

Rupture de cycles : Analyse de la finitude sur un échiquier réduit

ERRAHIMI Aya

Introduction :

Le paradoxe de l'infini : Du plateau fini au système divergent

Le système physique vs la réalité combinatoire

- **Espace fini** : 64 cases et 32 pièces.
- **Explosion combinatoire** : Nombre de Shannon $\approx 10^{120}$ positions (*J-P. Delahaye*).



Figure – échiquier 8x8

Le risque de divergence

Absence de règle de terminaison $\Rightarrow$ Trajectoires infinies.

- Formalisme de Berlekamp : « **Loopy Game** ».

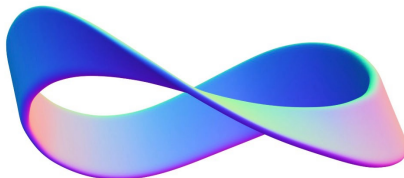


Figure – Représentation topologique d'un Loopy Game

Problématique :

Comment modéliser un échiquier sous forme de graphe pour prouver que la règle de répétition suffit à briser les boucles infinies ?

Plan de travail

① Modélisation Algébrique

- Du micro-échiquier au graphe d'états.
- Matrice d'adjacence et détection de cycles.

② Preuve Mathématique

- Présentation du Principe de Dirichlet.

③ Validation Informatique

- Algorithme de parcours (DFS).
- Confrontation entre la théorie et la simulation.

Modèle mathématique : Le mini-échiquier

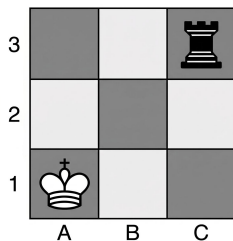


Figure – État initial sur grille 3x3

Espace d'états S

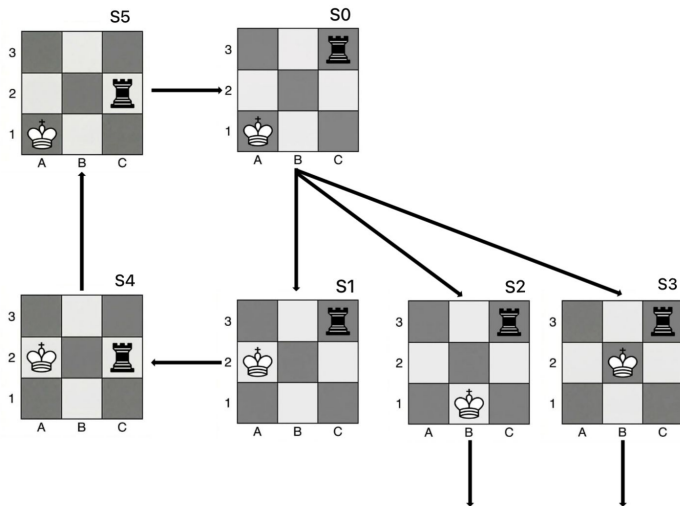
Ensemble fini de toutes les configurations légales possibles.

Définition d'un état $s \in S$

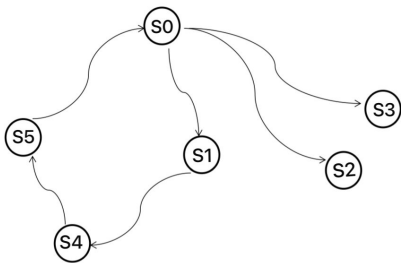
Chaque état est rigoureusement défini par un triplet :

- P_B : Position du Roi Blanc
- P_N : Position de la Tour Noire
- $t \in \{B, N\}$: Le trait

Mise en évidence d'une trajectoire cyclique :



Modélisation topologique : Le graphe d'états orienté



Modélisation topologique

Soit le graphe orienté fini
 $G = (V, E)$ où :

- **V (Sommets)** : Ensemble des états admissibles du jeu, définis par le triplet $s = (P_B, P_N, t)$.
- **E (Arêtes)** : Ensemble des transitions légales orientées $S_i \rightarrow S_j$.

Propriété fondamentale

Une répétition de configuration physique équivaut algébriquement à l'existence d'un **cycle** au sein du graphe G .

Modélisation Algébrique : La Matrice d'Adjacence A

1. Cardinalité et espace matriciel

L'espace d'états de notre échiquier réduit 3×3 est strictement fini :

$$N = 9 \text{ (Roi)} \times 8 \text{ (Tour)} \times 2 \text{ (Trait)} = \mathbf{144} \text{ configurations}$$

$$\text{Ainsi } A \in \mathcal{M}_{144}(\{0, 1\}).$$

2. Règle de construction

$$A_{i,j} = \begin{cases} 1 & \text{si la transition de l'état } S_j \text{ vers l'état } S_i \text{ est légal} \\ 0 & \text{sinon (mouvement impossible)} \end{cases}$$

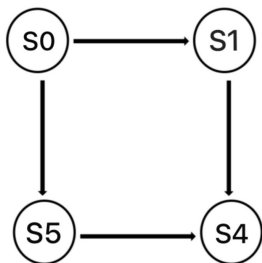


Figure – cycle détecté

3. Zoom sur le cycle détecté

Dans la base (S_0, S_1, S_4, S_5) , la sous-matrice de notre boucle infinie

$(S_0 \rightarrow S_1 \rightarrow S_4 \rightarrow S_5 \rightarrow S_0)$
s'écrit :

$$A_{cycle} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Détection Algébrique :

Principe : Simuler l'avancée du jeu

Résultat classique :

$$(A^k)_{i,j} \iff \text{Chemins de longueur } k$$

Application du résultat sur la boucle (S_0, S_1, S_4, S_5) :

$$(A_{cycle})^4 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}^4 = \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \end{pmatrix} = \mathbf{I}_4$$

Conclusion par l'indicateur Trace

Le retour systématique à l'état initial se lit sur la diagonale principale :

$$\text{Tr}(\mathbf{A}^4) = 4$$

Critère : Puisque $\text{Tr}(\mathbf{A}^4) > 0$, l'existence mathématique d'un cycle de longueur 4 est formellement démontrée.

Résolution Théorique : Le Principe de Dirichlet

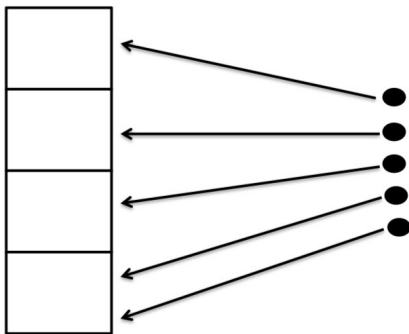


Figure - Principe des tiroirs

Application du principe de Dirichlet :

Formalisation du principe :

$n = 144$ (états possibles : les tiroirs)

$m = 145$ (positions visitées : les objets)

Application à la trajectoire

$m > n \implies$ **Répétition forcée**

D'où l'existence d'un cycle

Résolution Théorique : Mécanisme de la Rupture

1. Périodicité de la trajectoire

Notons $(s_k)_{k \in \mathbb{N}}$ la suite des états visités. Le principe de Dirichlet implique que cette suite est ultimement périodique. Il existe un rang d'entrée dans le cycle k_0 et une période $T \geq 1$ tels que :

$$\forall k \geq k_0, \quad s_{k+T} = s_k$$

2. Atteinte du seuil critique : Règle des trois répétitions

La règle de la FIDE impose l'arrêt à la troisième occurrence d'un même état.

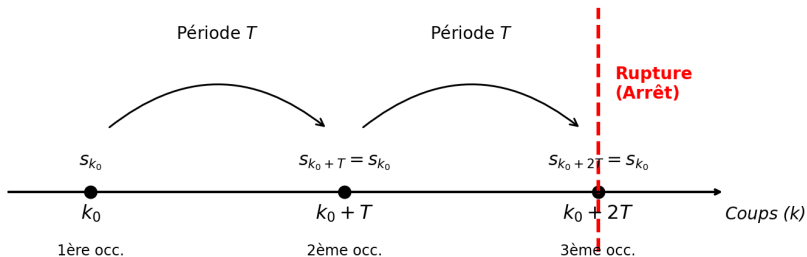


Figure – Troncature de la trajectoire au seuil critique $N = k_0 + 2T$

3. Finitude stricte

La règle agit comme une condition de troncature stricte sur la suite (s_k) . La partie s'achève **au plus tard** au coup $N = k_0 + 2T$, démontrant formellement l'impossibilité d'une trajectoire de jeu infinie.

Comparaison des règles d'arrêt

La FIDE dispose de plusieurs règles de terminaison. Nous comparons ici leur efficacité comme opérateurs de troncature :

Critère	Règle des 3 répétitions	Règle des 50 coups
Condition	Même état visité 3 fois	50 coups sans capture ni pion
Borne sur notre modèle	$N = k_0 + 2T$	$N \leq 50$
Type de détection	Topologique (cycle dans G)	Compteur linéaire
Sensibilité au graphe	Oui — dépend de la structure des cycles	Non — indépendante
Conclusion	Brise les cycles par définition	Brise les cycles par épuisement

Architecture générale du programme Python

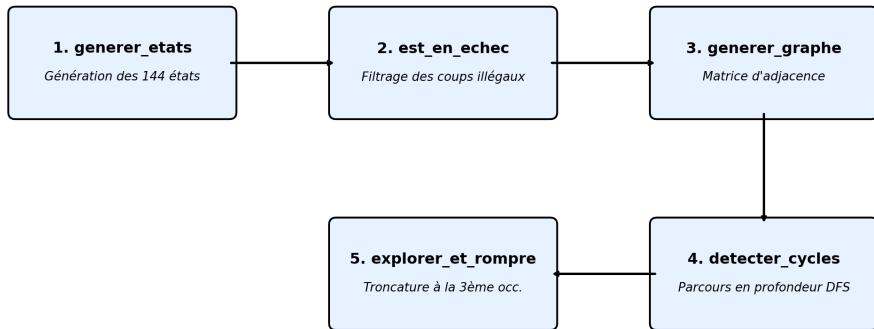


Figure – organigramme du programme

Résultats de la simulation globale

Exécution autonome de l'algorithme sur l'espace d'états :

```
> --- DÉBUT DE LA SIMULATION ---  
> Graphe généré avec 144 configurations  
admissibles.  
> Lancement depuis l'état : (1, 1, 3, 3, 1)  
> --- FIN DE LA SIMULATION ---  
> Cycles détectés           : 179  
> Longueur minimale        : 4  
> Longueur maximale        : 116  
> Conclusion : finitude du système prouvée  
informatiquement.
```

⇒ *La simulation confirme la théorie : la condition d'arrêt brise les cycles et empêche le jeu de tourner à l'infini.*

Grandeur	Théorie	Simulation
Nombre d'états	144	144
Longueur minimale de cycle	≥ 1	4
Longueur maximale de cycle	≤ 144	116
Finitude garantie	$N = k_0 + 2T$	Confirmée

Convergence théorie / simulation

Les résultats expérimentaux sont cohérents avec la borne théorique $N = k_0 + 2T$: la longueur maximale détectée de 116 coups est bien inférieure à la borne absolue de 144 états admissibles.

Bilan et conclusion

1. Modélisation

Graphe orienté
 $n = 144$ états

2. Théorie

Principe des tiroirs
Règle des 3
répétitions

3. Simulation

Algorithme DFS
Finitude vérifiée

Conclusion

La règle de triple répétition agit comme un **opérateur topologique strict**, garantissant la finitude absolue des trajectoires du système.

Limites et Perspectives

Limites du modèle

- **Modèle conservatif** : Exclusion des captures, du roque et des prises en passant.
- **Complexité** : Vulnérabilité face à l'explosion combinatoire.

Perspective

Déterminer une règle d'arrêt plus optimale que la triple répétition pour briser les cycles en un minimum de coups.

Merci pour votre attention

Annexe-A : Théorème de la matrice d'adjacence

Initialisation (pour $k = 1$)

La matrice $A^1 = A$ est la matrice d'adjacence du graphe. Par définition, le coefficient $A_{i,j}$ représente le nombre d'arêtes allant directement le sommet j au sommet i . Il s'agit donc bien du nombre de chemins de longueur 1 de j vers i . La propriété est vraie au rang $k = 1$.

Hypothèse de récurrence : Soit $A \in \mathcal{M}_n(\{0, 1\})$ et $k \geq 1$ entier. On suppose que le coefficient $(A^k)_{i,l}$ représente donc le nombre de chemins de longueur k allant du sommet i au sommet l . Montrons qu'elle est vraie au rang $k + 1$.

Hérédité : Montrons qu'elle est vraie au rang $k + 1$. On a : $A^{k+1} = A^k \times A$. En utilisant la formule du produit matriciel, on a donc :

$$(A^{k+1})_{i,j} = \sum_{l=1}^n (A^k)_{i,l} \times A_{l,j}$$

Or $A_{\ell j} = 1$ si et seulement si il existe une arête de j vers ℓ . Et par hypothèse de récurrence, $(A^k)_{i\ell}$ compte les chemins de longueur k de ℓ vers i .

Donc chaque terme non nul de la somme correspond à un chemin de longueur $k + 1$ de j vers i passant par ℓ :

$$j \xrightarrow{1 \text{ arête}} \ell \xrightarrow{k \text{ arêtes}} i$$

En sommant sur tous les sommets intermédiaires ℓ possibles, on obtient bien le nombre total de chemins de longueur $k + 1$ de j vers i .

Conclusion : Par le principe de récurrence, pour tout $k \geq 1$:

$$(A^k)_{ij} = \text{nombre de chemins de longueur } k \text{ de } j \text{ vers } i$$

Annexe-B : Justification du critère Trace

Un cycle de longueur k est un chemin fermé, c'est-à-dire un chemin qui part d'un sommet i et y revient. Ces chemins correspondent exactement aux coefficients diagonaux $(A^k)_{ii}$. Or la Trace est par définition la somme des coefficients diagonaux :

$$\text{Tr}(A^k) = \sum_{i=1}^n (A^k)_{ii}$$

Donc :

$$\begin{aligned} \text{Tr}(A^k) > 0 &\iff \exists i \text{ tel que } (A^k)_{ii} \geq 1 \\ &\iff \text{il existe un cycle de longueur } k \end{aligned}$$

Annexe-C : Principe des tiroirs de Dirichlet

Théorème : Principe des tiroirs de Dirichlet

Soient E et F deux ensembles finis tels que $\text{card}(E) > \text{card}(F)$.
Alors toute fonction $f : E \rightarrow F$ n'est pas injective, c'est-à-dire :

$$\exists (x, y) \in E^2, \quad x \neq y \text{ et } f(x) = f(y)$$

Hypothèse : Supposons que $f : E \rightarrow F$ est injective.

Donc :

$$\text{card}(E) \leq \text{card}(F)$$

Or, par hypothèse :

$$\text{card}(E) > \text{card}(F). \textbf{ABSURDE}$$

Conclusion : f ne peut pas être injective. Il existe donc nécessairement deux éléments distincts $x \neq y$ de E tels que $f(x) = f(y)$. $\square$

Annexe-D : Périodicité de la suite des états

Soit une suite (s_k) dans S , ensemble fini de cardinal n .

Considérons les $n+1$ premiers termes : $s_0, s_1, s_2, \dots, s_n$

Par le principe de Dirichlet (Annexe B) :

$$\exists (i, j) \in \{0, \dots, n\}^2, i < j \text{ tel que : } s_i = s_j$$

Puisque $s_i = s_j$, ces deux états sont identiques dans notre graphe.

Ils possèdent donc les mêmes transitions légales sortantes.

Ainsi : $s_i = s_j \implies s_{i+1} = s_{j+1}$

En appliquant ce raisonnement successivement, on obtient :

$$\text{pour tout } p \geq 0 : s_{i+p} = s_{j+p}$$

On pose $T = j - i \geq 1$. Alors pour tout $k \geq i$ et pour $p = k - i$:

$$s_{i+(k-i)} = s_{j+(k-i)} = s_k = s_{k+(j-i)} = s_{k+T}$$

La suite (s_k) est donc périodique à partir du rang $k_0 = i$, de période T . $\square$

Annexe-E1 : Génération de l'espace d'états et filtrage

```
1 def generer_etats_admissibles() -> list:
2     "Genere l'espace des 144 etats admissibles."
3     etats = [] # Un etat est un tuple : (x_roi, y_roi,
4               x_tour, y_tour, trait)
5     coords = [1, 2, 3] # Grille 3x3
6     for x_r in coords:
7         for y_r in coords:
8             for x_t in coords:
9                 for y_t in coords:
10                    if (x_r, y_r) != (x_t, y_t): # Regle physique:
11                        deux pieces ne partagent pas la meme case
12                        etats.append((x_r, y_r, x_t, y_t, 1)) #
13                        Trait aux Blancs
14                        etats.append((x_r, y_r, x_t, y_t, 0)) #
15                        Trait aux Noirs
16     return etats
17
18 def est_en_echec(x_r: int, y_r: int, x_t: int, y_t: int) ->
19     bool:
20     "Verifie si le Roi est menace par la Tour (meme ligne ou
21     colonne)."
22     return x_r == x_t or y_r == y_t
```

Annexe-E2 : Construction de la structure d'adjacence

```
1 def generer_graphe(etats: list) -> dict:
2     graphe = {etat: [] for etat in etats}
3     for etat in etats:
4         x_r, y_r, x_t, y_t, trait = etat
5         if trait == 1: # C'est au tour du Roi Blanc
6             for dx in [-1, 0, 1]:
7                 for dy in [-1, 0, 1]:
8                     if dx == 0 and dy == 0: continue
9                     nx_r, ny_r = x_r + dx, y_r + dy
10                    if 1 <= nx_r <= 3 and 1 <= ny_r <= 3:
11                        if (nx_r, ny_r) != (x_t, y_t) and not
est_en_echec(nx_r, ny_r, x_t, y_t):
12                            graphe[etat].append((nx_r, ny_r, x_t, y_t, 0))
13                else: # C'est au tour de la Tour Noire
14                    directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
15                    for dx, dy in directions:
16                        for pas in [1, 2]:
17                            nx_t, ny_t = x_t + dx * pas, y_t + dy * pas
18                            if 1 <= nx_t <= 3 and 1 <= ny_t <= 3:
19                                if (nx_t, ny_t) == (x_r, y_r): break
20                                graphe[etat].append((x_r, y_r, nx_t, ny_t, 1))
21                            else: break
22    return graphe
```

Annexe-E3 :Détection de cycles par parcours DFS

```
1 def detecter_cycles(graphe: dict, etat_initial: tuple):
2     # Au depart, tous les 144 etats sont vierges
3     couleurs = {etat: 'BLANC' for etat in graphe.keys()}
4     trajectoire_actuelle = []
5     cycles_detectes = []
6     def dfs(u: tuple):
7         couleurs[u] = 'GRIS'
8         trajectoire_actuelle.append(u)
9         for v in graphe[u]:
10             if couleurs[v] == 'BLANC':
11                 dfs(v) # La piece est non visitee, on avance
12             elif couleurs[v] == 'GRIS':
13                 # DETECTION DU CYCLE : On retombe sur nos pas !
14                 cycles_detectes.append(list(trajectoire_actuelle)
15                                     +[v])
16                 # L'etat est totalement explore, on le ferme et on
17                 # recule pour essayer un autre chemin
18                 couleurs[u] = 'NOIR'
19                 trajectoire_actuelle.pop()
20     # On lance le moteur depuis la position de depart
21     dfs(etat_initial)
22     return cycles_detectes
```

Annexe-E4 : Implémentation de la condition de troncature

```
1 def explorer_et_rompre_cycles(graphe, etat_de_depart):
2     couleurs = {etat: 'BLANC' for etat in graphe.keys()}
3     occurrences = {etat: 0 for etat in graphe.keys()}
4     chemin_en_cours = []
5     cycles_detectes = []
6     def dfs(etat):
7         if occurrences[etat] == 2:
8             return #Troncature de la trajectoire
9         couleurs[etat] = 'GRIS'
10        occurrences[etat] += 1
11        chemin_en_cours.append(etat)
12        for etat_suivant in graphe[etat]:
13            if couleurs[etat_suivant] == 'BLANC':
14                dfs(etat_suivant)
15            elif couleurs[etat_suivant] == 'GRIS':
16                idx = chemin_en_cours.index(etat_suivant)
17                cycle = chemin_en_cours[idx:]
18                cycles_detectes.append(len(cycle))
19                dfs(etat_suivant)
20        couleurs[etat] = 'NOIR'
21        occurrences[etat] -= 1
22        chemin_en_cours.pop()
23    dfs(etat_de_depart)
24    return cycles_detectes
```

Annexe-E5 : Test du code informatique

```
1 print("--- DEBUT DE LA SIMULATION ---")
2 etats = generer_etats_admissibles()
3 graphe = generer_graphe(etats)
4 print(f"Graphe genere avec {len(etats)} configurations
      admissibles.")
5 etat_depart = (1, 1, 3, 3, 1)
6 print(f"Lancement depuis l'etat : {etat_depart}")
7 cycles = explorer_et_rompre_cycles(graphe, etat_depart)
8 print("--- FIN DE LA SIMULATION ---")
9 print(f"Cycles detectes      : {len(cycles)}")
10 print(f"Longueur minimale   : {min(cycles)}")
11 print(f"Longueur maximale   : {max(cycles)}")
12 print("Conclusion : finitude du systeme prouvee
      informatiquement.")
```