Grand oral spé NSI : les échecs comme cas d

▶ SommaireSauter à une section

Tu as la spécialité **NSI** en **terminale** et tu cherches un **sujet de Grand Oral** qui te permette de montrer ce que tu sais vraiment faire : coder, raisonner sur la **complexité**, comprendre comment les machines décident. Cet **oral** compte au **coefficient 10** sur l'épreuve finale du **bac** ; c'est l'une des **épreuves** où tu peux le plus creuser l'écart en quelques **minutes**.

Parmi les **idées de sujets Grand Oral** en spécialité **NSI**, les échecs cochent toutes les cases du **programme** de **terminale** : algorithmique, structures de données, bases de données, intelligence artificielle. Face aux **idées de sujets** plus classiques (**sécurité informatique**, **réseaux sociaux**, langages de programmation), c'est un **sujet** rare qui combine code commentable et imaginaire culturel reconnu d'un **jury**.

Cet **article** te donne un kit complet, exploitable directement pour préparer ton **oral** : la méthodologie de l'**épreuve**, un plan minuté de **vingt minutes**, du code Python que tu peux expliquer ligne par ligne, une FAQ jury de quinze **questions** déjà classées par difficulté, une anti-sèche imprimable et des **conseils** de posture pour le **jour** J.

Comprendre l'épreuve : Grand Oral en spécialité NSI

Coefficient, durée, format

Le **Grand Oral du bac** est une **épreuve** terminale d'une durée totale de **20 minutes** (et 20 **minutes** de préparation). Le découpage est précis et le **jury** le respecte au chrono :

Temps	Phase	Ce que tu fais	Ce que le jury évalue
20 min	Préparation	On te donne deux **questions** issues de ton programme. Tu en choisis une. Brouillon autorisé.	,
5 min	Exposé debout	Tu présentes ta **question** debout, sans notes (ou très peu).	**Posture**, voix, structure
10 min	Échange avec le jury	Le **jury** te questionne sur le **sujet**, le **programme**, ta **méthode**.	Maîtrise, réactivité, ouverture
5 min	Projet d'orientation	Tu expliques comment ce **sujet** entre dans ton projet post-bac.	Cohérence parcours

Coefficient au bac général : 10 (sur un total de 100). Sur une moyenne de 12 à l'écrit, gagner 3 points au Grand Oral peut faire basculer une mention. C'est l'**épreuve** la plus rentable au ratio temps de préparation / impact note.

Les attendus officiels

Le **jury** est composé de **deux** professeurs : l'un de ta spécialité (souvent NSI), l'autre d'une discipline différente. Le second n'est **pas** un expert de ton sujet : ton exposé doit être compréhensible par un non-spécialiste. C'est un point que beaucoup de candidats ratent : ils déballent du jargon technique au lieu d'expliquer.

Le rapport officiel du Bulletin officiel (BO) précise que le **jury** évalue :

1. La qualité orale (élocution, **parole** posée, ton convaincu)
2. La maîtrise du **sujet** (capacité à expliquer, à justifier, à nuancer)
3. La construction de l'argumentation (problématique, plan, conclusion)
4. Le lien avec ton projet d'**orientation**

Un **sujet** comme les échecs en NSI passe les quatre filtres : le code se montre, l'argumentation se construit autour de la **complexité** algorithmique, et le lien avec une école d'ingénieur ou un DUT **informatique** est direct.

Ce qui distingue un bon sujet d'un sujet moyen

Un bon **sujet** Grand Oral NSI a trois propriétés que tu peux vérifier en cinq minutes :

- **Il contient du code visible** que tu peux décrire à voix haute (variables, boucles, **fonctionnement**, **questions** sur la **complexité**)
- **Il a un enjeu identifiable** au-delà de la technique (société, économie, éthique, science)
- **Il s'attache à un domaine que le jury reconnaît** (jeux, IA, web, sécurité, données massives)

Les échecs cochent les trois. Tu vas pouvoir montrer du Python concret, parler du basculement IA symbolique → IA connexionniste (enjeu sociétal), et appuyer sur **AlphaZero** que **tout jury** d'informatique connaît.

Pourquoi les échecs sont le bon sujet en spécialité NSI

Les échecs sont l'un des **enjeux** les plus étudiés de toute l'histoire de l'**informatique**. C'est un **sujet** où la **question** « les machines peuvent-elles penser ? » a été posée pour la première fois de manière opérationnelle : pas en philosophie abstraite, mais en lignes de code à exécuter.

Trois faits historiques que le jury appréciera

1. **1950**, Alan Turing écrit le premier programme d'échecs sur papier (faute de machine assez puissante pour l'exécuter). Il invente au passage une grande **partie** du vocabulaire de l'IA moderne.
2. **1950**, Claude Shannon publie *Programming a Computer for Playing Chess*. Il distingue deux stratégies pour jouer aux échecs sur machine : **type A** (force brute, exploration exhaustive) et **type B** (sélection guidée par l'expertise). Cette distinction structure toute l'IA contemporaine.
3. **2017**, DeepMind publie AlphaZero, qui bat Stockfish (alors le moteur le plus fort du monde) après **quatre heures** d'auto-apprentissage. C'est l'incarnation d'un changement de paradigme : du symbolique vers le connexionniste.

Tu vas donc pouvoir construire ton **oral** sur 67 ans d'histoire **informatique** condensée en un seul **exemple**, avec du code Python à montrer pour chaque étape.

Les concepts de spécialité que tu vas mobiliser

Ce **sujet** te permet de toucher **tout** le contenu de la **spécialité** NSI :

- **Algorithmique** : récursivité (minimax), élagage (alpha-bêta), mémoïsation (table de transposition)
- **Structures de données** : arbres n-aires, **systèmes** de tables de hachage, dictionnaires Python
- **Programmation Python** : fonctions récursives, opérateurs bit à bit, **langages** de description
- **Données et bases de données** : format PGN, requêtes SQL, encodage Zobrist
- **Intelligence artificielle** : apprentissage supervisé vs renforcement, réseaux de neurones
- **Complexité** : analyse en O(b^d) puis O(b^(d/2)), comparaison empirique
- **Architecture matérielle** : pourquoi les opérations bit à bit sont efficaces (cycles CPU)

Aucun autre **sujet** ne mobilise autant de **questions** du **programme** en aussi peu de **temps**.

Le piège à éviter

Le risque numéro un d'un **oral** sur les échecs en NSI, c'est de tomber dans le récit pur (Kasparov vs Deep Blue, anecdotes de **partie**). Le **jury** attend du code et de la rigueur **informatique**. Tu peux et dois citer Deep Blue, mais comme point de transition vers l'algorithme alpha-bêta, pas comme histoire à raconter.

Règle d'or : **chaque minute de récit doit être suivie d'une minute de code ou de calcul**.

Autres idées de sujets Grand Oral en spécialité NSI

Pour situer ton choix échecs par rapport aux autres **idées de sujets**, voici un comparatif rapide. Ces alternatives sont toutes valables ; certaines ont leurs propres **limites**.

Sujet	Forces	Limites	Niveau de risque
Sécurité informatique (chiffrement RSA, attaques)	Forte actualité, lié à l'**internet** moderne	Souvent traité, **jury** habitué, niveau math élevé pour RSA	Moyen
Réseaux sociaux et algorithmes de recommandation	Sujet de société fort, **idées** d'enjeux nombreuses	Difficile d'avoir du code à montrer, vite hors-programme	Élevé
Intelligence artificielle générative (ChatGPT, LLM)	Actualité brûlante	Risque de l'exposé non technique (que tu décris l'outil sans expliquer)	Élevé
Langages de programmation comparés	Technique pur	Peu d'enjeu sociétal, peu d'imaginaire	Faible
Systèmes embarqués (IoT, Raspberry Pi)	Très concret si tu as un projet	Demande un projet matériel à montrer	Moyen
Cryptographie quantique	Sujet de pointe, impressionnant	Niveau de difficulté très élevé, peu de candidats le tiennent	Élevé
Algorithmes de tri et **complexité**	Cœur du **programme**	Considéré « scolaire », difficile de **révolutionner** l'angle	Faible
Échecs et IA (notre choix)	Combine algo + structures + IA, imaginaire culturel	Risque du récit pur	Faible si tu suis ce guide
Bases de données géantes (Lichess, GitHub)	Cas concret de big data	Demande à manier SQL et statistiques	Moyen
Compression de données (PNG, MP3)	Math + structure	Sujet technique exigeant	Moyen

Quelle que soit ton **orientation**, garde en tête que le **jury** cherche un **sujet** où tu peux montrer du code, anticiper des **questions** techniques, et **révolutionner** ton angle pour ne pas réciter le cours. Les échecs combinent ces trois critères mieux que la plupart des autres **idées**.

Construire ta problématique : trois angles possibles

Le **jury** n'attend **pas** une dissertation : il attend une **question** précise qui orientera ton exposé. Trois angles fonctionnent particulièrement bien :

Angle 1, Le combat entre deux paradigmes

> ***« Comment les échecs ont-ils permis le passage de l'IA symbolique à l'IA par apprentissage ? »***

Cet angle te fait raconter Deep Blue (1997, symbolique) puis AlphaZero (2017, connexionniste) en t'appuyant sur le code des deux approches. C'est le **sujet** qui **révolutionne** le mieux la perception du **jury** sur ton niveau de réflexion.

Angle 2, La complexité comme contrainte

> ***« Pourquoi 10^120 positions possibles obligent-elles l'informatique à inventer de nouveaux algorithmes ? »***

Cet angle est plus mathématique. Tu démarres sur le nombre de Shannon (10^120 parties possibles aux échecs), tu montres que la force brute est impossible, et tu déroules les solutions : alpha-bêta, **systèmes** de pondération, MCTS. C'est l'angle préféré des candidats matheux.

Angle 3, Du jeu à la science fondamentale

> ***« Les échecs sont-ils un terrain d'expérimentation pour l'intelligence artificielle moderne ? »***

Angle large, orienté **enjeux** et **orientation**. Tu présentes les échecs comme un banc d'essai (AlphaZero a ensuite résolu Go, shogi, puis le repliement de protéines avec AlphaFold). C'est l'angle qui se prolonge le mieux dans la **partie** orientation post-bac.

Recommandation : choisis l'angle 1 si tu veux montrer du code, l'angle 2 si tu veux montrer ta maîtrise mathématique, l'angle 3 si tu vises une école d'ingénieur ou une prépa.

Plan minuté pour 20 minutes d'oral

Voici le plan détaillé que tu peux adapter. Les temps sont indicatifs mais tiennent au chrono.

Phase 1, Introduction (3 minutes)

- **Accroche** (30 s) : « En 1997, Garry Kasparov perd contre Deep Blue, et déclare : "j'ai vu un changement de paradigme". Vingt ans plus tard, AlphaZero apprend à jouer aux échecs en **quatre heures**, sans connaissance humaine. **Question** : que s'est-il passé entre les deux ? »
- **Contexte** (1 min) : rappel rapide de la place des échecs en **informatique** (Turing 1950, Shannon 1950, **systèmes** Deep Blue 1997, AlphaZero 2017).
- **Problématique** (30 s) : annonce claire de ta **question**.
- **Plan annoncé** (1 min) : « Je vais d'abord présenter l'algorithme minimax, puis les structures de données qui le supportent, puis le basculement vers l'apprentissage automatique. »

Phase 2, Algorithmes et complexité (6 minutes)

- **Minimax** (2 min) : présentation du code Python ligne par ligne sur tableau ou diapo.
- **Alpha-bêta** (2 min) : optimisation, coupure expliquée sur un arbre dessiné.
- **Complexité** (1 min) : O(b^d) → O(b^(d/2)), traduction en nombre de positions explorées.
- **Mémoïsation** (1 min) : table de transposition, dictionnaire Python.

Phase 3, Structures et représentation (4 minutes)

- **Tableau 2D vs bitboard** (2 min) : comparaison en termes de **temps** d'accès et d'opérations bit à bit.
- **Format PGN** (1 min) : données structurées, parsage avec regex Python.
- **Hachage Zobrist** (1 min) : XOR pour stocker une position en 64 bits.

Phase 4, IA et apprentissage (5 minutes)

- **Stockfish** (1 min) : approche symbolique, fonction d'évaluation explicite.
- **AlphaZero** (2 min) : réseau de valeur + réseau de politique + MCTS.
- **Apprentissage par renforcement** (1 min) : signal de récompense, auto-jeu.
- **Comparaison** (1 min) : tableau Stockfish vs AlphaZero, 28-0-72 sur 100 parties.

Phase 5, Conclusion (2 minutes)

- **Synthèse** (45 s) : les échecs ont permis à l'**informatique** de passer de l'algorithme explicite à l'apprentissage implicite.
- **Ouverture** (45 s) : transposition à AlphaFold (repliement de protéines), au Go, à d'autres jeux à information complète.
- **Pont vers l'orientation** (30 s) : ce que tu veux faire après le **bac** et pourquoi ce **sujet** y prépare.

Algorithmes et structures de données

L'algorithme minimax : récursivité pure

Le **programme** de **terminale** NSI inclut la récursivité comme concept fondamental. Le **minimax** est l'illustration la plus propre qui existe : une fonction qui s'appelle elle-même, sur un problème naturellement récursif (explorer un arbre de jeu).

Voici la structure en pseudo-code Python que tu peux expliquer ligne par ligne devant le **jury** :

Code (Python) - minimax (récursivité)

```
def minimax(position, profondeur, est_maximiseur):
    # Cas de base : profondeur atteinte ou partie terminée
    if profondeur == 0 or partie_terminee(position):
        return evaluer(position)

    coups = generer_coups_legaux(position)

    if est_maximiseur:
        meilleur = float('-inf')
        for coup in coups:
            nouvelle_pos = appliquer_coup(position, coup)
            valeur = minimax(nouvelle_pos, profondeur - 1, False)
            meilleur = max(meilleur, valeur)
        return meilleur
    else:
        meilleur = float('+inf')
        for coup in coups:
            nouvelle_pos = appliquer_coup(position, coup)
            valeur = minimax(nouvelle_pos, profondeur - 1, True)
            meilleur = min(meilleur, valeur)
        return meilleur
```

Ce code met en œuvre les notions du **programme** :

- **Récursivité** : l'appel `minimax(nouvelle_pos, profondeur - 1, ...)` à l'intérieur de la fonction
- **Cas de base** : la condition `profondeur == 0` qui arrête la récursion
- **Structure d'arbre** : l'exploration des fils d'un nœud avant de retourner la valeur au parent
- **Fonctions d'ordre supérieur** : `max()` et `min()` appliqués à un générateur

La **complexité temporelle** est $O(b^d)$, où b est le facteur de branchement moyen (≈ 35 aux échecs) et d la profondeur. Pour $d = 4$: $35^4 \approx 1{,}5 \times 10^6$ appels récursifs. Pour $d = 6$: $35^6 \approx 1{,}8 \times 10^9$. La croissance exponentielle est la première **question** que le **jury** ouvrira.

L'élagage alpha-bêta : optimisation algorithmique

L'élagage **alpha-bêta** est une optimisation du minimax qui évite d'explorer des branches dont on peut prouver qu'elles ne modifieront **pas** le résultat final. En NSI, c'est un **exemple** d'**algorithme d'élagage** (*pruning*), une technique générale d'optimisation.

L'idée : on maintient deux bornes, **alpha** (le meilleur score que le joueur maximisant peut garantir) et **beta** (le meilleur score que le joueur minimisant peut garantir). Si à un nœud on découvre que la valeur sera forcément pire que ce que l'on peut déjà garantir, on abandonne l'exploration de ce sous-arbre.

Code (Python) - alpha-bêta (élagage)

```
def alpha_beta(position, profondeur, alpha, beta, est_maximiseur):
    if profondeur == 0 or partie_terminee(position):
        return evaluer(position)

    coups = generer_coups_legaux(position)

    if est_maximiseur:
        for coup in coups:
            valeur = alpha_beta(appliquer_coup(position, coup),
                                profondeur - 1, alpha, beta, False)
            alpha = max(alpha, valeur)
            if beta <= alpha:
                break  # Coupure beta : inutile d'explorer plus
        return alpha
    else:
        for coup in coups:
            valeur = alpha_beta(appliquer_coup(position, coup),
                                profondeur - 1, alpha, beta, True)
            beta = min(beta, valeur)
            if beta <= alpha:
                break  # Coupure alpha : inutile d'explorer plus
        return beta
```

La ligne `break`, la **coupure**, est le cœur de l'optimisation. Dans le meilleur cas (coups ordonnés), l'alpha-bêta réduit la **complexité** à $O(b^{d/2})$: on explore environ la racine carrée du nombre de nœuds du minimax naïf. Pour $d = 6$, on passe d'environ $1{,}8 \times 10^9$ à $\sim 4{,}2 \times 10^4$ nœuds.

Pour la présentation devant le **jury**, dessine un petit arbre à 3 niveaux et montre en direct quelle branche est coupée et pourquoi. C'est l'**exemple** où le **jury** posera le plus de **questions**.

Les arbres comme structure de données

Le **programme** inclut les **arbres binaires** : définition, parcours (préfixe, infixe, suffixe), hauteur. L'arbre des coups aux échecs est un **arbre n-aire** (chaque nœud a jusqu'à 35 fils), mais le même vocabulaire s'applique.

Points importants pour le **jury** :

- La **racine** est la position initiale
- Les **nœuds internes** sont les positions en cours de partie
- Les **feuilles** sont les positions finales (mat, nulle, abandon) ou les positions à la profondeur maximale d'exploration
- La **hauteur** de l'arbre est le nombre de coups restant à explorer
- Un **parcours en profondeur** (DFS) est ce que le minimax effectue naturellement via la pile d'appels récursifs

La différence avec un arbre binaire de recherche : dans un arbre des coups, on ne cherche **pas** une valeur, on **propage des scores** de bas en haut. C'est ce qu'on appelle la **rétropropagation des valeurs**, un concept qui réapparaît dans les réseaux de neurones (backpropagation).

Programmation Python : représenter l'échiquier

Tableau 2D : la représentation naïve

La représentation la plus directe d'un échiquier en Python est un tableau bidimensionnel 8×8 :

Code (Python) - échiquier en tableau 2D

```
# Représentation de l'échiquier en début de partie
# P=pion, T=tour, C=cavalier, F=fou, D=dame, R=roi
# Majuscule = blanc, minuscule = noir

echiquier = [
    ['t', 'c', 'f', 'd', 'r', 'f', 'c', 't'],  # rangée 8 (noirs)
    ['p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'],  # rangée 7
    ['.', '.', '.', '.', '.', '.', '.', '.'],   # rangée 6
    ['.', '.', '.', '.', '.', '.', '.', '.'],   # rangée 5
    ['.', '.', '.', '.', '.', '.', '.', '.'],   # rangée 4
    ['.', '.', '.', '.', '.', '.', '.', '.'],   # rangée 3
    ['P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'],  # rangée 2 (blancs)
    ['T', 'C', 'F', 'D', 'R', 'F', 'C', 'T'],  # rangée 1
]

# Accès à la case e4 (colonne 4, rangée 3 dans notre index 0-7)
def get_piece(echiquier, col, rang):
    return echiquier[7 - rang][col]
```

Cette représentation est simple à comprendre et à présenter. Elle permet d'illustrer les notions de **tableau 2D**, d'**indexation** et de **slicing** Python. Son inconvénient : les opérations sur les cases (vérifier si une pièce est attaquée, générer les coups légaux) nécessitent des boucles imbriquées, ce qui est lent pour un vrai moteur.

Bitboard : la représentation efficace

La représentation **bitboard** est utilisée dans **tout** moteur professionnel (Stockfish, Leela Chess Zero). Elle illustre directement le **programme** sur la **représentation binaire** et les **opérateurs logiques bit à bit**.

Principe : chaque type de pièce (pions blancs, tours noires, etc.) est représenté par un **entier de 64 bits**. Le bit *i* vaut 1 si la pièce occupe la case *i* (cases numérotées de 0 à 63, de a1 à h8).

Code (Python) - bitboard (opérations bit à bit)

```
# En Python, les entiers ont précision arbitraire
# On peut simuler un bitboard à 64 bits

# Position initiale : pions blancs sur les cases 8 à 15
# Cases 8-15 = bits 8 à 15 activés
pions_blancs = 0xFF00  # 65280 en décimal

# Vérifier si un pion blanc est sur la case e2 (case 12 dans notre numérotation)
case_e2 = 1 << 12  # 0b0001000000000000
pion_sur_e2 = (pions_blancs & case_e2) != 0
print(pion_sur_e2)  # True

# Déplacer un pion de e2 à e4 (case 12 → case 28)
pions_blancs = (pions_blancs & ~case_e2) | (1 << 28)
```

Ce code mobilise les **opérateurs bit à bit** Python : `&` (ET), `|` (OU), `~` (NON), `<<` (décalage gauche). Ces opérateurs sont au **programme** et sont ici directement utiles. Un moteur peut calculer tous les coups légaux d'une pièce avec quelques opérations bit à bit en microsecondes, alors qu'un tableau 2D nécessite des boucles.

Intelligence artificielle : de Stockfish à AlphaZero

Stockfish : l'approche classique symbolique

Stockfish est un moteur open source, l'un des plus forts au monde. Il repose sur l'algorithme alpha-bêta avec des optimisations massives :

- **Table de transposition** : une table de hachage qui stocke les positions déjà évaluées pour éviter de les recalculer (mémoïsation)
- **Iterative deepening** : on explore d'abord jusqu'à profondeur 1, puis 2, puis 3… pour obtenir rapidement une bonne réponse même si le **temps** vient à manquer
- **Move ordering** : les coups sont triés avant exploration (captures d'abord) pour maximiser les coupures alpha-bêta

La **table de transposition** est un **exemple** direct de structure de données clé-valeur (dictionnaire en Python) utilisée pour la mémoïsation :

Code (Python) - table de transposition (mémoïsation)

```
# Table de transposition simplifiée
transposition_table = {}

def minimax_memo(position, profondeur, est_max):
    # Hasher la position pour en faire une clé
    cle = (hash(str(position)), profondeur, est_max)

    if cle in transposition_table:
        return transposition_table[cle]  # Résultat déjà calculé

    # ... calcul normal du minimax ...
    resultat = ...

    transposition_table[cle] = resultat  # Stocker pour réutilisation
    return resultat
```

Le concept est celui de la **programmation dynamique** : plutôt que de recalculer des sous-problèmes identiques, on stocke leurs solutions. Les échecs produisent des positions identiques par des séquences de coups différentes (*transpositions*), ce qui rend la mémoïsation particulièrement efficace.

AlphaZero : l'approche par apprentissage

En 2017, DeepMind a publié AlphaZero, un programme radicalement différent de Stockfish. Pour NSI, la distinction fondamentale est la suivante :

	Stockfish	**AlphaZero**
Approche	Programmation explicite	Apprentissage par renforcement
Évaluation	Formule codée par des experts	Réseau de neurones appris
Connaissance humaine	Oui (ouvertures, stratégie)	**Non** (règles uniquement)
Entraînement	Aucun	**4 heures** d'auto-jeu
Résultat vs Stockfish	/	28 victoires, 0 défaites (100 parties, 2017)
Positions / seconde	200 millions	~60 000

AlphaZero utilise deux réseaux de neurones simultanément :

1. **Le réseau de valeur** (*value network*) : donne une évaluation numérique de la position
2. **Le réseau de politique** (*policy network*) : donne une distribution de probabilité sur les coups légaux

Ces deux réseaux guident la **Monte Carlo Tree Search (MCTS)** : au lieu d'explorer l'arbre de manière uniforme, MCTS concentre les ressources sur les coups que le réseau de politique juge prometteurs, et corrige cette estimation grâce aux résultats des simulations.

Pour NSI, l'argument clé est celui du **paradigme** : Stockfish dit *explicitement* à la machine ce qui est une bonne position. AlphaZero *apprend* ce qu'est une bonne position. Le premier paradigme est celui de la **programmation impérative**, le second celui de **l'apprentissage automatique**: tous deux au **programme**.

Machine learning vs algorithmes classiques

Les échecs permettent d'illustrer la distinction entre les grandes familles d'approches :

Apprentissage supervisé : on fournit des millions de positions annotées (position → meilleur coup), et le réseau apprend à reproduire ces annotations. C'est comme apprendre à résoudre des exercices à partir de corrigés.

Apprentissage par renforcement (AlphaZero) : on ne fournit que les règles du jeu et un signal de récompense (gagner = +1, perdre = -1). Le programme joue contre lui-même, améliore sa stratégie en fonction des résultats, sans jamais voir de partie humaine.

Cette distinction est l'une des **questions** les plus probables du **jury** sur un **sujet** IA/NSI.

Données et bases de données

Le format PGN : données structurées

Le **PGN** (*Portable Game Notation*) est le format standard de stockage des parties d'échecs. Un fichier PGN contient des **métadonnées structurées** entre crochets, puis la liste des coups :

Exemple (PGN) - fichier texte structuré

```
[Event "World Chess Championship 1997"]
[Date "1997.05.03"]
[White "Deep Blue"]
[Black "Kasparov, G"]
[Result "1-0"]
[WhiteElo "?"]
[BlackElo "2795"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6
8. c3 O-O 9. h3 Nb8 10. d4 Nbd7 ... 1-0
```

Pour NSI, ce format illustre plusieurs notions :

- **Format de données structurées** : les métadonnées respectent un schéma prévisible
- **Parsage** : on peut extraire les informations avec des expressions régulières
- **Encodage** : la notation des coups (e4, Nf3, O-O) est un **langage** formel avec sa propre grammaire

Un **parseur PGN** simple en Python :

Code (Python) - parser PGN (regex)

```
import re

def extraire_metadata(pgn_text):
    """Extrait les métadonnées d'une partie PGN."""
    pattern = r'\[(\w+)\s+"([^"]+)"\]'
    return dict(re.findall(pattern, pgn_text))

pgn = '[Event "Test"][Date "2026.04.28"][White "Alice"][Black "Bob"]'
metadata = extraire_metadata(pgn)
# {'Event' : 'Test', 'Date' : '2026.04.28', 'White' : 'Alice', 'Black' : 'Bob'}
```

SQL sur une base de parties

Lichess publie une **base de données publique** de plus de 3 milliards de parties au format PGN. Cette base est un **exemple** parfait de **grandes données** (*big data*) pour NSI :

```
-- Schéma simplifié d'une base de parties d'échecs
CREATE TABLE parties (
    id INTEGER PRIMARY KEY,
    date DATE,
    joueur_blanc VARCHAR(50),
    joueur_noir VARCHAR(50),
    elo_blanc INTEGER,
    elo_noir INTEGER,
    ouverture VARCHAR(10),
    resultat CHAR(3),
    nb_coups INTEGER
);

-- Requête : taux de victoire des Blancs par ouverture
SELECT
    ouverture,
    COUNT(*) AS nb_parties,
    ROUND(100.0 * SUM(CASE WHEN resultat = '1-0' THEN 1 ELSE 0 END) / COUNT(*), 1) AS
pct_blanc
FROM parties
GROUP BY ouverture
ORDER BY nb_parties DESC
LIMIT 10;
```

Cette requête SQL illustre les notions de **bases de données relationnelles**, d'**agrégation**, de **filtrage**, et de **tri**.

Représentation numérique et compression

Pour stocker efficacement une position, les moteurs utilisent le **hachage de Zobrist** : on associe à chaque (pièce, case) un entier aléatoire de 64 bits, et la valeur de hachage de la position entière est le XOR de tous ces entiers.

Code (Python) - hachage de Zobrist (XOR)

```
import random

ZOBRIST_TABLE = {}
pieces = ['P', 'p', 'T', 't', 'C', 'c', 'F', 'f', 'D', 'd', 'R', 'r']
for piece in pieces:
    for case in range(64):
        ZOBRIST_TABLE[(piece, case)] = random.getrandbits(64)

def calculer_hash(echiquier_liste):
    """Calcule le hash Zobrist d'une position."""
    h = 0
    for case, piece in enumerate(echiquier_liste):
        if piece != '.':
            h ^= ZOBRIST_TABLE[(piece, case)]
    return h
```

Ce code illustre les notions d'**opérateurs XOR**, de **tables de hachage**, et de **collision de hachage**.

Anti-sèche imprimable : à plier dans ta poche

Trois chiffres à retenir absolument :

- 35 : facteur de branchement moyen aux échecs
- 10^120 : nombre de parties possibles (nombre de Shannon)
- 28-0-72 : score d'AlphaZero vs Stockfish (2017, 100 parties)

Trois dates :

- 1950 : Turing et Shannon, fondations algorithmiques
- 1997 : Deep Blue bat Kasparov (symbolique, calcul brut)
- 2017 : AlphaZero bat Stockfish en 4h (connexionniste, apprentissage)

Trois complexités :

- O(b^d) : minimax pur
- O(b^(d/2)) : alpha-bêta dans le meilleur cas
- O(1) en moyenne : accès table de transposition

Trois noms : Turing, Shannon, Silver (DeepMind).

Trois mots-clés : récursivité, mémoïsation, renforcement.

Conseils pour réussir le jour J

Posture et voix

Tu es debout pendant les cinq premières **minutes** de ton exposé. La **posture** compte plus que tu ne crois : pieds ancrés à largeur d'épaules, mains visibles (**pas** dans les poches), regard distribué entre les **deux** membres du **jury**.

Évite la **parole** trop rapide : l'angoisse fait toujours accélérer. Repère un fait technique précis (« le facteur de branchement est de 35 ») et utilise-le comme balise pour ralentir. Si tu sens que tu vas trop vite, marque une pause d'**une heure** humaine (une seconde réelle) après chaque chiffre clé.

Gestion du stress avant l'épreuve

Le **stress** d'avant **oral** est normal. Trois techniques rapides :

1. **Respiration carrée** : 4 secondes inspire, 4 secondes pause, 4 secondes expire, 4 secondes pause. Trois cycles, juste avant d'entrer.
2. **Ancrage corporel** : sens tes pieds dans le sol, ferme les poings 5 secondes puis relâche. Reconnecte au corps.
3. **Réécoute mentale de ton accroche** : la première phrase doit être automatique. Si tu démarres sans hésiter, le reste suit.

Ce qu'il faut absolument éviter

- **Lire ses notes** pendant l'exposé : disqualifiant. Tu peux les avoir mais à peine les regarder.
- **Répondre « je ne sais pas »** sans rien proposer. Préfère « je ne suis pas certain, mais je dirais que… ». Le **jury** valorise la tentative.
- **Critiquer le sujet** ou dire que tu n'aimes pas ta spé. Tu as choisi ce **sujet**, défends-le.
- **Dépasser le temps** : à 5 minutes pile, tu dois conclure même si tu n'as **pas** fini. Le **jury** coupera sinon.

Lien avec ton projet d'orientation

Les cinq dernières **minutes** portent sur ton projet post-bac. Prépare une **réponse** courte et cohérente :

- Si tu vises une **école d'ingénieur** : « Ce **sujet** m'a fait découvrir l'algorithmique avancée, et c'est ce que je veux approfondir en classe préparatoire / INSA / UTC. »
- Si tu vises un **DUT informatique** ou **BUT** : « J'ai aimé travailler le code concret, et je veux continuer dans un cursus qui privilégie la pratique. »
- Si tu vises une licence **maths-info** : « La modélisation mathématique de la **complexité** m'a intéressé, je veux la creuser en université. »

Conseil : ne mens **pas** sur ton projet. Le **jury** sent l'incohérence. Préfère honnêteté sur tes doutes ; un projet en construction est mieux accepté qu'un projet inventé.

Le jour J : checklist

La veille au soir :

- [ ] Relire l'anti-sèche imprimable (cinq minutes max, **pas** plus)
- [ ] Préparer la tenue (pantalon + chemise / chemisier ; **pas** de jean troué)
- [ ] Vérifier l'heure et l'adresse de la convocation
- [ ] Se coucher tôt (le **stress** dérègle déjà le sommeil)

Le matin :

- [ ] Manger normalement (**pas** plus, **pas** moins ; le ventre vide accentue le **stress**)
- [ ] Arriver 30 minutes en avance
- [ ] Eau dans la salle d'attente, **pas** de café à jeun

Pendant l'épreuve :

- [ ] Phase de préparation : 20 minutes pour choisir et organiser. Ne perds **pas** plus de 2 minutes à choisir.
- [ ] Exposé : 5 minutes debout, voix posée, regard distribué.
- [ ] Échange : 10 minutes. Si tu ne comprends **pas** une **question**, fais-la reformuler.
- [ ] **Orientation** : 5 minutes. Reste cohérent avec ton dossier Parcoursup.

Après :

- Aucune relance utile. Le **jury** délibère, tu n'as plus rien à faire.

Sources et références

- **Shannon, C. E. (1950).** *Programming a Computer for Playing Chess.* *Philosophical Magazine*, 41(314). (Fondation de l'algorithme minimax appliqué aux échecs.)
- **Silver, D., et al. (DeepMind, 2018).** *A general reinforcement learning algorithm that masters chess, shogi, and Go.* *Science*, 362(6419). (AlphaZero : apprentissage par renforcement et MCTS.)
- **Knuth, D. & Moore, R. (1975).** *An Analysis of Alpha-Beta Pruning.* *Artificial Intelligence*, 6(4), 293-326. (Analyse formelle de la complexité de l'élagage alpha-bêta.)
- **Zobrist, A. L. (1970).** *A New Hashing Method with Application for Game Playing*. ICCA Journal. (Invention du hachage de Zobrist pour les tables de transposition.)
- **Bulletin officiel, Note de service 2020-014.** Modalités du Grand Oral au baccalauréat général. (Cadre réglementaire de l'épreuve.)
- **Documentation Stockfish.** *Stockfish Chess Engine : Source code.* GitHub. (Code source du moteur open source, référence pour l'implémentation réelle.)
- **Lichess Open Database.** *lichess.org/database*. (Base de données publique de parties d'échecs au format PGN.)

- **Sadler, M. & Regan, N. (2019).** *Game Changer : AlphaZero's Groundbreaking Chess Strategies.* New In Chess. (Analyse approfondie du style d'AlphaZero, accessible aux lycéens.)

À retenir

- Le minimax est le cas concret de récursivité le plus pur qui soit, codable en 10 lignes Python
- Un échiquier peut être représenté comme un tableau 2D (simple) ou un entier 64 bits (bitboard, avancé)
- Le format PGN illustre directement la notion de données structurées parsables
- L'élagage alpha-bêta réduit O(b^d) à O(b^(d/2)) : argument de **complexité** idéal pour le **jury**
- AlphaZero vs Stockfish = apprentissage automatique vs programmation explicite : la distinction centrale de l'IA au **programme**
- Le **Grand Oral** se gagne autant sur la maîtrise que sur la **posture**, la **parole** et la **qualité** de l'échange avec le **jury**

Après lecture : enregistre-toi en **une prise** sur le minimax (tableau blanc ou IDE, **trois minutes max**) ; si tu bloques sur une ligne de code, c'est là que le **jury** creusera.

Questions fréquentes

Pourquoi les échecs sont-ils un sujet idéal pour un Grand Oral spécialité NSI ?

Les échecs incarnent les quatre piliers du programme de spé NSI : les algorithmes (minimax, alpha-bêta, récursivité), les structures de données (arbres de jeu, graphes), la programmation (représentation de l'échiquier en Python, fonctions récursives), et l'intelligence artificielle (passage de Stockfish à AlphaZero). C'est l'un des rares sujets qui permet de montrer concrètement le lien entre la théorie algorithmique et une application réelle connue du jury.

Comment coder un minimax en Python pour un Grand Oral NSI ?

La structure de base du minimax en Python est une fonction récursive : def minimax(pos, depth, is_max): if depth == 0: return evaluer(pos) ; coups = generer_coups(pos) ; if is_max: return max(minimax(appliquer(pos,c), depth-1, False) for c in coups) ; else: return min(minimax(appliquer(pos,c), depth-1, True) for c in coups). Cette structure met en œuvre la récursivité, la gestion de l'arbre et les fonctions d'ordre supérieur vus en terminale NSI.

Qu'est-ce que le format PGN et comment l'utiliser en NSI ?

Le PGN (Portable Game Notation) est le format standard de stockage des parties d'échecs. C'est un fichier texte structuré : des métadonnées entre crochets ([Event 'Partie test'][Date '2026.04.28']) suivies de la liste des coups (1. e4 e5 2. Nf3 Nc6...). En NSI, il illustre la notion de format de données structurées, parsable en Python avec des expressions régulières ou une bibliothèque dédiée. Il peut aussi être stocké et interrogé dans une base de données SQL.

Qu'est-ce qu'un bitboard aux échecs et pourquoi est-ce important en NSI ?

Un bitboard représente l'état d'un type de pièce sur l'échiquier avec un entier de 64 bits. Chaque bit correspond à une case : le bit i vaut 1 si la pièce occupe la case i, 0 sinon. Par exemple, les pions blancs en position initiale occupent les cases 8 à 15, soit l'entier 0xFF00 en hexadécimal. Les opérations sur les bitboards (ET, OU, décalage de bits) sont des opérations entières très rapides.

C'est une application directe du programme NSI sur la représentation binaire et les opérateurs logiques.

Quelle différence entre Stockfish et AlphaZero pour un Grand Oral NSI ?

Stockfish utilise un algorithme minimax avec élagage alpha-bêta et une fonction d'évaluation programmée par des experts humains (approche symbolique, 'top-down'). AlphaZero utilise un réseau de neurones profond entraîné uniquement par auto-apprentissage, sans connaissance humaine initiale (approche connexionniste, 'bottom-up'). Pour NSI, la différence clé est celle entre la programmation explicite (règles codées) et l'apprentissage automatique (règles découvertes par l'expérience). AlphaZero a battu Stockfish en moins de 24 heures d'entraînement en 2017.

Quelles questions le jury peut-il poser sur les échecs en spécialité NSI ?

Les cinq questions les plus probables sont : (1) Quelle est la complexité temporelle du minimax ? (O(b^d), exponentielle). (2) Comment l'élagage alpha-bêta réduit-il cette complexité ? (O(b^(d/2)) dans le meilleur cas). (3) Qu'est-ce qu'une table de transposition et pourquoi est-elle utile ? (Mémoïsation des positions déjà calculées). (4) Pourquoi utilise-t-on des bitboards plutôt qu'un tableau 2D en Python pour un moteur sérieux ? (Vitesse des opérations bit à bit vs accès tableau). (5) Quelle est la différence entre apprentissage supervisé et apprentissage par renforcement dans le contexte d'AlphaZero ?