Sujet Grand Oral NSI : Comment un programme informatique peut-il jouer aux échecs ?

▶ SommaireSauter à une section

Bonjour. Je vais vous parler des échecs vus comme un problème d'**informatique**. Pas du jeu en lui-même, mais de la **question** suivante : comment un programme informatique peut-il prendre des décisions pour jouer aux échecs ?

Cette **question** m'intéresse parce qu'elle croise trois piliers du **programme** de spécialité **NSI** en **terminale** : la récursivité, les structures de **données** arborescentes, et l'analyse de **complexité**. Les échecs sont l'un des plus vieux **sujets** étudiés par l'**informatique** : Alan Turing écrit le premier programme d'échecs sur papier en 1950, et soixante-quinze ans plus tard, AlphaZero démontre qu'une machine peut apprendre à jouer sans aucune intervention humaine. Entre les deux, presque toute l'histoire de l'intelligence artificielle.

Pour répondre à ma **question**, je vais procéder en trois étapes. D'abord, je modéliserai le jeu comme un arbre de décision et je présenterai l'algorithme minimax qui l'explore. Ensuite, je m'appuierai sur la fonction d'évaluation, c'est-à-dire la manière dont le programme attribue un score à une position. Enfin, je discuterai les **limites** de cette approche et ce qu'AlphaZero a changé.

L'arbre de décision et l'algorithme minimax

L'idée fondamentale est de modéliser le jeu d'échecs comme un arbre. La racine de cet arbre est la position de départ. Chaque branche représente un coup possible. Chaque nœud fils représente la nouvelle position obtenue après ce coup. En théorie, on peut développer cet arbre jusqu'à la fin de la partie : les feuilles sont alors les positions de mat, de pat ou de nulle.

Le **programme** doit choisir le meilleur coup parmi tous les coups possibles. Mais comment ? L'algorithme **minimax**, formalisé par Claude Shannon en 1950, repose sur une intuition très simple : je joue pour gagner, mon adversaire joue pour me faire perdre. Donc à chaque tour, j'alterne entre maximiser mon score (mon tour) et minimiser ce même score (le tour de mon adversaire, supposé jouer optimalement).

Voici comment cela se traduit en Python.

Python Copier

```python
def minimax(position, profondeur, maximise):
    if profondeur == 0 or position.est_terminee():
        return evaluer(position)
    if maximise:
        meilleur = -float('inf')
        for coup in position.coups_possibles():
            position.joue(coup)
            score = minimax(position, profondeur - 1, False)
            position.annule(coup)
            meilleur = max(meilleur, score)
        return meilleur
    else:
        meilleur = +float('inf')
        for coup in position.coups_possibles():
            position.joue(coup)
            score = minimax(position, profondeur - 1, True)
            position.annule(coup)
            meilleur = min(meilleur, score)
        return meilleur
```

Trois éléments du **programme** de spécialité **NSI** apparaissent ici. Premièrement, la **récursivité** : la fonction `minimax` s'appelle elle-même, sur une position transformée par un coup. Deuxièmement, le **cas de base** : la condition `profondeur == 0` ou `position.est_terminee()` qui arrête la récursion. Sans ce cas de base, on aurait une récursion infinie. Troisièmement, la structure d'**arbre** : chaque appel récursif explore les fils d'un nœud avant de remonter le résultat au parent. C'est exactement un parcours en profondeur.

La **complexité** de l'algorithme est cruciale à comprendre. Aux échecs, chaque joueur dispose en moyenne de trente-cinq coups légaux. Si je veux explorer toutes les variantes jusqu'à une profondeur de quatre demi-coups, je dois examiner trente-cinq puissance quatre positions, soit environ un million cinq cent mille. À profondeur six, on dépasse le milliard. La **complexité** est exponentielle, en notation grand O, on écrit O(b puissance d), où b est le facteur de branchement et d la profondeur.

Or les échecs ont environ dix puissance cent vingt parties possibles, plus que d'atomes dans l'univers observable. Il est physiquement impossible d'explorer l'arbre entier. Le **programme** doit donc se contenter d'une exploration partielle, à profondeur fixée, et utiliser une fonction d'évaluation pour estimer la qualité des positions atteintes à cette profondeur limite.

La fonction d'évaluation : juger une position sans tout calculer

C'est ici que se joue toute la subtilité. Quand le **programme** arrête sa recherche à une profondeur donnée, il atteint des positions qui ne sont pas des fins de partie. Il faut leur attribuer un score numérique pour pouvoir comparer les branches entre elles.

La fonction d'évaluation classique aux échecs combine plusieurs critères. Le plus simple est la valeur matérielle des pièces : la dame vaut neuf points, la tour cinq, le fou et le cavalier trois, le pion un. Si je calcule la somme pour les Blancs et que je soustrais la somme pour les Noirs, j'obtiens un déséquilibre matériel positif si les Blancs ont l'avantage.

Mais le matériel ne suffit pas. Un **programme** sérieux ajoute des bonus pour la position : contrôle du centre, sécurité du roi, mobilité des pièces, structure des pions. Voici une version simplifiée en Python.

Python Copier

```python
VALEURS = {'P': 1, 'C': 3, 'F': 3, 'T': 5, 'D': 9, 'R': 0}

def evaluer(position):
    score = 0
    for case in position.cases():
        piece = position.piece_sur(case)
        if piece is None:
            continue
        v = VALEURS[piece.type]
        if piece.couleur == 'blanc':
            score += v
        else:
            score -= v
    score += bonus_position(position)
    score += bonus_securite_roi(position)
    return score
```

Cette fonction est l'intelligence du **programme**. Elle encode la connaissance humaine du jeu, accumulée pendant des siècles. C'est aussi sa principale faiblesse : la fonction d'évaluation est écrite par des humains, donc elle reflète des intuitions parfois inexactes ou incomplètes. Si la fonction sous-estime la valeur du contrôle du centre, le **programme** jouera systématiquement des coups passifs.

Pour limiter ce problème, les moteurs modernes comme Stockfish utilisent une version optimisée du minimax appelée élagage alpha-bêta. L'idée est de couper certaines branches dont on peut prouver qu'elles ne changeront pas le résultat final. Cette optimisation est purement algorithmique : elle ne change pas la fonction d'évaluation, mais elle permet d'explorer plus profondément en moins de temps. Dans le meilleur des cas, l'élagage divise la **complexité** par la racine carrée, ce qui est énorme.

Sur ma machine personnelle, un **programme** Python naïf peut explorer un arbre à profondeur quatre en quelques secondes. Stockfish, écrit en C++ avec toutes les optimisations possibles, atteint régulièrement des profondeurs de vingt à trente coups en quelques secondes. Ce gain provient à la fois de meilleures fonctions d'évaluation, d'un meilleur élagage et de structures de **données** efficaces comme les bitboards, qui représentent l'échiquier en un seul entier de soixante-quatre bits.

Limites de l'approche minimax et rupture AlphaZero

Tout ce que je viens de décrire repose sur deux hypothèses fortes. Premièrement, on suppose qu'on peut écrire une bonne fonction d'évaluation à la main. Deuxièmement, on suppose qu'on peut explorer suffisamment profond pour que l'évaluation soit fiable. Ces deux hypothèses ont des **limites**.

La fonction d'évaluation humaine est forcément imparfaite. Aucun grand maître ne peut formaliser l'intégralité de son intuition du jeu en une formule. Pendant cinquante ans, les meilleurs **programmes** étaient des suites de petites améliorations sur la même formule de base. Cela a fini par plafonner.

La rupture vient de DeepMind, en deux mille dix-sept, avec **AlphaZero**. Contrairement à Stockfish, AlphaZero n'a aucune fonction d'évaluation écrite par des humains. Le **programme** apprend à jouer en jouant contre lui-même, des millions de fois, et en ajustant un réseau de neurones à chaque partie. Aucune connaissance humaine n'est injectée : seules les règles du jeu sont fournies.

Le résultat est spectaculaire. En quatre **heures** d'entraînement sur du matériel spécialisé, AlphaZero a battu Stockfish vingt-huit fois sur cent, n'a jamais perdu, et a fait nulle dans toutes les autres parties. Le **programme** a redécouvert seul des ouvertures classiques, en a inventé de nouvelles, et a joué dans un style que les commentateurs humains ont qualifié de profondément original.

Sur le plan **informatique**, c'est un changement de paradigme. L'approche minimax est de la **programmation impérative** : le développeur dit explicitement à la machine quoi faire. L'approche AlphaZero est de l'**apprentissage automatique** par renforcement : le développeur fournit un objectif, et la machine découvre seule comment l'atteindre. Ces deux approches sont au **programme** de **NSI**, et les échecs offrent la meilleure illustration du passage de l'une à l'autre.

Pour comprendre cette différence, on peut comparer deux **systèmes**. Stockfish dit : « voici la valeur d'une dame, voici l'importance du centre, voici comment chercher ; cherche jusqu'à profondeur quinze ». AlphaZero dit : « voici les règles ; joue des millions de parties contre toi-même ; tu trouveras ta propre manière d'évaluer les positions ». Les deux fonctionnent, mais ils ne pensent pas pareil.

Conclusion et ouverture

Pour répondre à ma **question** initiale, un **programme** informatique peut prendre des décisions pour jouer aux échecs de deux manières principales. La première, classique, est l'algorithme minimax appuyé sur une fonction d'évaluation écrite par des experts. Cette approche est limitée par la **complexité** exponentielle du jeu et par la qualité de la fonction d'évaluation. La seconde, moderne, est l'apprentissage par renforcement : le **programme** joue contre lui-même et ajuste un réseau de neurones, sans connaissance humaine initiale. Cette approche a démontré, avec AlphaZero, qu'elle pouvait dépasser cinquante ans d'expertise humaine en quatre heures.

L'ouverture la plus naturelle, c'est de se demander si ce changement de paradigme se limite aux jeux. La réponse semble être non : les mêmes techniques ont été appliquées au repliement des protéines avec AlphaFold, prix Nobel de chimie deux mille vingt-quatre, et à d'autres **systèmes** où l'**informatique** doit prendre des décisions complexes sans pouvoir tout énumérer. Les échecs sont peut-être un petit jeu, mais ils sont le laboratoire d'une révolution **informatique** beaucoup plus large.

Je vous remercie pour votre attention et je suis prêt à répondre à vos **questions**.