Intelligence Artificielle
2024 - 2025

Université Paris Cité

Rapport de Projet

Implémentation d'une Intelligence Artificielle basée sur l'algorithme Minimax et l'élagage alpha-bêta pour le Neuf Hommes de Morris

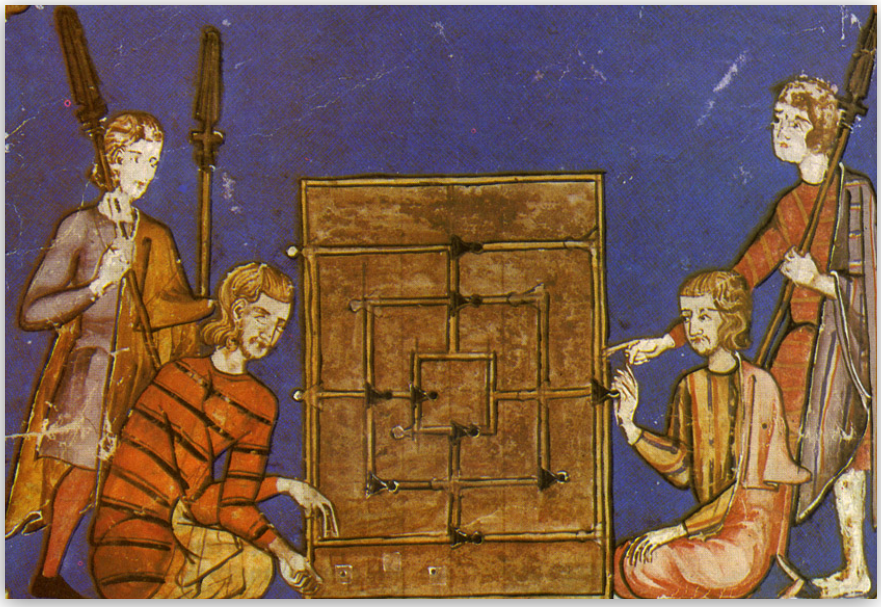


Illustration du jeu joué avec des dés, XIIIe siècle

Aidoudi Aaron

Tables des matières

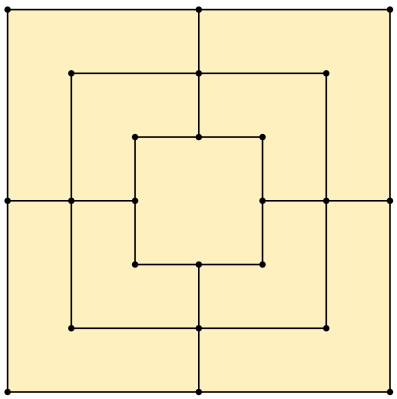
I. Description du jeu

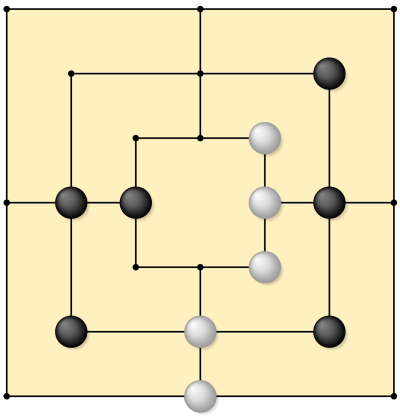
Le jeu du Moulin, également connu sous les noms de Neuf Hommes de Morris ou Jeu de Mérelles, est un jeu de société traditionnel européen dont les origines remontent à l'Antiquité. Il était particulièrement populaire durant le Moyen Âge et se jouait sur des plateaux gravés dans la pierre ou fabriqués en bois. Ce jeu stratégique pour deux joueurs se joue sur un plateau constitué de trois carrés imbriqués reliés par des lignes, offrant 24 points d'intersection où les pions peuvent être placés ou déplacés. Son attrait réside dans la simplicité de ses règles et la profondeur de ses stratégies, qui en font un sujet d'étude privilégié pour l'intelligence artificielle appliquée aux jeux classiques.

Ce document présente ainsi la conception, l'implémentation et l'évaluation d'IA jouant au jeu, en s'appuyant sur l'algorithme Minimax optimisé par l'élagage alpha-bêta, et en comparant trois niveaux d'heuristiques d'évaluation. Après une présentation détaillée des règles et de la modélisation informatique du jeu, le rapport expose les choix d'implémentation, décrit les différentes heuristiques, puis analyse les résultats d'un tournoi entre IA. Enfin, il discute les limites observées et propose des perspectives d'amélioration pour des IA plus performantes et adaptatives.

- **Matériel**

Ce jeu, lorsqu'il est joué physiquement et non via le terminal d'un ordinateur, nécessite que peu de matériel. Ce sont un plateau de jeu, 9 pions blancs et 9 pions noirs, ainsi que deux joueurs. Les figures suivante représente des configurations possibles du plateau.





- **Règles**

Le but est de réduire l'adversaire à deux pions ou de bloquer ses mouvements, tout en formant des alignements de trois pions, que l'on appelle moulins, pour capturer ses pièces. Le jeu se déroule en trois phases distinctes : la pose, le mouvement et le saut.

Phase 1, la pose : les joueurs déterminent qui commence et placent, à tour de rôle, un pion sur une intersection libre. Lorsqu'un joueur forme un moulin (trois pions alignés sur une même ligne, horizontalement ou verticalement), il peut retirer un pion adverse du plateau, sauf si ce dernier fait partie d'un moulin actif. Une fois que les deux joueurs ont placé leurs neuf pions, la phase de mouvement débute.
Phase 2, le mouvement : les joueurs déplacent leurs pions, à tour de rôle, vers une intersection adjacente libre en suivant les lignes tracées sur le plateau. Si un joueur forme un moulin pendant cette phase, il peut capturer un pion adverse selon les mêmes règles que lors de la pose. Cette phase continue jusqu'à ce qu'un joueur soit réduit à trois pions ou ne puisse plus effectuer de mouvements légaux.

Phase 3 : le saut : lorsqu'un joueur n'a plus que trois pions, il peut déplacer ses pièces librement sur n'importe quelle intersection libre du plateau.

Un joueur gagne la partie si son adversaire est réduit à deux pions, ce qui l'empêche de former des moulins; ou bien que celui-ci est incapable d'effectuer le moindre mouvement avec ses pièces restantes. Aussi, la partie peut se terminer sur un match nul si les joueurs effectuent un certain nombre de mouvements identiques ou sans réaliser aucune prise.

II. Structures de données utilisées

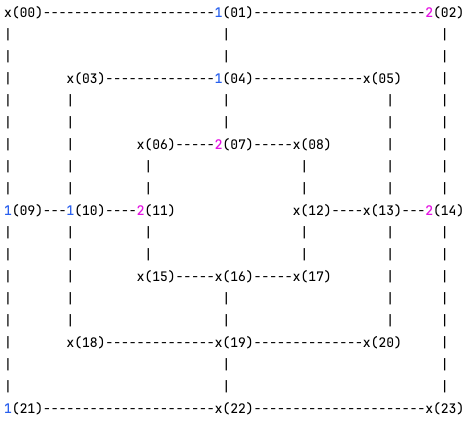
Dans le cadre de l'implémentation du jeu en Python, plusieurs structures de données ont été utilisées pour représenter les éléments essentiels du jeu, tels que le plateau, les positions des pions, et les relations entre les intersections. Ces structures permettent une manipulation efficace des données et facilitent la mise en œuvre des règles du jeu.

• Représentation du plateau

Le plateau est représenté par une liste contenant 24 éléments, correspondant aux 24 intersections possibles sur le plateau, et affiché via la fonction *printTableau*. Chaque élément de la liste peut prendre l'une des valeurs suivantes :

- *'x'* : intersection vide.
- *'1'* : intersection occupée par un pion du joueur 1.
- *'2'* : intersection occupée par un pion du joueur 2.

Ci-contre l'affichage de celui-ci lors d'une partie, avec les pions du Joueur 1 en bleu et ceux du Joueur 2 en rose. Les numéros entre parenthèses indiquent quant à eux le nom de la position.



• Relation entre positions adjacentes

Les relations entre les positions adjacentes sur le plateau, c'est-à-dire les intersections directement connectées par une ligne, sont stockées dans une liste imbriquée via la fonction *positionsAdjacentes*. Chaque position est associée à une sous-liste contenant les indices des positions adjacentes. Par exemple, les positions adjacentes à la position 3 sont 4 et 10, donc *adjacences[8]* est *[4,10]*.

• Détection des moulins

Pour vérifier si un joueur a formé un moulin, une série de combinaisons prédéfinies est utilisée. Ces combinaisons sont implémentées sous forme de tuples représentant les indices des positions qui forment un moulin potentiel. Par exemple, les moulins faisables avec la position 3 sont ceux avec la position 4 et 5, et ceux avec 10 et 18. Ainsi, *moulin[3]* est *[(3,4,5), (3,10,18)]*.
Cette détection de moulins est liée à plusieurs fonctions, entre autres :

- *prochainMoulin()* : détermine si un joueur peut former un moulin en ajoutant un pion sur une position donnée.
- *moulinCree()* : vérifie si un moulin a été formé sur une position donnée.

D'autres fonctions sont implémentées à propos des moulins. Entres-autres :

- *nombrePossibleMoulins()* : donnant le nombre de moulins potentiellement réalisables pour un joueur donné
- *moulinEnFormation()* : vérifiant si un pion est en formation pour un moulin potentiel et utilisée dans *nombrePiecesMoulinEnFormation()*.

• Générations des mouvements possibles

Les mouvements possibles pour chaque phase du jeu sont générés dynamiquement à l'aide de fonctions qui manipulent le plateau et les relations entre positions. Ces fonctions renvoient une liste d'états possibles, où chaque état est une copie du plateau après un mouvement valide.
Pour la phase de placement, la fonction *mouvementsPossiblesEtape1*() génère la liste des plateaux possibles en plaçant le pion du joueur à chacune des positions libres, tout en gérant le cas où un moulin est formé et le joueur doit alors retirer un pion adverse.
Pour la phase 2 et 3, les fonctions *mouvementsPossiblesEtape2()* et *mouvementsPossiblesEtape3()* génèrent la liste des plateaux possibles en déplaçant chaque pion du joueur aux positions possibles. Le choix entre l'utilisation de ces deux fonctions se fait par la fonction mouvementsPossiblesEtape2ou3(), calculant le nombre de pions du joueur et renvoyant les mouvements possibles en fonction.
On utilise dans ces fonctions, de sorte à manipuler efficacement différents états du plateau sans affecter l'état actuel, la fonction *deepcopy*() bibliothèque *copy* créant des copies indépendantes du plateau.
Enfin, une approche via la génération de plateau "inverse" a été mise en place pour générer les mouvements de l'adversaire, avec les fonctions *plateauInverse()* et *genereListePlateauInverse()*. Par exemple, un plateau avec les positions *[1, x, x, 2, x]* générera une position adverse possible telle que *[2, x, x, 1, x]* (inversion) → *[2, 1, x, 1, x]* (mouvement possible) → *[1, 2, x, 2, x]* (ré-inversion). Cette approche est utilisé dans l'algorithme Minimax pour générer les positions du joueur minimisant.

• Interactions

Les interactions entre l'utilisateur et le terminal sont gérées via des fonctions basiques telles que *int(input())* dans la fonction *demanderPosition*(). Celle-ci est utilisée dans les fonctions *jouerTourHumain*() et *retirerPionAdverse*(), elles-mêmes utilisées dans les fichiers *HumainVsHumain.py* (pour le jeu entre humains uniquement) et *HumainVsIA.py* (pour jouer contre une IA). L'exécution de ces scripts lance le jeu dans le mode choisi, avec la difficulté d'IA choisie. Plus d'informations sont données dans le *README*, et des captures d'écran du jeu contre l'IA sont données dans l'annexe.
Enfin, une fonction dédiée au tournoi des IA a été créée, donnant pour deux heuristiques et profondeurs données le résultat (victoire, défaite, égalité) pour 50 parties.

III. Implémentation de l'IA

L'intelligence artificielle développée pour ce projet repose sur l'algorithme Minimax, optimisé par la technique d'élagage alpha-bêta. Cet algorithme permet à l'IA d'explorer les différentes configurations possibles du jeu afin de déterminer le meilleur coup à jouer à chaque tour.

- **Fonctionnement détaillé de Minimax**

L'algorithme Minimax est une méthode récursive qui parcourt l'arbre des états possibles du jeu jusqu'à une certaine profondeur définie à l'avance. À chaque état (ou nœud) de cet arbre, l'algorithme attribue une valeur numérique évaluant la qualité du plateau selon une fonction heuristique donnée. Le joueur humain (lorsque le jeu est lancé en mode humain contre IA) est considéré comme le joueur maximisant (cherchant à maximiser son score), alors que l'IA représente le joueur minimisant (cherchant à minimiser le score du joueur humain).

L'implémentation se base sur la classe *Evaluer*, contenant deux attributs :
- *evaluer*, une valeur numérique représentant la qualité estimée d'un état du plateau.
- *plateau*, la configuration du plateau associée à cette évaluation.

La fonction *minimax()* prend en paramètres le plateau actuel, la profondeur restante à explorer dans l'arbre des possibilités, un booléen indiquant si c'est au tour du joueur maximisant ou minimisant, les valeurs alpha et beta utilisées pour l'élagage alpha-bêta, un booléen indiquant si le jeu est encore dans sa première phase, et une fonction heuristique permettant d'évaluer les plateaux.

À chaque appel récursif, la fonction vérifie d'abord si une condition terminale est atteinte :
- la profondeur maximale fixée est atteinte.
- un joueur possède moins de trois pions sur le plateau (condition de fin du jeu).

Dans ce cas, elle retourne immédiatement la valeur calculée par la fonction heuristique fournie.
Si aucune condition terminale n'est remplie, la fonction génère alors tous les mouvements possibles en fonction de la phase actuelle du jeu, via les fonctions *mouvementsPossiblesEtape1* ou *mouvementsPossiblesEtape2ou3*, donnant l'ensemble des mouvement possibles.
Pour chaque mouvement possible, la fonction s'appelle récursivement en inversant le rôle du joueur (maximisant devient minimisant et inversement), tout en diminuant la profondeur restante d'un niveau.
Chaque plateau obtenu alors est évalué grâce à l'heuristique choisie afin d'estimer sa qualité.

Cependant, le nombre d'états possibles croît exponentiellement avec la profondeur de recherche. Par exemple, avec 23 positions libres (lorsque l'IA joue son premier coup) et une profondeur fixée à 5, l'algorithme évalue précisément 6 436 343 états différents : $O(b^m) = 23^5$ (avec b le nombre maximal de coups possibles et m la profondeur maximale de l'arbre, formule donnée dans le cours).
Cette explosion combinatoire rend rapidement impossible une exploration complète en un temps raisonnable.

Pour pallier ce problème majeur, une optimisation essentielle est intégrée : l'élagage alpha-bêta.

Cette optimisation permet d'éviter d'explorer certaines branches inutiles dans l'arbre des possibilités.
Deux valeurs sont utilisées :
- alpha : meilleure valeur trouvée jusqu'ici pour le joueur maximisant
- beta : meilleure valeur trouvée jusqu'ici pour le joueur minimisant.

À chaque étape récursive, lorsque c'est au tour du joueur maximisant, on met à jour la valeur alpha. Si à un moment donné, on obtient une valeur supérieure ou égale à beta, cela signifie que le joueur minimisant ne choisira jamais cette branche. On peut donc arrêter immédiatement l'exploration de cette branche, et l'on vient de réaliser un élagage. Lorsque c'est au tour du joueur minimisant, on applique le même raisonnement en inversant les rôles.
Cette optimisation permet ainsi de réduire drastiquement le nombre total d'états explorés, améliorant grandement les performances et réduisant le temps nécessaire aux calculs. La complexité devient alors $O\left(b^{\frac{m}{2}}\right) = 23^{\frac{5}{2}} = 2537$ (dans le meilleur cas, si l'arbre est parfaitement équilibré et ordonné).

Cette implémentation rigoureuse et optimisée permet donc à l'intelligence artificielle de prendre des décisions efficaces et rapides tout en conservant un haut niveau stratégique dans ses choix.

- **Fonctionnement détaillé des heuristiques**

Les heuristiques jouent un rôle central en permettant à l'intelligence artificielle d'évaluer la qualité d'un état du plateau de jeu. Ces fonctions d'évaluation fournissent une estimation numérique qui guide l'IA dans ses décisions stratégiques, notamment lorsqu'elle doit choisir le meilleur coup à jouer. Plusieurs heuristiques ont été développées, chacune avec un niveau de sophistication croissant.

i. Heuristique naïve

L'heuristique naïve évalue un plateau en fonction des moulins potentiels (c'est à dire deux pions alignés sur trois) et du nombre de pions restants pour chaque joueur. Elle est simple mais efficace pour des scénarios basiques. Cette heuristique attribue des poids différents aux moulins potentiels en fonction du nombre de pions sur le plateau. Par exemple, lorsqu'un joueur a moins de quatre pions, les moulins adverses sont considérés comme plus menaçants, et leur poids est donc augmenté dans le calcul.
Bien que simple, cette approche présente des limites, notamment dans les situations complexes où plusieurs facteurs doivent être pris en compte simultanément.

ii. Heuristique avancée

L'heuristique avancée introduit plusieurs facteurs supplémentaires pour évaluer plus précisément la qualité d'un plateau. Ces facteurs incluent :

- les pions sur le plateau : une différence significative entre les pions des deux joueurs peut indiquer un avantage ou un désavantage.
- les moulins formés : le nombre total de moulins déjà créés par chaque joueur.
- les moulins potentiels : comme dans l'heuristique naïve, mais avec un poids ajusté.
- la mobilité : le nombre de mouvements possibles pour chaque joueur (uniquement après la phase 1).
- le blocage des mouvements adverses : les situations où les pions adverses sont immobilisés.

Cette heuristique utilise des coefficients (ou "poids") pour pondérer l'importance relative de chaque facteur. Par exemple, les moulins formés ont un poids plus élevé que les mouvements possibles, car ils représentent une menace stratégique plus importante.

iii. Heuristique experte

L'heuristique experte s'inspire des recommandations avancées trouvées dans diverses études (voir sources) et intègre une analyse encore plus fine des configurations du plateau. Elle prend en compte :

- les moulins fermés récemment : les moulins formés lors du dernier coup jouent un rôle crucial.
- le nombre total de moulins : un joueur avec plusieurs moulins a un avantage stratégique évident.

- les pions bloqués : le nombre de pièces adverses immobilisées.
- le nombre total de pions : Comme dans l'heuristique avancée.
- les configurations à 2 ou 3 pièces : les configurations où deux ou trois pions sont alignés ou proches d'un moulin.
- les double moulins : les situations où deux moulins se chevauchent, créant une menace double.
- configurations gagnantes ou perdantes : si un joueur est proche d'une victoire ou d'une défaite (moins de trois pions ou immobilisation totale).

Cette heuristique ajuste dynamiquement les poids des différents facteurs en fonction de la phase du jeu (phase 1 ou phase 2/3). Par exemple, en phase 1 (placement), les configurations à deux pièces et les moulins potentiels sont prioritaires. Là où en phase 2/3 (déplacement), les double-moulins et les configurations gagnantes deviennent critiques.
Grâce à sa complexité et à sa précision, cette heuristique permet à l'IA d'adopter une stratégie très compétitive contre des joueurs humains expérimentés.

IV. Tournois des IA

• Fonctionnement du tournoi

Le tournoi consiste à faire s'affronter trois versions de l'IA, chacune utilisant une heuristique différente (naïve, avancée, experte). Chaque paire d'heuristiques joue 50 parties, soit un total de 150 parties (3 combinaisons possibles : naïve vs avancée, naïve vs experte, avancée vs experte). Seul le nombre de victoires est pris en compte pour évaluer les performances. Les autres métriques (temps par coup, nombre moyen de moulins formés) ne sont pas utilisées. Aussi, un match est considéré comme nul si un cycle de déplacement de pions est répété 5 fois.

La profondeur de recherche d dans l'algorithme Minimax influence directement la performance et le temps de calcul. Ainsi il nous faut définir celle-ci de manière équitable mais adaptée à chaque heuristique.
Pour l'heuristique naïve, il est décidé de prendre une profondeur : $d = 3$: cette heuristique est simple et rapide à évaluer. Une profondeur plus élevée entraînerait une explosion combinatoire sans apporter de bénéfices significatifs dans la prise de décision.
Pour l'heuristique avancée, la profondeur est légèrement augmentée avec $d = 4$: l'ajout de facteurs comme la mobilité et les moulins formés nécessite un peu plus de profondeur pour exploiter pleinement ces informations. Une profondeur modérée permet d'équilibrer précision et temps de calcul.
Quant à l'heuristique experte, une profondeur $d = 5$ est nécessaire : cette heuristique complexe prend en compte des configurations stratégiques avancées. Une profondeur plus importante est nécessaire pour maximiser son efficacité et anticiper des scénarios complexes.

• Résultats

Match-up (1 vs 2)	Victoires1	Victoires 2	Matchs nuls
Naïve vs Avancée	4	9	37
Naïve vs Experte	3	17	30
Avancée vs Experte	5	11	34

Plusieurs observations peuvent être faites de ces résultats :

Prévalence des matchs nuls :

Les matchs nuls dominent dans tous les affrontements. Cela indique que les IA sont capables de bloquer efficacement les mouvements adverses, mais peinent à conclure une victoire. Ce phénomène est cohérent avec les observations de différents articles scientifiques, où il est mentionné que le Jeu du Moulin tend vers des configurations équilibrées lorsque les joueurs adoptent des stratégies défensives solides. De plus, les IA ayant tendances à éviter les sacrifices offensifs risqués et bloquer systématiquement les formations adverses

Performance de l'heuristique experte :

L'heuristique experte remporte ses affrontements contre la naïve (17 victoires) et avancée (11 victoires), confirmant son efficacité dans l'exploitation des configurations stratégiques complexes. Son taux élevé de victoires contre la naïve montre qu'elle excelle dans les scénarios où l'adversaire manque de profondeur stratégique.

Limites de l'heuristique avancée :

Bien qu'elle surpasse la naïve, elle peine face à l'experte ne remportant que 5 parties sur 50. Cela suggère que ses critères d'évaluation, bien qu'améliorés par rapport à la naïve, restent insuffisants pour rivaliser avec une analyse plus fine des configurations du plateau.

Faiblesse de l'heuristique naïve :

L'IA utilisant cette heuristique ne remporte que quelques parties. Cela reflète ses limites dans la gestion des phases complexes du jeu, notamment en phase de mouvement et de saut, où une stratégie plus sophistiquée est nécessaire.

- **Conclusions**

Les résultats confirment que la qualité de l'heuristique utilisée est déterminante dans le jeu du Moulin. L'heuristique experte montre une nette supériorité grâce à sa prise en compte des configurations stratégiques avancées, comme les double-moulins et les blocages adverses.

Aussi, le nombre élevé de matchs nuls, particulièrement entre les heuristiques avancée et experte, souligne la nature défensive du jeu lorsque deux IA compétentes s'affrontent. Cela reflète également le caractère équilibré du jeu du Moulin tel qu'observé dans les études académiques.

Enfin, de sorte à réduire le nombre de matchs nuls et améliorer les performances générales, il pourrait être intéressant d'introduire des mécanismes d'apprentissage adaptatif ou d'ajuster dynamiquement la profondeur en fonction de la phase du jeu.

V. Bilan

Ce projet a permis de concevoir et d'évaluer des intelligences artificielles jouant au jeu du Moulin en s'appuyant sur l'algorithme Minimax avec élagage alpha-bêta et trois niveaux d'heuristiques : naïve, avancée et experte. Les résultats du tournoi, opposant ces IA dans un total de 150 parties, ont mis en évidence l'importance cruciale des heuristiques dans la performance stratégique.
Le nombre élevé de matchs nuls, en particulier entre les IA avancée et experte, reflète la nature défensive du jeu du Moulin lorsqu'il est joué à un niveau stratégique élevé. Ces résultats confirment les observations de travaux académiques, qui souligne l'équilibre inhérent au jeu lorsque les deux adversaires adoptent des stratégies optimales.
En conclusion, ce projet met en lumière l'efficacité des heuristiques sophistiquées et ouvre la voie à des améliorations futures, telles que l'intégration de techniques d'apprentissage automatique pour adapter dynamiquement les stratégies.

VI. Sources

À propos des règles et de l'histoire du Neuf Hommes de Morris :

- en.wikipedia.org/wiki/Nine_men%27s_morris

Evaluation du plateau de jeu :

- kartikkukreja.wordpress.com/2014/03/17/heuristicevaluation-function-for-nine-mens-morris/
- dasconference.ro/papers/2008/B7.pdf
- cs.brandeis.edu/~storer/JimPuzzles/GAMES/NineMensMorris/INFO/GasserArticle.pdf
- chessprogramming.org/Nine_Men%E2%80%99s_Morris

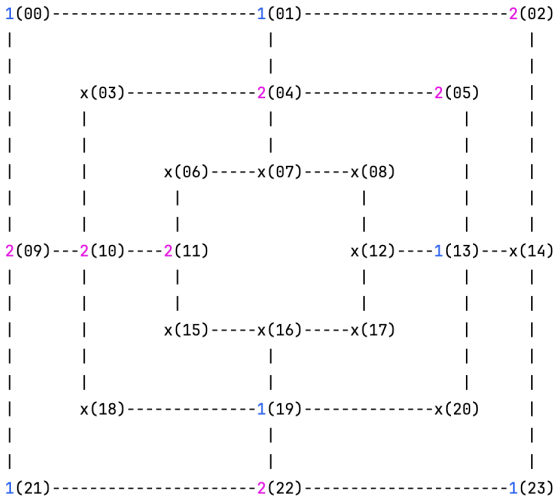
VII. Annexe

Cette partie du document regroupe quelques captures du jeu au fil des parties, exposant l'interface (via le terminal) ainsi que des exemples types de configurations obtenues via les IA.

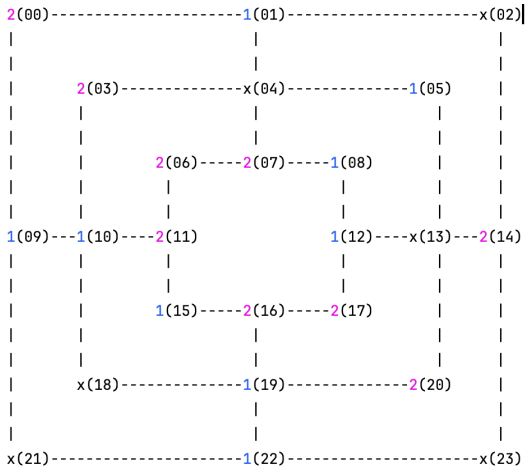
```
Bienvenue dans le Jeu du Neuf Hommes de Morris - Mode Humain contre IA !

Première phase : placement des pions.

Il vous reste chacun 9 pions à placer.

x(00)----------------------x(01)----------------------x(02)
|                           |                           |
|                           |                           |
|       x(03)--------------x(04)--------------x(05)     |
|       |                   |                   |       |
|       |                   |                   |       |
|       |        x(06)-----x(07)-----x(08)      |       |
|       |         |                 |         |       |
|       |         |                 |         |       |
x(09)---x(10)----x(11)              x(12)----x(13)---x(14)
|       |         |                 |         |       |
|       |         |                 |         |       |
|       |        x(15)-----x(16)-----x(17)      |       |
|       |                   |                   |       |
|       |                   |                   |       |
|       x(18)--------------x(19)--------------x(20)     |
|                           |                           |
|                           |                           |
x(21)----------------------x(22)----------------------x(23)
```

Interface lors du lancement de la partie en mode Humain contre IA.

Ci-contre, le Joueur 2, via l'heuristique experte, a formé un double moulin sur les triplets (9, 10, 11) et (3,4,5). C'est une position qui lui assure une victoire, à moins que l'adversaire vienne bloquer la rotation aux positions 3 et 10...



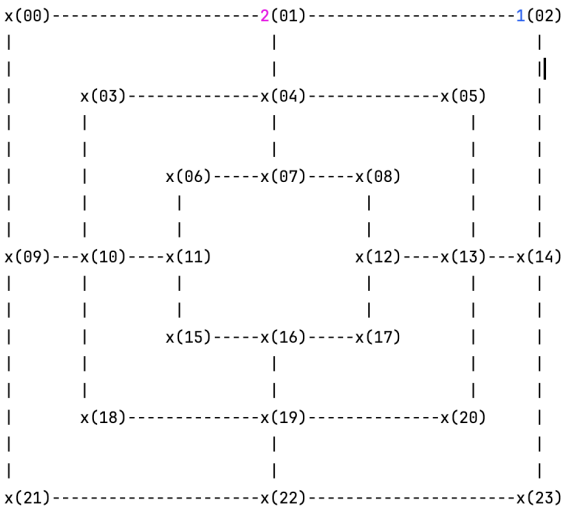

```
Joueur 1 : Placez une pièce sur le plateau.
Entrez une position : 22

L'IA réfléchit...
L'IA place un pion en position 16.

Deuxième phase : déplacement des pions.
```



Ici, l'IA anticipe le mouvement du pion en position 15 (lors de la phase 2, le joueur 1 ayant posé tous ses pions) vers la position 16 en y posant un pion.

Dans le cas d'une erreur de saisie, le programme soulève l'erreur et indique à l'utilisateur de réessayer.

```
Il vous reste chacun 8 pions à placer.
```




```
Joueur 1 : Placez une pièce sur le plateau.
Entrez une position : 1
Position occupée. Réessayez.
Entrez une position :
```