Projet III de « Algorithmique, Structures de données et Programmation II »

Les dames

1 But du projet

Le but de ce projet est d'écrire un programme C sous UNIX permettant de jouer aux dames. Le programme devra permettre le jeu entre deux joueurs humains mais devra également gérer des affrontements humain vs. ordinateur et ordinateur vs. ordinateur via une interface réseau qui vous est fourni.

2 Règles des dames

2.1 Le but du jeu

La partie est gagnée si :
- on prend toutes les pièces de l'adversaire ;
- on bloque toutes les pièces de l'adversaire ;
- l'adversaire abandonne :-).

La partie est nulle (ou remise) si :
- aucun des deux joueurs ne peut prendre toutes les pièces adverses (par exemple 3 dames contre 1) ;
- les deux joueurs sont d'accord.

2.2 Disposition initiale

Le jeu damier se déroule sur un damier de 10 cases sur 10 orienté avec une case foncée en bas à gauche et en haut à droite.

Chaque joueur possède 20 pions placés sur les 4 premières rangées. Les joueurs jouent chacun à leur tour. Les blancs commencent toujours.

2.2.1 Disposition du Damier

Le jeu de dames international se pratique sur les cases foncées. Le damier doit être placé de telle façon que la grande diagonale foncée ait une extrémité à la gauche de chaque joueurs.

2.2.2 La Notation Manoury

Différents mode de notation, plus ou moins ingénieux et pratiques, ont été imaginés depuis la publication des premiers traités. Le système le plus simple est celui de Manoury,célèbre « damiste » du XVIII° siècle. Il consiste tout bonnement à numéroter de 1 à 50 les cases utilisées (voir la figure 1).La grande diagonale, qui doit être orienté de gauche à droite en remontant, comprend dont les cases de 46 à 5. Au début de la partie, les pions sont placés sur les cases de 1 à 20 pour les *Noirs* et les *Blancs* sont sur les cases de 31 à 50. La numérotation des cases permet de noter les opérations qui se déroulent sur le damier. Déplacements et prises sont indiqués par deux numéros, celui de la case de départ et de la case d'arrivée.

Exemples : 35–30 signifie que le pion 35 est avancé sur la case 30. 36x27 signifie que le pion de la case 36 passe à la case 27 en effectuant une prise.

Dans votre programme, un coup sera stocké dans un tableau de `char` de taille 25. La première case de ce tableau permettra de déterminer le type de message. Si cette première case contient un :
- `'B'` : cela indique un coup joué par les *blancs* ;
- `'N'` : cela indique un coup joué par les *noirs* ;
- `'Q'` : cela indique une demande de partie nulle ;
- `'P'` : cela indique une acceptation de partie nulle ;

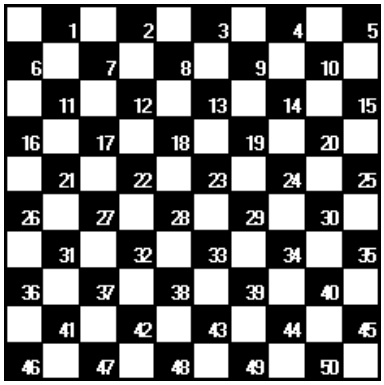

FIGURE 1 – Notation de MANOURY

— `'A'` : cela indique un abandon.

Lorsque le tableau commence par un `'B'` ou un `'N'`, les case suivantes indiqueront le coup joué de la manière suivante. La case 1 du tableau contiendra le numéro de case du damier, selon la numérotation de MANOURY, occupé par le pion qui se déplace. Puis la ou les cases suivantes contiendront les différents numéros de case du damier que le pion va traverser lors du coup. Enfin la case tableau suivant directement la case contenant la position finale du pion sera le caractère spécial `'\0'` qui permettra d'indiquer la fin du coup.

Exemples : Si un pion *blanc* se déplace de la case 42 à la case 38. Votre tableau sera de la forme :

`'B'`	42	38	`'\0'`																					

Si un pion *noir* se déplace de la case 8 à la case 20 en prenant deux pions *noirs* en case 13 et 23. Votre tableau contiendra :

`'N'`	8	19	20	`'\0'`																				

2.3 Le mouvement des pions

Les pions se déplacent d'une case vers l'avant en diagonale (on ne joue que sur les cases foncées). Un pion qui arrive sur la dernière rangée et s'y arrête est promu dame. On le couvre alors d'un pion de la même couleur.

2.4 Le mouvement de la dame

La dame se déplace sur une même diagonale d'autant de cases qu'elle le désire, en avant et en arrière. Son mouvement est seulement limité par les autres pièces sur le damier.

2.5 La prise par le pion

Pour prendre avec un pion, il faut :

1. être placé à côté d'un pion adverse ;
2. sauter par dessus le pion adverse et se rendre sur la case vide située derrière celui-ci ;
3. enlever le pion sauté ;

La prise peut également s'effectuer en arrière.

La prise est obligatoire.

Si, après avoir pris un premier pion, vous vous retrouvez de nouveau en position de prise, vous devez continuer, jusqu'à ce que cela ne soit plus possible. Les pions doivent être enlevés à la fin de la prise et non pas un par un au fur et à mesure.

2.6 La prise par la dame

Puisque la dame a une plus grande marge de manoeuvre, elle a aussi de plus grandes possibilités pour les prises. La dame doit prendre tout pion situé sur sa diagonale (s'il y a une case libre derrière) et doit bifurquer à chaque fois qu'une nouvelle prise est possible. On ne peut passer qu'une seule fois sur un même pion. En revanche, on peut passer deux fois sur la même case.

2.7 La prise majoritaire

C'est la règle la moins connue. Elle est pourtant essentielle. Lorsque plusieurs prises sont possibles, il faut **toujours prendre du côté du plus grand nombre de pièces**. Cela signifie que si on peut prendre une dame ou deux pions, il faut prendre les deux pions. La qualité de la pièce prenante ou des pièces à prendre n'intervient pas. Une dame vaut un pion.

Chaque joueur a obligation d'appliquer cette dernière règle, Si un joueur tente d'invalider cette règle, votre programme l'en empêchera en lui spécifiant la fameuse phrase : « *Souffler n'est pas jouer* ».

3 Programmer un jeu (en général) ... de dames (en particulier)

3.1 Bref historique

D'abord sont apparus les « partenaires électroniques » de jeu de dames, qui ont suivi de peu leurs homologues plus commerciaux de jeu d'échecs. Au début des années 80, ces consoles dédiées au jeu avaient des niveaux plutôt faibles. Peu après, avec la banalisation du PC (Personnal Computer), sont apparus les premiers logiciels.

Jusqu'à une date récente, les performances de ces appareils n'étaient pas à la hauteur de leurs autres avantages, en particulier leur calme (pas de crise de colère quand ils perdent), leur patience (ils vous laissent reprendre trente-six fois votre coup) et leur disponibilité (prêts à jouer même à trois heures du matin).

Les progrès ont cependant été très rapides, et il semble inéluctable que dans quelques années, plus aucun joueur ne puisse gagner de partie contre les machines.

Au jeu d'échecs, en 1996, la défaite de Kasparov contre Deeper Blue a fait couler beaucoup d'encre, et ébranlé bien des certitudes.

Déjà, aux « *checkers* » (jeu de dames sur 64 cases), le programme Chinook , né en 1988, avait été admis à participer aux éliminatoires pour le titre de champion du monde 1992, et il avait battu tous ses adversaires jusqu'en finale, où il s'était finalement incliné devant Marion Tinsley, champion du monde depuis 1954, par 4 défaites, 2 victoires, et 33 remises sur 40 parties. Cela était d'autant plus remarquable que Tinsley n'avait jusque là perdu que 7 parties officielles en 30 ans de carrière, sur plusieurs milliers jouées. En 1994, à Boston, Tinsley dut abandonner la revanche pour des problèmes de santé, et Don Lafferty, son second ne put faire mieux que match nul (10 à 10 sur 20 parties, dont 18 nulles). En 1995 à Petal (Mississipi), Chinook gagna enfin, 16.5 à 15.5 sur 32 parties, dont 31 nulles (gain de Chinook à la 31ème partie).

Il n'existe pas encore de super-ordinateur programmé pour le jeu de dames, mais les logiciels pour PC, et les PC eux-mêmes font des progrès extraordinairement rapides.

Il semble donc que pour les dames telles que nous les connaissons, la domination des machines doive être totale avant peu de temps (5, 10, 15 ans ?). Alors, comment fonctionnent ces machines qui seront bientôt nos futurs vainqueurs ? Connaître son adversaire est une des règles de tous les stratèges, et ceux qui comprendront le « raisonnement » de leur adversaire électronique pourront peut-être retarder leur échéance personnelle de défaite inéluctable contre les machines. Nous allons donc essayer de lever le voile sur ces mystérieux joueurs et leur mode de pensée, au fil des différentes étapes de ce projet.

3.2 Généralités sur l'évaluation

Nous allons d'abord présenter des généralités sur le raisonnement que font suivre les programmeurs à leurs logiciels, raisonnement constitué de ce que l'on appelle algorithmes, c'est à dire de suites d'opérations élémentaires. Vous entendrez, dans cette matière, parler également de méthodes « heuristiques ».

Ce sont des méthodes, souvent fondées sur des calculs précis, mais aussi utilisant des techniques empiriques, qui permettent de trouver des voies viables se rapprochant par étapes de la solution finale.

On peut distinguer plusieurs types d'algorithmes, utilisés simultanément par les programmes, à travers deux phases imbriquées du « raisonnement » :

1. la démarche dynamique, qui correspond aux prévisions de mouvements de l'adversaire, aux réponses de l'ordinateur, aux contre-réponses de l'adversaire, etc..., et ce, le plus loin possible dans l'avenir. Cette vision de l'ordinateur est « tactique », car elle évalue les mouvements des pions, sans vue d'ensemble.

   Si les processeurs avaient une capacité et une vitesse plusieurs milliards de fois supérieures à leurs performances actuelles, cette démarche suffirait à l'ordinateur qui, dès le premier coup, analyserait toutes les variantes possibles jusqu'au dernier coup, et jouerait à chaque tour le meilleur coup.

Hélas (ou heureusement), les ordinateurs n'en sont pas là ; il faut que la machine arrête son analyse au bout de quelques coups, et se livre alors au deuxième type d'analyse.

2. l'évaluation statique de la situation, qui correspond à la vision « coup d'oeil » que l'on peut avoir d'une position. Ainsi, indépendamment des coups que vous n'auriez pas vus, vous pouvez dire, en regardant une partie en cours, « les noirs sont mieux que les blancs ». C'est cette évaluation statique que l'on doit apprendre au programme. Cette évaluation statique est la plus facile à expliquer. Sachez aussi que c'est la moins « scientifique », car elle est faite de méthodes très subjectives (tout comme votre appréciation d'une position peut être discutée par votre voisin). On l'assimile à une vision « stratégique » de la position.

3.3 L'évaluation statique

Il s'agit en fait de quantifier objectivement l'impression subjective que chacun de nous peut avoir en jetant un coup d'oeil à une position en cours de jeu, sans se livrer à une analyse des possibilités de développement.

Vous vous êtes sans doute souvent livrés à cet exercice, en regardant par-dessus l'épaule des joueurs quelles étaient leurs positions respectives.

La subjectivité de la chose vous est sans doute apparue quand votre voisin (qui regardait par-dessus l'autre épaule) a émis un jugement différent du votre. Comment faire dire à l'ordinateur « les noirs sont mieux que les blancs » (ou l'inverse) avec un risque faible d'erreur ?

Examinons les critères que l'on va prendre en compte. Ils sont de deux sortes, souvent associées, le nombre de pions et la position.

Analysons chacun de ces critères :

3.3.1 la différence du nombre de pions.

Cela paraît facile (il suffit de compter), mais se pose là une éternelle question souvent débattue au Café du Commerce (celui qui est juste en face de la fac) : combien la dame vaut-elle de pions ?

Je vous laisse bien sûr la réponse, mais sachez que les programmeurs prennent en général une valeur voisine de trois, avec des variantes quand leur programmation le leur permet (c'est à dire quand elle accepte des nombres non entiers), souvent un peu moins de trois en début de partie, un peu plus de trois en fin de partie.

3.3.2 la position.

La position s'évalue en fonction d'un certain nombre de critères secondaires :

La valeur des cases occupées : Il s'agit là de valoriser l'occupation des cases du damier en fonction de l'importance stratégique que vous leur accordez.

Si vous voulez que votre ordinateur joue une partie classique, il sera bon de privilégier la case trois (supposons qu'il ait les noirs) ainsi que les cases centrales en début et milieu de partie (en fin de partie les choses sont bien sûr différentes), mais d'autres cases seront aussi valorisées, à des valeurs moindres.

Pour effectuer cette évaluation, il faut d'abord affecter une valeur à chaque case, en fonction de l'importance qu'on lui accorde. Ensuite, à un moment quelconque du jeu, on regardera quelle est la couleur du pion posé sur chaque case, et, si on est du côté des noirs, on ajoutera les valeurs des cases occupées par les noirs, et on retranchera celles occupées par les blancs.

La valeur associée à l'occupation par un pion blanc ou un noir n'est en général pas la même ; ainsi, la case 6 occupée par un pion noir aura une valeur positive de quelques points, tandis que, occupée par un pion blanc, elle aura une très forte valeur négative, le plus souvent opposée à celle de la case 45 quand elle est occupée par les noirs.

Pour les spécialistes des dames (du jeu évidemment pas des personnes du sexe faible), ces valeurs permettent ainsi de privilégier une position classique, ou une Roozenburg, ou une partie de flanc (difficile pour un ordinateur), etc...

Un bon programme comprendra plusieurs valeurs, en fonction du type de partie (qu'il pourra reconnaître d'après l'occupation de certaines cases caractéristiques), mais aussi en fonction de la phase du jeu (début, milieu, fin)

Ceci n'est cependant aussi simple que cela ; en effet, les valeurs choisies doivent respecter deux critères :

1. être représentatives de la valeur de la position,

2. respecter des écarts logiques entre des cases correspondant à des mouvements possibles.

Les formations : Il est souvent intéressant de donner quelques points supplémentaires aux positions qui permettent des marchands de bois, des flèches, des trèfles, ... et qui évitent, dans le doute, les lunettes. Le bon programme tiendra compte de tout cela. Là encore ces termes sont familiers aux « damistes ».

On peut aussi privilégier l'enchaînement de l'adversaire, mais ceci est particulièrement ardu, car en l'absence de jugement humain, il est souvent malaisé de définir qui est l'enchaîneur et qui est l'enchaîné, et le risque de retrouver rapidement l'enchaîneur enchaîné est difficile à faire évaluer par l'ordinateur. Inutile de préciser que si les bonnes formations de l'ordinateur ajoutent des points, celles de l'adversaire vont en enlever.

Les degrés de liberté : Le principe est que, si on ne regarde pas loin dans l'avenir, il vaut mieux avoir un grand nombre de possibilités parmi les pions à jouer (on appelle cela les degrés de liberté), que peu, car on aura alors plus de choix, et donc plus de chances de trouver une bonne solution.

Il faudra bien sûr chercher à diminuer le nombre de degrés de liberté de l'adversaire, mais il faut remarquer ici que l'analyse n'est pas symétrique. En effet, on ne peut parler de degrés de liberté que pour le joueur qui a le trait, le nombre de degrés de liberté de l'autre pouvant être modifié par le coup du premier. Dans la pratique, on ne fait pas toujours cette distinction, pour rendre possible la comparaison entre des positions qui n'ont pas le même trait.

Les temps : Cette notion ressemble aux degrés de liberté, mais elle correspond à un déroulement dans le temps, et non dans l'espace. Un bon joueur préfère toujours être sûr qu'il peut encore jouer plusieurs coups avant d'être contraint à des échanges, qui peuvent casser sa position, ou lui faire perdre des pions. Il convient donc de vérifier que notre jeu nous laisse suffisamment de temps d'avance, et en laisse moins à l'adversaire. Ce critère n'est pas toujours pris en compte par les programmes de dames, pour deux raisons :

— il est propre au jeu de dames, et les principales recherches dans le domaine de la programmation se sont surtout intéressées aux échecs, le négligeant ;

— il contient une notion importante de déroulement dans le temps, donc ne peut pas être correctement traité par une évaluation purement statique. Il convient d'ajouter à celle-ci une composante dynamique de l'évolution du jeu, ou de se contenter d'un ersatz discutable, par exemple le nombre de cases vides devant les pions, avant le contact avec l'adversaire.

Mais, le moyen le plus simple pour approcher cette notion reste le comptage des temps tel qu'expliqué par Cantaluppo dans les années 40, puis repris par d'autres :

On calcule le nombre de pions situé sur la ligne du fond + 2 fois le nombre sur la $2^{ème}$ ligne + 3 fois le nombre sur la $3^{ème}$ + ... etc. Même chose pour l'adversaire. On fait la différence (ordinateur-adversaire).

Dans un début de partie classique, il est bon d'obtenir le chiffre le plus bas possible (on dit qu'on a des temps de retard, mais le terme est mal choisi). Plus la partie avance, plus la valeur de cette différence doit diminuer, et elle est inverse à la fin.

Autres : On peut encore ajouter d'autres critères pour valoriser la position.

Nous pouvons citer par exemple le regroupement des pions (il vaut mieux bien sûr qu'ils soient ensemble qu'étalés sur le damier), l'existence ou non de couloirs faibles pouvant mener à dame, le déséquilibre des forces sur un flanc, etc...

Je vous laisse proposer d'autres critères, qui seront d'autant meilleurs que l'ordinateur pourra les évaluer sans se livrer à des calculs trop compliqués.

Quand l'ordinateur a fait toutes ces évaluations que fait-il ?

Et bien il les mélange, il les malaxe, il les cuisine, et il en sort une valeur qui lui permet de comparer les différentes positions étudiées.

En fait il peut en tirer 2 valeurs séparées, celle du nombre de pions et celle de la position, et privilégier alors totalement le nombre de pions par rapport à la position ; cela oblige alors à traiter les gambits dans la démarche dynamique, nous y reviendrons. `Dames 2020` (un très vieux programme) fonctionne ainsi, mais la plupart de ses concurrents (dont `Quidam` et `Windames`), s'inspirent des principes développés pour les programmes d'échecs où la perte d'une pièce est tolérable pour obtenir un gain positionnel, et ajoutent la valeur issue du comptage des pions à celle issue de l'analyse de la position, en leur affectant des coefficients appropriés, pour n'en tirer qu'un seul nombre (par exemple la valeur positionnelle, plus 100 fois la différence des pions). Cette méthode classique a aussi l'avantage de ne faire manipuler qu'une valeur au lieu de deux.

Vous avez bien compris que tout ceci est une alchimie qui laisse une large place aux appréciations du programmeur et de ses conseillers techniques. Mais il peut être intéressant, lorsqu'on joue contre un ordinateur, d'essayer de retrouver quels sont les aspects privilégiés et ceux négligés, et donc d'en déduire les points faibles du programme.

3.4 La démarche dynamique

L'objectif de cette démarche est d'examiner, au-delà de la situation actuelle, ce qui peut se passer dans le futur. L'idéal est de trouver, parmi les chemins possibles, celui qui mène à une victoire, imparable par l'adversaire.

Nous avons vu que pour l'instant l'ordinateur ne peut pas atteindre cet idéal dès le début, et doit se contenter de regarder dans un avenir limité.

Nous allons tenter de bien comprendre la démarche en suivant tout d'abord le raisonnement d'un programme un peu limité (pour ne pas dire demeuré). Pour se faire, nous allons représenter les choix possibles par un « arbre », et comme d'habitude cet arbre a la tête en bas et le tronc en haut, c'est comme cela que nous informaticiens (qui connaissons sans doute mal la nature) les représentons. Nous sommes d'ailleurs si incultes que nous appelons racine le sommet (la base du tronc) et noeud chaque départ de branche. Dans le cas représenté ci-dessous par le schéma n°2, il n'y a d'ailleurs qu'un noeud, confondu avec la racine.

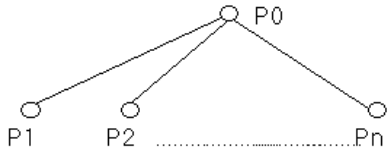

FIGURE 2 – Premier niveau

Examinons cet arbre. L'ordinateur est dans une position P_0. Il a n possibilités de jeu. La première possibilité le conduit à la position P_1, la seconde à la position P_2, ... et la nième à la position P_n.

Il « imagine » qu'il joue le premier pion, pour se retrouver dans la position P_1. Il place alors dans sa mémoire tous les pions conformément à cette position (mais, bien sûr, il ne montre rien de cela sur l'écran) et il applique à cette position l'évaluation statique que nous avons présentée précédemment. Il trouve A_1. Effectuant la même démarche pour la deuxième possibilité, il trouve une valeur A_2, ainsi de suite jusqu'à la valeur A_n de la $N^{ième}$ possibilité. Il compare alors ces valeurs A_1, A_2, ... A_n, regarde laquelle est la plus élevée (c'est à dire la plus favorable pour lui), et choisit donc de la jouer.

Nous avons fait là une démarche dynamique de premier niveau, c'est à dire que nous avons étudié un organigramme s'arrêtant au premier demi-coup. J'ai qualifié notre programme d'un peu limité, car il est évident qu'un joueur, même mauvais, poussera plus loin la réflexion et se demandera quelles sont les réponses possibles à chaque mouvement.

Nous allons donc aller un peu plus loin dans la réflexion.

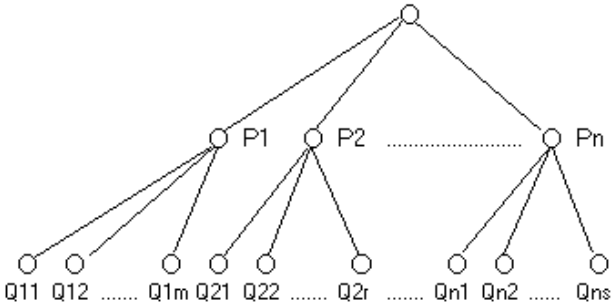

FIGURE 3 – Deuxième niveau

Nous voici devant un organigramme de deuxième niveau (schéma n°3), où chaque possibilité P est suivie de diverses possibilités Q de réponse.

Les points P et Q, d'où partent ou peuvent partir des branches de l'organigramme sont les noeuds dont nous avons déjà parlé (ils en ont d'ailleurs la tête).

L'ordinateur va évaluer chacune des positions obtenues en Q_{11},...,Q_{1m},Q_{21},...,Q_{2r}, ... ,Q_{n1},...,Q_{ns}. Vous voyez que cela fait déjà beaucoup d'évaluations statiques au niveau 2.

Il convient bien sûr d'aller plus loin, et d'envisager la réponse de l'ordinateur à son adversaire, puis de l'adversaire à l'ordinateur, etc..., et on augmente ainsi la taille de l'organigramme de 2 niveaux (celui de l'ordinateur et celui de l'adversaire) par coup.

L'organigramme prend rapidement des proportions gigantesques. Ainsi, si une moyenne de 10 possibilités par niveaux peut-être retenue, le total de positions à évaluer est de :

1. niveau 1 : 10
2. niveau 2 : 100
3. niveau 3 : 1000
4. niveau 4 : 10000
5. niveau 5 : 100000
6. niveau 6 : 1 million
7. niveau 7 : 10 millions
8. niveau 8 : 100 millions
9. etc...

Vous voyez qu'à ce rythme pourtant modéré (10 possibilités par demi-coup), on atteint le million de possibilités en 6 niveaux, c'est à dire en 3 coups, et le milliard en 9 niveaux (4 coups $\frac{1}{2}$).

Il faudra visiblement agir avec méthode (l'ordinateur a de la mémoire, mais il vaut mieux éviter de lui faire retenir et comparer des milliards de chiffres) et aussi envisager de faire de l'élagage dans cet arbre.

3.5 La méthode du Min-Max

Nous avons vu qu'il fallait, pour exploiter correctement l'arbre de décision, devenu plus compliqué et plus dense qu'une forêt, de la méthode, et une bonne scie pour élaguer.

Commençons par la méthode, qui nous évitera de nous retrouver avec des milliards de chiffres dont nous ne saurons que faire.

La méthode universellement utilisée (programmes de dames, d'échecs, de go, d'othello, de bridge,...) s'appelle le **Min-Max**.

Le Min-Max schématise le processus décisionnel que devrait suivre tout joueur : supposer que chaque joueur va, à chaque moment, prendre la décision qui est la meilleure pour lui (ceci est l'axiome de base de toute la théorie des jeux).

Ceci signifie donc que l'ordinateur, contrairement à ce que peut parfois laisser croire son jeu, ne va pas vous tenter de vous tendre un piège dans l'espoir que vous tombiez dedans. Par contre, il prendra et maintiendra des positions fortes pour lui, qui peuvent ressembler à des pièges pour vous.

Examinons un cas précis, pour comprendre comment « raisonne » l'ordinateur.

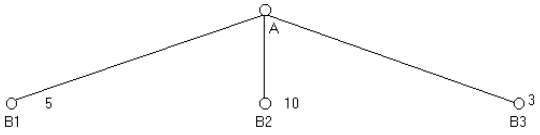

FIGURE 4 – Niveau 1

Dans cet organigramme qui s'arrête au niveau 1, nous supposons que la position initiale de l'ordinateur soit identifiée par la lettre A, et que l'ordinateur a 3 possibilités de jeu, qui l'amènent respectivement aux positions B_1, B_2, B_3. Nous avons associé à ces positions la valeur que l'ordinateur trouve en effectuant leur évaluation statique. Vous voyez que l'ordinateur peut choisir entre les valeurs 5, 10, 3, trouvées en B_1, B_2 et B_3. Il prendra naturellement la solution la plus avantageuse pour lui, c'est à dire celle correspondant à la valeur la plus élevée (ici 10 en B_2).

Examinons maintenant un organigramme de niveau 2 :

Dans ce cas, les évaluations effectuées en B_1, B_2, B_3 n'ont pas de signification, puisque aux coups suivants (amenant aux positions C_1 à C_9) ce sont d'autres valeurs qui sont trouvées, les pions ayant forcément bougé. Votre vision dans l'avenir va donc plus loin, et c'est cet avenir, le plus lointain possible, qu'il faut examiner.

L'ordinateur choisira bien sûr au début entre les coups amenant à B_1, B_2 et B_3. S'il joue le coup l'amenant à B_1, ce sera son adversaire qui choisira la réponse, amenant aux positions C_1, C_2 ou C_3. De même, si l'ordinateur joue le coup amenant en B_2, l'adversaire choisira entre C_4, C_5 et C_6, et si l'ordinateur choisit

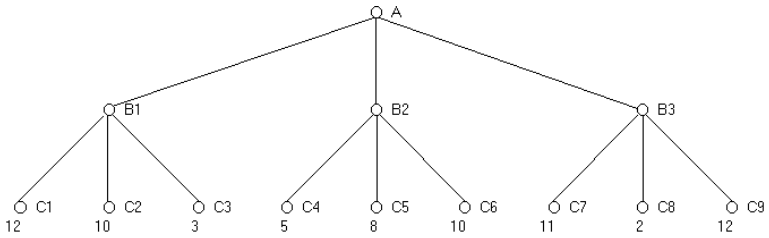

FIGURE 5 – Niveau 2

B_3, l'adversaire choisira entre C_7, C_8 et C_9. Comme l'adversaire n'est pas là pour vous faire de cadeaux, et s'il a les mêmes critères d'évaluation que l'ordinateur (ce qu'il faut bien supposer), il va choisir dans chaque cas la solution qui désavantage l'ordinateur, donc celle correspondant aux valeurs les plus faibles des évaluations statiques. L'ordinateur ne pourra donc jamais atteindre les positions tentantes telles que C_1 ou C_9.

Ainsi, si l'ordinateur joue le coup amenant à la position B_1, l'adversaire répondra par le coup amenant en C_3 qui donne une valeur de 3, si l'ordinateur choisit B_2, l'adversaire répondra par la position C_4 qui donne une valeur de 5, si l'ordinateur choisit B_3, il répondra par C_8, qui donne une valeur de 2. Nous sommes donc ramenés à un organigramme de type suivant :

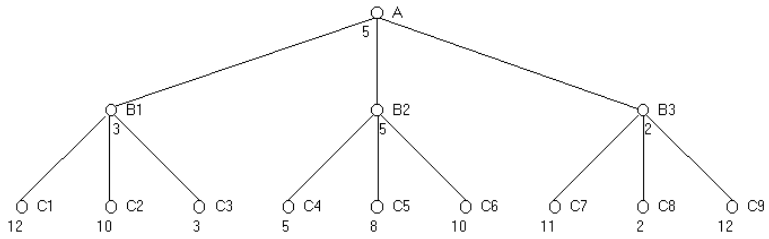

FIGURE 6 – Niveau 3

Entre les 3 valeurs correspondant à B_1, B_2 et B_3, l'ordinateur a intérêt à choisir celle qui lui donne le plus de points, donc la voie B_2, qui lui rapportera 5, son adversaire étant amené à lui répondre par le coup amenant à la position C_4.

De même, si nous poursuivions notre raisonnement au niveau 3, nous verrions que le choix du coup étant alors, en C_1, C_2, ..., C_9 du ressort de l'ordinateur, celui-ci choisirait alors dans chaque branche la position correspondant à la plus forte des valeurs possibles (voir le schéma n°7).

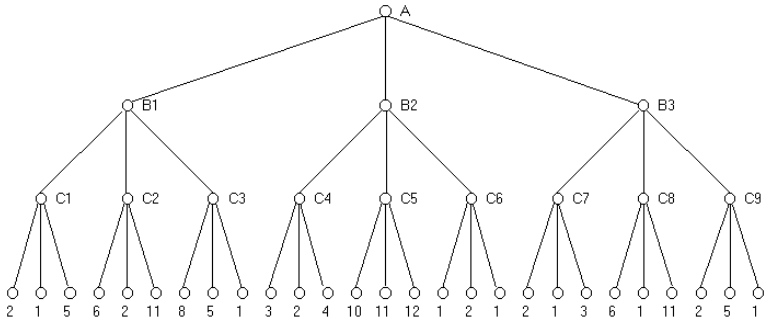

FIGURE 7 – Niveau 4

Plaçons-nous à la position C_1. L'ordinateur a le choix entre 3 possibilités, l'amenant à des positions (que nous aurions pu appeler D_1, D_2, D_3), respectivement valorisées 2, 1, 5. Il en déduit que, s'il est amené à se retrouver dans la position C_1, le meilleur coup qu'il pourra jouer l'amènera dans une position de valeur 5 points. Il peut donc d'ores et déjà affecter cette valeur de 5 à la position C_1. De même, il peut affecter 11 à C_2, et 8 à C_3.

Cependant, depuis la position B_1, ce sera l'adversaire qui choisira entre C_1, C_2 et C_3. Logiquement, celui-ci devrait prendre la solution la moins avantageuse pour l'ordinateur, c'est à dire celle issue de C_1. Ceci permet alors d'affecter la valeur 5 en B_1. De même, nous allons remonter la valeur 2 en B_2 (issue de C_6), et 3 en B_3 (issue de C_7). Le meilleur choix pour l'ordinateur est donc la branche B_1, qui lui donne un score de 5 en passant par C_1 (choix probablement imposé par l'adversaire s'il joue bien).

Ainsi, le processus utilisé par le programme correspond à l'alternance des décisions (ordinateur / adver-

saire), et alterne les niveaux dits « maximisants », où le choix du coup à jouer appartient à l'ordinateur, et les niveaux dits « minimisants », où le choix du coup appartient à son adversaire. La solution remontant au sommet est, avec certitude, le meilleur choix initial, compte tenu des meilleures réponses possibles de l'adversaire dans chaque cas.

Vous voyez que seules les évaluations de fin de branche sont nécessaires, les résultats remontant ensuite au sommet par alternance de maximisations et de minimisations.

Le Min-Max permet d'examiner avec méthode l'arbre complexe des décisions, il ne permet pas d'élaguer. Nous allons aborder les méthodes d'élagage plus loin.

Cependant, je vous convie à un petit exercice que nous corrigerons ensemble, et qui vous fera mieux comprendre la notion suivante : le magnifique, l'indispensable algorithme alpha-bêta.

A partir du schéma n°7 :

1. Vérifiez tout d'abord que, en affectant n'importe quelles valeurs aux 3 branches issues de C_5, on ne modifie en rien le choix de l'ordinateur.
2. Donnez le nombre de branches qui peuvent ainsi prendre n'importe quelle valeur sans changer le choix final de l'ordinateur (valeur 5, et chemin passant par B_1 et C_1).

Lorsque nous aurons déterminé qu'il existe des branches qui peuvent prendre des valeurs quelconques sans changer la solution, nous pourrons nous dire que nous aurions mieux fait de ne pas examiner ces branches, en effectuant un élagage adéquat.

3.6 L'Alpha-Bêta

Nous avons vu que la méthode du Min-Max nous permettait de remonter avec certitude le meilleur choix, mais nous n'avons pas résolu le problème du nombre gigantesque de possibilités à examiner.

Faut-il examiner toutes les solutions ? Et bien je vous rassure tout de suite : non, il existe une méthode miracle qui permet d'élaguer à coup sûr dans l'arbre touffu des possibilités : il s'agit de l'**Alpha-Bêta**.

Le principe est le suivant : si, sur un coup, l'adversaire a une possibilité de réponse qui rend ce coup mauvais, il est inutile d'examiner les autres possibilités de réponse à ce même coup, puisqu'il ne les choisira que si elles sont pires.

Ceci n'est peut-être pas très clair, aussi nous illustrer le propos par l'exemple précédent.

Examinons le schéma n°7.

L'ordinateur examine, comme on le lui a appris, la première extrémité de branche. Il trouve 2, qu'il remonte en C_1. La valeur suivante est 1, l'ordinateur juge inutile de la remonter puisque C_1 est un niveau maximisant et que la valeur actuelle, 2, est supérieure. La valeur suivante est 5, qu'on remonte en C_1 et qui prend la place de 2.

L'examen de la branche issue de C_1 est terminé, nous pouvons remonter provisoirement la valeur 5 en B_1, et examiner la branche issue de C_2.

La première valeur trouvée est 6, que l'on remonte provisoirement en C_2. C'est à ce point qu'intervient Alpha-Bêta. En effet, C_2 est un niveau « maximisant » (l'ordinateur choisit, et prend la meilleure solution). Donc la valeur qui sera retenue sera au moins égale au 6 déjà trouvé.

Par contre, le niveau B_1 est un niveau minimisant (l'adversaire choisit, et prend la plus mauvaise solution vue de l'ordinateur). La valeur issue de C_2 (6 ou plus) ne sera jamais retenue puisque la valeur issue de C_1 (5) est inférieure. Il est donc inutile de poursuivre plus avant l'examen des autres branches issues de C_2.

De même, la première branche issue de C_3 donne 8, ce qui implique que l'ordinateur n'ira jamais dans cette direction (il préférera la branche C_1). Cela élimine donc cette branche.

Nous avons donc terminé l'examen des branches issues de B_1. Retenons 5 comme valeur d'évaluation que nous remontons provisoirement en A.

Nous examinons les branches issues de B_2, en commençant par C_4. L'examen des 3 valeurs est nécessaire, et nous remontons en C_4 la valeur la plus élevée, 4.

B_2 étant un niveau minimisant, nous en déduisons que la valeur définitive sera inférieure ou égale à 4.

A est par contre un niveau maximisant, qui a déjà provisoirement la valeur 5. La valeur issue de B_2 (4 ou moins) ne présentera donc pas d'intérêt, car si l'ordinateur s'oriente vers là, il sait que l'adversaire peut l'amener à une position de valeur 4 (ou moins). La branche B_2 peut d'ores et déjà être éliminée, car moins intéressante que B_1, qui, si l'adversaire répond au mieux, amènera au pire à une position valant 5.

De même, les valeurs 2, 1, 3 issues de C_7 permettent de remonter 3 en B_3, et d'éliminer B_3 pour les mêmes raisons sans poursuivre l'examen des autres parties de cette branche.

Au total, nous avons examiné 11 positions, sur 27 possibles, soit un gain de 60%, et nous avons trouvé 16 positions inutiles à étudier, leurs valeurs ne changeant pas le choix final. Pour ceux qui attendent le résultat de l'exercice du précédent chapitre, je peux donc dire que 16 est la bonne réponse, et je félicite les heureux gagnants.

L'exemple que nous avons choisi était particulièrement favorable, puisque les « bonnes » valeurs étaient trouvées dès le début (en fait, nous devions arriver au résultat en un nombre d'évaluations compris entre 11 et 27). L'Alpha-Bêta permet toujours un gain de temps plus important quand les solutions les meilleures sont examinées d'abord. Cette propriété est utilisée par d'autres algorithmes, qui cherchent à ordonner les solutions selon leur valeur probable avant de les examiner vraiment. Plusieurs méthodes sont utilisées pour cela, mais je n'en citerai que deux :

1. La méthode des coups meurtriers, qui consiste à essayer en premier les coups qui ont déjà, dans d'autres branches, permis des élagages Alpha-Bêta. Ce sont souvent des départs de combinaison, que l'on retrouve plusieurs fois sauf si l'on bouche le « trou », et il est naturel de chercher à savoir si le mouvement que l'on a fait maintient ou élimine la menace qui existait auparavant ; s'il la maintient, cela permet d'éliminer rapidement la branche.
2. La recherche itérative, sur laquelle je reviendrai plus tard, dans le cadre d'un autre de ses avantages.

Le gain final dans le cas d'un arbre ordonné est appréciable : si d est le nombre de branches par niveaux, et p le nombre de niveaux, le nombre minimal d'évaluations est de l'ordre de $2 \times d^{\frac{p}{2}}$, alors que le nombre maximal d'évaluations (correspondant à toutes les branches) est d^p (soit un gain de $0,5 \times d^{\frac{p}{2}}$). Le gain généralement constaté se situe entre les deux, et est d'autant plus important que le nombre de niveaux examinés est élevé.

Différents essais effectués sur `DAMES 2020`, puis sur `QUIDAM`, qui effectuent des évaluations jusqu'à des niveaux 20, font estimer le gain moyen à plus de 80% du temps de calcul.

Quelle a été la démarche ? Et bien les calculs ont été arrêtés lorsque les valeurs remontées étaient inférieures à une borne minimale (appelée α) aux niveaux maximisants (choix du coup du ressort de l'ordinateur), ou supérieures à une borne maximale (appelée β) aux niveaux minimisants (choix du coup du ressort de l'adversaire).

Je vous conseille, pour bien comprendre cette démarche fondamentale, de faire plusieurs essais avec des valeurs différentes en bout d'arbre, et éventuellement de comparer à des évaluations menées sans effectuer d'élagage Alpha-Bêta, pour vérifier que l'élagage ne perturbe absolument pas le résultat.

D'autres types d'élagages sont possibles, mais ils sont moins « logiquement purs » que Alpha-Bêta ; certains peuvent néanmoins le compléter. Quelques uns sont décrits dans le paragraphe suivant, mais bien d'autres restent à découvrir et à implanter ci cela vous dit.

3.7 Les élagages préalables

L'élagage Alpha-Bêta permet des économies de calculs substantielles, mais on peut aller encore plus loin.

Nous examinerons plusieurs types d'élagages préalables, c'est à dire d'élagages utilisés avant toute évaluation dans la branche.

3.7.1 Les sacrifices visiblement inutiles

Quand vous jouez, il y a un certain nombre de solutions que vous éliminez rapidement (parfois inconsciemment) car elles sont visiblement trop coûteuses. Essayons d'« expliquer » à l'ordinateur ce raisonnement.

Il s'agit de lui faire éliminer une solution lorsqu'il s'aperçoit qu'elle conduit à donner ou à perdre des pions sans espoir de les récupérer à court terme. Il faut bien sûr différencier ce cas des dons de pions qui constituent des phases initiales de combinaisons. L'ordinateur peut faire cette distinction : tant que le joueur donne des pions et que son adversaire est en prise, ou n'a qu'un nombre limité de coups à jouer (1 ou 2), il y a une possibilité de combinaison ou de forcing. Dès que l'adversaire a un certain nombre de réponses possibles (plus de 2 qui ne soient pas des prises), on n'est probablement plus dans une phase de combinaison possible, et tous les pions qui ont été perdu jusque là l'ont été en vain.

Vous remarquerez que j'ai utilisé là quelques précautions, et que je me réfugie derrière des probabilités. En effet, nous ne sommes plus dans des certitudes logiques mais dans des raisonnements de type empiriques, c'est à dire fondés sur la seule expérience (l'heuristique vous dis-je). De plus, ce raisonnement présente, j'en suis conscient, un certain nombre de failles, dont deux importantes :

1. l'éventualité de gambit laissant espérer un gain ultérieur ;

2. la possibilité de donner des pions sans gain immédiat mais pour permettre, dans un avenir supérieur à celui envisagé par l'ordinateur, un passage à dame.

Ces deux difficultés peuvent être amoindries (mais non totalement annulées) en fixant assez haut la barre du sacrifice inutile (3 pions par exemple, le gambit gagnant de plus de trois pions étant tout de même exceptionnel, et un prix de quatre pions ou plus pour passer à dame étant le plus souvent cher payé, sauf peut-être en fin de partie). On peut aussi fixer une barre habituelle plus basse, qui ne monte que lorsque l'adversaire est près de la ligne de dames, à condition de savoir déterminer cette distance à dame de façon fiable.

Remarquons que ces algorithmes doivent être neutralisés en fin de partie, sinon, la dame valant environ 3 pions, l'ordinateur n'envisagera jamais de donner deux ou trois dames pour en prendre une après un temps de repos (blocage sur la grande diagonale et sur le tric-trac).

3.7.2 les solutions trop ouvertes

Nous sommes là encore dans des domaines incertains, qui jouent davantage sur les probabilités et sur l'expérience que sur une imparable logique (encore l'heuristique).

Imaginez que, à un niveau N de l'examen des avenirs possibles, l'adversaire a le choix entre 2 solutions. La première autorise 4 réponses de l'ordinateur, suivies chacune de 4 réponses de l'adversaire, chacune suivie de 4 possibilités. La seconde autorise 20 réponses, suivies chacune de 20 réponses de l'adversaire, et chacune suivie de 20 possibilités.

Si les évaluations finales vues du temps présent sont équivalentes dans chacun des cas (ce qui ne signifie pas que, lorsque vous serez dans l'avenir N et que l'ordinateur les examinera à nouveau, elles soient encore équivalentes, puisque le programme verra alors plus loin), il y a une chance sur $4 \times 4 \times 4$, soit une chance sur 64, que l'une des solutions soit choisie si l'adversaire prend la solution une.

Par contre, s'il prend la deuxième solution, il n'y a qu'une chance sur 1000 que chacune soit retenue. L'ordinateur aura donc passé un temps 15 fois plus élevé à examiner des solutions 15 fois moins probables.

On peut en déduire que, plus l'arbre des possibilités se rétrécie, plus la probabilité de détecter le chemin effectivement retenu augmente. A l'inverse, plus il s'ouvre, plus chacune des solutions est peu probable, mais aussi plus il est vraisemblable qu'on pourra trouver, quand on arrivera là, une solution convenable, puisque les possibilités sont plus nombreuses.

On peut donc examiner plus loin chaque solution d'un arbre étroit (puisqu'elle est plus probable), ou (c'est équivalent, mais plus choquant à dire) limiter la réflexion dans un arbre large. Le critère de coupure peut être par exemple la multiplication à chaque niveau du nombre de branches viables ; si on coupe à 10 millions, on pourra examiner 7 niveaux de largeur moyenne 10 ou 13 niveaux de largeur moyenne 6 (en fait on n'examinera de toute façon pas toutes les branches, compte tenu de l'application de l'Alpha-bêta et des autres élagages préalables).

Cette méthode, théoriquement peu satisfaisante car elle peut laisser passer à des niveaux élevés des possibilités intéressantes, fait néanmoins gagner un temps appréciable en présentant un risque limité. Elle n'est évidemment pas contradictoire avec la valorisation des degrés de liberté, mais peut être complémentaire. Cette méthode, utilisée par `Dames 2020`, a été peu décrite dans la littérature jusqu'ici, peut-être parce qu'elle est moins adaptée aux autres jeux.

3.7.3 La recherche focalisée

Cette méthode consiste à faire une recherche exhaustive jusqu'à un certain niveau, puis à ne garder que 80% ou 90% des solutions (les meilleures), puis, au niveau suivant à focaliser encore sur les 80% ou 90% les meilleures, etc...

Le défaut lié à cette méthode est que, si les meilleures solutions à un niveau N le restent souvent au niveau $N + 1$, ce n'est pas systématiquement le cas, et dans les 10% ou 20% qui restent, peut se cacher la voie retenue par l'adversaire, celle qui, justement, fait mal, et qu'on ne saura pas éviter si l'arbre est étroit.

Cette méthode est insatisfaisante et il est souvent souhaitable de lui préférer l'élagage des solutions trop ouvertes, qui est de même nature, mais paraît mieux ciblée.

3.7.4 Les solutions déjà examinées

Ah ! Là, vous allez vous sentir mieux : c'est du sûr.

Vous avez sans doute compris de ce qui précède que la démarche dynamique étudie des mouvements, alors que l'évaluation statique s'intéresse à des positions. Or, des enchaînements différents de mouvements peuvent aboutir à des positions identiques. Par exemple, les coups suivants :

— 1. 33–28, 18–23
  2. 31–26, 16–21
— ou
  1. 31–26, 18–23
  2. 33–28, 16–21
— ou
  1. 31–26, 16–21
  2. 33–28, 18–23
— ou
  1. 33–28, 16–21
  2. 31–26, 18–23

amènent, au niveau 4, c'est à dire au bout de deux coups, exactement à la même position.

On pourra donc s'efforcer de détecter si la position à étudier a déjà été rencontrée, ce qui permet de lui attribuer des caractéristiques déjà calculées, et d'élaguer toute les branches qui en partent. Pour profiter de cette caractéristique, il suffira de conserver les positions étudiées, et de comparer chaque nouvelle position à celles-ci. Il existe bien sûr un optimum dans le nombre de positions à conserver, car, au-delà d'un certain nombre, il est plus rapide de réétudier la branche complète que de comparer chaque position aux 10 ou 20 mille autres déjà enregistrées. Pour aider dans cette approche, le jeu de dames possède une particularité importante (qu'on ne trouve pas aux échecs), son caractère unidirectionnel. Cela signifie qu'une position donnée ne peut se retrouver qu'à un niveau donné, et pas avant ni après puisque les pions doivent avancer et ne peuvent pas reculer (je simplifie, en ne considérant pas le mouvement des dames, qui, effectivement, détruit cette unidirectionnalité, mais je ne m'expose qu'à étudier plusieurs fois des positions où les dames ont fait des aller-retour). Ainsi, l'exemple considéré ci-dessus donne une position que l'on atteint en 4 niveaux (2 coups), mais impossible à obtenir en 3 ou 5 niveaux. On pourra profiter de cette caractéristique en stockant les positions étudiées par niveau, ce qui limite le volume des comparaisons.

3.8 Perfectionnons le programme

Un programme qui saurait effectuer toutes les analyses qui ont été exposées, et qui surtout saurait les effectuer rapidement, serait sans doute un bon programme. Mais il souffrirait de quelques défauts :

3.8.1 L'effet d'horizon

Si votre ordinateur est programmé pour arrêter son évaluation au niveau N, vous comprendrez très bien qu'il ne se préoccupe pas de ce qu'il se passe au niveau $N+1$ et au-delà. Or il peut s'y passer des choses très intéressantes. Nous allons voir un exemple (figure n°8).

Au niveau $N-3$, les noirs ont les cases 7,11,14,20. Les blancs ont la case 8.

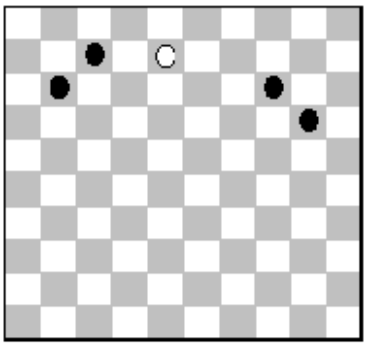

FIGURE 8 – Effet d'horizon

Noir voit que, s'il joue les pions 11, 14 ou 20 au niveau $N-2$, son adversaire fera une dame au coup $N-1$, qu'il aura toujours au coup N. Or une dame vaut 3 pions, et même plus en fin de partie, lui a-t-on

expliqué. Par contre, en jouant le pion 7 au coup $N-2$, l'ordinateur perd certes 2 pions, mais son adversaire ne fait pas de dame au coup $N-1$ (il prend). L'ordinateur choisira donc cette solution, pensant aboutir à une position plus favorable. Et, ainsi, il transformera une victoire (la dame aurait été perdue au niveau $N+1$ sur 11–16 et 20–25) en défaite (les pions 14 et 20 ne passeront jamais).

Comment éviter cet effet d'horizon ? Et bien il est impossible de le supprimer tout à fait, il y aura toujours un horizon. Mais on peut le reculer, en examinant plus loin les solutions sensibles. Il convient en particulier de regarder plus loin les chemins qui passent par des prises, par exemple en augmentant le niveau maximum d'une unité chaque fois qu'il y a une prise, ou, c'est équivalent, en n'incrémentant pas le niveau actuel lors des prises (en maintenant cependant une limite raisonnable pour éviter que l'ordinateur n'examine le don de tous les pions du damier). Dans le cas que nous avons étudié ci-dessus, l'ordinateur aurait alors vu que, au niveau $N+1$, l'adversaire faisait de toute façon une dame après le sacrifice des 2 pions.

Cette méthode concernant le recul de l'horizon après les prises offre de plus l'avantage de nous mettre généralement à l'abri des contre-coups qui peuvent suivre des combinaisons.

On peut aussi regarder plus loin certaines positions, indépendamment de l'enchaînement de coups (avec ou sans prises) qui nous y a conduits. Ainsi, nous avons vu dans un chapitre précédent que les branches étroites devaient être explorées plus loin. De même, les positions où un joueur est prés de la ligne de dame sont à étudier plus à fond.

3.8.2 La maîtrise du temps

En fait, beaucoup de programmes ne maîtrisent pas correctement leur temps de jeu (en particulier `Dames 2020`). Ils vous offrent de jouer à un certain degré de difficulté, mais vous ne savez pas s'ils vont vous répondre en 5 secondes ou en 3 minutes.

La recherche itérative, qui aurait pu être vous présenter dans le chapitre concernant l'Alpha-bêta car elle en constitue un perfectionnement, permet une meilleure régularité dans les temps de réponse.

Contrairement aux techniques présentées jusqu'ici, la recherche itérative ne diminue pas le nombre de boucles examinées, mais les augmente. Quel est alors son intérêt, me direz-vous ?

Suivez-moi bien, et vous comprendrez. La recherche itérative se déroule de la façon suivante :

- On étudie l'arbre des solutions d'abord jusqu'à un niveau N (par exemple 2).
- Puis on ordonne les solutions trouvées.
- On réexamine depuis le début l'arbre jusqu'à un niveau M, supérieur à N (généralement $N+1$). Cet examen au niveau M sera généralement plus rapide que si on l'avait entrepris directement dès le début, puisqu'on le fait selon l'ordre des solutions du niveau N, et qu'une bonne solution au niveau N a une certaine probabilité de rester bonne au niveau M. Selon la théorie précédemment exposée de l'alpha-bêta, une recherche commençant par les bonnes solutions donne lieu à plus d'élagages, et donc se déroule plus rapidement.
- On ordonne les solutions.
- On recommence depuis le début jusqu'au niveau P, supérieur à M.
- ... et ainsi de suite.

Même si nous supposons que nous n'avons rien gagné dans les élagages Alpha-bêta, l'augmentation du temps de calcul est relativement faible. En effet, si nous supposons que le nombre moyen de branches est de d, et que le nombre de niveaux à examiner est de p, une recherche itérative jusqu'au niveau 1, puis 2, ..., puis p se traduit, sans Alpha-bêta, par l'examen de $d^p + d^{p-1} + ... + d$ noeuds terminaux, soit $\frac{d^{p+1}-d}{d-1}$, au lieu de d^p si on n'avait pas utilisé cette méthode. Le rapport entre les 2 nombres de solutions examinées est équivalent à $\frac{d}{d-1}$ lorsque p est grand, et est d'autant plus proche de 1 que d est grand (ce qui signifie que la rentabilité est meilleure aux échecs où d moyen vaut voisin de 35 alors qu'aux dames, d moyen vaut voisin de 10).

Mais me direz-vous, quel est le rapport avec la maîtrise du temps ?

Et bien, avec une telle méthode, l'ordinateur peut toujours proposer une solution jugée bonne jusqu'à un certain niveau d'examen. Ainsi, si son temps de réflexion est limité à 5 secondes, il examinera par exemple le jeu jusqu'au niveau 2 en un centième de seconde, jusqu'au niveau 3 en dixième de secondes, jusqu'au niveau 4 en une seconde, commencera l'examen du niveau 5 et l'arrêtera faute de temps. Il aura alors eu le temps d'examiner plus à fond la meilleure solution de niveau 4, la jouera si elle est toujours bonne, jouera la seconde si ce n'est plus le cas (en fait, il préférera alors souvent réexaminer le niveau 4 en éliminant la première solution, car la seconde solution, lorsqu'on a procédé à un élagage Alpha-bêta, n'a pas été vue à fond, et n'est pas forcément le véritable deuxième meilleur choix ; cela lui prendra donc une seconde de plus).

Ceci permet à l'ordinateur de vous répondre dans le temps indiqué (par exemple 50 coups en 2 heures). Vous comprendrez donc qu'un programme muni de ce perfectionnement joue différemment selon l'ordinateur

sur lequel vous l'installez. Ainsi, placé sur un vieux *PC 8086*, tel programme vous parait mauvais lorsqu'il réfléchit moins d'une minute. Placé sur un *486 DX4-100*, 60 fois plus rapide, vous obtenez la même qualité de réponse en une seconde, et si vous le laissez réfléchir une minute par coup, comme sur l'ancien, le niveau est celui anciennement obtenu en une heure. Mieux encore, sur un *Pentium III 1000*, 40 fois plus rapide encore, vous obtenez en une minute le résultat obtenu en 40 heures sur le *8086*!

Si votre binôme vous dit qu'il bat le programme T..., alors que celui-ci vous flanque une pâtée à chaque match, demandez-lui donc sur quel type d'ordinateur il s'entraîne !

3.8.3 La recherche sur temps adverse

Quand vous jouez contre un adversaire, vous profitez généralement de son temps de réflexion pour réfléchir de votre côté. Certains programmes savent en faire autant, ce qui leur permet, si leur gestion du temps est bien faite (et en particulier s'ils sont munis du perfectionnement présenté ci-dessus) d'avoir le plus souvent examiné la réponse de l'adversaire et les contre-réponses possibles avant que l'adversaire n'ait joué. Ceci permet ensuite, dans le même temps, d'examiner plus loin l'arbre des solutions.

Essayons maintenant de faire un bilan de tout ce que nous avons appris du jeu des ordinateurs, et de voir la démarche globale qu'ils utilisent lorsqu'ils sont munis de tous les algorithmes présentés :

— l'ordinateur examine l'arbre des solutions jusqu'au niveau N ;
— au départ de chaque branche, il regarde si la solution a déjà été examinée ;
— il vérifie que la solution conduisant à cette branche n'est pas absurde (sacrifice inutile) ;
— il fait une évaluation statique de la position obtenue à chaque extrémité de la branche ;
— il remonte cette solution aux niveaux supérieurs par la méthode du Min-max ;
— il examine s'il peut effectuer à chaque niveau un élagage Alpha-bêta ;
— ayant examiné tout l'arbre au niveau N, il ordonne les solutions obtenues ;
— il recommence ce travail au niveau $N + 1$, ... ;
— et ainsi de suite, jusqu'à ce que le temps qui lui a été imparti soit atteint ;
— il joue la solution qu'il a retenue comme la meilleure dans sa recherche ;
— il poursuit sa recherche pendant que l'adversaire réfléchit, de façon à être prêt quand son tour viendra, sans handicaper son temps de réflexion.

Voilà, ce « bref » descriptif sur la programmation des jeux (en général) et du jeu de dames en particulier est terminé, il ne vous reste plus qu'à battre votre ordinateur maintenant que vous connaissez ses forces et ses faiblesses, et à écrire un petit programme, qui vous serve d'adversaire les jours où vous n'avez pas de projet de LI5 à préparer.

4 Description de l'interface réseau

La partie réseau de ce projet n'est pas à votre charge. Il vous est fourni un module compilé (un point `o`), un fichier d'entêtes décrivant les fonctionnalités du module compilé et un programme d'exemple d'utilisation de ce module compilé.

L'ensemble de ces fichiers se trouve dans le répertoire : `/home/licence/potcommun/DamesReSo`.

Quatre fonctions sont mises à votre disposition :

1. `void connexion(char couleur,char * nom_machine)` : tente d'établir une connexion avec la machine de nom `nom_machine`. La couleur passé en paramètre est soit le « *define* » `BLANC`, soit le « *define* » `NOIR` (définis dans le fichier d'entêtes) et correspond à la couleur avec laquelle la machine locale va jouer. La machine jouant avec les blancs doit **toujours** initiée la communication.
2. `void deconnexion(void)` : permet de terminer la connexion. Les deux machines doivent obligatoirement clore leur communication par cette fonction.
3. `void envoi(char demicoup[SIZE_DEMI_COUP])` : envoi à l'adversaire le coup stocké dans le tableau `demicoup`. Le tableau doit être organisé comme décrit au paragraphe 2.2.2.

   La constante `SIZE_DEMI_COUP` est défini dans le fichier d'entêtes et vaut 25.
4. `void reception(char demicoup[SIZE_DEMI_COUP])` : cette fonction attend la réception d'un coup. Le coup joué par la machine distante sera stocké au retour de cette fonction dans le tableau `demicoup`.

5 Projet à rendre

Ce projet est à réaliser en binôme. Votre programme devra obligatoirement contenir un algorithme de type Min-Max avec élagage Alpha-Bêta. Par contre, vous êtes libres en ce qui concerne l'implantation des autres techniques d'élagages plus ou moins propres aux dames et vous êtes également libres de sophistiquer à « l'infini » votre heuristique d'évaluation d'une position.

Ce projet est à rendre lors de la dernière séance TP où un mini-championnat de dames sera organisé afin de déterminer le meilleur « damiste » parmi vos programmes. La durée de réflexion entre chaque coup sera limité pour ce tournoi. Tout dépassement du temps équivaudra à une défaite. Le projet sera accompagné d'un rapport où sera notamment détaillé votre fonction d'évaluation.

6 Bibliographie

1. BA. K. Dewdney : Récréations informatiques, les ordinateurs qui jouent aux dames ; dans Pour la Science, Novembre 1984 - (N.B. : il s'agit ici de checkers et non pas de dames européennes).
2. J.L. Lauriere : Intelligence artificielle, résolutions de problèmes par l'homme et la machine ; Eyrolles 1987.
3. David Levy : Les jeux et l'ordinateur ; série de 20 articles dans L'Ordinateur Individuel, n°16 (avril 1980) à 35 (mars 1982).
4. David Levy : Chess and Computers, Bastford, London 1976.
5. Jonathan Schaeffer, Norman Treloar, Paul Lu and Robert Lake : Man versus machine for the World Checkers Championship ; dans A.I. Magazine, summer 1993.
6. Fédération Française d'Othello, BP 147, 75062 Paris CEDEX 02. Cette fédération possède une importante documentation sur le sujet.
7. Jean-Bernard Alemanni : CHECKERS PROGRAMS PAGE