Université de Caen Basse-Normandie

Projet annuel de L3

Implémentation d'algorithmes d'intelligence artificielle pour le jeu de Gygès

Auteurs :
Valentin Lemière
Guillaume Desquesnes

Enseignant :
Grégory Bonnet

20 mars 2013

Table des matières

Table des figures

1 | Contexte

1.1 Intérêt

La création d'intelligences artificielles (IA) de jeu présente plusieurs intérêts qui rendent leur utilisation utile pour la recherche en intelligence artificielle.

Elles peuvent servir de benchmark, en effet un jeu permet de comparer de façon simple des IA entre elles ou contre des joueurs humain. Des facteurs tels que le nombre de coups nécessaire pour la victoire ou encore le ratio victoire/défaite sont des indicateurs de qualité facilement calculables et interprétables.

De plus cette branche de l'intelligence artificielle fut beaucoup étudiée, il est donc aisé de trouver des algorithmes ou recherches sur lesquels se baser. Certains jeux furent l'objets de plus de travaux que d'autres, notament le jeu d'échecs qui fut le jeu par excellence pour la recherche en IA pendant plus de 40 ans, ou plus récemment le jeu de go.

FIGURE 1.1 – Deep blue : ordinateur ayant battu Kasparov aux échecs

1.2 Présentation

Le Gygès est un jeu de stratégie combinatoire abstrait, c'est à dire un jeu où deux joueurs jouent à tour de rôle, où tous les éléments du jeu sont connus et où le hasard n'intervient pas.

Ce jeu se joue sur un plateau 6×6, plus deux bases, avec des pions de valeurs 1 à 3 (représentés par des anneaux sur le pions) correspondant à leur capacité de déplacement. Les pions peuvent se déplacer en ligne ou en colonne mais pas en diagonale.

Les pions n'appartiennent à aucun joueur, chaque joueur ne pouvant déplacer que les pions se trouvant sur la ligne non vide la plus proche de sa base.

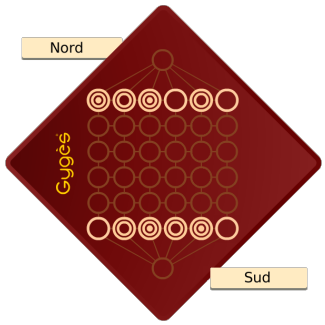

FIGURE 1.2 – Exemple de plateau de Gygès

Le but Le but du jeu est de réussir à capturer la base adverse avec un pion, le premier joueur réussissant cet objectif gagne et met fin à la partie.

Les règles Le Gygès possède un certain nombre de règles qui le démarque de jeux plus « standard ».

1. Un pion se déplace d'autant de case que ça valeur, ni plus ni moins (voir figure 1.3–1.5) ;
2. Un pion peut se déplacer sur les lignes et les colones, mais pas les diagonales ;
3. Un pion ne peut pas passer par une case occupée ;
4. Si la dernière case du mouvement d'un pion est libre alors il se pose dessus et le tour du joueur s'achève ;

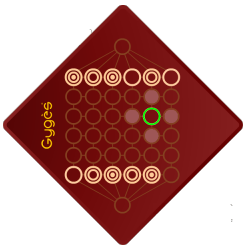

FIGURE 1.3 – Déplacement d'un pion 1

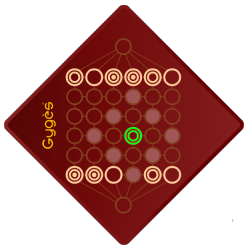

FIGURE 1.4 – Déplacement d'un pion 2

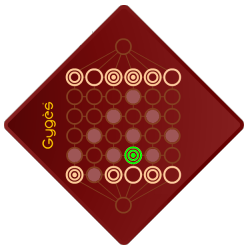

FIGURE 1.5 – Déplacement d'un pion 3

5. Sinon le joueur a deux possibilités :
    a. rebondir : Le pion rebondit sur le pion d'arrivée d'autant de case que ce dernier a d'anneaux ; ou

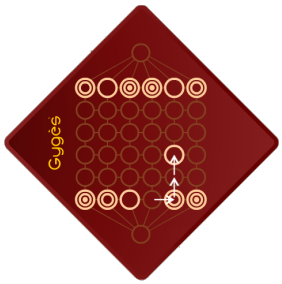

FIGURE 1.6 – Exemple de rebond

    b. remplacer : Le pion remplace le pion d'arrivée, ce dernier pourra alors être reposé sur n'importe quelle case libre du plateau à l'exception des lignes avant la première ligne de l'adversaire.

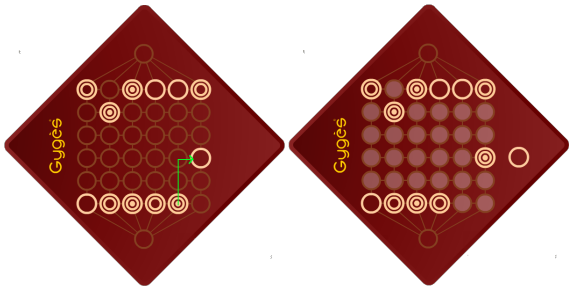

FIGURE 1.7 – Exemple de remplacement

6. Un pion ne peut passer sur une ligne que dans un sens ;
7. Un pion ne peut pas passer par une base, il ne peut que y aboutir ; et
8. Pour gagner un pion doit finir exactement sur la base adverse, il ne doit pas lui rester de mouvement.

Déroulement d'une partie Le joueur sud pose ses pions sur sa première ligne suivi du joueur nord. Chaque joueur déplace ensuite à tour de rôle, en commencant par le joueur sud, un pion de leur ligne. La partie s'arrête lorsqu'un joueur a capturé la base adverse.

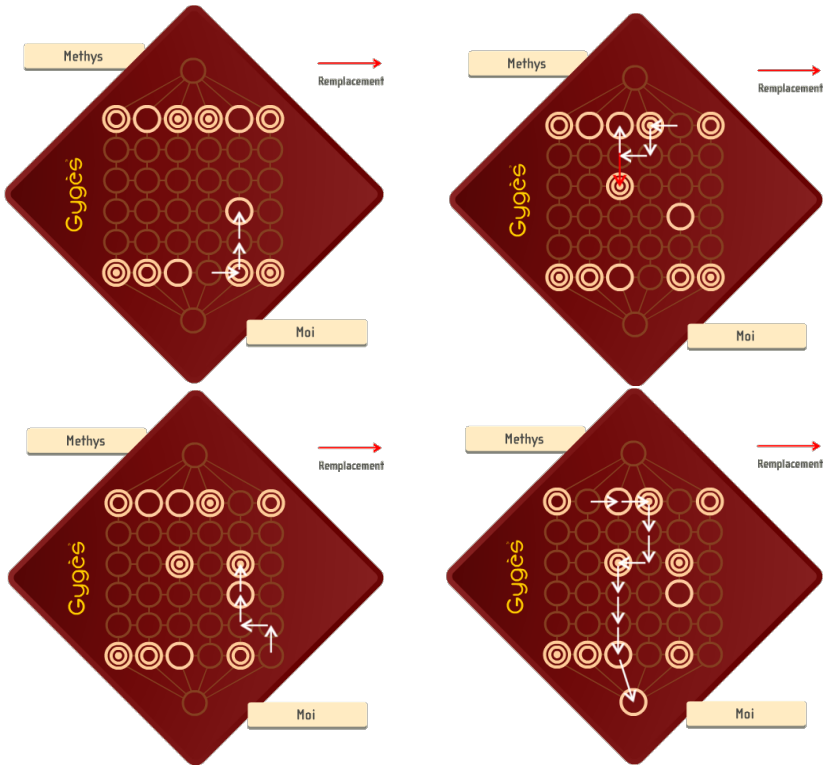

FIGURE 1.8 – Example de déroulement d'une partie de Gygès

Les parties sont composées de mouvements complexes mais ne durent généralement pas plus d'une dizaines de coups.

1.3 Difficultées

Traditionnellement les techniques d'intelligence artificielle pour les jeux listent l'intégralité des coups possible, les notent et choisissent le meilleur ; cela en regardant plusieurs coups à l'avance. Voyons donc pourquoi avec ces techniques le jeu de Gygès est un jeu difficile.

1.3.1 Point de vue technique

Nombreux coups possible Le grand nombre de coups évoqué ci-avant pose problème d'un point de vue temps de calcul. En effet, évaluer chaque coup en en regardant plusieurs en avance est très long. Hors pour que la machine soit efficace, il faut qu'elle soit capable d'anticiper plusieurs coups à l'avance.

Coups semblables Beaucoup de coups, notamment les remplacements, se ressemblent. Il est de plus possible d'accèder à un même plateau via plusieurs déplacements de pions différents. Il faudrait donc être capable de supprimer efficacement les doublons

Cycle Dans le Gygès, il y a beaucoup de cycle possible. Par exemple, il est possible de déplacer aucun pion au delà de la première ligne , voire de rester dans la même configuration que le plateau précedent. Enfin il est possible d'annuler le déplacement d'un pion de l'adversaire par un déplacement symètrique. Il faut donc savoir éviter ou détecter (pour les arrêter) ces cycles afin d'empêcher le jeu de tourner en rond.

Chemins asymétrique Les chemins construits sont différents selon les joueurs, cela veut dire que pour un plateau les deux n'ont pas les mêmes probabilités de gagner, il faut donc prendre en compte les deux sens d'un plateau lors de son évaluation.

Mais le Gygès n'est pas seulement un jeu complexe pour les IA, en effet un joueur humain aussi cherchera à envisager les différents coups possible et à en choisir le meilleur.

1.3.2 Point de vue humain

Appartenance des pions Les pions n'ont pas de propriétaire. Les deux joueurs ont la possibilité de déplacer chaque pion, cela implique de construire

une stratégie ou chaque pièce pourra être déplacé. Par exemple, le pion que vous souhaitiez déplac au prochain tour pourra être déplacé alors qu'il se trouve sur votre ligne.

Nombreux coups possible À partir d'un plateau un joueur peut avoir accès à un très grand nombre de coups possible. Par exemple les possibilités d'ouverture d'une partie varie entre 150 et 600. Bien que parmi ces nombreuses possibilités beaucoup soient similaires, il reste difficile de trouver un coup utile n'ouvrant pas de chemin à l'adversaire.

Complexité du jeu Les déplacements complexes de pions et les parties rapides rendent difficile l'apprentissage du jeu pour les joueurs humains. De plus, le grand nombre de coup possible rend l'anticipation d'un coup plus difficile que dans les jeux traditionnels (échecs, dames).

Chemins asymétrique Les chemins construits par les joueurs ne sont pas forcément utilisable dans les deux sens. Il faut donc constamment s'assurer que les chemins construits ne donnent pas un avantage déterminant à l'adversaire.

1.4 Cahier des charges

Pour ce projet nous avions plusieurs étapes a réaliser, les étapes 1–4 permettent d'avoir un module de jeu complet permetant le déroullement d'une partie, les étapes 5 et 6 de jouer de façon « intelligente », les étapes 7 et 8 d'améliorer la performance de l'IA et finallement l'étape 9 de pouvoir facilement laisser un joueur humain affronter l'IA.

Nous avions comme objectifs :

1. de commencer par étudier le jeu de Gygès pour pouvoir le modéliser ;
    a. lister les règles spéciales ;
    b. coder le mouvement des pions ; et
    c. déterminer quels pions chaque joueur peut déplacer.

2. de créer un module de jeu permettant le déroulement d'une partie ;
    a. création d'un plateau ;
    b. initialisation des IA ; et

c. déroulement tour à tour jusqu'a la victoire d'un des deux joueurs.

3. de lister les coups possible à partir d'un plateau : pour chaque pions déplaçable lister quelles sont les positions attégniables ;

4. de déterminer si un coup est légal : existe-t-il un mouvement permettant de passer du plateau de départ à celui d'arrivé ;

5. de créer une fonction d'évaluation de plateau ;

6. d'implémenter un algorithme de recherche alpha-beta, permettant de ne pas jouer au hasard ;

7. de trouver et d'implémenter des fonctions d'évaluation plus performantes, pour jouer de meilleurs coups ;

8. de trouver et implémenter des algorithmes de recherche plus performants, permettant une recherche plus profonde et donc de meilleurs coups ; et

9. de développer une interface graphique pour pouvoir jouer facilement contre une IA.

2 | Algorithmes

Pour pouvoir choisir le meilleur coup à jouer il faut d'abord les lister, c'est à cela que servent les algorithmes de recherche. Nous allons commencer par voir comment représenter un plateau de Gygès, puis les principes fondamentaux de la recherche de coup et enfin nous présenterons l'algorithme MTD-f.

2.1 Représentation du plateau

Le plateau du jeu est divisé en deux partie : la grille principale de 6 cases par 6 cases ainsi que 2 bases, une par joueur.

Les cases sont réprésentées par un entier correspondant à la valeur du pion présent sur la case, $\{1, 2, 3\}$, ou à 0 si la case est vide. La grille principale est stockée sous la forme d'une matrice 6×6 et les bases dans un simple tableau 1×2, les deux contenant des cases sous forme d'entier.

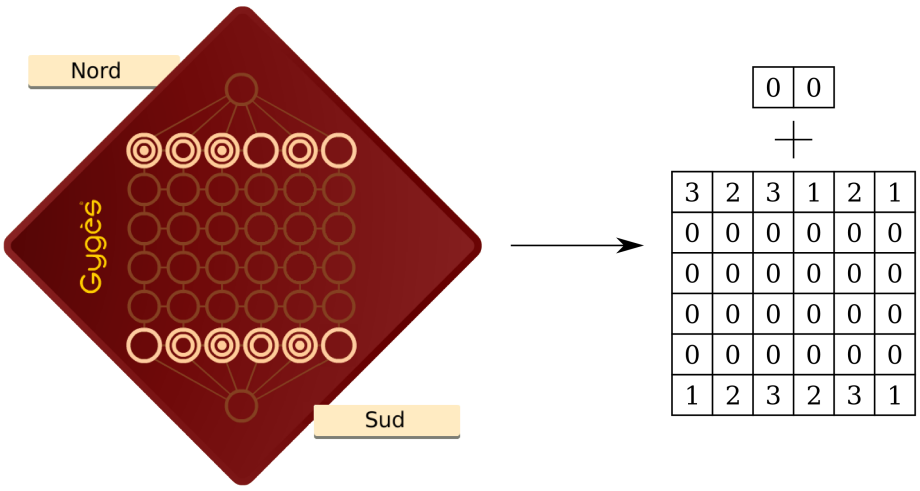

FIGURE 2.1 – Représentation d'un plateau

2.2 Concepts fondamentaux

2.2.1 Arbre de recherche

La grande majorité des algorithmes de jeu représente une partie sous la forme d'un arbre de recherche, ce qui correspond plus ou moins à un arbre bicolore. La racine représente le plateau en cours, ses sous-nœuds, de couleur verte, les coups possibles du joueurs ; leurs sous-nœuds, de couleurs rouge, les coups de l'adversaire ; et ce jusqu'aux feuille de l'arbre. La taille de l'arbre correspond à la profondeur de recherche.

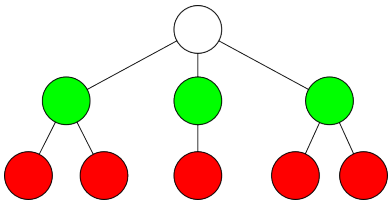

FIGURE 2.2 – Exemple d'arbre de recherche

2.2.2 Minimax

Le minimax est un algorithme de recherche simple, il affecte une valeur à chaque noeud, pour une feuille cela représente le résultat de l'évaluation du plateau et pour les autres nœud il faut distinguer deux cas :

1. le nœud est vert, il représente un coup du joueur, alors sa valeur vaut la valeur minimal de ses sous-nœuds : minimiser le prochain coup adverse ; ou
2. le nœud est rouge, il représente un coup de l'adversaire, alors sa valeur vaut la valeur maximal de ses sous-nœuds : maximiser son prochain coup.

La racine est de couleur rouge, elle représente le coup précédent de l'adversaire.

On dit que le minimax consiste à minimiser la perte maximum, c'est à dire qu'il choisit un coup maximisant le gain du joueur tout en minimisant celui de l'adversaire.

Exemple de l'utilisation du minimax On commence par affecter des valeurs aux noeuds feuilles en utilisant une heuristique déterminant la qualité, ici, du plateau de jeu.

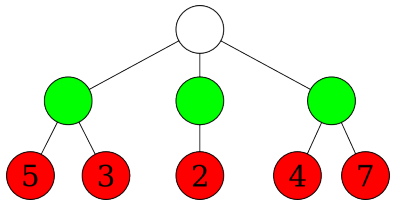

FIGURE 2.3 – Minimax étape 1

Ensuite on remonte leurs valeurs sur leur nœud parent, ici un nœud vert, on choisit donc la valeur minimale $valeur = min(sousnoeuds)$.

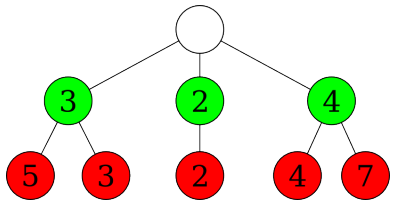

FIGURE 2.4 – Minimax étape 2

Finalement on arrive à la racine, toujours rouge, ou l'on choisit la valeur maximale $valeur = max(sous_noeuds)$.

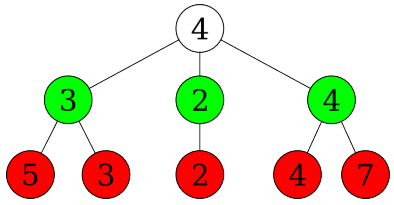

FIGURE 2.5 – Minimax étape 3

Nous avons donc déterminé qu'il faut jouer le 3^{e} coup, de valeur 7.

Optimisation Pour optimiser le parcours de l'arbre on effectuera un parcours en profondeur de l'arbre, affectant au fur et à mesure les valeurs des nœuds.

2.2.3 Negamax

Le negamax est une variante du minimax, qui se sert d'une propriété des jeux à somme nulle, pour en simplifier le code. Grâce au fait que maximiser f équivaut à minimiser $-f$, principe du dualité, on a :

$$max(a, b) = -min(-a, -b)$$

on peut donc retirer le partie conditionnelle de l'algorithme et effectuer uniquement une fonction max sur l'opposé de la valeur, d'où le nom de negamax.

2.2.4 Élagage alpha-beta

Pour des jeux complexes tels que les échecs, le go ou le Gygès il n'est pas possible de représenter l'intégralité de l'arbre de recherche, on se limite donc à une profondeur donnée, mais ce n'est pas la seule technique possible pour réduire le temps de parcours de l'arbre de recherche.

L'élagage alpha-beta est un algorithme visant à réduire le nombre de nœuds évalués par un algorithme minimax, pour cela il limite son exploration de l'arbre de recherche en n'évaluant pas les nœuds qui ne contribueront pas au calcul du gain de la racine. Plus exactement il ne va pas explorer plus profondement une branche dont le résultat sera forcement inférieur à un nœud déjà évalué.

L'algorithme alpha-beta tient son nom de ces deux paramètres alpha et beta représentant respectivement le minimum souhaité et le maximum espéré. Donc lorsqu'on tombe sur un coup ayant une valeur moins élevée (resp. plus élevée) que alpha (resp. beta) alors on sait que ce nœud ne participe pas au calcul et donc peut être élagé.

2.3 MTD-f

2.3.1 Présentation

Le MTD-f est une amélioration des algorithmes de types minimax permettant de réduire le temps de calcul d'un coup mais augmente l'utilisation totale de mémoire.

L'évaluation de plusieurs milliers, voire millions de plateaux, selon la profondeur, rend le calcul d'un coup très long. Or nous avons besoin de

réduire au maximum le temps passé à évaluer les plateaux.

Cependant en évaluant l'ensemble des plateaux, nous allons évaluer plusieurs fois les mêmes plateaux, ce qui est une perte de temps. Nous pourrions donc utiliser des tables de transposition pour garder en mémoire la valeur d'un plateau et ainsi ne pas le réévaluer.

Tables de transposition Les tables de transpostion correspondent à des tableaux associatifs reliant un couple (plateau, profondeur) à l'évaluation du plateau.

L'avantage des tableaux associatifs est que l'ajout et la lecture d'éléments est en $O(1)$

2.3.2 Algorithme

Le pseudo-code de la fonction AlphaBetaAvecMémoire est disponible en anglais sur le site `http://people.csail.mit.edu/plaat/mtdf.html`

Voir figure 2.6 pour le pseudo-code du MTD-f.

Différences Le minimax est l'algorithme le plus simple pour résoudre le problème d'un arbre de recherche, il fonctionne en effectuant un parcours en profondeur de l'arbre. Il fut amélioré avec l'élagage alpha-beta notamment.

Le MTD-f est différent de l'élagage alpha-beta, en effectuant uniquement des recherche à fenêtre nulle cela lui permet d'élager bien plus et donc d'être plus rapide, mais cela ne lui permet pas d'avoir autant d'information, seulement une fenêtre sur le résultat. Pour avoir le résultat exact MTD-f appel donc un algorithme alpha-beta plusieurs fois jusqu'à trouver la bonne valeur.

Le coup lié à la réexploration d'une partie de l'arbre de recherche, dû aux multiples appels à un algorithme alpha-beta, est compensé par l'utilisation de table de transposition évitant la réévaluation de nœuds déjà évalués.

Une recherche à fenêtre nulle correspond à l'algorithmes alpha-beta où le paramètre alpha vaut beta-1.

```
fonction MTDF(racine, f, d)
    g = f
    limiteMax = +∞
    limiteMin = −∞

    tant que limiteMin < limiteMax alors
        si g = limiteMin alors
            β = g + 1
        sinon
            β = g
        fin si

        g = AlphaBetaAvecMémoire(racine, β − 1, β, d)

        si g < β alors
            limiteMax = g
        sinon
            limiteMin = g
        fin si
    fin tant que

    retourner g
fin fonction
```

FIGURE 2.6 – Pseudo-code de l'algorithme MTD-f

3 | Heuristiques

Un des points important d'une intelligence artificielle de jeu est de pouvoir attribuer une note à un coup pour pouvoir ensuite déterminer le meilleur coup possible à jouer. Pour cela on utilise des heuristiques.

Nous allons donc voir comment nous avons trouvé nos fonctions d'évaluation, puis nous expliquerons leur fonctionnement.

3.1 Méthodologie de recherche

Valeur d'un coup Pour établir des fonctions heuristiques nous avons du déterminer une manière d'évaluer un plateau, c'est à dire une façon de noter un coup afin de pouvoir déterminer un meilleur coup parmis l'ensemble des coups possibles. Pour cela nous avons essayé diverses stratégies contre des intelligences artificielles déjà existantes ainsi que contre d'autres joueurs plus ou moins expérimentés. Nous avons ensuite gardé celles qui étaient efficace dans la plupart des cas.

Plateau symètrique Puisque le plateau est symètrique, nous avons décidé de noter uniquement les plateaux appartenant du joueur Nord. Ainsi pour évaluer les plateaux du joueur Sud nous effectuons une rotation du plateau.

Soit P le plateau initial :

$$P = \begin{pmatrix} 1 & 2 & 1 & 2 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 2 & 1 & 1 & 2 \end{pmatrix} + (1, 0)$$

alors Q sera sa rotation :

$$Q = \begin{pmatrix} 2 & 1 & 1 & 2 \\ 0 & 0 & 4 & 0 \\ 0 & 4 & 0 & 0 \\ 1 & 2 & 1 & 2 \end{pmatrix} + (0, 1)$$

Mouvements asymètriques Le fait que le jeu soit asymètrique offre plusieurs façon d'évaluer les plateaux. Il est ainsi possible d'évaluer uniquement sur les points positifs (resp. négatifs) du joueur ou bien, de prendre la différence des points positifs (resp négatifs) du joueur et de son adversaire. Dans le cas de la différence, ce calcul est possible car

$$valeurJ_1 \neq -1 * valeurJ_2$$

Cependant, cela implique une double évaluation du plateau qui augmente donc le temps de calcul et diminue la profondeur de recherche en un temps fixe.

3.2 Heuristiques

Il est possible de créer des fonctions d'évaluations selon plusieurs critères, certaines étant plus rapide, d'autres permettant plus facilement d'être élagué et enfin certaines donnent des résultats plus précis.

3.2.1 Basic eval

Cette heuristique a pour but d'éviter d'avoir les pièces $n \in \{1, 2, 3\}$ sur la n^e ligne du joueur tout en essayant de les placer sur la n^e ligne adverse. En effet, si un pion de taille 2 est sur la deuxième ligne d'un joueur, ce joueur risquera de perdre dès lors que le pion précédent sera accessible à son adversaire.

On ne pénalisera pas les pièces $n \in \{2, 3\}$ sur la $(n-1)^e$ ligne puisqu'elles sont plus difficile d'accès et ont moins de chemin possible vers la base adverse. De plus cette version de l'heuristique fonctionne très mal à faible profondeur, puisque la plupart des coups ont la même valeur.

Enfin, comme toute les autres heuristiques, elle favorise fortement la victoire tout en pénalisant la défaite.

$$
\begin{aligned}
f(x,i) &= \begin{cases} 1 & \text{si } x = i+1 \\ 0 & \text{sinon} \end{cases} \\
base(P) &= \begin{cases} 10000 & \text{si la base sud de P est occupée} \\ 0 & \text{sinon} \end{cases} \\
value(P) &= base(P) + \sum_{i=0}^{2} \sum_{j=0}^{5} f(P_{i,j}, i) \\
basicEval(P,Q) &= value(P) - value(Q)
\end{aligned}
$$

FIGURE 3.1 – Formule de basic eval avec P la matrice représentant le plateau et Q la matrice représentant le plateau retourné.

3.2.2 Distance eval

Le but de cette heuristique est de faire avancer les pions le plus possible afin de se rapprocher de la base adverse pour créer des opportunités. Il est en effet évident que plus les pions sont proche de la base adverse, et par conséquent loin de la notre, plus les chances de victoires sont grandes.

$$
\begin{aligned}
f(x,i) &= \begin{cases} i & \text{si } x \neq 0 \\ 0 & \text{sinon} \end{cases} \\
base(P) &= \begin{cases} 10000 & \text{si la base sud de P est occupée} \\ 0 & \text{sinon} \end{cases} \\
value(P) &= base(P) + \sum_{i=0}^{5} \sum_{j=0}^{5} f(P_{i,j}, i) \\
distanceEval(P,Q) &= value(P) - value(Q)
\end{aligned}
$$

FIGURE 3.2 – Formule de distance eval avec P la matrice représentant le plateau et Q la matrice représentant le plateau retourné.

3.2.3 Maxpath eval

Cette heuristique est la plus complexe des trois, puisqu'elle note un plateau en fonction du nombre de chemin disponible. Cela implique donc de pouvoir calculer tout les chemins possible à partir des pions de notre ligne et de celle de l'adversaire.

Posséder plusieurs chemin permettant d'accèder à plusieurs endroits similaire est une des clefs de la réussite au Gygès. Si vous construisez deux chemins distincts menant à la base adverse alors la victoire est assurée.

$$
\begin{aligned}
score(C) &= \sum_{i=0}^{5}\sum_{j=0}^{5} \begin{cases} i^{C_{i,j}} & \text{si } C_{i,j} \neq 0 \\ 0 & \text{sinon} \end{cases} \\
base(P) &= \begin{cases} 10000 & \text{si la base sud de P est occupée} \\ 0 & \text{sinon} \end{cases} \\
value(P) &= base(P) + \sum_{m \in M} score(C_m) \\
maxpathEval(P,Q) &= value(P) - value(Q)
\end{aligned}
$$

FIGURE 3.3 – Formule de maxpath eval avec M l'ensemble des pions déplacables pour le joueur, C_m l'ensemble des chemins atteignables à partir du pion m, P la matrice représentant le plateau et Q la matrice représentant le plateau retourné.

3.3 Perceptron

Trouver une fonction d'évaluation correcte est difficile, il est cependant possible d'en approximer une par apprentissage via un perceptron.

3.3.1 Le perceptron multicouche

Le perceptron multicouche est composé d'un réseau de neurones, reliés ensemble par des liens pondérés, organisé en plusieurs couches, d'une fonction d'activation non linéaire, telle que la tangente hyperbolique, et d'un algorithme de rétropropagation du gradient de l'erreur pour permettre de résoudre des problèmes non-linéaires.

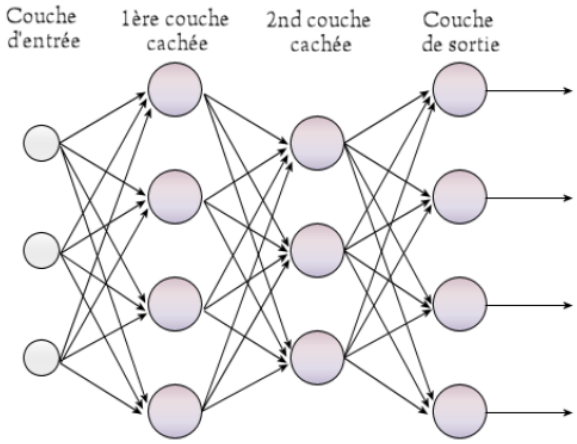

FIGURE 3.4 – Exemple d'un perceptron multicouche

3.3.2 Abandon

L'approximation d'une fonction par un perceptron multicouche nécessite de fournir des données déjà traitées. Tout d'abord, il est difficile de déterminer les plateaux intéressant et étant donné la quantité de plateau a traité. En effet il serait impossible de le faire sans fonction d'évaluation au vu de la limitation de temps du projet. Nous avons donc décidé d'abandonner l'implémentation du perceptron.

4 | Statistiques

4.1 Cadre expérimental

Machine de test Les statistiques sur le projet ont été effectuées sur une machine dédiée disposant de 1.75Go de RAM et un processeur 1GHz 64bits, avec java version 7u3-2.1.6-1 et linux version 3.2.0-3-amd64.

quels tests

utilisation des résultats

4.2 Valeur des fonction d'evaluation

4.2.1 Nombre moyen de nœuds explorés selon la profondeur

Un des points important d'une fonction d'évaluation est sa capacité à créer des valeurs qui vont être élagées, en effet moins de nœuds sont évalués plus rapide sera le choix du coup. Nous avons donc comparé le nombre de nœuds évalué en fonction de la profondeur pour chacune des fonctions d'évaluation.

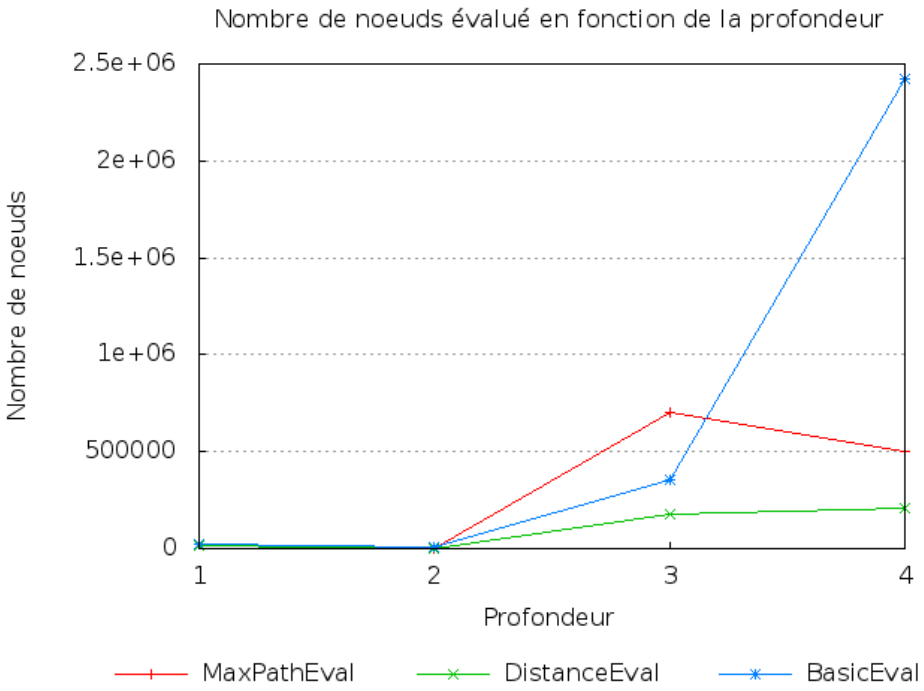

FIGURE 4.1 – Nœuds/profondeur

On remarque qu'aux profondeurs 1 et 2 le faible nombre de nœuds évalués ne permet pas de différencier les différentes fonctions d'évaluation. À profondeur 3 et 4 la taille de l'arbre de recherche permet un plus grand élagage. Distance eval est la fonction qui permet le plus grand élagage, suivie par Maxpath eval. On découvre que Basic eval s'élague très mal, cela est dû au fait que les valeurs crées ne sont pas assez distinctes.

4.2.2 Temps moyen de calcul selon la profondeur

Un autre point important d'une fonction d'évaluation est sa vitesse, en effet on évalue jusqu'a plusieurs millions de nœuds, la vitesse est donc cruciale. Pour ce test nous avons mesuré le temps utilisé par la fonction sur une partie et nous l'avons divisé par le nombre de coups.

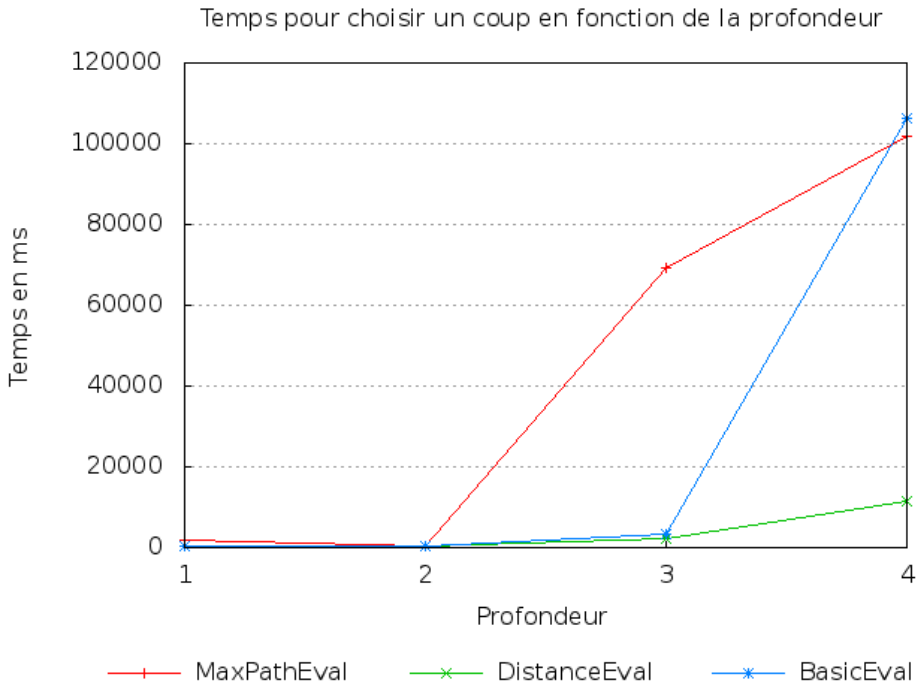

FIGURE 4.2 – Temps/profondeur

On remarque de nouveau que Distance eval est la fonction la plus rapide, notamment dû au faible nombre de nœuds évalués ainsi qu'à sa simplicitée. Pour Basic eval le temps est aussi directement lié au nombre de nœuds évalués. Enfin pour MaxPath eval on observe que malgré un nombre plus faible d'évaluations le temps requis à la profondeur 4 est plus grand, probablement dû à la complexité de la fonction.

4.2.3 Nombre de coup par milliseconde selon la profondeur

En faisant le ratio des deux premières statistiques, nous avons donc le nombre de nœuds évalués par milliseconde en fonction de la profondeur.

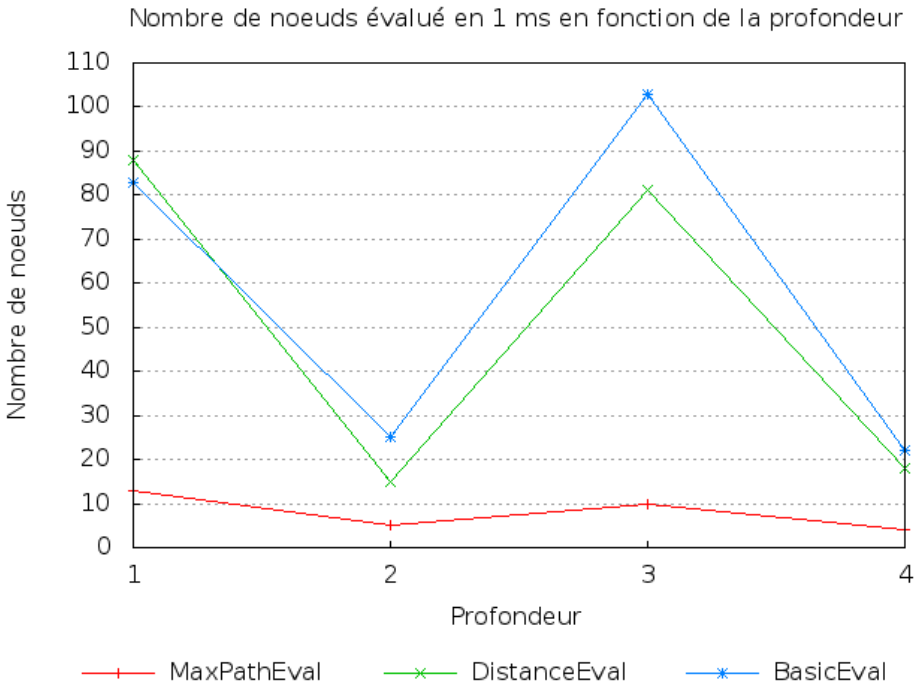

FIGURE 4.3 – Coups/profondeur

On observe que MaxPath eval a un nombre assez constant quelque soit la profondeur, ce qui n'est pas le cas de Basic eval et Distance eval, mais leurs valeurs n'oscillent que entre 20 et 100, ce n'est donc pas vraiment influent sur la capacité des fonctions.

4.2.4 Nombre moyen de coups pour gagner selon la profondeur

Au final le point le plus important d'une fonction d'évaluation est sa capacité à gagner. Nous avons donc fait combatre les fonctions d'évaluation contre une fonction jouant au hasard et nous avons regarder le nombre de coups nécessaire pour gagner.

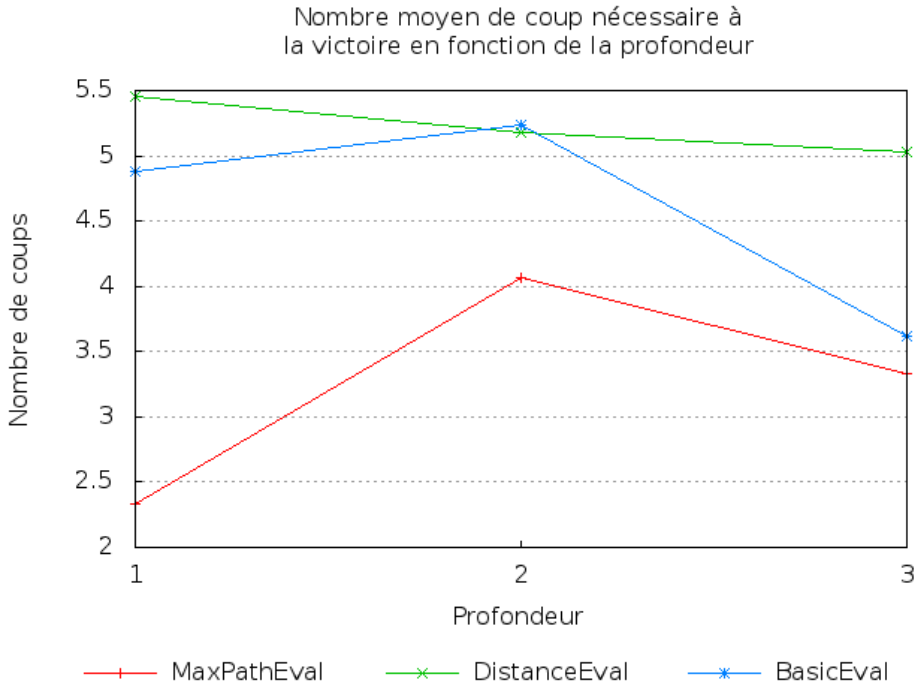

FIGURE 4.4 – Coups/profondeur

On note que MaxPath eval qui est la fonction la plus complexe est aussi la fonction la plus performante, nécéssitant que de 2 à 4 coups pour gagner. A partir de la profondeur 3 Basic eval gagne aussi rapidement. Mais Distance eval, la fonction la plus simple et la plus rapide, a les scores les plus mauvais, nécéssitant plus de 5 coups pour gagner.

4.3 Comparaison des fonction d'éval

4.3.1 Nombre victoires à profondeur fixe

Nous avons aussi comparer les fonctions entre elles, pour avoir des résultats ayant plus de signification que contre une evaluation jouant au hasard.

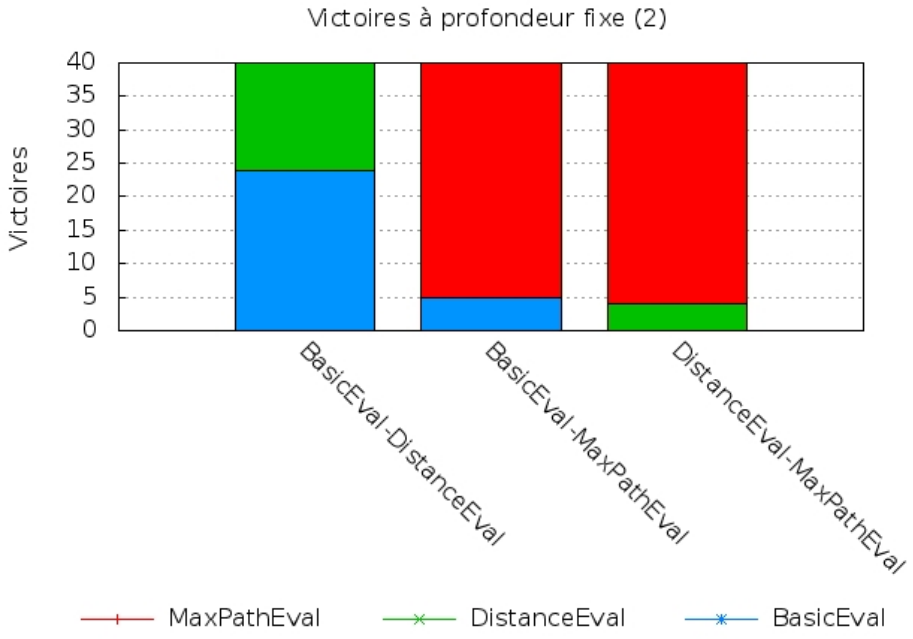

FIGURE 4.5 – Victoires

On constate que Basic eval et Distance eval ne sont pas loin du 50/50, cela confirme les résultats précédents qui montraient une équivalence entre les deux fonctions. On voit aussi que MaxPath eval est bien la fonction la plus performante avec un nombre de victoire contre Basic eval et Distance eval élevé.

5 | Conclusion

5.1 Problèmes

Nous avons rencontré des difficultés à trouver des fonctions d'évaluation pertinentes, en effet le jeu de Gygès étant un jeu complexe que nous ne maîtrisons pas suffisament, il fut difficile de trouver une stratégie de jeu efficace et de la retranscrire en fonction mathématique.

5.2 Limites du projet

Nous n'avons pas eu le temps d'optimiser les différents algorithmes de recherche ni les fonctions d'évaluation, ce qui aurait permis d'augmenter la vitesse, ou la profondeur, de recherche.

Aussi il n'est pas possible d'annuler un coup ou de voir comment l'IA a joué (chemin de déplacement du pion) à cause du temps limité du projet.

5.3 Perspectives

On pourrait envisager l'ajout de nouveaux algorithmes de recherche, non basé sur le minimax, et comparer leur performances avec le MTD-f.

Il pourrait aussi être intéressant d'améliorer l'interface graphique pour pouvoir choisir les algorithmes et fonctions d'évaluation directement depuis l'interface. Voir aussi d'ajouter un mode multijoueur via internet.

Finalement le rajout d'indications sur la qualité du coup que l'on vient de jouer serait un plus pour l'entrainement au Gygès, mais cela requiert la connaissance complète de l'arbre de recherche et est donc impossible.

5.4 Bilan

Ce projet fut l'occasion pour nous de réaliser un projet de a à z, comprenant un cahier des charges à prendre en compte ainsi que des limites de temps à respecter.

Cela a été l'opportunité de travailler en équipe et d'utiliser un gestionnaire de version.

Cela nous a aussi permis d'implémenter des algorithmes et concepts vu en cours et de mieux comprendre leur utilisation dans un projet de grande taille.

Finalement ce projet nous a permis de mieux comprendre les principes et difficultés de la création d'une intelligence artificielle de jeu.

Annexes

A | Glossaire

Chemin
Ensemble de pions permettant de se déplacer via rebond.

Cycle
Ensemble de coups remettant le jeu dans sa position précédente.

Élagage Alpha-Beta
Algorithme visant à réduire le nombre de nœuds évalués par un algorithme minimax en excluant les nœuds qui ne contriburont pas au calcul du gain de la racine.

Intelligence artificielle
Système informatique mimant les capacités de réflexions d'un être humain. Ici fait référence à la partie du programme émulant un joueur.

Facteur de branchement
Nombre de plateaux possible depuis un plateau départ.

Fonction d'évaluation
Fonction prenant en entrée un plateau de jeu et sortant la valeur de ce plateau.

Heuristique
Algorithme qui fourni une approximation d'un résultat de façon plus rapide qu'un algorithme exacte.

Horizon
Limite de calcul dans l'arbre de recherche. Fait référence au fait qu'un très bon coup peut se cacher derrière le dernier coup calculé, mais qu'il est impossible de le savoir.

Jeu combinatoire abstrait
Type de jeu où deux joueurs ou équipes d'affrontent et jouent à tour de rôle, tous les éléments sont connus et où le hasard n'intervient pas.

Minimax
Simple, cet algorithme assigne à un nœud la valeur minimale ou maximale de ses nœuds fils, respectivement lorsque le nœud resprésente le tour de l'adversaire ou le tour du joueur. Pour un nœud feuille sa

valeur correspond à la valeur retournée par la fonction d'évaluation pour le plateau représenté par ce nœud. Cet algorithme cherche donc à minimiser les pertes tout en maximisant les gains.

MTD-f
: Une optimisation du *minimax*, le *MTD-f* se sert de tables de transposition pour garder en mémoire la valeur d'un plateau et ainsi ne noter qu'une seule fois les plateaux identiques. C'est une des variantes du *minimax* la plus rapide.

Negamax
: Une variante du *minimax*, en se basant sur le fait que $max(a, b) = -min(-a, -b)$ et que la valeur de coup pour le joueur adverse et la négation du coup pour le joueur courant, pour simplifier l'algorithme en n'utilisant que des *max* au lieu d'une alternance *max*, *min*.

Neurone
: Un neurone artificiel est composé d'entrées pondérées, plus le poids est fort plus l'entrée influence la valeur de sortie. Les valeurs des entrées sont combinés grâce à une fonction de combinaison, le plus souvent la fonction somme suivante : $net_j = \sum_{i=0}^{n} x_i w_{ij}$. Enfin la valeur de sortie du neurone est calculée de la façon suivante : $O_j = \tanh(net_j)$. L'utilisation de la tangente hyperbolique, une fonction Sigmoïde, permet une activation non linéaire et possède une sortie dans l'intervale -1 à 1.

Perceptron multicouche
: Réseau de neurones formel organisé en plusieurs couche utilisant une fonction de sortie de type sigmoïde. Utiliser comme classifieur non linéaire ou pour la régression de fonction inconnue.

Table de transposition
: Permet de conserver en mémoire l'évaluation d'un nœud et ainsi éviter une réévalutation coûteuse.

B | Crédits photographiques

Deep blue, Figure 1.1
James the photographer
`http://flickr.com/photos/22453761@N00/592436598/`

Plateaux de Gygès
`http://www.gyges.com/`

Perceptron multicouche, Figure 3.4
Wikipedia
`http://fr.wikipedia.org/wiki/Perceptron_multicouche`

C | Manuel

C.1 Ligne de commande

On utilise le logiciel en utilisant la commande `java -jar gyges.jar`, cette commande prend plusieurs paramètres pour régler la partie.

On peut choisir la fonction d'évaluation utilisée avec les paramètres `--eval1 NomDeLaClasse` et `--eval2 NomDeLaClasse`, permettant de changer la fonction pour le joueur 1 et le joueur 2. Sont disponible les fonctions suivante : *BasicEval*, *DistanceEval* et *MaxPathEval*.
C'est la fonction basic eval qui est utilisé par defaut si aucune fonction n'est passée en paramètes ou si elle est incorrecte.

De même on peut choisir l'algorithme de recherche avec les paramètres `--ai1 NomDeLaClasse` et `--ai2 NomDeLaClasse`. Sont disponible les algorithmes suivant : *NegamaxMover* et *MtdfMover*.
C'est l'algorithme MTD-f qui est utilisé par defaut.

Ces algorithme prennent en paramètre la profondeur de recherche qui peuvent être configurés avec les paramètres `--depthJ1 Profondeur` et `--depthJ2 Profondeur` ou avec `--depth Profondeur` pour régler les deux en même temps.
La valeur par défaut est de 2.

Il est possible de définir les positions de départ avec les paramètres `-pos1 Position` et `-pos2 Position` où *Position* est une suite de six chiffres représentant les pions.

C.2 Interface graphique

On peut lancer une interface graphique avec l'option `--ui`, dans ce cas on peut rajouter des joueurs humains en passant la valeur `human` à un ou aux

deux paramètres `--ai`. La création de joueurs humains passent forcement le jeu en mode interface graphique.

Le joueur humain sélectionne un pion avec un clic droit de souris, effectue un rebond avec un clic de molette et pose le pion avec le clic gauche.

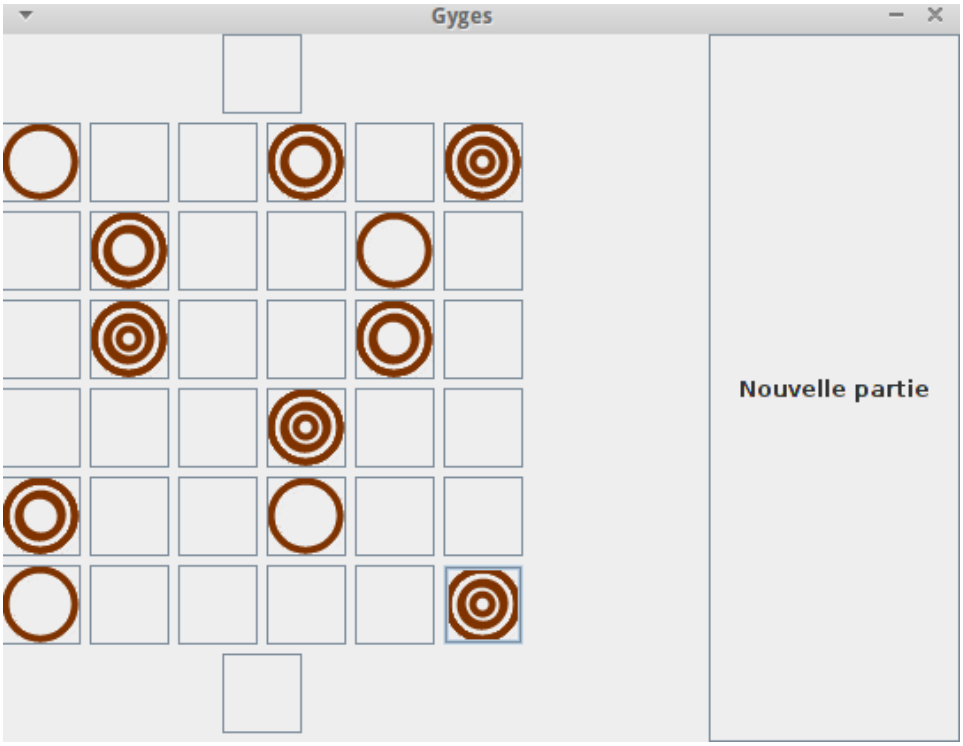

FIGURE C.1 – Capture d'écran du jeu