IA pour les Jeux
Composants d'une résolution arborescente
Licence Informatique et Vidéoludisme

Nicolas Jouandeau

n@up8.edu

2024

représentation courante du matériel (jeu de plateau)

- un état du jeu (*i.e.* une position du jeu)
  - des constantes identifiant les types de pièces
  - une grille 1D ou 2D (pour le plateau)
  - une taille de grille
  - une valeur de tour
- une pièce
  - un type
  - une position sur la grille
- un coup
  - une position intiale
  - une position finale

fonctions utiles (jeu de 1 à N joueurs)

- initialiser l'état du jeu
- afficher l'état courant
- lister les coups possibles
- jouer un coup choisi (MAJ l'état courant)
- jouer un coup aléatoire (MAJ l'état courant)
- déjouer un coup (MAJ l'état courant)
- dire si un état est terminal
- donner le score d'un état terminal
- donner une évaluation heuristique d'un état non terminal
- donner une clé identifiant un état
- jouer un playout (partie aléatoire)

fonctions utiles (jeu à 2 joueurs)

- joueur random
- joueur minimax à profondeur fixée
- jouer 1 partie avec deux joueurs
- jouer N parties (joueurA, joueurB)
- jouer N parties (joueurB, joueurA)
- afficher le nombre de victoires de chaque joueur sur N parties

premiers tests

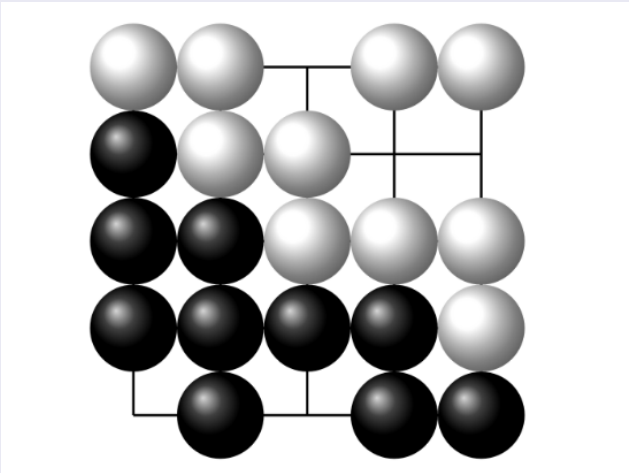
- réduction du problème (plateau 5x4)
  - évaluation de la profondeur pour une résolution minimax complète
  - test du joueur aléatoire contre le joueur minimax

développement

- revenir sur un plateau + grand
  - test de la fonction heuristique pour l'évaluatiuon des états non terminaux
  - ajout de nouvelles fonctions heuristiques
  - ajout de nouveaux joueurs (*i.e.* de nouveaux algorithmes)

représentation "bitboard" pour jouer plus vite

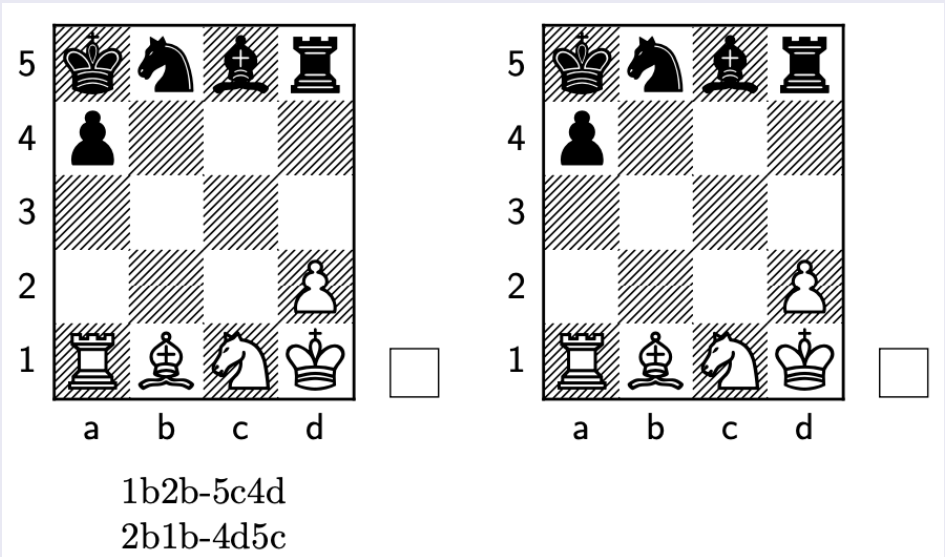
- conversion des pions en représentation binaire
- 1 position => - de mémoire => - de swap => + rapide
- exemple sur un plateau de go



- 11011011000011110000100000 (pour blanc)
- 00000100001100011111001011 (pour noir)
- plateau 5x5 -> 2x25 bits par position
- appliquer un coup = appliquer 2 opérations 32bits
- obtenir les coups possibles avec `not(blanc xor noir)` (`~(blanc ^ noir)` en C avec les entiers non signés)

cycles (une particularité de certains jeux)

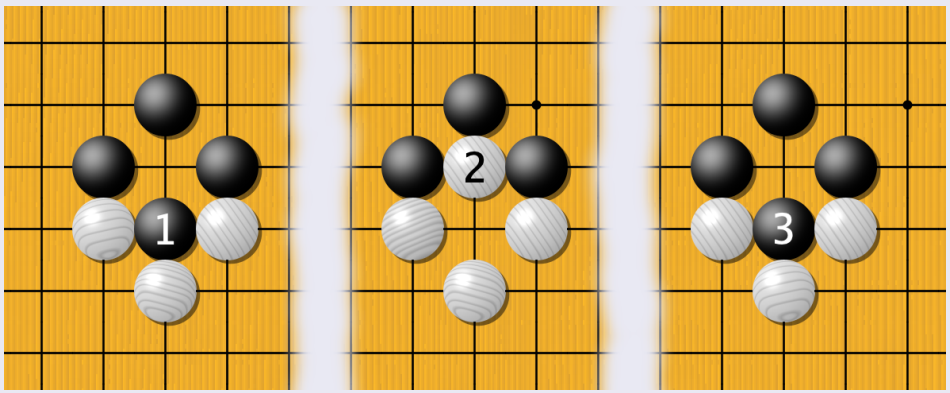
- cycle = séquence de coups revenant sur une position précédente
- exemple sur un plateau de Microchess



  - considérer le tour de jeu dans l'état produit des positions différentes, ce qui fait que deux positions identiques pourrait ne pas avoir les mêmes statistiques
  - les cycles produisent des boucles infinies dans les descentes

éviter les cycles (la règle du ko au go)

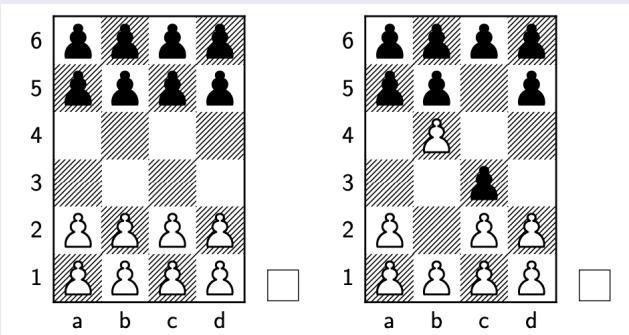
- certains jeux interdisent les cycles dans leurs règles
- règle du ko : interdiction de jouer un coup revenant sur la position précédent la position courante
- exemple sur un plateau de go



  - 1 et 2 sont des coups légaux
  - 3 est interdit par la règle du ko

transpositions

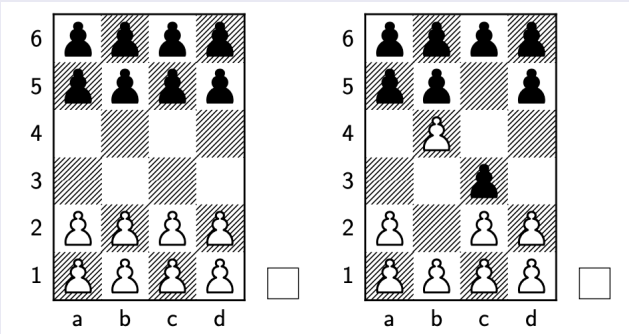
- variation d'une séquence qui mène à la même position
  - deux positions distinctes A et B
  - deux séquences distinctes 1 et 2
  - partant de A, appliquer la séquence 1 mène à B
  - partant de A, appliquer la séquence 2 mène à B
  - A est une transposition de B
  - B est une transposition de A
- exemple à breakthrough



- séquences de coups pour blanc (sans considérer noir)
  - `2b3b` puis `3b4b` (abrégée `f` puis `f`)
  - `2b3a` puis `3a4b` (abrégée `l` puis `r`)
  - `2b3c` puis `3c4b` (abrégée `r` puis `l`)

transpositions

- coups de blanc en minuscule et coups de noir en majuscule
- `f` pour forward, `l` pour left et `r` pour right



- 7 transpositions considérant les pions partant de `b2` et `c5`
  - `fFfF`, `fLfR`, `fRfL`
  - `lFrF`, `lRrL`
  - `rFlF`, `rRlL`

en pratique

- ne pas considérer les transpositions
  - crée des positions identiques avec des statistiques incomplets
  - multiplie les positions inutilement en mémoire
- considérer les transpositions
  - stocker les positions dans $\mathcal{H}$
    - une table de hashage (*i.e.* `unordered_map`)
    - un arbre (*i.e.* `map`)
      arbre binaire de recherche ABR
      arbre à équilibrage automatique AVL
      arbre bicolore
  - identifier une position par une clé
    - rapidité de la fonction de hashage
    - taille en mémoire de la table des positions
    - question des collisions (une clé pour deux positions)

pour minimiser l'espace mémoire nécessaire

- une position -> une clé
- une clé -> une position
- $\mathcal{H}$[clé] contient les statistiques de la position
- recherche, ajout et suppression en $O(log(n))$ au pire
- partant d'une position
  - calculer la liste des coups possibles
  - jouer chaque coup pour obtenir la nouvelle position
  - rechercher les statistiques de chaque nouvelle position
  - décider du meilleur coup

résoudre un jeu

- prédire le résultat (victoire/défaite) à partir de n'importe quelle position en supposant que les joueurs jouent parfaitement
- résolution ultra-faible
  - démontrer le résultat pour le premier joueur à partir de la position initiale
  - par application d'un principe
- résolution faible
  - avoir un algorithme qui donne les coups à jouer à partir de la position initiale
  - prédit la victoire pour un des joueurs
  - pas obligatoirement optimal contre un joueur qui joue mal
- résolution forte
  - avoir un algorithme qui donne les meilleurs coups à jouer à partir de n'importe quelle position

quelques jeux résolus

- tic tac toe (résolution triviale)
  - chaque joueur peut forcer un match nul
- tic tac toe 3D
  - achevée en 1980 par O. Patashnik et V. Allis
  - résolution faible
  - le premier joueur gagne
- awalé (variante Oware)
  - achevée en 2002 par J.W. Romein et H.E. Bal
  - résolution forte en 51h de calcul avec 144 cpu
  - évaluation de 889 milliards d'états
  - la partie parfaite se solde par un match nul
- dames
  - achevée en 2007 par J. Schaeffer
  - déplacements courts de dames uniquement
  - résolution faible en 18 ans de calcul
  - évaluation de 10^{14} états
  - la partie parfaite se solde par un match nul