Rapport de projet : Jeu Trolls & Châteaux

Yacine SAHNOUNE   Azouaou LOUAIFI

Mai 2025

Résumé

L'objectif de ce TP est de créer un programme informatique capable de déterminer, pour un joueur donné et à n'importe quelle étape du jeu Trolls & Châteaux, la stratégie prudente (minimax) optimale. Nous présenterons d'abord la modélisation du problème : ensemble des actions possibles, fonctions de gain. Ensuite, nous décrirons l'algorithme mis en œuvre, basé sur l'analyse des sous-arbres de décision et la recherche des équilibres de Nash en stratégie mixte. Enfin, nous illustrerons ses performances par des simulations numériques sur des configurations de jeu variées.

1 Présentation théorique du code

Paramètres du jeu

Chaque état du jeu « Trolls & Châteaux » est entièrement décrit par quatre paramètres :

— x : nombre de pierres restantes pour le joueur 1,
— y : nombre de pierres restantes pour le joueur 2,
— t : position actuelle du troll (0 au départ, augmente ou diminue à chaque coup),
— M : nombre de cases entre les deux châteaux (doit être impair)

Pour déterminer la stratégie prudente (minimax) optimale à chaque étape du jeu « Trolls & Châteaux », notre programme suit essentiellement deux phases :

1. **Calcul des gains et des cas de base.** La fonction `base_gain(x,y,t)` identifie d'abord si l'état (x, y, t) est terminal et renvoie directement le gain -1, 0 ou $+1$. Les principaux cas sont :
   — `if t == M+1` : le troll atteint la case $(M + 1)$ (château joueur 1) $\rightarrow$ gain $+1$.
   — `if t == -(M+1)` : le troll atteint la case $-(M + 1)$ (château joueur 2) $\rightarrow$ gain 1.
   — `if x == 0 and y == 0` : plus aucune pierre $\rightarrow$ gain $=$ sign(t).
   — `if x == 0` : joueur 1 sans pierres $\rightarrow$
     — $t > y \rightarrow +1$,
     — $t < y \rightarrow 1$,
     — $t = y \rightarrow 0$.
   — `if y == 0` : symétrique pour joueur 2 sans pierres $\rightarrow$
     — $x > |t|$ ou $t > 0 \rightarrow +1$,
     — $x < |t|$ et $t < 0 \rightarrow 1$,
     — sinon $\rightarrow 0$.
   — Sinon (`return None`) $\rightarrow$ état non terminal.

2. **Application des principes de l'algorithme de Kuhn.** Pour tout état non terminal, on construit la matrice de gains J dont chaque entrée

$$J_{ij} = V\big(x - i,\ y - j,\ t + \mathrm{sign}(i - j)\big)$$

est la valeur du sous-jeu résultant si le joueur 1 retire i jetons et le joueur 2 en retire j. On applique alors l'algorithme de Kuhn, qui consiste à chercher la stratégie mixte de jeu en résolvant un programme linéaire de maximisation de la valeur garantie V.

L'implémentation finale recourt à une fonction récursive mémoïsée (`@lru_cache`) qui stocke, dans deux tables de hachage (`V_CACHE` et `P_CACHE`), la valeur du jeu et la distribution des probabilités (stratégie mixte optimale) pour chaque triplet (x, y, t). Cette distribution permet de déterminer, pour le joueur 1, la probabilité de choisir chacun des coups possibles.

Bloc principal et affichages

Dans le bloc `if name == "main"`, le programme realise un test complet pour un etat donne :
— Affichage de la matrice des gains ;
— Affichage de la valeur du jeu ;
— Affichage de la distribution des probabilites pour chaque coup du joueur 1 ;
— Selection aleatoire d'un coup selon cette distribution (fonction `bestmove`) et affichage du coup choisi.

2 Résultats

2.1 Tableaux de probabilité

m	$j_1(x)$	j_2(y)	t	Tableau de probabilités
3	15	15	0	[0.06, 0.18, 0.0, 0.18, 0.01, 0.24, 0.01, 0.32, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
5	15	15	0	[0.0, 0.46, 0.05, 0.49, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
7	15	15	0	[0.17, 0.6, 0.22, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
9	15	15	0	[0.0, 0.99, 0.01, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
> 9	15	15	0	[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Table 1 – Tableaux de probabilités en fonction des paramètres de jeu

3 Simulation

Des simulations (à 1000 itérations) ont été réalisées pour confronter la stratégie prudente à une stratégie aléatoire. Le tableau 2 présente les ratios de victoire du joueur 1 obtenus pour chaque configuration de jeu :

Nombre de cases	Nombre de pierres	Stratégie prudente	Stratégie aléatoire
5	15	75.9%	17.2%
7	15	96.1%	2.0%
7	30	95.8%	3.1%
15	30	100.0%	100.0%
15	50	100.0%	100.0%

Table 2 – Ratios de victoires en confrontation de stratégies

4 Discussion

Analyse des stratégies prudentes

Les résultats obtenus montrent que la stratégie prudente implémentée par notre programme se comporte de manière cohérente avec l'intuition du jeu :

— Pour un petit nombre de cases(par exemple $m = 3$), la distribution privilégie généralement les mouvements intermédiaires (lancer 2 ou 4 pierres) plutôt que des extrêmes (1, 3 ou 5 pierres), ce qui minimise le risque de rapprocher le troll trop près de son château.

— À mesure que la taille du plateau augmente (cas $m > 9$), la stratégie devient de plus en plus conservatrice, recommandant systématiquement de n'avancer qu'une seule pierre. Cela reflète la nécessité de préserver ses ressources lorsque le troll est encore loin des extrémités.

— On observe une préférence marquée pour les nombres pairs de pierres dans certaines configurations ($m = 7$ ou $m = 9$), sans doute parce qu'ils équilibrent mieux l'écart de position du troll en cas de réponse adversaire.

Limites de l'étude

Bien que robuste pour des valeurs modérées de x, y et m, notre approche présente plusieurs limites :

— La complexité de résolution croît rapidement avec le nombre de pierres, car la taille de la matrice de gains est $ximesy$ et chaque sous-état est recalculé ou mémoïsé.

— Le choix de la taille du plateau (M fixe et impair) limite la généralisation à d'autres configurations plus complexes (multi-dimensionnelles, plusieurs trolls, etc.).

5 Perspectives

Plusieurs axes d'amélioration et d'extension peuvent être envisagés :

1. Introduire un historique des coups de l'adversaire pour adapter dynamiquement la stratégie.
2. Generaliser le plateau à plus de dimensions (graphes ou grilles) et étendre l'algorithme aux jeux séquentiels plus complexes, notamment par découpage en sous-problèmes et heuristiques d'approximation.
3. Optimiser la performance via l'utilisation de solveurs plus rapides (CPLEX, Gurobi) ou d'approches mixtes (méthodes de Monte Carlo Tree Search couplées à la PL).
4. Étudier la résistance de la stratégie mixte à des adversaires imparfaits (joueurs aléatoires, heuristiques humaines) et mesurer l'avantage compétitif dans des simulations tournantes.