ENST
Brique IAMR
Pascal Corpet
Alain Grumbach

Pylos®

I. Le jeu

Pylos® est un jeu de plateau produit par Gigamic. Il se joue à deux sur un plateau spécial avec 30 billes colorés : 15 billes blanches et 15 billes noires. Les règles sont relativement simples et une partie relativement rapide ce qui en fait un bon outil pour notre étude. Par ailleurs un joueur débutant acquiert rapidement un niveau suffisant élevé par une simple réflexion, mais les joueurs aguerris peuvent apprendre des coups et jouer un niveau supérieur.

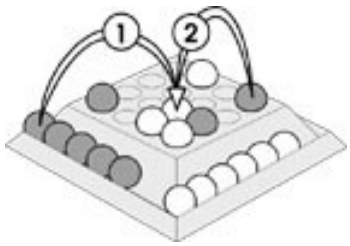
J'ai déjà essayé lors d'un travail personnel en classe préparatoire de coder un joueur intelligent artificiel qui puisse jouer contre un humain au Pylos®. Le joueur artificiel utilisait des raisonnements très simples à partir d'un grand nombre de coefficients que j'ai ensuite optimisé par un algorithme génétique. Cette approche a été plutôt décevante : n'importe quel joueur débutant était largement au-dessus du niveau de cette première intelligence artificielle. Il est donc intéressant d'essayer d'appliquer un modèle de raisonnement un peu plus général et plus intelligent. Je n'ai pu trouvé malgré des recherches appliquées sur internet d'autres exemples de modèles de raisonnement appliquées à ce problème. Mais mon modèle de raisonnement s'inspire largement de la théorie des jeux et de l'algorithme min-max vu en cours.

Le matériel :

le plateau est un support (généralement en bois) comportant 16 cases réparties en une grille carrée de quatre fois quatre. Ce plateau permet de supporter une pyramide de 30 billes comme indiqué sur l'image ci-dessus (16 billes au premier niveau, 9 au deuxième, 4 au troisième et une au sommet). Au début de la partie, le plateau est vide. Chaque joueur ne joue qu'avec les billes de sa couleur et empile chacun à son tour un bille jusqu'à construire la pyramide. Le but est de poser la dernière bille.

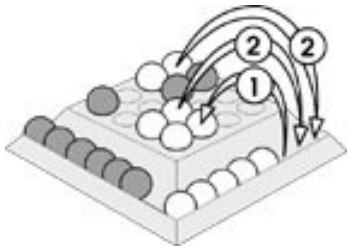
Les règles :

- Chacun est obligé de jouer à son tour de jeu.
- Un joueur qui n'a plus de billes dans sa réserve a perdu.
- On ne peut poser une bille que sur un emplacement stable.
- On ne peut ôter (voir plus loin) une bille que si elle ne supporte pas une autre.
- Empilement sur un carré.



Si un carré forme un emplacement stable libre le joueur peut choisir de poser une bille de sa réserve sur le carré (1), mais il peut aussi choisir de monter une bille libre d'un niveau inférieur (2) et ainsi économiser une bille de sa réserve.

- Carré à sa couleur.



Un joueur qui réalise un carré à sa couleur (1), reprend tout de suite soit une, soit deux de ses billes présentes sur le plateau (2) qu'il remet dans sa réserve. Cette règle peut être combinée avec la précédente : monter une bille, former ainsi un carré et retirer une ou deux billes.

Les tactiques humaines :

le jeu est très court (en moyenne une vingtaine de coups par personne) et il est difficile d'économiser des billes. Par conséquent la tactique la plus évidente consiste à se défendre au maximum et à essayer d'économiser le maximum de billes. Un débutant va donc essayer de jouer au Pylos®, comme il joue au morpion ou au Puissance 4 en empêchant son adversaire de faire un carré ou de monter une bille.

Mais le Pylos® ne se termine pas pour la première bille économisée : parfois il faut savoir laisser l'autre économiser une bille pour se retrouver en meilleure position et en économiser deux au coup suivant. La force des adversaires se mesure alors au nombre de coups d'avance que chacun est capable de « lire ». Mais personne ne peut lire plus de 15 coups à la suite, donc généralement le début d'une partie se fait souvent un peu au hasard : on essaie d'économiser des billes pour la suite tout en positionnant ses billes à des endroits plus favorables que l'adversaire.

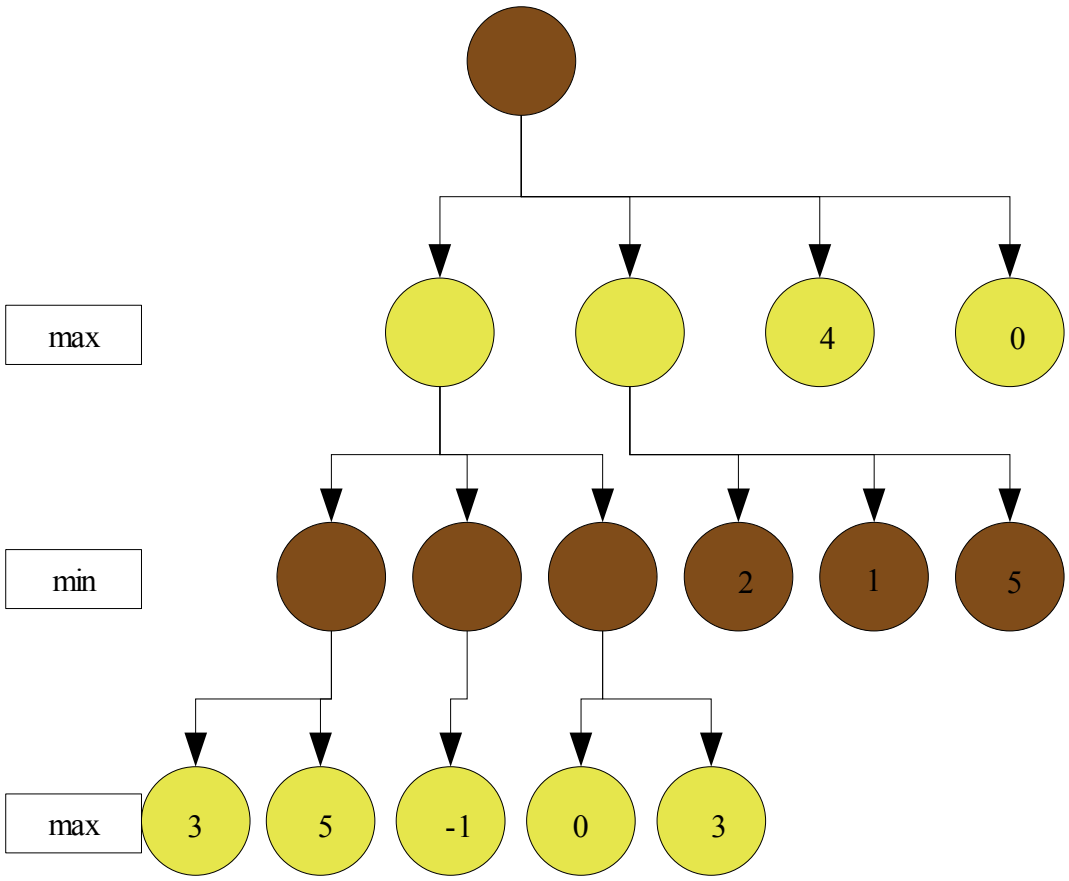
L'algorithme proposé ci-après propose d'utiliser la puissance de calcul d'un ordinateur pour lire le maximum de coups à l'avance ainsi que de rationnaliser les règles d'endroits favorables. Comme nous le verrons après nous utiliserons les règles du débutant pour minimiser la recherche en évitant d'étudier une série de coups qui nous est très défavorable dès le début. On part du principe qu'on ne peut pas remonter plus de 4 billes perdues face à un adversaire intelligent.

II. L'algorithme

L'algorithme choisit a été trouvé en réfléchissant de quelle manière un joueur humain réfléchit lorsqu'il joue au Pylos® en essayant de calquer le même modèle de raisonnement tout en profitant de la puissance de calcul et de mémorisation de l'ordinateur. Je vais donc expliquer dans cette partie la réflexion d'un être humain et l'algorithme associé.

Revenons un peu sur le détail de la réflexion d'un joueur que j'ai abordé un peu avant. J'ai dit que la puissance d'un joueur revenait aux nombre de coups qu'il pouvait prévoir à l'avance. Un joueur humain, lorsque c'est à lui de jouer, considère les différentes possibilités qui s'offrent à lui et choisit la meilleure pour lui en supposant que son adversaire jouera toujours de façon à le faire perdre. Il s'agit ici de l'algorithme du min-max :

Je cherche le meilleur coup pour moi en supposant que mon adversaire cherchera au contraire le coup qui m'arrange le moins. Si on peut donner une note ou une évaluation à chaque situation cela reviendra à prendre le maximum ou le minimum selon le niveau :



Et en effet, c'est ce que fait un joueur humain : il ne calcule pas jusqu'au bout qui va gagner ou perdre dans tel cas, mais plutôt si telle situation lui semble bien meilleure que telle autre parce que j'ai plus de billes en réserve par exemple, ou même cette situation lui semble très bonne.

Quelle est l'évaluation d'un joueur humain ? La solution la plus évidente paraît être de compter le nombre de billes que j'ai en plus de mon adversaire en réserve. Cette évaluation a deux inconvénients majeurs :

- La différence du nombre de billes en réserve est souvent faible et peut varier de 2 en un seul coup. Ce qui implique que même à une grande profondeur d'arbre comme ci-dessus, il est possible qu'en regardant un cran plus loin l'évaluation se dégrade.
- Dans la plupart des cas, les évaluations seront semblables et il sera difficile de distinguer les

coups.

Le premier problème est un problème qui se trouvera de toute manière : à part trouver l'évaluation parfaite, il est obligatoire qu'un coup qui semble un peu bon si l'on regarde à une profondeur donnée puisse devenir un peu mauvais si l'on regarde un tout petit peu plus loin. Comment fait l'être humain alors ?

Tout simplement il « sent » les bons ou les mauvais coups arriver : son évaluation est plus continue d'une profondeur à l'autre par exemple en prenant en compte non seulement la différence du nombre de billes, mais en comptant aussi le nombre d'occasions qu'il y a pour l'un ou l'autre de gagner des billes. Il faut alors évaluer une occasion par rapport à une autre pour essayer de conserver la continuité de l'évaluation. Après un grand nombre de parties voici les règles avec lesquelles il me semble que j'évalue une situation donnée en me plaçant juste avant de jouer :

1. si je n'ai plus de billes, j'ai perdu et si l'adversaire n'a plus de billes, j'ai gagné.
2. 10 points par bille d'avance.
3. 8 points si je vais pouvoir terminer un carré, et 4 points de plus pour chaque carré que je peux terminer en plus.
4. -8 points si l'adversaire peut terminer deux carrés (même en le bloquant ce tour ci, il pourra en terminer un juste après), et -4 points pour chaque carré qu'il peut terminer en plus.
5. 2 points pour chaque carré que je peux terminer avec deux billes.
6. -3 points pour chaque carré que l'adversaire peut terminer avec deux billes.
7. + ou – 3 points pour celui qui possède la bille du milieu au premier étage.

Par ailleurs, il me semble que quand je joue je ne fais pas un algorithme min-max à profondeur fixe, mais que je vais raffiner une ou plusieurs branches de l'arbre si elles me paraissent intéressantes en laissant les autre tomber dès qu'elles me semblent perdantes.

L'algorithme que j'ai implémenté essaie donc de suivre ce raisonnement : un algorithme min-max qui utilise les coupes alpha et bêta pour accélérer son traitement, une évaluation qui suit scrupuleusement les règles annoncées ci-dessus. Je n'ai par contre pas implémenté la dernière partie avec une profondeur d'arbre dynamique.

III. Le code

L'algorithme est codé en LISP. L'algorithme lui-même est optimisé pour sa rapidité, mais le code ne l'est pas ni au niveau temps processeur, ni au niveau mémoire. Les résultats que l'on peut avoir sont donc probablement plus lents que s'il était codé de manière compilé avec des optimisations de code. Le gros avantage est que l'on garde une bonne transparence du code par rapport à l'algorithme initial.

Les fonctions outils

La première partie du code consiste en de nombreuses fonctions qui servent à manipuler le plateau de jeu et à faciliter l'écriture des fonctions plus complexes. Le plateau de jeu est défini comme une liste d'étages, chaque étage étant une liste de rangées, chaque rangée étant une liste d'entiers {-1, 0 ou 1}.

- 0 représente un emplacement libre
- 1 représente une bille blanche
- -1 représente une bille noire

Un emplacement est défini par un vecteur de trois entiers : le numéro dans la rangée, le numéro de rangée dans l'étage et le numéro d'étage. Sont alors définies très simplement les fonction :

- *get_case* : donne l'entier {0, -1, 1} d'une case d'un plateau
- *plateau_vide* : renvoie une liste au format d'un plateau rempli de 0
- *set_case* : renvoie un plateau identique à celui d'origine avec la case choisie modifiée
- *otable* : vérifie qu'une bille est ôtable
- *stable* : vérifie qu'un emplacement est libre et stable
- *carre* : compte le nombre de billes total et de chaque joueur dans un carré
- *cree_carre* : vérifie si jouer à un endroit crée un carré de la couleur du joueur
- *case_premiere* : renvoie la case du haut
- *case_suivante* : renvoie la case d'après celle donnée en argument
- *case_derniere* : renvoie vrai si l'on est sorti du plateau

Les trois dernières fonctions permettent de parcourir le plateau.

Les fonctions de plateau

La deuxième partie du code met en place un cadre de jeu sans définir aucune stratégie ou une intelligence artificielle. Ces fonctions prennent en compte les règles décrites dans la première partie. Un coup est défini par une liste d'action. Les actions possibles sont :

- *POSE (x y z)* pour poser une bille
- *OTE (x y z)* pour ôter une bille

Pour jouer plusieurs coups à la suite, on peut utiliser l'action *SUIVANT* qui prend en compte le changement de joueur. On peut ainsi décrire une partie simplement en indiquant le joueur qui commence et la liste des coups joués. On définit les fonctions suivantes :

- *droit* : vérifie qu'un coup est jouable et qu'il est complet à partir d'un plateau donné

- *joue* : renvoie le plateau donné modifié par le coup donné
- *jouable* : renvoie l'ensemble des possibilités de jeu pour un joueur à partir d'un plateau donné
- *compte* : fait des statistiques sur une configuration de jeu (nombre de billes posées, nombre de billes restantes pour blanc, pour noir, nombre d'empilement sur carré possibles par blanc, par noir, nombre de carré à la couleur en cours de construction pour blanc, pour noir).
- *fin* : renvoie 1000 si blanc a gagné, -1000 si noir a gagné, faux sinon.

Les fonctions d'IA

La partie la plus intéressante est celle qui code l'algorithme décrit plus haut. Le but est de définir une fonction meilleur_coup avec pour arguments :

- plateau : la situation actuelle du plateau de jeu
- joueur : le numéro du joueur (1 ou -1) dont c'est le tour
- niveau : le niveau d'intelligence

et pour sortie une liste :

- la confiance qu'on a dans ce coup (1000 = gagné à coup sur, -1000 = perdu à coup sur)
- le coup à jouer
- le coup qu'il pense que son adversaire va jouer
- le coup qu'il pense jouer après son adversaire
- ...

Les fonctions de Jeu

Il est difficile d'utiliser la fonction meilleur_coup à la main pour chaque coup, j'ai donc créé un petit environnement (malheureusement textuelle et non graphique) pour pouvoir plus facilement faire des parties :

- *affiche_joue* : affiche un coup et renvoie le plateau modifié par ce coup
- *ia* : cherche le meilleur coup à jouer à une intelligence donnée, l'affiche et le joue
- *commande* : demande une commande interactive à l'utilisateur s'il est humain.
- *jeu* : rentre dans un mode interactif où les joueurs rentrent leurs commandes s'ils sont humains ou lance l'ia s'ils sont virtuels. Les commandes sont :
  - ➔ `QUIT` sort de ce mode interactif
  - ➔ `JOUABLE` affiche tous les coups jouable
  - ➔ `IA` n joue le meilleur coup à l'intelligence n
  - ➔ `MEILLEUR` n affiche le meilleur coup à l'intelligence n
  - ➔ `SAUVE` nom enregistre la partie en tant que 'nom'
  - ➔ `CHARGE` nom récupère une partie à partir de 'nom'
  - ➔ un coup jouable joue le coup
  - ➔ autre chose affiche le plateau en cours
- *jeu_auto* : définit la force des joueurs (humain ou niveau d'intelligence)

IV. Les conclusions

L'algorithme exposé ci-dessus ne s'est pas fait en une seule fois, et les conclusions que je tire ici prennent en compte les différents essais précédents.

Évaluations

Dans mes premiers essais, j'ai essayé d'être le plus fidèle possible à un joueur neutre : ne pas faire intervenir comment je sentais le jeu. Pour cela l'évaluation qui me semblait la moins biaisée était la différence de billes entre les joueurs. Mais cette solution n'était vraiment pas satisfaisante et se faisait rapidement avoir même par un débutant lorsqu'il ne travaillait pas à une profondeur suffisante (plus de 6), ce qui donnait des temps de calculs faramineux et donc inacceptables.

Par contre dès l'élaboration de ce min-max basique, je me suis aperçu que la parité de la profondeur de réflexion jouait beaucoup sur son efficacité. En effet puisque le Pylos® est plutôt un jeu de défense où à la moindre faille on peut prendre un retard difficilement rattrapable , il vaut mieux essayer de trouver le moins bon coup pour l'adversaire que le meilleur coup pour soi.

L'évaluation finale qui utilise les différentes règles a été facile à imaginer, mais beaucoup plus dur à préciser, notamment la force de chaque coefficient et donc de chaque règle par rapport aux autres. J'ai essayé d'affiner ces règles en jouant contre l'IA et en m'arrêtant à chaque fois que je sa logique ne me convenait pas. Pour être très performant, il faudrait appliquer une optimisation de ces coefficients par un autre processus (algorithme génétique ou autre). Par ailleurs les coefficients optimaux dépendent probablement de la profondeur à laquelle l'algorithme travaille ou de qui commence la partie.

Résultats

Quoiqu'il en soit l'algorithme ainsi formé est bon, voire très bon sur les derniers coups où il sait renverser la situation s'il existe une possibilité. A une profondeur de quatre il peut battre presque tout débutant, en commençant ou en laissant l'autre commencer avec des temps de calcul inférieurs aux temps de réflexion d'un joueur normal. A une profondeur de cinq, il est difficile de le battre sans rester extrêmement concentré. Au-delà je n'ai pas eu la patience de jouer contre lui parce qu'il est trop lent, mais à priori il est bien meilleur qu'un joueur humain (le fait de jouer à une profondeur paire l'avantage, donc le niveau 6 est bien au-dessus du niveau 5).

Lorsqu'il joue contre lui-même, le premier joueur part avec un désavantage au score, handicap qu'il garde tout au long de la partie et qu'il ne parvient pas à remonter sauf s'il joue contre plus faible que lui.

Un des problèmes classique du min-max est que le joueur artificiel va repousser les coups perdants pour lui au-delà de son horizon ce qui peut lui porter tort : aux échecs par exemple, si un de ses fous est perdu mais qu'il peut jouer autre chose, le joueur artificiel va jouer autre chose pour ne pas « voir » la perte de son fou (trop loin dans le futur). Au Pylos®, on ne retrouve pas le même problème : le plateau étant plus petit, et l'évaluation portant non seulement sur les boules mais aussi sur leurs dispositions locale et globale, on ne peut pas laisser comme cela un coup important (bon ou mauvais) dans un coin et faire bouger les choses dans l'autre.

Au final, même si le Pylos® n'est pas résolu (en gros plus de 10^{12} positions), il est très possible de le faire et donc un joueur artificiel devient rapidement meilleur qu'un humain notamment grâce à son plateau en trois dimensions qui perd un peu le joueur humain en cachant certaines boules. Par ailleurs dans notre cas nous avons essayé d'appliquer au joueur artificiel, le modèle de raisonnement humain : comme la puissance de calcul et la mémoire que l'ordinateur peut réserver au jeu est plus grande que celle d'un humain, il est certain qu'il batte son concepteur. Peut-

être aurait-il moins de chances contre quelqu'un qui joue très différemment. Mais j'en doute.

Améliorations

Voilà quelques améliorations possibles à ce projet :

- ➔ faire une interface graphique : jouer en mode texte est vraiment fatiguant surtout à un jeu en trois dimensions
- ➔ finir d'implémenter l'algorithme tel qu'il a été décrit en rajoutant le concept de profondeur dynamique ce qui le rendrait dans un premier temps moins bon, mais comme il serait plus rapide à une profondeur maximum donnée, on pourrait aller fouiller à une profondeur plus grande
- ➔ affiner les coefficients relatifs aux règles par un algorithme d'optimisation