Pré-rapport du projet de Gygès

Valentin Lemière - Guillaume Desquesnes

1 Présentation

Pour ce projet, il nous a été demandé de créer une intelligence artificielle pour le jeu de Gygès.

1.1 Le jeu de Gygès

Présentation Le Gygès est un jeu de stratégie combinatoire abstrait, c'est à dire un jeu à deux joueurs jouant 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 6x6, 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.

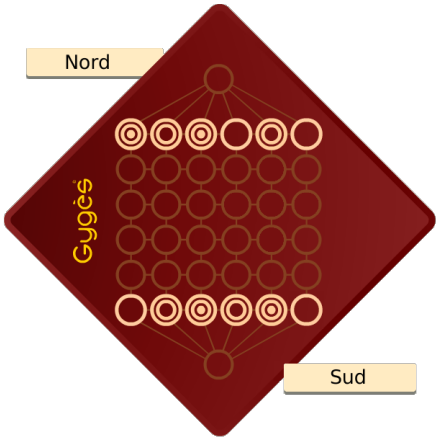

Fig. 1: Exemple de plateau de Gygès

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 » :

– Un pion se déplace d'autant de case que ça valeur, ni plus ni moins ;

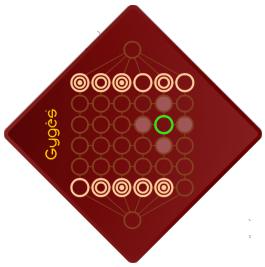

FIG. 2: Déplacement d'un pion 1

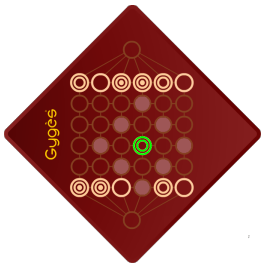

FIG. 3: Déplacement d'un pion 2

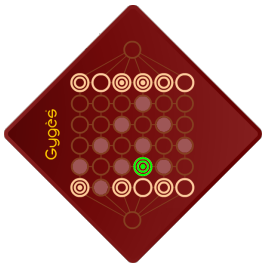

FIG. 4: Déplacement d'un pion 3

FIG. 5: Déplacement des pions

- Un pion peut se déplacer sur les lignes et les colones, mais pas les diagonales ;
- Un pion ne peut pas passer par une case occupée ;
- Si la dernière case du mouvement d'un pion est libre alors il se pose dessus et le tour du joueur s'achève ;
- Sinon le joueur a deux possibilités :

  Rebondir Le pion rebondit sur le pion d'arrivée d'autant de case que ce dernier a d'anneaux ;

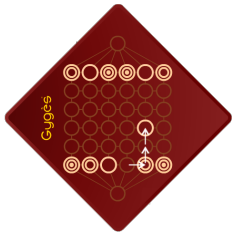

FIG. 6: Exemple de rebond

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.

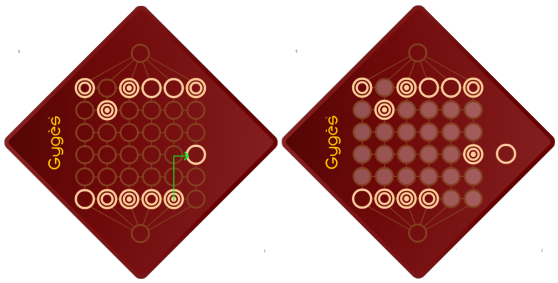

FIG. 7: Exemple de remplacement

– Un pion ne peut passer sur une ligne que dans un sens ;
– Un pion ne peut pas passer par une base, il ne peut que y aboutir ;
– 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.

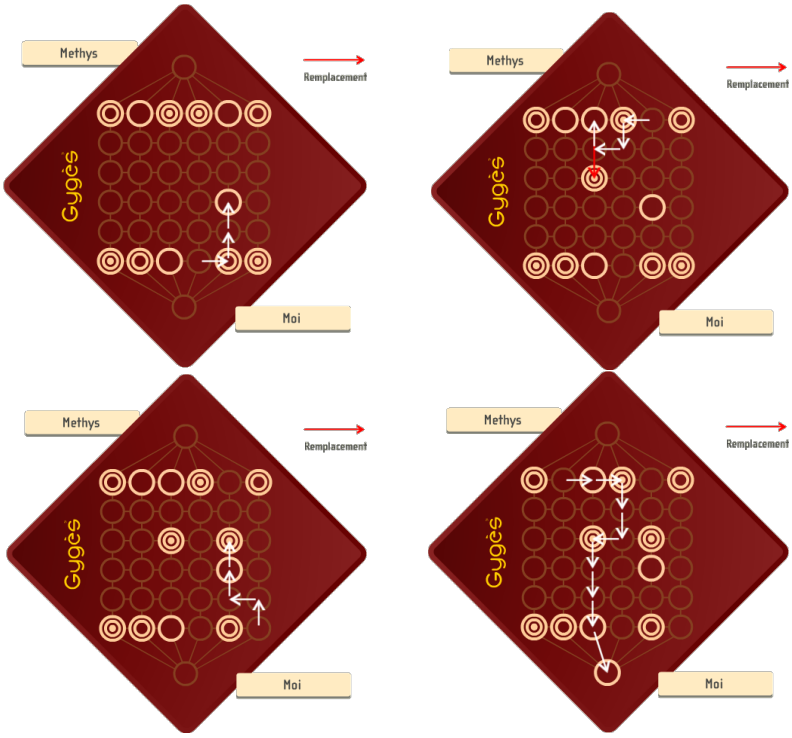

FIG. 8: Example de déroulement d'une partie de Gygès

Nous développons ce projet pour Grégory BONNET. Il pourra être utilisé par des joueurs souhaitant s'entrainer ou jouer seul.

1.2 Objectifs

Nous avons comme objectifs :

– De commencer par étudier le jeu de Gygès pour pouvoir le modéliser :
  – lister les règles spéciales ;
  – coder le mouvement des pions ;
  – déterminer quels pions chaque joueur peut déplacer.

– Ensuite de créer un module de jeu permettant le déroulement d'une partie :
  – création d'un plateau ;

  - initialisation des IA ;
  - déroulement tour à tour jusqu'a la victoire d'un des deux joueurs.

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

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

- créer une fonction d'évaluation de plateau ;

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

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

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

- développer une interface graphique pour pouvoir jouer facilement contre une IA.

2 Le point de départ

2.1 Logiciels

À ce jour, nous n'avons trouvé qu'une seule application permettant de jouer contre une intelligence artificielle au Gygès, on peut la trouver sur le site suivant : `www.gyges.com`. Cependant, cette application n'est pas capable de trouver tout les coups. De plus cette application n'ayant pas son code source disponible, il nous est impossible de l'utiliser pour ajouter et tester différents algorithmes, dans le but de trouver le plus adapté pour résoudre les problèmes du Gygès.

Nous n'avons donc aucun composant sur lequel baser notre projet, il va falloir tout développer.

2.2 Algorithmes

Il existe aujourd'hui divers algorithmes permettant de trouver un bon coup dans un jeu, il va donc falloir déterminer quel sera le plus adapté. Voici les algorithmes qui nous semblent les plus pertinants, d'après nos recherches, pour notre problème :

Minimax Un des plus simples est le *minimax*, 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.

Search EXtension Une amélioration du *minimax* est le *Search EXtension*, cet algorithme permet d'avoir une profondeur de recherche variable. Plus le coup semble intéressant, plus la branche sera explorée et au contraire, moins le mouvement est intéressant, moins on explorera la branche.

Élagage Alpha-Beta Une optimisation du *minimax*, l'élagage *Alpha-Beta* permet de réduire le nombre de nœuds évalués. L'algorithme évite d'évaluer des nœuds dont on est sûr que leur qualité sera inférieure à un nœud déjà évalué, il permet donc d'optimiser grandement l'algorithme minimax sans en modifier le résultat.

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*.

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.

MCTS La recherche arborescente Monte-Carlo est une recherche où l'on construit un arbre de recherche nœuds à nœuds en fonction du résultat de parties simulées aléatoirement.

3 Le point d'arrivée espéré

3.1 Ce que nous allons réaliser

Nous allons réaliser un logiciel doté d'une interface graphique permettant à un ou deux joueurs de jouer au jeu de Gygès. L'interface comportera un menu permettant de sélectioner l'intelligence artificielle que le joueur veut affronter, pour jouer il suffira de cliquer sur le pion à déplacer puis de cliquer sur sa destination. De plus pour chaque intelligence artificielle, il sera possible de régler la profondeur de recherche ou le temps maximal alloué pour permettre de doser la difficulté.

Enfin il sera possible de générer des parties IA contre IA sans interface graphique afin de pouvoir avoir accès à des données de jeu qui sont facilement exploitables.

3.2 Ce que nous allons développer

Nous allons développer un moteur de jeu, celui ci permettra de générer les coups légaux à partir d'un plateau et de vérifier si un coup est légal, c'est à dire qu'il est possible de passer du plateau de départ à celui d'arrivé.

Nous définirons aussi plusieurs fonctions d'évaluation de plateaux, permettant ainsi d'avoir plusieurs comportements possible pour l'intelligence artificielle et/ou plusieurs vitesses de calcul.

Enfin, nous implémenterons plusieurs algorithmes de recherche permettant de déterminer le meilleur coup à jouer d'après notre intelligence artificielle pour une profondeur de recherche donnée, afin de pouvoir trouver lesquels se prettent le plus aux problèmes que pose le Gygès : très fort facteur de branchement et règles atypiques.

Nous allons, dans un premier temps, développer les algorithmes Negamax et MTD-f, avec un élagage Alpha-Beta ainsi qu'un perceptron multicouche comme l'une des fonctions d'évaluations.

3.3 Comment cela répond-il aux objectifs ?

Ce produit permettra le déroulement d'une partie, quelque soit le nombre de joueur humain. Il pourra lister l'ensemble des coups possible à partir d'un plateau et vérifier si un coup est légal.

De plus, il contiendra plusieurs fonctions d'évaluation aux caractéristiques variées, performantes sur le temps ou sur la qualité de l'évaluation. De même, il possèdera plusieurs algorithmes de recherches variés performants d'un point de vue temporel ou d'utilisation de la mémoire.

Enfin, il possèdera une interface graphique qui permet de jouer facilement au jeu.

3.4 Ce que nous avons déjà développé

Nous avons déjà développé le moteur de jeu ainsi qu'un système générique permettant d'implémenter aisément différents types d'algorithme de recherche, ainsi qu'un système similaire pour les fonctions d'évaluations.

Nous n'avons implémenté qu'un seul algorithme de recherche pour le moment : un algorithme *negamax* avec élagage alpha-beta.

Nous avons développé plusieurs fonctions d'évalution se basant sur la position des pions sur le plateau, indépendament les uns des autres ou non. Nous avons ainsi pu déterminer que trouver une fonction d'évaluation pertinante est très compliqué. C'est pour cela que nous allons implémenter un perceptron multicouche, qui est un algorithme permettant de régresser une fonction inconnue, ici celle d'évaluation du plateau de Gygès, grace à un apprentissage à base d'analyse de parties déjà jouées.

4 Organisation du travail

4.1 Techniques utilisées

Nous utilisons une technique de développement agile : *l'extreme programming*. Cette méthode se base sur des cycles de développement rapides, ce qui nous permet d'avoir de fréquentes versions fonctionnelles et donc d'avoir des retours régulier du client sur le projet et de pouvoir ainsi s'adapter facilement à ses nouveaux besoins.

De plus cela s'adapte bien à notre programation en binôme.

4.2 Répartition du travail

D'un point de vue recherche, chacun recherche différents algorithmes permettant d'accèlerer le temps de recherche alpha-beta, ainsi que différentes fonctions d'évaluation de plateaux.

Du point de vue de l'implémentation, nous travaillons majoritairement ensemble, sur un seul ordinateur. Cela nous permet de corriger et de trouver plus rapidement les bugs, ainsi que d'optimiser et harmoniser le code.

De plus cela facilite aussi la compréhension globale du code afin que chacun puisse aisément implémenter de nouvelles fonctionnalité de manière séparée.

De même nous nous efforçons de rendre le code le plus modulaire possible afin de pouvoir facilement rajouter de nouvelles fonctions d'évaluation ou de nouveaux algorithmes de recherches.

4.3 Problèmes

Plusieurs problèmes se posent, tout d'abord il est difficile de trouver une fonction d'évaluation pertinante pour noter un plateau selon un joueur, le Gygès étant asymètrique, qui ne soit pas trop complexe car cette fonction sera appelée un très grand nombres de fois.

Ensuite il nous faut implémenter des algorithmes de parcours, cependant ces algorithmes d'apparence simple deviennent complexe lorsqu'il s'agit de les implémenter. De plus il est difficile de déterminer si un algorithme est adapté pour résoudre le Gygès, il faut donc le tester pour savoir si il est adapté ou non.

Enfin il faudra optimiser le code et les algorithmes au maximum de nos capacités, afin de reduire la complexité et la mémoire utilisée, pour soit reduire le temps de calcul ou augmenter la profondeur de recherche.

5 Conclusion

En conclusion, ce projet est avant tout un projet de recherche visant à trouver et tester différents algorithmes de recherche de coup ainsi que des fonctions d'évaluations sur le jeu de Gygès. Il nous faudra effectuer de nombreux tests pour classer les algorithmes selon leur temps de calul et les fonctions d'évaluation selon leur performances et leur capacité à faire gagner le joueur. L'extreme programming est un modèle de développement qui s'adapte bien à la nature itérative de notre projet.

A Lexique

Algorithme Search EXtension
: Une amélioration du *minimax* est le *Search EXtension*, cet algorithme permet d'avoir une profondeur de recherche variable. Plus le coup semble intéressant, plus la branche sera explorée et au contraire, moins le mouvement est intéressant, moins on explorera la branche.

Élagage Alpha-Beta
: Une amélioration du *minimax* est le *Search EXtension*, cet algorithme permet d'avoir une profondeur de recherche variable. Plus le coup semble intéressant, plus la branche sera explorée et au contraire, moins le mouvement est intéressant, moins on explorera la branche.

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.

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*.

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.

Recherche Monte-Carlo (Monte-Carlo Tree Search)
: La recherche arborescente Monte-Carlo est une recherche où l'on construit un arbre de recherche nœuds à nœuds en fonction du résultat de parties simulées aléatoirement.

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

B Sources

Toutes les images de plateau de Gygès proviennent du site : `www.gyges.com`.

C Tests

Nous avons effectué un test comparant le temps d'exécution d'un coup, suivant les fonctions d'évaluation, en fonction de la profondeur de recherche.

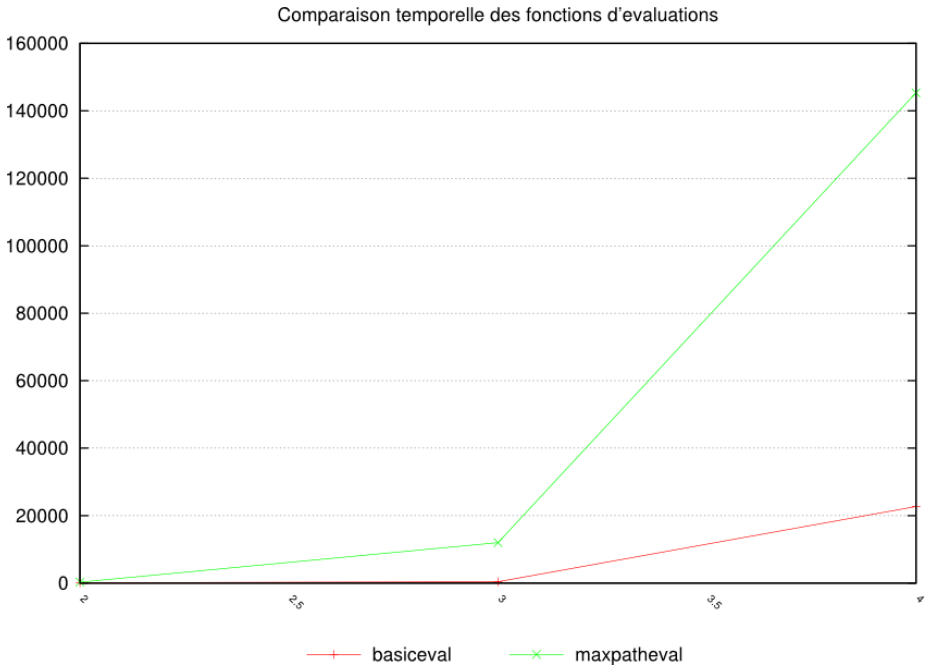

FIG. 9: Comparaison des fonctions d'évaluations

On remarque donc que la seconde fonction voit son temps requis augmenter exponentiellement avec la profondeur, elle est donc inadaptée à notre projet de Gygès car il nous faut une fonction d'évaluation rapide pour pouvoir tester un très grand nombre de plateaux.