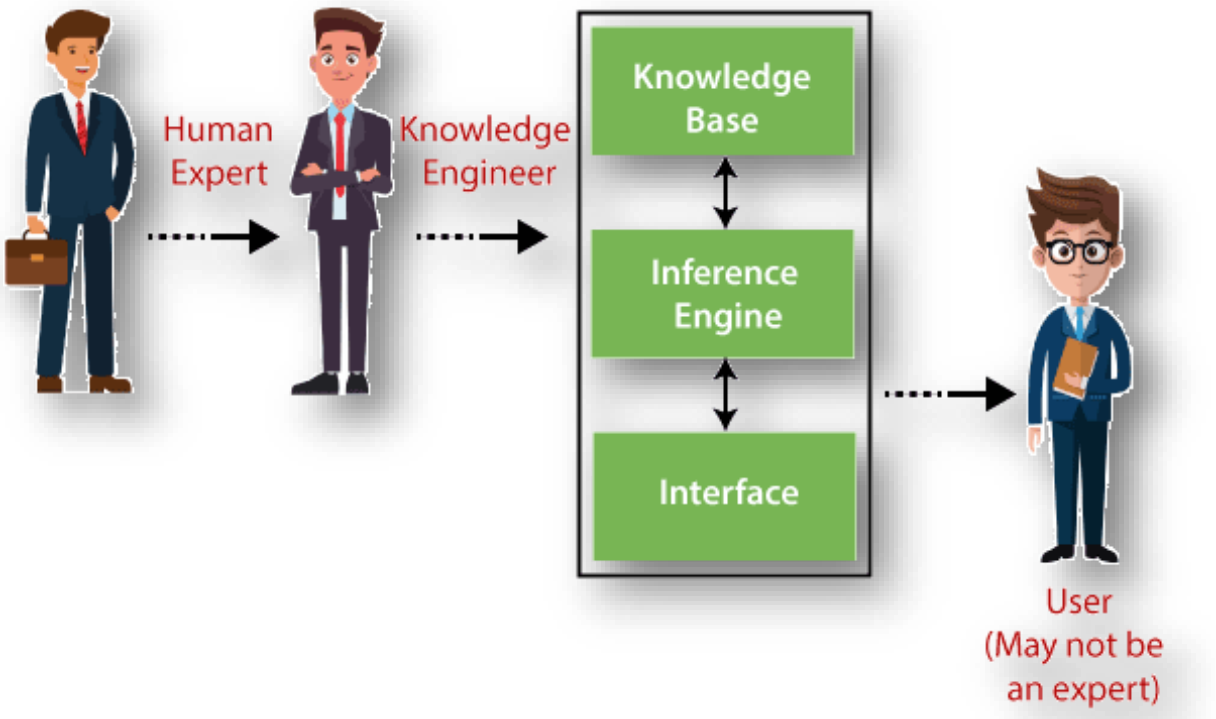
Machine-Learning : Projet-1

Système Expert



AL NATOUR Mazen et HERVOUET Léo

ML 2024-2025

Table des matières

Introduction : 3
I. Présentation du Problème et du Système Expert : 4
A. Description du problème des tours de Hanoï 4
B. Base de connaissances 4
C. Représentation du jeu 5
D. Stratégie de résolution et génération des situations uniques 5
II. Développement des Composants du Système : 6
A. Modélisation du jeu des tours de Hanoï 6
1. Structure de la classe Jeu_Hanoi 6
2. Méthodes principales 6
B. Stratégie de résolution : Moteur d'inférence en chaînage avant, Algorithme de sélection et d'application des règles 7
1. Redéfinition des règles de base 7
2. Algorithme de sélection et d'application des règles 7
3. Processus d'inférence basé sur les règles 9
4. Optimisation par réorganisation des règles 9
III. Résultats et Analyse : 10
A. Analyse de la Performance 10
1. Séquence de coups optimale : 10
B. Comparaison avec notre approche 12
IV. Propositions d'amélioration et extension du Jeu : 13
A. Optimisation des déplacements 13
B. Extension du jeu et propositions d'amélioration pour des configurations plus complexes 14
1. Augmentation du nombre de disques 14
2. Ajout de plus de pics 14
Conclusion : 15

Introduction :

Les tours de Hanoï sont un célèbre problème de logique inventé par le mathématicien français Édouard Lucas en 1883. Ce jeu consiste à déplacer une pile de disques de tailles décroissantes d'une tour de départ vers une tour d'arrivée, en passant éventuellement par une tour intermédiaire, tout en respectant des règles précises : un seul disque peut être déplacé à la fois, et un disque ne peut être placé que sur un autre disque de taille supérieure ou sur une tour vide. Le défi réside dans la recherche d'une solution optimale en un minimum de coups, ce qui en fait un problème classique de récursivité et de réflexion.

Notre but dans ce projet est de résoudre le problème des tours de Hanoï à l'aide d'un système Expert. À la différence de l'approche algorithmique classique, nous établissons un moteur d'inférence qui repose sur des règles préétablies, ce qui permet à notre programme de prendre des décisions en fonction de l'état du jeu. La réalisation de ce projet, en Python, repose sur une base de règles qui oriente les déplacements des disques, tout en évitant les situations déjà analysées pour garantir une progression vers la solution.

L'objectif principal de ce rapport est de détailler la mise en œuvre du système expert, d'analyser son efficacité en termes de nombre de coups nécessaires pour résoudre le problème, et de comparer cette solution à l'algorithme optimal. Nous explorerons également les possibilités d'extension et d'optimisation du système.

I. Présentation du Problème et du Système Expert :

A. Description du problème des tours de Hanoï

Les tours de Hanoï constituent un casse-tête mathématique où l'objectif est de déplacer une pile de disques de tailles décroissantes d'une tour de départ vers une tour d'arrivée, en passant par une tour intermédiaire. Le jeu doit être résolu en respectant les règles suivantes :

1. Un seul disque peut être déplacé à la fois.
2. Un disque ne peut être placé que sur un autre disque de taille supérieure ou sur une tour vide.

Dans le projet, nous travaillons avec trois tours (ou « pics ») et trois disques de tailles différentes (3 cm, 2 cm, et 1 cm de diamètre). Au départ, tous les disques sont empilés sur le premier pic dans l'ordre décroissant (le disque le plus grand en bas). Le but est de déplacer l'ensemble des disques vers le troisième pic dans le même ordre, tout en minimisant le nombre de déplacements.

Le nombre minimum de coups pour résoudre ce problème, selon l'algorithme optimal, est de 7 pour 3 disques. Notre objectif est de mettre en place un système expert capable de résoudre ce problème à l'aide de règles basées sur une logique déterministe.

B. Base de connaissances

La base de connaissances de notre système expert est constituée des règles définies pour déplacer les disques en respectant les contraintes du jeu. Voici un exemple de règles extraites de notre base :

- **Règle 1** : Si le pic 0 contient au moins un disque et que le disque supérieur du pic 0 peut être déplacé sur le pic 1 (i.e., la taille du disque supérieur du pic 1 est supérieure ou le pic est vide), et que cette nouvelle situation n’a pas été observée auparavant, alors déplacer le disque du pic 0 vers le pic 1.
- **Règle 2** : Si le pic 1 contient au moins un disque et que le disque supérieur peut être déplacé sur le pic 2 dans les mêmes conditions, alors réaliser le déplacement.

Ces règles sont appliquées en fonction de l'état actuel du jeu. Le moteur d'inférence les évalue à chaque étape pour déterminer le déplacement à effectuer. Si plusieurs règles sont applicables, une métarègle stipule que la règle de plus petit indice est sélectionnée.

C. Représentation du jeu

Afin de représenter l'état du jeu à chaque instant, nous utilisons une structure Python, modélisée par une classe appelée Jeu_Hanoi. Cette classe contient deux attributs principaux :

- **pic** : un tableau Numpy de dimensions 3x3, où chaque ligne représente un pic, et les colonnes représentent les disques empilés sur ce pic. Par exemple, pic[0, 0] contient le disque de plus grande taille sur le pic 0, tandis que pic[0, 2] contient le plus petit disque sur le pic 0.
- **nombre_palet** : un tableau de taille 3, où chaque élément indique le nombre de disques actuellement présents sur un pic donné. Par exemple, nombre_palet[0] = 3 signifie que le premier pic contient trois disques.

À l'initialisation, tous les disques sont placés sur le premier pic. L'état final désiré est celui où tous les disques sont correctement empilés sur le troisième pic.

D. Stratégie de résolution et génération des situations uniques

Pour éviter les boucles infinies durant la résolution, il est nécessaire de mémoriser les configurations de jeu déjà observées. Chaque configuration de jeu est convertie en un nombre unique, appelé « identifiant de situation », qui est stocké dans une liste appelée 'situations'. Pour générer cet identifiant unique, nous utilisons une méthode basée sur la concaténation de la position de chaque disque se trouvant dans la matrice 'pic'.

Cette stratégie permet d'éviter de répéter des déplacements qui mèneraient à une situation déjà étudiée, garantissant ainsi une progression continue vers la solution.

II. Développement des Composants du Système :

A. Modélisation du jeu des tours de Hanoï

La classe **Jeu_Hanoi** est au cœur du système expert que nous avons développé pour résoudre le problème des tours de Hanoï. Elle permet de modéliser l'état du jeu, d'appliquer les règles, et de gérer les déplacements des disques entre les pics.

1. Structure de la classe Jeu_Hanoi

La classe est composée de plusieurs méthodes qui permettent de gérer l'état des trois pics et d'effectuer des déplacements tout en respectant les règles du jeu.

- **Attributs principaux :**
  - **pic**: Un tableau Numpy de dimensions 3x3 représentant l'état des trois pics. Chaque ligne correspond à un pic, et les valeurs des colonnes indiquent les disques empilés, les positions vides étant représentées par des zéros.
  - **nombre_palet**: Un tableau d'entiers qui indique le nombre de disques présents sur chaque pic. Par exemple, nombre_palet[0] = 3 signifie que le premier pic contient trois disques.
  - **situations**: Une liste qui stocke les configurations de jeu déjà visitées afin d'éviter de répéter les mêmes situations.

2. Méthodes principales

- **get_situation()**: Retourne une représentation de l'état actuel des pics sous forme d'une chaîne de caractères. Cette méthode est utilisée pour suivre les situations déjà visitées afin d'éviter les boucles infinies.
- **pic_vide**(indice_pic): Vérifie si un pic est vide en s'assurant que toutes les cases correspondantes du tableau pic sont égales à 0.
- **get_pic_top**(indice_pic): Retourne le disque qui se trouve au sommet d'un pic. Si le pic est vide, cette méthode retourne None. Cela permet d'identifier le disque à déplacer.
- **regle_jeu**(indice_pic1, indice_pic2): Cette méthode vérifie si un déplacement d'un disque du pic indice_pic1 vers le pic indice_pic2 est valide. Le déplacement est considéré valide si le pic source n'est pas vide, et si le pic cible est vide ou le disque au sommet du pic source est plus petit que celui au sommet du pic cible.

- **deplacer**(indice_pic1, indice_pic2): Effectue le déplacement d'un disque du pic indice_pic1 vers le pic indice_pic2. Cette méthode met à jour l'état du tableau pic et du tableau nombre_palet en conséquence.
- **effectue_deplacement**(indice_pic1, indice_pic2): Si le déplacement est valide, il est exécuté, et la nouvelle situation est enregistrée dans la liste des situations visitées. Cette méthode retourne True si le déplacement a été effectué, sinon False.
- **afficher()**: Affiche l'état actuel des trois pics, ce qui permet de visualiser les déplacements effectués.

B. Stratégie de résolution : Moteur d'inférence en chaînage avant, Algorithme de sélection et d'application des règles

Le moteur d'inférence de notre système expert repose sur une approche en **chaînage avant**, c'est-à-dire que l'on part d'une situation initiale donnée, et on applique les règles de manière itérative pour progresser vers l'état final souhaité. L'idée principale est de sélectionner et d'appliquer des règles qui modifient l'état du jeu jusqu'à ce que la condition finale (tous les disques déplacés sur le troisième pic) soit atteinte.

1. Redéfinition des règles de base

Dans le cadre du jeu des tours de Hanoï, les règles se résument aux déplacements possibles entre les trois pics. Chaque règle décrit une transition d'un état à un autre, basée sur les règles du jeu :

- **Règle 1** : Un disque peut être déplacé d'un pic à un autre uniquement si le pic source n'est pas vide.
- **Règle 2** : Un disque ne peut être déplacé sur un pic cible que si ce dernier est vide, ou si le disque au sommet du pic cible est plus grand que le disque déplacé.

Ces deux règles garantissent que les déplacements respectent les contraintes du jeu.

2. Algorithme de sélection et d'application des règles

L'algorithme de chaînage avant que nous utilisons pour sélectionner et appliquer les règles de déplacement fonctionne de manière itérative et se base sur l'état actuel du jeu. Voici le processus :

1. **Identification de la situation courante** :
    - À chaque étape de la résolution, l'état actuel des pics est évalué par la méthode get_situation(), qui permet de représenter l'état sous forme d'une chaîne de caractères.

- Cet état est comparé à l'état final attendu, pour déterminer si le but du jeu est atteint.

2. **Génération des déplacements possibles** :
   - À partir de la situation courante, une liste de **déplacements potentiels** est générée. Ces déplacements sont définis par les combinaisons possibles entre les pics ((0, 1), (0, 2), (1, 2), etc.).
   - Chaque déplacement est ensuite évalué pour vérifier sa validité en utilisant la méthode regle_jeu(indice_pic1, indice_pic2). Cette méthode applique les règles du jeu définies précédemment (vérification des disques au sommet des pics).
3. **Sélection de la règle à appliquer** :
   - Parmi les déplacements valides, l'algorithme sélectionne celui qui n'a pas encore été effectué dans la séquence actuelle de déplacements.
   - Si un déplacement est jugé valide et n'entraîne pas de retour en arrière immédiat (éviter de revenir directement au dernier état visité), il est sélectionné pour être appliqué.
4. **Application du déplacement** :
   - Le déplacement sélectionné est exécuté en modifiant l'état du jeu à l'aide de la méthode deplacer(indice_pic1, indice_pic2). Cela consiste à retirer le disque du sommet du pic source et à le placer au sommet du pic cible.
   - Une fois le déplacement effectué, l'état du jeu est enregistré dans la liste des situations visitées (situations), afin d'éviter les boucles où la même situation serait explorée à nouveau.
5. **Mise à jour de l'historique des déplacements** :
   - Après chaque déplacement, l'état actuel du jeu est sauvegardé dans une liste moves_history. Cela permet de suivre l'évolution du jeu pas à pas et de visualiser les différents états par lesquels il passe avant d'atteindre l'état final.
6. **Itération du processus** :
   - Le processus se répète jusqu'à ce que l'état final soit atteint, c'est-à-dire que tous les disques soient empilés sur le troisième pic, ou que l'algorithme détecte qu'il n'existe plus de déplacements valides (dans le cas d'une erreur ou d'un blocage).

3. Processus d'inférence basé sur les règles

Le processus d'inférence dans notre système se base donc sur l'application successive de règles qui permettent de transformer l'état initial en état final. Voici un résumé de ce processus :

- **Étape initiale** : Le moteur d'inférence démarre avec l'état initial du jeu, où tous les disques sont empilés sur le premier pic.
- **Détection des situations déjà visitées** : À chaque étape, l'état courant est comparé à ceux déjà rencontrés, pour éviter de revenir sur une situation antérieure.
- **Sélection des déplacements valides** : Les règles de jeu sont appliquées pour filtrer les déplacements possibles, et un déplacement est sélectionné.
- **Application des déplacements** : Le déplacement sélectionné est appliqué, et le nouvel état du jeu est enregistré.
- **Arrêt du processus** : Le moteur d'inférence s'arrête lorsque l'état final est atteint, c'est-à-dire que tous les disques sont correctement empilés sur le troisième pic, ou lorsqu'il est impossible d'effectuer d'autres déplacements.

4. Optimisation par réorganisation des règles

Un aspect important de notre moteur d'inférence est la possibilité d'optimiser la résolution en modifiant l'ordre des règles appliquées. Dans la fonction solve, l'ordre des déplacements est défini dans la liste moves. En réorganisant cette liste, il est possible de minimiser le nombre de déplacements nécessaires.

III. Résultats et Analyse :

Le sujet nous demandait d'appliquer une suite de déplacements prédéfinie en utilisant la plus petite règle éligible à chaque étape :

- **Règle 1** : Essayer de déplacer un disque du pic 0 au pic 1.
- **Règle 2** : Essayer de déplacer un disque du pic 1 au pic 2.
- **Règle 3** : Essayer de déplacer un disque du pic 2 au pic 1.
- **Règle 4** : Essayer de déplacer un disque du pic 1 au pic 0.

En appliquant cette séquence de règles dans notre programme, nous obtenons une solution en 26 coups. Ce nombre élevé s'explique par l'absence d'optimisation dans l'ordre des mouvements, conduisant à plusieurs déplacements inutiles avant la résolution complète du problème.

A. Analyse de la Performance

Le nombre minimal de coups pour résoudre les tours de Hanoï avec trois disques est de 7, selon la stratégie optimale. En comparaison, notre méthode utilisant une simple évaluation des règles sans priorisation optimale entraîne des mouvements redondants, ce qui allonge considérablement la résolution.

1. Séquence de coups optimale : ***(Cette partie a été réalisé en bonus)***

Voici la séquence parfaite de déplacements pour résoudre le problème en 7 coups :

1. Déplacer le plus **petit disque** du pic 0 au pic 2.
2. Déplacer le **disque moyen** du pic 0 au pic 1.
3. Déplacer le plus **petit disque** du pic 2 au pic 1.
4. Déplacer le plus **grand disque** du pic 0 au pic 2.
5. Déplacer le plus **petit disque** du pic 1 au pic 0.
6. Déplacer le **disque moyen** du pic 1 au pic 2.
7. Déplacer le plus **petit disque** du pic 0 au pic 2

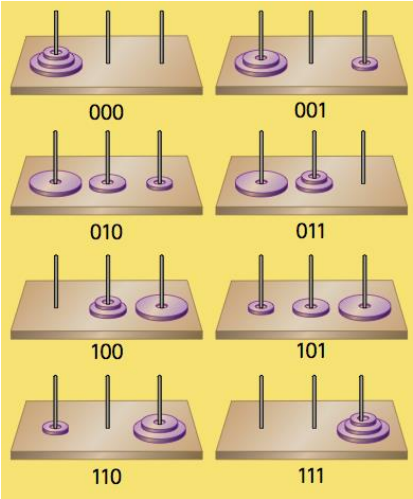


Figure 1 : Résolution des tours d'Hanoï en 7 coups

La méthode que nous avons employée repose sur un algorithme récursif bien connu, qui garantit une solution en un nombre minimal de coups. Pour une configuration de trois disques, la solution optimale nécessite exactement 7 déplacements, selon la formule classique :

Nombre minimal de coups = $2^{\mathbf{n}} - 1$ où **n** représente le nombre de disques.

- Le principe de L'algorithme récursive est le suivant :
  - On déplace les n-1 disques du pic source vers un pic auxiliaire, en utilisant le pic cible comme intermédiaire.
  - Ensuite, on déplace le dernier disque (le plus grand) directement du pic source vers le pic cible.
  - Enfin, on déplace les n-1 disques du pic auxiliaire vers le pic cible, en utilisant le pic source comme intermédiaire.

  Cette approche repose sur la décomposition du problème global en sous-problèmes plus petits, qui sont eux-mêmes résolus de manière récursive.

- Voici les grandes étapes du code de la fonction optimal dans le fichier *Resolve_Hanoi.py* :

  ➔ **Fonction hanoi_recursive :** Cette fonction implémente l'algorithme récursif. Elle prend comme paramètres le nombre de disques à déplacer (n), les indices des pics (source, cible et auxiliaire), l'état du jeu (une instance de la classe Jeu_Hanoi) et une liste de mouvements qui enregistre chaque état après un déplacement. Chaque appel récursif gère un sous-ensemble de disques, déplaçant successivement les n-1 disques, puis le plus grand disque, et enfin repositionnant les n-1 disques restants.

  ➔ **Affichage des déplacements :** À chaque déplacement effectué, l'état actuel du jeu est affiché pour montrer visuellement la progression du casse-tête. Cette approche permet également de suivre chaque coup et de vérifier si le jeu se déroule de manière correcte.

Donc cette solution, qui suit un schéma optimal, atteint l'objectif sans mouvements superflus, réduisant ainsi le nombre de coups nécessaires et la durée de la résolution.

B. Comparaison avec notre approche

Notre solution présente des problèmes d'efficacité en raison de l'application rigide des règles sans une réévaluation dynamique des meilleures options de déplacement. Par conséquent, on effectue 19 coups supplémentaires par rapport à la solution idéale. Ceci met en évidence la nécessité d'une méthode de sélection plus complexe dans le moteur d'inférence.

En résumé, l'amélioration des règles, que ce soit par une réorganisation des priorités ou l'intégration d'une stratégie plus avancée, permettrait de diminuer considérablement le nombre de déplacements et d'atteindre la solution optimale en un temps réduit.

IV. Propositions d'amélioration et extension du Jeu :

Pour améliorer les performances de notre système expert, la première intuition est que l'ordre des coups joue un rôle crucial dans l'efficacité de la résolution. Afin de vérifier cette hypothèse, nous avons utilisé la fonction solve_find_best(...) pour chercher la meilleure configuration possible en appliquant les règles de déplacements fournies par le sujet. Cette fonction prend en entrée un état initial du jeu, un état final cible, et une liste de déplacements possibles.

Dans un premier temps, nous avons donné à cette fonction la liste de coups standards : [(0, 1), (1, 2), (2, 1), (1, 0)]. Cependant, malgré les combinaisons testées, le résultat reste constant : **26 coups** pour résoudre les tours de Hanoï. Ce résultat montre que l'algorithme, dans sa forme actuelle, ne permet pas d'optimiser le nombre de déplacements avec les règles de base.

A. Optimisation des déplacements

Une piste d'amélioration évidente serait de permettre davantage de types de déplacements. Jusqu'ici, notre système expert ne permettait que des déplacements entre des pics voisins, c'est-à-dire :

- De pic 0 à pic 1
- De pic 1 à pic 2
- De pic 2 à pic 1
- De pic 1 à pic 0

Cela laissait de côté deux déplacements potentiellement utiles : **de pic 0 à pic 2** et **de pic 2 à pic 0**. En intégrant ces nouveaux mouvements, nous augmentons les possibilités d'optimisation.

Nous avons alors modifié la liste des déplacements en ajoutant ces nouveaux coups et en les plaçant en tête de liste : [(2, 0), (0, 2), (0, 1), (1, 2), (2, 1), (1, 0)]. Grâce à cette simple modification, le système parvient à résoudre le problème en **9 coups**, une amélioration significative par rapport à la solution initiale de 26 coups.

Pour affiner davantage cette stratégie, nous avons réutilisé la fonction solve_find_best(...) afin de trouver la meilleure séquence possible. Avec cette approche, nous avons obtenu une solution en **8 coups**, avec la séquence suivante : [(1, 2), (0, 2), (0, 1), (2, 1), (2, 0), (1, 0)]. Cette séquence montre que même de petites optimisations dans l'ordre des déplacements peuvent se rapprocher fortement de la solution optimale en 7 coups.

B. Extension du jeu et propositions d'amélioration pour des configurations plus complexes

L'ajout de règles pour améliorer la résolution à trois disques nous montre qu'il est possible d'optimiser encore plus notre système expert. Dans ce cadre, il est pertinent de réfléchir à l'extension du système pour résoudre des configurations plus complexes avec un plus grand nombre de disques ou de tours.

1. Augmentation du nombre de disques

Lorsque le nombre de disques augmente, le nombre de coups nécessaires suit une relation exponentielle : pour **n disques**, le nombre minimal de déplacements est de **2^n - 1**. Par conséquent, un système expert capable de résoudre des instances avec plus de disques nécessitera un moteur d'inférence plus puissant.

Piste d'amélioration :

- **Réorganisation des règles** en fonction de la taille des disques, afin de prioriser les déplacements des disques les plus petits pour minimiser les étapes intermédiaires.

2. Ajout de plus de pics

Une autre extension intéressante serait d'augmenter le nombre de pics. L'ajout d'un quatrième pic, par exemple, ouvre de nouvelles stratégies pour déplacer les disques plus efficacement. Dans ce cas, le jeu ne suit plus la solution classique à trois pics, et le nombre minimal de coups peut être réduit. Pour gérer ce type de configuration, nous pourrions envisager :

- **Nouvelles règles spécifiques au déplacement entre quatre pics**. Par exemple, en réduisant le nombre de déplacements disponibles pour permettre une meilleure résolution.

Conclusion :

Grâce à ce projet, nous avons pu étudier l'option d'utiliser un système expert afin de résoudre un problème classique de façon différente de l'approche purement récursive. Malgré les inefficacités de notre solution initiale, elle a souligné l'importance d'améliorer les règles d'inférence et de varier les stratégies de déplacement afin d'améliorer les performances.

En analysant les résultats obtenus en comparaison avec l'algorithme optimal, nous avons repéré des éléments d'amélioration évidents, tels que l'amélioration des déplacements et l'élargissement de la base de connaissances afin de prendre en compte plus de configurations de jeu. En outre, l'intégration de nouvelles caractéristiques, telles que l'extension à des versions plus complexes ou une interface graphique, pourrait apporter une amélioration future au projet.

Ce travail illustre ainsi la pertinence des systèmes experts dans la résolution de problèmes complexes, et les possibilités d'amélioration via des stratégies d'optimisation et d'extension du modèle initial.