# Intelligence artificielle pour le jeu de stratégie Pogo

-

Rapport de projet de l’UV IA41



Luc CADORET

Tristan GIERCZYNSKI

Belkacem LAHOUEL

## SOMMAIRE

[Introduction 3](#_Toc389668703)

[I – Comment jouer à PoGo 4](#_Toc389668704)

[1) Le but 4](#_Toc389668705)

[2) Les déplacements autorisés 4](#_Toc389668706)

[II – Fonctionnement de l’intelligence artificielle 6](#_Toc389668707)

[1) Problématique amenée par le jeu 6](#_Toc389668708)

[2) La représentation d’un état au sein du fichier prolog 7](#_Toc389668709)

[3) Les prédicats de manipulation des états et des coups 8](#_Toc389668710)

[a) Passer d’un état à un autre 8](#_Toc389668711)

[b) Rechercher tous les états/coups possibles d’un joueur 8](#_Toc389668712)

[4) L’évaluation d’un état 8](#_Toc389668713)

[5) L’algorithme minmax 8](#_Toc389668714)

[a) Méthode de recherche dans l’arbre de jeu 8](#_Toc389668715)

[b) Rapidité d’exécution, grâce à l’élagage alpha-bêta 8](#_Toc389668716)

[III – Quelques situations concrètes 9](#_Toc389668717)

[Conclusion 10](#_Toc389668718)

# Introduction

# I – Comment jouer à PoGo

### Le but

L’état initial d’une partie de Pogo se présente de la façon suivante : neuf cases, 6 pions noirs, 6 pions blancs.



Pour comprendre le but du jeu, il faut comprendre la notion de « couleur de pile ». A Pogo, une pile appartient à un joueur si le pion qui est tout en haut de cette dernière appartient au joueur. Ainsi, une pile peut comporter des pions noirs ou blancs, mais seul le pion au sommet définira l’appartenance de cette pile.

Un joueur ne peut jouer qu’avec les piles qui lui appartiennent. Le but du jeu est donc de posséder toutes les piles du plateau (ou la pile, puisqu’il peut n’en rester qu’une seule).

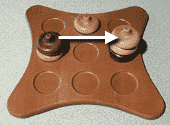
### Les déplacements autorisés

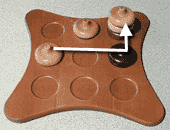
Pour parvenir à ses fins, les joueurs ont le droit de déplacer leurs pions selon des règles précises :

* On ne peut prendre qu’un, deux ou trois pions en même temps
* On peut déplacer des piles de pions, la longueur du déplacement doit être égale au nombre de pions dans la pile qu’on déplace.
* Lors de ce déplacement, la ligne droite n’est pas forcée : on peut effectuer 1 coude (si 2 ou 3 pièces prises), ou 2 coudes (si 3 pièces prises). Faire deux coudes revient à se déplacer d’une seule case.

Voici les images correspondant aux déplacements possibles :



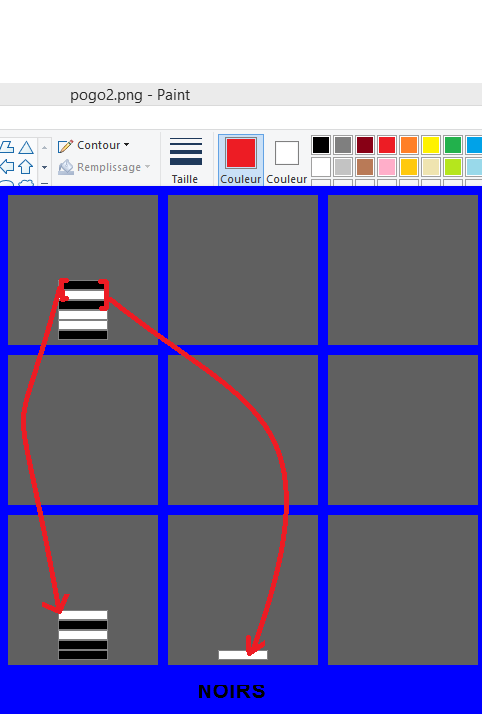
 

# II – Fonctionnement de l’intelligence artificielle

## Problématique amenée par le jeu

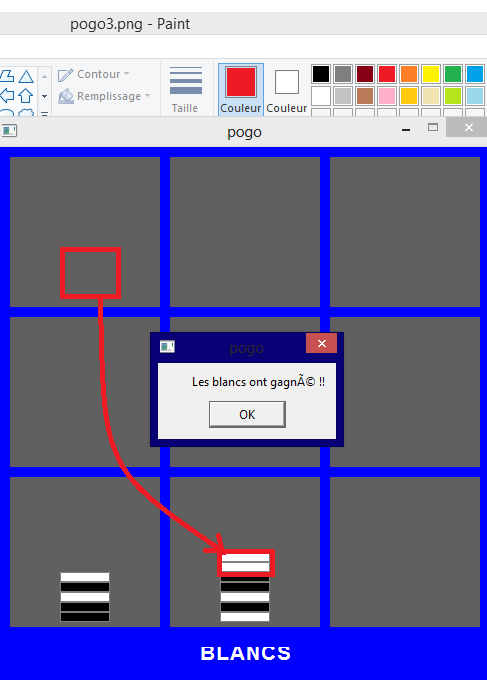
Bien que relativement simples, les règles apportent cependant une très grande dimension stratégique. Evaluer un état du jeu peut s’avérer plus complexe que prévu : il n’y a pas seulement la couleur de la pile qui importe, mais aussi les pièces qui se libèrent quand on déplace la pile. Bien vite, on atteint une problématique : comment évaluer la valeur d’une pile ? Une pile qui m’appartient, mais qui contient beaucoup de pions adverses a-t-elle beaucoup de valeur ? A-t-elle plus de valeur qu’une plus petite pile, qui contient seulement des pions à moi ?

Voici un des cas précis qui nous a permis de nous rendre compte que la mise en place d’une évaluation juste était primordiale.



*C’est le tour du joueur noir, il a deux choix (enfin, plus, en réalité, mais considérons seulement ces deux-là). S’il choisit d’aller sur le pion blanc le plus en bas à gauche, il prendra donc deux pions, ce qui lui laissera la possession de la pile la plus en haut à gauche, et lui fera gagner la possession de la pile de destination. En revanche, le pion blanc en bas à droite, bien que tout seul, représente un danger pour les noirs au tour suivant.*

Supposons donc qu’il décide de prendre la pile non pas en bas à gauche, mais celle en bas à droite :



Ce choix de déplacement aura libéré une pile de trois pions pour les blancs, et mènera le joueur noir directement à sa perte.

C’est là toute la stratégie du Pogo : il y a énormément de possibilités différentes. Le joueur noir aurait même pu forcer le blanc à jouer en faisant un mouvement totalement différent des deux possibles présentés, comme aller tout en haut à droite… etc.

## La représentation d’un état au sein du fichier prolog

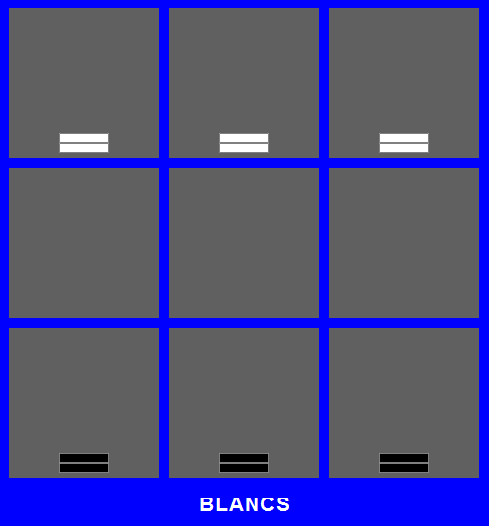
Dans les fichiers C++, contenant l’interface graphique, les piles sont représentées par des listes : le pion le plus en bas de la pile est le premier élément de la liste. La pile appartient donc au joueur qui possède le pion à la toute fin de la liste.

Dans Prolog, nous avons choisi de représenter cette liste dans le sens inverse, afin de diminuer les calculs : pour chaque case, le premier élément de la liste est le pion tout au-dessus. Cela offre une plus grande rapidité de calcul :

* l’index du pion qui sera la base de la pile qu’on souhaite déplacer, correspond aussi à la taille de la pile.
* On n’a pas besoin de parcourir toute la pile en partant de la fin pour connaitre quels seront les 1, 2 ou 3 éléments qui seront dans la pile qu’on veut déplacer.

Ce genre d’amélioration, même si elle peut paraître minime, s’avérera bien pratique lorsqu’il faudra effectuer des milliers de calculs d’états.

Voici la représentation de l’état initial dans le C++, ainsi que dans le fichier prolog :

  
L’état initial, vu depuis l’interface graphique

C:\Users\Luc\Pictures\prologinit.png

L’état initial, vu depuis le fichier prolog. Les 1 représentent les pions du joueur blanc, les 0 ceux du joueur noir. Les -1 servent à indiquer à prolog que c’est la fin de la liste des pions sur la case.

## Les prédicats de manipulation des états et des coups

### Passer d’un état à un autre

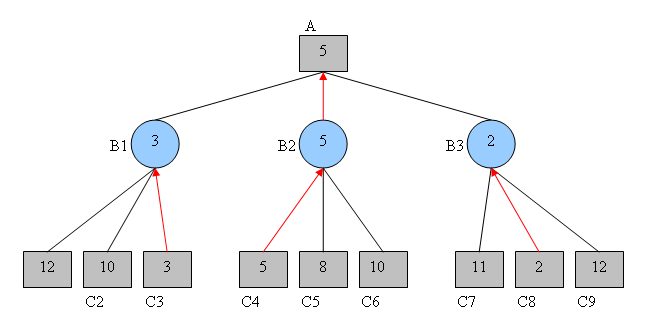
### Rechercher tous les états/coups possibles d’un joueur

## L’évaluation d’un état

## L’algorithme minmax

### Son fonctionnement

Pour parvenir à faire une IA correcte, nous avons décidé d’implémenter l’algorithme Minmax, relativement simple à comprendre. Il consiste à descendre tout en bas de l’arborescence (jusqu’au moment où il n’est plus possible d’obtenir un nouvel état, OU si on a atteint la profondeur renseignée), puis faire remonter le meilleur état jusqu’à la racine, sachant qu’un des joueurs va décider de minimiser l’évaluation de cet état (dans notre programme, les noirs), et l’autre va chercher à la maximiser (les blancs). En voici un exemple (les carrés maximisent, les ronds minimisent) :



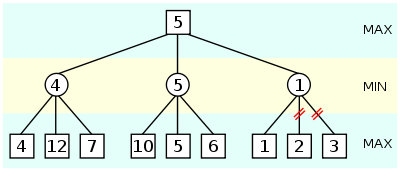
Les B sont choisis par rapport aux C minimums, et le A est choisi par rapport au B maximum.

Cet algorithme, bien que simple à comprendre, nous a été plutôt difficile à implémenter, à cause du fonctionnement de prolog : uniquement récursif, impossible de faire une simple boucle « for » pour trouver les nœuds à l’évaluation minimum ou maximum. Mais grâce aux nombreux exemples mis en ligne sur internet, nous avons réussi à en trouver un qu’on l’on a pu adapter à notre programme. Il nous a été également compliqué de le débugguer, puisque trouver la branche précise qui empêche le bon fonctionnement du prédicat relève de beaucoup de patience : un joueur peut avoir plus ou moins de 16 coups possibles (à l’état initial). On atteint donc rapidement un nombre de branches énormes, et encore plus de calculs.

Nous avons cherché à implémenter la simplification de Minmax, Négamax. Quelques test se sont révélés très concluants (à une profondeur de 4, Négamax était 4 fois plus rapide que Minmax !), cependant nous ne sommes pas sûrs que Négamax soit applicable à notre jeu, puisqu’il réclame certaines conditions. Par précaution, nous avons donc gardé Minmax.

### Rapidité d’exécution, grâce à l’élagage alpha-bêta

L’algorithme Minmax n’est rien sans son élagage alpha-bêta, qui permet d’augmenter les performances de ce dernier, en y greffant un questionnement simple « Si je sais que cette branche est déjà pire qu’une branche que je connais déjà, pourquoi continuer à la visiter ? ». Couper une branche nous évitera donc de faire des calculs supplémentaires, sans changer le résultat de Minmax. Voici un exemple ci-dessous :



Là encore, il n’a pas été facile d’implémenter cet élagage, mais grâce à plusieurs exemples trouvés sur internet, nous avons finalement réussi à l’ajouter à Minmax.

Là où Minmax sans élagage possède une complexité de bd, avec b facteur de ramification, et d la profondeur, un Minmax avec élagage possède une complexité (en moyenne) de b3d/4, ce qui n’est pas négligeable pour les grosses profondeurs.

# III – Quelques situations concrètes

# Conclusion