SIM202 : Jeux d'échecs

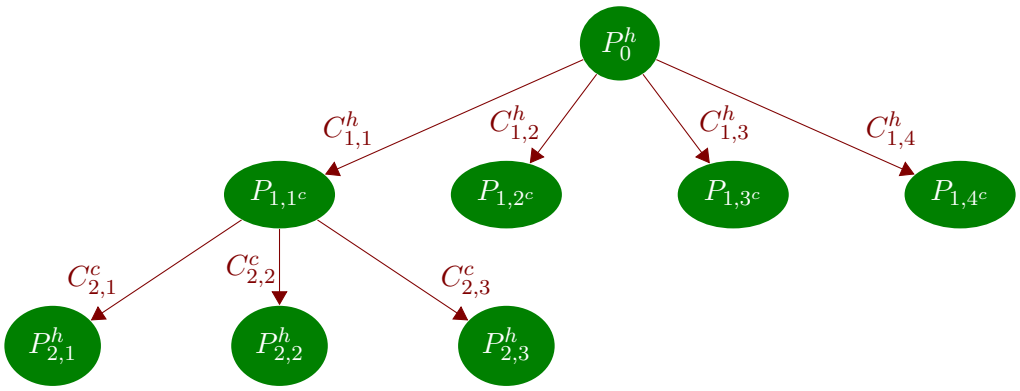
sujet proposé par Nicolas KIELBASIEWICZ : nicolas.kielbasiewicz@ensta-paris.fr

18 janvier 2022

Il s'agit dans ce projet de réaliser un jeu d'échecs permettant à un joueur humain d'affronter l'ordinateur. Ce dernier utilisera un algorithme de type min-max pour jouer. Dans une première étape, on développera une version non graphique et dans une seconde étape si le temps le permet, on développera une interface graphique s'appuyant sur les outils développés dans la première phase. La réalisation de ce projet suppose un minimum de connaissances des échecs (au moins les règles). Il n'est pas question, vu le temps imparti, de faire jouer la machine comme un grand maître ! Dans ce qui suit ne sont énoncés que les principes généraux et quelques éléments sur les classes à utiliser. Bien évidemment, il vous appartient de préciser et de compléter ces classes, voire d'en ajouter de nouvelles si le besoin s'en fait sentir.

1 Principe de l'algorithme du min-max

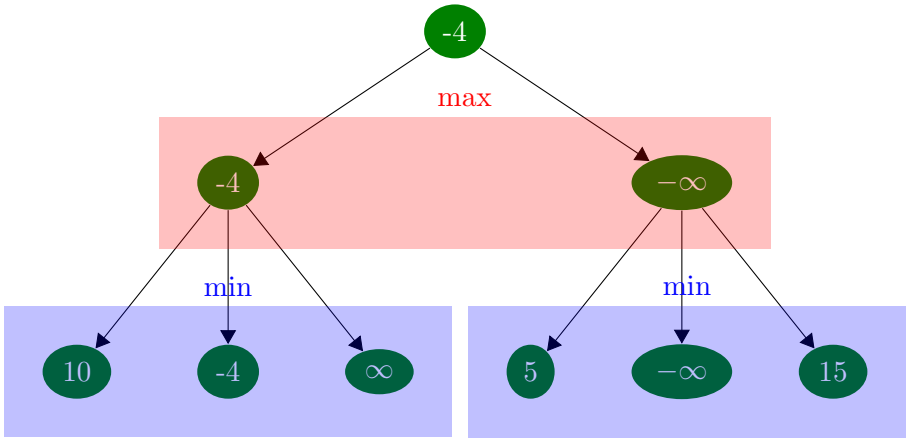
Cet algorithme n'est pas spécifique, il s'applique à tout jeu à deux joueurs au tour par tour. Un tel jeu peut être vu comme une succession de positions, résultats des coups alternés des joueurs. Nous noterons P_i^h une position que doit jouer le joueur humain et P_i^c une position que doit jouer l'ordinateur. A un instant donné de la partie, supposons que le joueur humain doit jouer à partir de la position P_0^h. Il a en général un nombre fini de coups possibles $C_{0,k}^h$ qui vont produire autant de positions $P_{1,k}^c$ que l'ordinateur va devoir jouer et ainsi de suite. Ce qui nous donne un arbre de toutes les possibilités d'évolution du jeu.



On définit maintenant une fonction valeur d'une position $V(P)$. La valeur d'une position gagnante pour le joueur humain vaut ∞, d'une position perdante $-\infty$ et d'une position de match nul 0. La valeur d'une position autre sera calculée en fonction d'heuristiques liées au jeu. Par exemple aux échecs, on pourra prendre la différence des valeurs des pièces encore présentes sur l'échiquier entre les 2 joueurs à laquelle on ajoute la différence du nombre de cases contrôlées par les 2 joueurs. On définit alors la fonction MinMax d'une position quelconque P de la façon suivante :

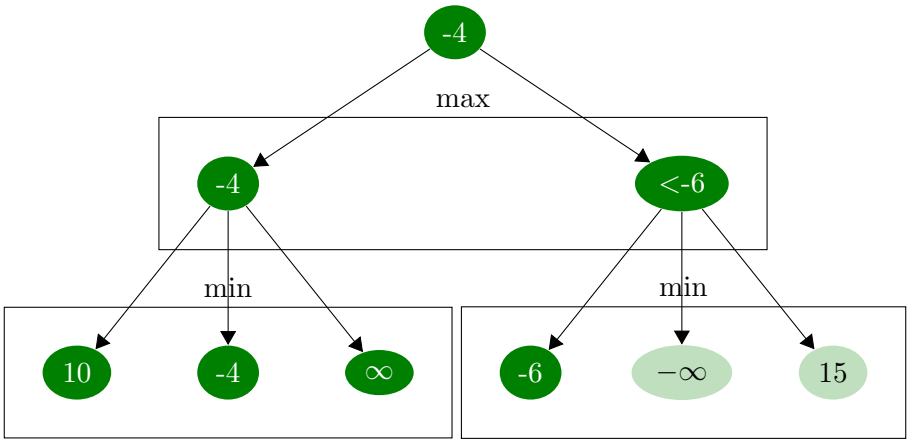
$$\text{MinMax}(P) = \begin{cases} V(P) & \text{si } P \text{ est une position terminale} \\ \max\left(\text{MinMax}(P_1), \ldots, \text{MinMax}(P_k)\right) & \text{si } P \text{ est une position du joueur} \\ \min\left(\text{MinMax}(P_1), \ldots, \text{MinMax}(P_k)\right) & \text{si } P \text{ est une position de l'opposant} \end{cases}$$

$P_1, \ldots, P_k$ désignant toutes les positions obtenues après 1 coup. L'algorithme du min-max consiste à calculer la fonction MinMax de toutes les positions issues du premier coup du joueur (humain dans l'exemple) et à jouer le coup qui donne une position maximale.



1.1 La variante $\alpha - \beta$

La variante $\alpha - \beta$ consiste à stopper l'exploration d'une branche dès lors qu'à un niveau correspondant à une phase de minimisation, on trouve une valeur inférieure à une valeur min-max du niveau précédent. En effet, on peut abandonner l'exploration car on est sûr que le joueur cherchant à maximiser sa position ne jouera pas un coup produisant une valeur inférieure à celle qu'il a déjà trouvée. De même, on stoppera l'exploration d'une branche si en phase de maximisation, on trouve une valeur supérieure à une valeur min-max du niveau précédent. Dans l'exemple suivant :



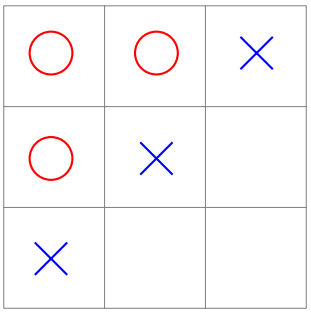
l'exploration de la deuxième branche est stoppée car -6 < -4.

1.2 Implémentation de l'algorithme du min-max

Pour écrire d'un algorithme du min-max, il suffit de disposer d'une classe `Position` qui doit au moins fournir :

- une fonction qui renvoie la valeur d'une position
- un pointeur sur la position sœur
- un pointeur sur la première position fille
- un indicateur sur le joueur qui doit jouer cette position
- générateur de toutes les positions suivantes à une profondeur donnée
- destructeur de toutes les positions filles d'une position (par récursivité, toutes les positions seront détruites)

On pourra valider l'algorithme sur un jeu très simple, par exemple le tic-tac-toe (premier à aligner 3 pions dans un carré 3x3) :



2 Application aux échecs : éléments de conception

2.1 Définition des pièces et déplacements

On commencera par définir une classe `TypePiece` ayant les attributs suivants :

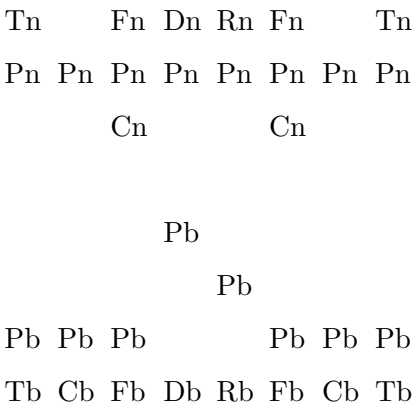
- un type valant roi, reine, four, cavalier, tour ou pion
- un déplacement relatif
- une valeur : 0 pour le roi, une valeur v arbitraire pour le pion, $9v$ pour la reine, $5v$ pour la tour, et $3v$ pour le fou et le cavalier.

On définira ensuite une classe `Piece` ayant les attributs suivants :

- le type de la pièce
- la couleur de la pièce (noir ou blanc)
- la position de la pièce sur l'échiquier

2.2 Définition de l'échiquier

On définira une classe `Echiquier` qui sera un tableau 8x8 de pointeurs de pièces. Si une case de l'échiquier n'est pas occupée, la case correspondante du tableau sera un pointeur nul, sinon, ce sera le pointeur sur la pièce occupant la case. Cela permet d'avoir en permanence une « image » de l'échiquier. On pourra adjoindre à cette classe une fonction print permettant de suivre l'évolution de la partie à l'écran ou dans un fichier texte.



2.3 Définition d'un coup

Afin de suivre le déroulement de la partie, on définira une classe `Coup`, ayant les attributs suivants :
— la couleur
— la pièce jouée
— l'ancienne position
— la nouvelle position
— s'il y a prise de pièce
— si c'est un coup spécial (roque)

Pour la position d'une pièce, on utilisera un couple (i, j). Il pourra être utile d'avoir une fonction générant le codage standard d'un coup (FouC1-D2, Dame D1-D7+,O-O,O–O), voir sur internet.

2.4 Définition d'une position

Il s'agit d'un état temporaire de l'échiquier. Comme il serait trop volumineux de stocker l'état de l'échiquier après chaque coup, on va définir une classe `Position` décrivant une position relative par rapport à un échiquier de référence (le dernier état connu de l'échiquier) en indiquant seulement la liste des coups permettant de passer de l'échiquier de référence à la position temporaire. Lorsque l'on aura besoin d'avoir tout l'échiquier, on en génèrera un temporaire. Bien évidemment, cette classe sera munie de tous les attributs et fonctionnalités nécessaire à l'algorithme du min-max.

2.5 Génération de toutes les positions

2.6 Valeur d'une position

La valeur d'une position sera ∞ pour une position gagnante et $-\infty$ pour une position perdante (échec et mat), 0 pour une partie nulle (pat) et sera donnée pour une autre position par :

$$\gamma_p \left(\text{val\_piece}(j_1) - \text{val\_piece}(j_2)\right) + \gamma_c \left(\text{cont}(j_1) - \text{cont}(j_2)\right)$$

où $\text{val\_piece}(j)$ représente la valeur des pièces encore présentes du joueur j et $\text{cont}(j)$ le nombre de cases contrôlées par le joueur j.

La première partie correspond à un critère quantitatif essentiel aux échecs et la second à un critère qualitatif sur l'occupation de l'échiquier. Les paramètres γ_p et γ_c pondèrent l'importance d'un critère par rapport à l'autre et cette formule permet à l'ordinateur de jouer un peu mieux qu'un débutant. Une grande partie de la qualité d'un jeu d'échecs par ordinateur dépend du choix de ces critères.

2.7 Définition d'une partie

— initialisation de l'échiquier

— séquence de 2 coups : pré-calcul des coups possibles pour le joueur 1 (humain), coup joué par le joueur 1, détermination du meilleur coup du joueur 2 (ordinateur) à l'aide du min-max, coup du joueur 2
— mémorisation de l'historique des coups joués et fonction d'annulation d'un coup
— affichage du déroulement de la partie

3 Validation

On validera chacune des fonctionnalités du programme. On portera une attention particulière au coût calcul. En effet, il y a en moyenne 30 à 40 coups possibles en début et milieu de partie. Si on veut que l'ordinateur joue correctement, il faut explorer au moins 4 coups (2 par joueurs), ce qui donne de l'ordre de 1 million de positions.

4 Interface graphique

S'il reste du temps, où si vous êtes 4, on pourra essayer de développer une interface graphique avec un outil comme Kylix ou Qt Creator en réutilisant ce qui a été développé précédemment.