessin ci-contre, s'analyse à l'aide des calculs suivants :

1 = 0 0 1 en binaire
3 = 0 1 1 "
5 = 1 0 1 "
7 = 1 1 1 "

Si on effectue les sommes des chiffres du binaire colonne par colonne en base dix, on trouve :

S = 2 2 4

La stratégie dans le cas où celui qui prend le dernier objet gagne commence à l'identique, mais il faut viser $S = 4$ ou $S = 2$ au final.

Tous les chiffres de S sont pairs, le joueur qui débutera la partie perdra si son adversaire prend le soin de conserver cette propriété de S tant que ce nombre possède au moins deux chiffres. En fin de partie, il convient de choisir $S = 3$ ou $S = 1$ pour l'emporter.

Si on représente les différentes combinaisons de jeu, on voit qu'il existe un chemin (on parle de kernel : ensemble de nœuds) de coups gagnants. Ainsi, un joueur qui se trouve dans une situation gagnante et qui connaît l'astuce est sûr de gagner à la fin. Le joueur qui ne commence pas est dans le kernel (S pair) et est donc sûr de gagner quelles que soient les actions de l'adversaire (pourvu qu'il connaisse l'astuce, i.e., qu'il reste dans le kernel). Son adversaire n'a pas son destin entre ses mains. Ceci est vrai que le jeu soit classique (dernière allumette = perdu) ou non classique (dernière allumette = gagné).

Exercice 10.1

Programmez le jeu de Marienbad avec une interface en ligne de commande (entrez les coups au clavier selon un code que vous indiquerez).

10.3. Utilisation de la force brute : Mastermind

Ce jeu de déduction se présente généralement sous la forme d'un plateau perforé de 10 rangées de quatre trous pouvant accueillir des boules de couleurs. Il y a également des pions blancs et noirs utilisés pour donner des indications à chaque tour de jeu.

Un joueur commence par placer des boules sans qu'elles soient vues de l'autre joueur.

L'autre doit trouver quelles sont les quatre boules, c'est-à-dire leur couleur et leur position.

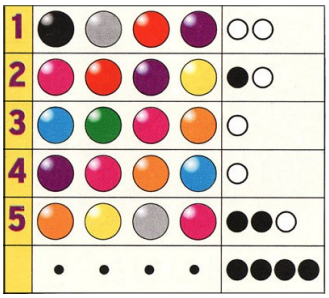
Pour cela, à chaque tour, le joueur doit se servir de boules pour remplir une rangée selon l'idée qu'il se fait des boules dissimulées. L'autre joueur indique alors :

1. le nombre de boules de la bonne couleur bien placées en utilisant le même nombre de pions noirs ;

2. le nombre de boules de la bonne couleur, mais mal placées, avec les pions blancs.

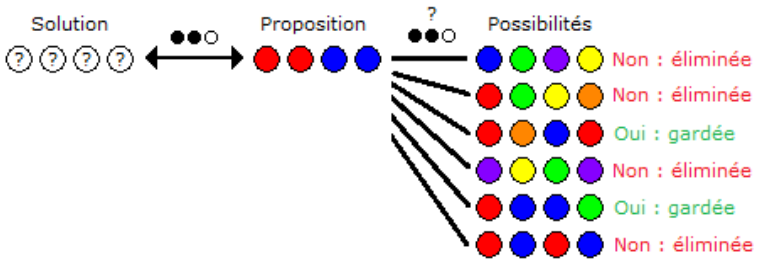
Exercice 10.2

Quelle est la combinaison gagnante ?



Comment l'ordinateur trouve la solution ?

Ce jeu exige beaucoup de déduction et de concentration de la part d'un humain. Cependant, il est facile pour un ordinateur de trouver la combinaison gagnante. Il va au début construire une liste de toutes les solutions possibles (il y en a k^n, où n est le nombre d'emplacements et k le nombre de couleurs des boules). Il va ensuite éliminer de cette liste toutes les configurations impossibles en fonction des pions noirs et blancs, comme l'indique le schéma ci-dessous :

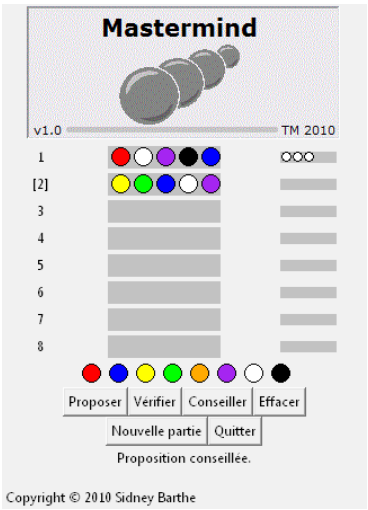


Au bout d'un certain nombre de tours, il ne restera plus qu'une solution possible.

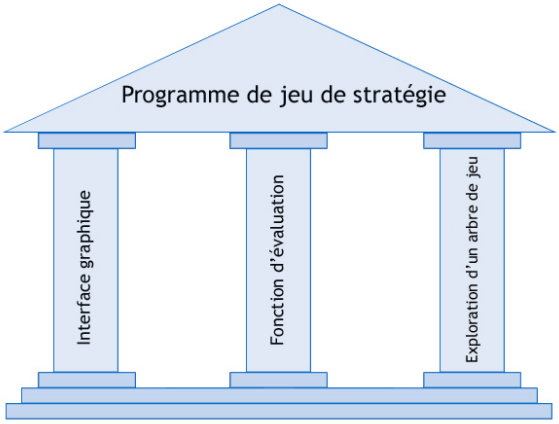
C'est ce qu'on appelle la force brute : on passe en revue toutes les possibilités et on élimine celles qui ne vont pas.

Exercice 10.3

Téléchargez et testez le programme de Sydney **Barthe** disponible sur le site compagnon. Pour voir comment le programme joue, pressez alternativement les touches *Conseiller* puis *Proposer*.



10.4. Les trois piliers d'un jeu de stratégie



10.4.1. Interface graphique

Une interface graphique (en anglais *GUI* pour *graphical user interface*) est un dispositif de dialogue homme-machine, dans lequel les objets à manipuler sont dessinés sous forme de pictogrammes à l'écran, que l'usager peut opérer en imitant la manipulation physique de ces objets avec un dispositif de pointage, le plus souvent une souris. Ce type d'interface a été créé par Xerox en 1981 pour remplacer les interfaces en ligne de commande, puis popularisé par Apple avec l'ordinateur Macintosh quelques années plus tard.

Pour les jeux, cette interface graphique peut être plus ou moins élaborée. Elle est simple pour des jeux comme les dames ou les échecs, et très complexes pour des jeux en trois dimensions. Ci-dessous, l'interface graphique de *Myth II* :

10.4.2. Fonction d'évaluation

Comme son nom l'indique, cette fonction a pour but d'évaluer une position : un nombre très grand sera l'indice d'une excellente position, tandis qu'un nombre très petit (généralement négatif, mais cela dépend de l'échelle choisie) traduira une position catastrophique.

Comment obtient-on une « bonne » fonction, c'est-à-dire une fonction qui traduit fidèlement la réalité ? Une fonction d'évaluation *f* est généralement une somme pondérée de la forme :

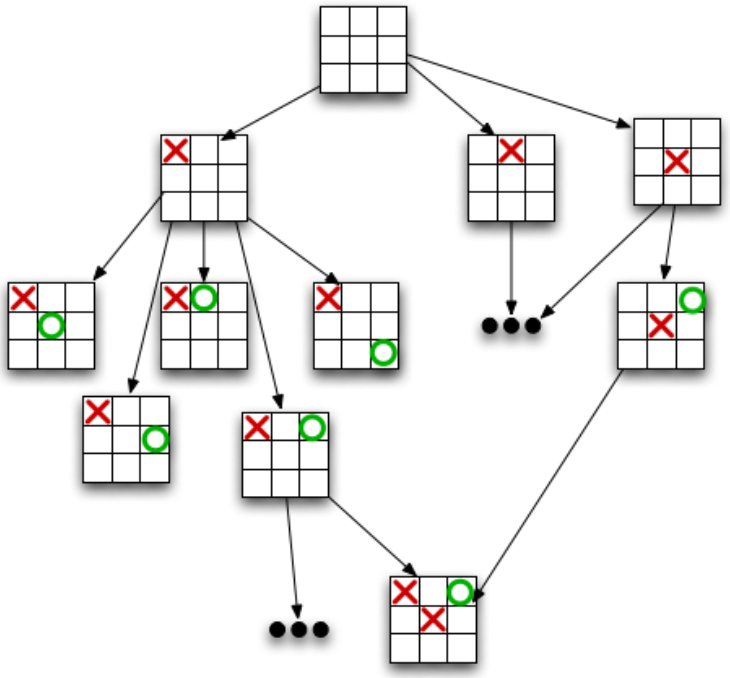
$$f(P) = a_1 \cdot c_1(P) + a_2 \cdot c_2(P) + \ldots + a_n \cdot c_n(P)$$

où les a_i sont des *poids* (des nombres réels) et les c_i des *caractéristiques* d'une position P (par exemple, aux échecs, le nombre de pions doublés, les pions occupant le centre, le nombre de fous pris à l'adversaire, le nombre de fous perdus, etc.). Trouver les caractéristiques d'un jeu n'est généralement pas chose facile, et les programmeurs ont souvent recours à des experts pour les y aider. Quant aux poids, qui traduisent le fait qu'une caractéristique est plus importante qu'une autre, leur ajustement fait plus appel à l'empirisme et à l'expérience du programmeur qu'à une méthode rigoureuse.

Une fonction d'évaluation peut aussi être « apprise » à l'aide d'un algorithme d'apprentissage à partir d'un ensemble de parties jouées.

10.4.3. Arbre de jeux

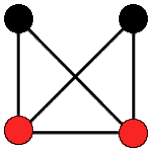
Un arbre de jeu consiste à représenter virtuellement, sous forme d'un arbre orienté, toutes les positions que l'on peut atteindre à partir de celles du coup précédent. Selon le jeu, cet arbre devient gigantesque à partir de peu d'étages déjà. Aux échecs par exemple, les blancs ont le choix entre 20 coups (16 coups de pions et 4 coups de cavaliers) pour débuter la partie. Cela fait donc déjà 20 branches qui partent du sommet. Ensuite, les noirs ont aussi 20 coups à disposition. Cela fait déjà 400 positions possibles après 2 coups, puisque 20 branches partent des 20 positions précédentes !



Autant dire, qu'il est impossible de dessiner l'arbre de jeu complet, sauf pour des jeux très simples. Aussi doit-on élaguer l'arbre pour éliminer les positions manifestement mauvaises pour le joueur. Nous verrons plus tard comment s'y prendre.

Il est à noter qu'un arbre de jeu est une représentation virtuelle de tous les coups possibles. Dans le programme de jeu, il ne s'agira pas d'implémenter un arbre comme nous l'avions fait au chapitre 6.

Exercice 10.4



Vous voyez ci-contre le tablier du jeu **Pong Hau K'i**. Chaque joueur bouge à tour de rôle un des jetons de sa couleur sur la seule intersection disponible. Le but du jeu est de coincer l'adversaire.

Ce jeu est l'un des rares assez simples pour que l'on puisse dessiner l'arbre de jeu complet.

1. Combien y a-t-il de positions possibles ? Combien sont gagnantes ?
2. Dessinez l'arbre de ce jeu, en supposant que les noirs commencent. Que constatez-vous ?

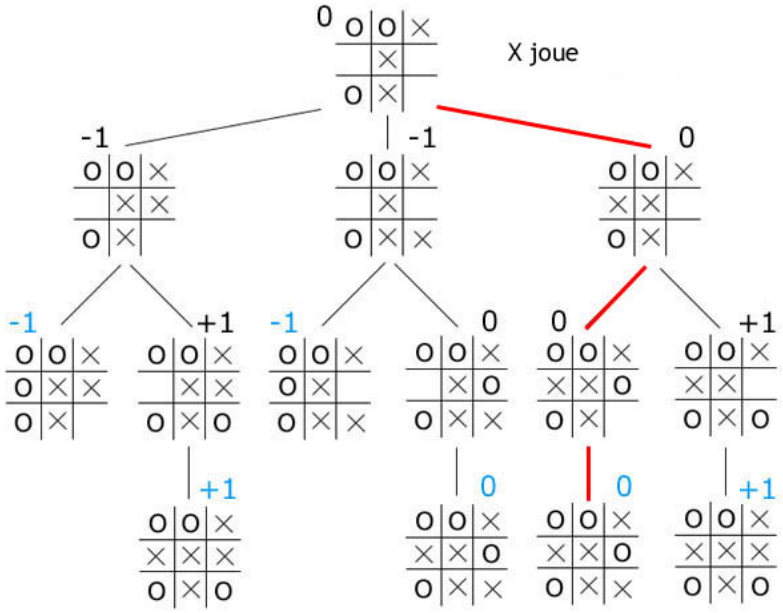
3. Dessinez l'arbre de ce jeu, en supposant que les rouges commencent. Que constatez-vous?

Conseil : utilisez une feuille A3!
Remarque : dessinez un graphe plutôt qu'un arbre, c'est-à-dire que vous pouvez relier une position à une position déjà rencontrée, et ainsi « remonter » dans l'arbre.

10.4.4. Algorithme minimax

L'algorithme minimax est un algorithme qui s'applique à la théorie des jeux pour les jeux à deux joueurs à somme nulle. Dans un jeu à somme nulle, la somme des gains de tous les joueurs est égale à 0. Par exemple, si l'on définit le gain d'une partie de tic-tac-toe comme 1 si on gagne, 0 si la partie est nulle et –1 si on perd, le tic-tac-toe est un jeu à somme nulle.

Cet algorithme amène l'ordinateur à passer en revue toutes les possibilités pour un nombre limité de coups et à leur assigner une valeur qui prend en compte les bénéfices pour le joueur et pour son opposant. Le meilleur choix est alors celui qui minimise les pertes du joueur tout en supposant que l'opposant cherche au contraire à les maximiser.



Fin d'une partie de Tic-tac-toe. Si les deux adversaires jouent juste (chemin rouge), la partie sera nulle (score de 0).
+1 indique une victoire des x.
-1 indique une victoire des o.

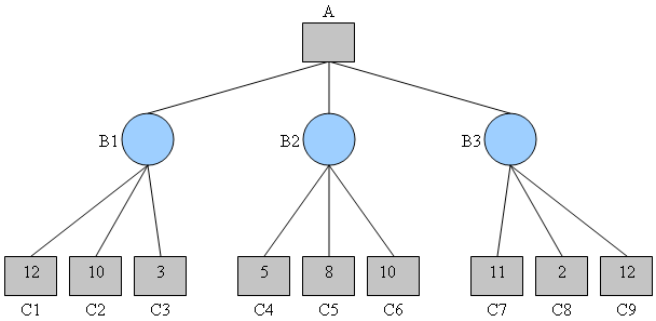
Il existe différents algorithmes basés sur le minimax permettant d'optimiser la recherche du meilleur coup, en limitant le nombre de nœuds visités dans l'arbre de jeu. Le plus connu est l'élagage alpha-bêta. En pratique, l'arbre est souvent trop vaste pour pouvoir être intégralement exploré (comme par exemple pour le jeu d'échecs ou de go). Seule une fraction de l'arbre est alors explorée.

Principe

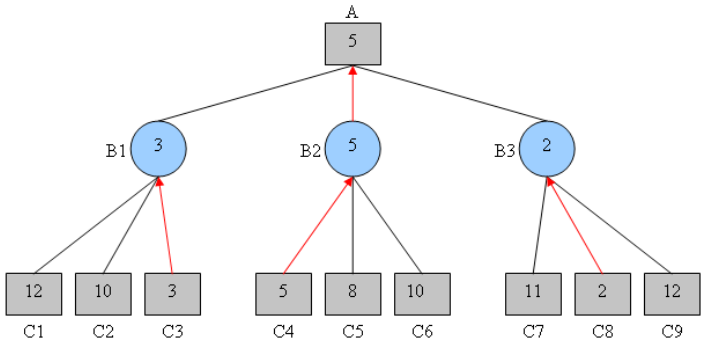
L'algorithme minimax est très simple : on visite l'arbre de jeu pour faire remonter à la racine une valeur (appelée « valeur du jeu ») qui est calculée récursivement. Soit p un nœud de l'arbre de jeu et f est une fonction d'évaluation de la position du jeu. Alors :

- valeur(p) = f (p) si p est une feuille de l'arbre
- valeur(p) = MAX(valeur(O_1), …, valeur(O_n)) si p est un nœud Joueur avec pour fils O_i
- valeur(p) = MIN(valeur(O_1), …, valeur(O_n)) si p est un nœud Opposant avec pour fils O_i

Exemple



Dans le schéma ci-dessus, les nœuds gris représentent les nœuds Joueur et les bleus les nœuds Opposant. Pour déterminer la valeur du nœud A, on choisit la valeur maximum de l'ensemble des nœuds B (A est un nœud Joueur). Il faut donc déterminer les valeurs des nœuds B qui reçoivent chacun la valeur minimum stockée dans leurs fils (les nœuds B sont Opposant). Les nœuds C sont des feuilles, leur valeur peut donc être calculée par la fonction d'évaluation.



Le nœud A prend donc la valeur 5. Le joueur doit donc jouer le coup l'amenant en B2. En observant l'arbre, on comprend bien que l'algorithme considère que l'opposant va jouer de manière optimale : il prend le minimum. Sans ce prédicat, on choisirait le nœud C1 qui propose le plus grand gain et le prochain coup sélectionné amènerait en B1. Mais alors on prend le risque que l'opposant joue C3 qui propose seulement un gain de 3.

En pratique, la valeur théorique de la position *P* ne pourra généralement pas être calculée. En conséquence, la fonction d'évaluation sera appliquée sur des positions non terminales. On considérera que plus la fonction d'évaluation est appliquée loin de la racine, meilleur est le résultat du calcul. C'est-à-dire qu'en examinant plus de coups successifs, nous supposons obtenir une meilleure approximation de la valeur théorique donc un meilleur choix de mouvement.

C'est de cette façon que l'on déterminera la force d'un programme : plus il descendra bas dans l'arbre, plus il sera redoutable (en supposant qu'il a une bonne fonction d'évaluation). La profondeur ne devra d'ailleurs pas forcément être la même pour tous les nœuds.

10.4.5. Élagage alpha-bêta

L'élagage alpha-bêta (*alpha-beta pruning* en anglais) est une technique très utilisée permettant de réduire le nombre de nœuds évalués par l'algorithme minimax.

L'algorithme minimax effectue en effet une exploration complète de l'arbre de recherche jusqu'à un niveau donné, alors qu'une exploration partielle de l'arbre est généralement suffisante : lors de l'exploration, il n'est pas nécessaire d'examiner les sous-arbres qui conduisent à des configurations dont la valeur ne contribuera sûrement pas au calcul du gain à la racine de l'arbre. L'élagage α-β nous permet de réaliser ceci.

Plus simplement, l'élagage α-β évite d'évaluer des nœuds dont on est sûr que leur qualité sera inférieure à un nœud déjà évalué, il permet donc d'optimiser grandement l'algorithme minimax sans en modifier le résultat.

Principe

On prend α et β appartenant au domaine d'arrivée de la fonction d'évaluation tel que $\alpha < \beta$. On définit la fonction AlphaBeta ainsi :

- AlphaBeta(P, α, β)=$g(P)$ si P est une feuille de l'arbre et g la fonction d'évaluation du nœud
- AlphaBeta(P, α, β)=min(β, max(−AlphaBeta(O_i, −β, −α))) où les O_i sont les fils du nœud P

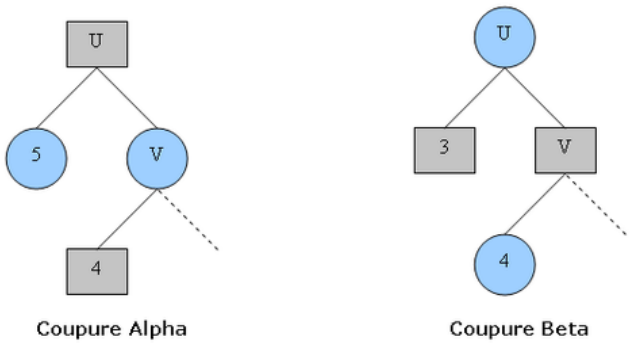
On appelle fenêtre α-β le couple (α, β) où α et β sont les deux paramètres d'appel de la fonction. Les nœuds élagués sont ceux qui seraient appelés avec une fenêtre tel que $\alpha \geq \beta$. Il existe 3 types de nœuds ne pouvant donc pas être élagués :

- *Nœud de type 1* : fenêtre d'appel : $(-\infty, +\infty)$
- *Nœud de type 2* : fenêtre d'appel : $(-\infty, \beta)$ avec $\beta \neq +\infty$
- *Nœud de type 3* : fenêtre d'appel : $(\alpha, +\infty)$ avec $\alpha \neq -\infty$

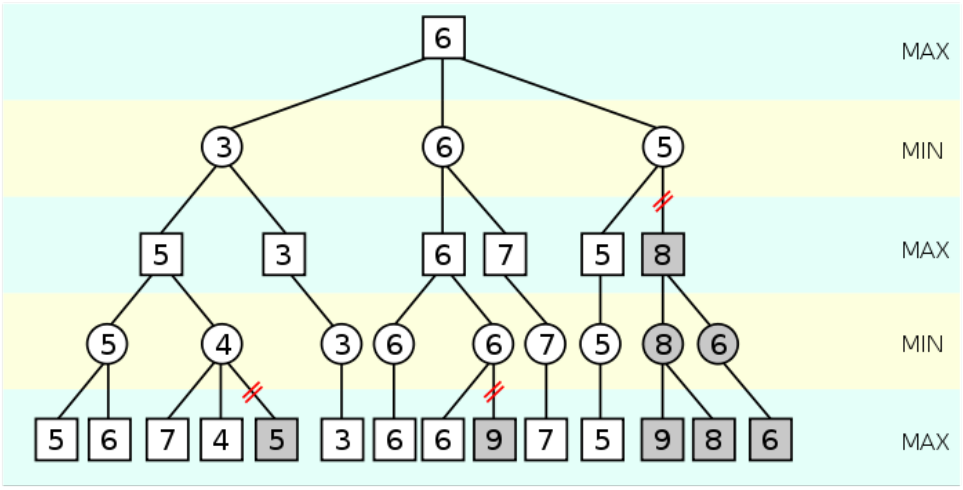
Le schéma ci-dessous présente les deux types de coupures possibles. Les nœuds Min sont représentés par un rond bleu et les nœuds Max par un carré gris. Rappel : les nœuds Min prennent la valeur minimum de leurs fils (et respectivement maximum pour les nœuds Max).

Coupure Alpha : le premier fils du nœud Min V vaut 4 donc V vaudra au plus 4. Le nœud Max U prendra donc la valeur 5 (maximum entre 5 et une valeur inférieure ou égale à 4).

Coupure Beta : le premier fils du nœud Max V vaut 4 donc V vaudra au minimum 4. Le nœud Min U prendra donc la valeur 3 (minimum entre 3 et une valeur supérieure ou égale à 4).



Voici le résultat de l'élagage sur l'arbre ci-dessous déjà étiqueté avec les valeurs d'un minimax.



Minimax avec élagage alpha-bêta

Trois coupures ont pu être réalisées :

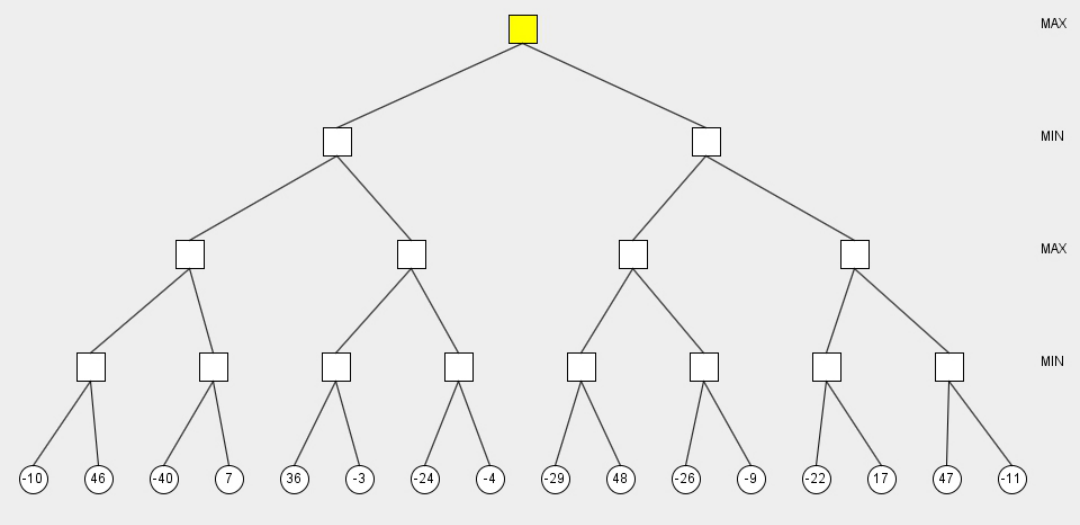
1. Le nœud MIN vient de mettre à jour sa valeur courante à 4. Celle-ci, qui ne peut que

baisser, est déjà inférieure à α=5, la valeur actuelle du nœud MAX précédent. Celui-ci cherchant la valeur la plus grande possible, ne la choisira donc de toute façon pas.

2. Le nœud MIN vient de mettre à jour sa valeur courante à 6. Celle-ci, qui ne peut que baisser, est déjà égale à α=6, la valeur actuelle du nœud MAX précédent. Celui-ci cherchant une valeur supérieure, il ne mettra de toute façon pas à jour sa valeur que ce nœud vaille 6 ou moins.
3. Le nœud MIN vient de mettre à jour sa valeur courante à 5. Celle-ci, qui ne peut que baisser, est déjà inférieure à α=6, la valeur actuelle du nœud MAX précédent. Celui-ci cherchant la valeur la plus grande possible, ne la choisira donc de toute façon pas.

Exercice 10.5

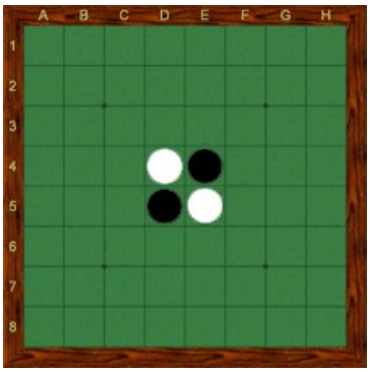
1. Remplissez les valeurs des nœuds de l'arbre ci-dessous en suivant l'algorithme minimax.
2. Refaites ensuite l'exercice avec l'élagage alpha-bêta.



10.5. Othello

Othello est basé sur le jeu Reversi qui a été inventé en 1883 par l'Anglais Lewis **Waterman**, et acquis une popularité considérable en Angleterre à la fin du XIXe siècle. Le jeu d'Othello sous sa forme actuelle a été commercialisé pour la première fois au Japon en 1971.

C'est un jeu de stratégie à deux joueurs : Noir et Blanc. Il se joue sur un damier vert de 64 cases, 8 sur 8.



Dans le Reversi original, il n'y a aucun pion posé en début de partie, les ouvertures sont donc libres. Une partie d'Othello débute avec 4 pions placés au centre.

Ces joueurs disposent de 64 pions bicolores, noirs d'un côté et blancs de l'autre. Par commodité, chaque joueur a devant lui 30 pions (2 de sa couleur sont déjà placés sur le plateau au début de la partie, comme montré sur le schéma ci-dessus).

But du jeu

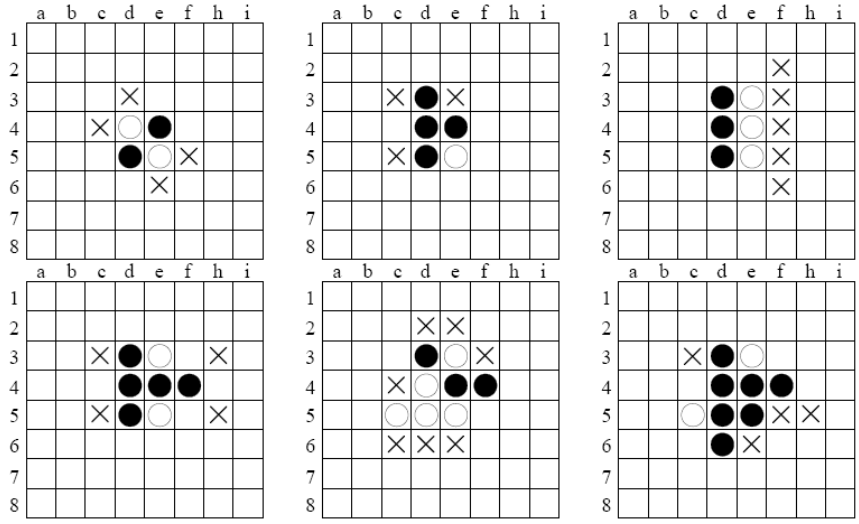
Avoir plus de pions de sa couleur que l'adversaire à la fin de la partie.
Celle-ci s'achève quand aucun des deux joueurs ne peut plus jouer de coup légal.

La pose d'un pion

À son tour de jeu, le joueur doit poser un pion de sa couleur sur une case vide du plateau, adjacente à un pion adverse. En posant son pion, il doit encadrer un ou plusieurs pions adverses entre le pion qu'il pose et un pion de sa couleur, déjà placé sur le plateau, que ce soit sur une ligne horizontale, verticale ou diagonale.

Les pions encadrés sont alors retournés en pions de couleur inverse. Les pions ne sont jamais retirés du plateau, ni déplacés d'une case à l'autre.

Si un joueur ne peut pas poser de pions, il passe son tour.



6 premières positions d'une partie d'Othello. La position initiale du jeu est celle en haut à gauche et c'est le joueur noir qui débute la partie. Le premier coup de noir est 'd3', la réplique de blanc est 'e3', puis noir joue en 'f4' et blanc en 'e5', et le dernier coup joué par noir est 'd6'.

Force des programmes

Depuis 1995 et le programme *Logistello* de Michael **Buro**, l'ordinateur est beaucoup plus fort que l'humain au jeu d'Othello.

Les programmes d'Othello sont relativement simples à écrire. Aussi est-il devenu classique de développer un programme d'Othello pendant ses études d'informatique.

10.5.1. Analyse du programme Othello-py

http://ow.ly/5iIy8

Nous allons analyser un programme jouant à Othello[2]. Il a été francisé et modifié par l'auteur pour être compatible avec Python 3. Allez sur le site compagnon pour télécharger cette version remaniée.

Le programme se compose de quatre modules :

- « minimax »
  - explore l'arbre de jeu
- « othello »
  - implémente le jeu Othello (coups légaux, retournement des pions, fin de partie, etc.)
  - contient la fonction d'évaluation (« edge_eval »)

2 Voir [10]

- « game2 » :
  - gère la partie elle-même (tour de jeu, temps de réflexion, gain de la partie, etc.)
  - permet de tester des parties ordinateur contre ordinateurs
  - peut être exécuté sans passer par l'interface graphique
- « othello_gui »
  - est l'interface graphique
  - à exécuter pour qu'un humain joue contre l'ordinateur

Exercice 10.6

1. Téléchargez ces quatre modules et analysez-les attentivement. En particulier,
   - comment fonctionne l'élagage alpha-bêta dans le module « minimax » ?
   - comment est calculée la fonction d'évaluation ?
2. Modifiez la fonction d'évaluation pour tenter de rendre le programme encore plus fort. Testez votre fonction comme indiqué dans les commentaires en bas du module « game2 ».

Sources

[1] Wikipédia, « Intelligence artificielle », <http://fr.wikipedia.org/wiki/Intelligence_artificielle>

[2] Wikipédia, « Programme d'échecs », <http://fr.wikipedia.org/wiki/Programme_d'échecs>

[3] Wikipédia, « Backgammon », <http://fr.wikipedia.org/wiki/Backgammon>

[4] Wikipédia, « Jeu de Go », <http://fr.wikipedia.org/wiki/Jeu_de_go>

[5] Wikipédia, « Jeu de Marienbad », <http://fr.wikipedia.org/wiki/Jeu_de_Marienbad>

[6] Barthe Sidney, *Mastermind. Un jeu compliqué ?*, Travail de maturité, février 2010

[7] Wikipédia, « Algorithme Minimax », <http://fr.wikipedia.org/wiki/Algorithme_minimax>

[8] Wikipédia, « Elagage alpha-bêta », <http://fr.wikipedia.org/wiki/élagage_alpha-bêta>

[9] Fédération française d'Othello, <http://www.ffothello.org/>

[10] Othello-py, <http://code.google.com/p/othello-py/>