Réalisation d'un Jeu de Dames en Objective Caml

Nicolas Bernard
n.bernard@lafraze.net

Note (19 février 2017) : ce travail a été réalisé au premier semestre de l'année universitaire 2000-2001. Cette version est compilée à partir de fichiers retrouvés qui sont incomplets et/ou ne constituent pas nécessairement la version la plus définitive ayant existé. Les images en annexe sont manquantes.

Le programme décrit dans ce document a été réalisé dans le cadre du projet d'informatique de deuxième année de DEUG Sciences à la Faculté des Sciences d'Epinal.

Le programme ci-décrit a été développé sous Linux (Caml 2.99 - GNU Emacs) et sous Windows (Caml 2.99 - Wordpad).
Ce document a été réalisé avec LaTeX 2ε.
Le code-source a été inséré avec la version japonaise de *lgrind*.
Les captures d'écrans ont été réalisé sous windows avec *Sinfo 2.04*

Table des matières

Chapitre 1

Introduction et historique(s)

1.1 Historique et règles du jeu de Dames

1.1.1 Un peu d'histoire

Il semble que le jeu de dames ait été créé en Europe au cours du Moyen-Age et se soit réellement répandu à partir du XIV^e siècle. Ce serait à l'origine une déformation du jeu d'échecs. Il en existe de nombreuses versions : italienne, polonaise, française, russe... Le premier traité sur le jeu de dames date de 1547 et a été écrit par Antonio Torquemada. En France, le premier ouvrage spécialisé est rédigé par Pierre Mallet en 1688.

1.1.2 Règles

Il n'est évidemment pas possible de décrire les règles de toutes les versions. Intéressons-nous simplement à deux d'entre elles :

La version française

Dans la version française, le jeu se déroule sur un damier de 64 cases, chaque joueur disposant de 12 pions au départ, répartis sur les cases noires des trois lignes les plus proches du joueur. Les noirs commencent la partie. Un pion se déplace en avant en diagonale d'une case à la fois. Un pion prend (en avant toujours) un pion adverse en passant par-dessus à condition qu'une case soit libre derrière et, de là, il peut éventuellement en prendre un ou plusieurs autres dans le même tour. Une prise multiple ainsi réalisée s'appelle une *rafle*. Lorsqu'il est possible de prendre, il faut prendre. De même, si le joueur n'est pas obligé de choisir le coup qui prendra le plus de pièces, lorsqu'il a commencé à bouger l'une d'elles, il doit prendre toutes celles qui

se présentent. Lorsqu'un pion d'une couleur arrive à la ligne de départ des pions adverses, il y a "Dame". Il est alors chevauché par un autre pion de la même couleur. Une dame peut se déplacer dans les deux sens de plusieurs cases à la fois, toujours en diagonale.

La version polonaise

Paradoxalement, c'est la version polonaise qui est la plus connue en France, peut-être parce qu'il s'agit de la version utilisée par la Fédération Internationale. Les règles sont les mêmes si ce n'est que cette version se joue sur un damier de 100 cases avec 20 pions par camp et que les pions peuvent prendre en arrière. De plus,le joueur peut ne pas respecter les règles concernant la prise des pièces : le joueur adverse pourra alors prendre la pièce fautive puis jouer normalement. C'est la règle dite du *Souffler n'est pas Jouer*. C'est cette version que nous avons choisi d'implémenter.

1.2 Pourquoi un jeu de Dames ?

L'intérêt du jeu de dames est qu'il s'agit d'un jeu simple pour un être humain : les règles sont peu nombreuses et faciles à apprendre, il est relativement facile de jouer à un niveau moyen. Pour un ordinateur, les choses sont nettement plus compliquées ; mathématiquement parlant, c'est un problème soluble en un temps exponentiel : il est impossible pour l'ordinateur de jouer en ayant prévu toutes les possibilités. La force brute qui fait la puissance calculatoire d'un ordinateur ne lui est ici que de peu d'utilité, il faut avoir recours à d'autres méthodes. C'est cela qui a fait du jeu de dames l'un des premiers problèmes étudiés par l'Intelligence Artificielle, dans les années soixante. A titre indicatif de la difficulté de la tâche, notons qu'il a fallu 15 ans à Arthur Samuel, l'un des pionniers de l'IA, pour écrire un jeu de dames d'un niveau de championnat (ce qui n'est pas notre but !)...

1.3 Brève histoire des débuts l'IA

On peut évidemment discuter sur l'origine de l'intelligence artificielle (IA ou AI en anglais), l'intérêt de l'homme pour les machines douées d'intelligence remontant aux premières machines. Il y a deux dates qui peuvent servir de début : en 1950 le mathématicien et logicien anglais Alan Turing, que l'on considère parfois comme le créateur de la science informatique avec la machine qui porte son nom, a publié dans la revue *Mind* un article in-

titulé *Computing Machinery and Intelligence* ([7]) [1] dans lequel il réfute les objections que l'on pourrait opposer à l'idée d'une machine pensante. Dans ce même article, il crée le célèbre *jeu de l'imitation* ou *test de Turing* qui doit servir à déterminer si une machine est intelligente sans avoir à définir ce qu'est l'intelligence. L'autre date pouvant tenir lieu de naissance de l'IA est celle de la conférence de Dartmouth, en 1956. Cette conférence réunit alors la plupart de ceux qui allaient devenir les pionniers d'une discipline nouvelle. Nous citerons notamment Allen Newell, Herbert Simon, Marvin Minsky et John McCarthy, ce dernier ayant alors proposé le terme d'Intelligence Artificielle [2]. C'est lors de cette conférence que fut présenté un programme de démonstration de théorèmes conçu par Newell, Simon et Shaw, *Logic Theorist*, qui est généralement reconnu comme étant le premier programme d'intelligence artificielle. Il faut noter que, pour faire ce programme, Newell, Simon et Shaw ont créé un langage de programmation, IPL [3], qui fut le premier à implémenter le traitement de liste. Cette notion est peut-être plus importante que le programme lui-même... C'est de cette époque que date l'algorithme que nous allons utiliser. Nous ne nous étendrons donc pas plus, malgré les progrès considérables faits depuis, sur l'histoire passionnante de l'intelligence artificielle, cette histoire pouvant occuper un livre entier sans problème. Le lecteur intéressé pourra se reporter à [4] pour l'histoire ainsi qu'à [5] pour les concepts fondamentaux.

1.4 Les différents types de programmation

La programmation informatique n'est pas *unie*. C'est-à-dire que tous les langages de programmation qui existent (la liste qu'en a fait l'université de Genève [4] en recense plus de 2500 !) ne se ressemblent pas, ne s'utilisent ni de la même façon ni pour faire la même chose. On peut distinguer deux paradigmes majeurs, fonctionnels et impératifs, auxquels est venue s'ajouter plus récemment l'Orientation Objet qui permet une représentation des données et de leurs interactions plus proches du *monde réel*. Le langage Objective Caml que nous utilisons combine toutes ces possibilités.

1. Les références des ouvrages cités ou utilisés dans le cadre de ce projet se trouvent dans la bibliographie située à la fin de ce document
2. Claude Shannon était également présent, mais il est plus connu pour sa Théorie de l'Information
3. Information Processing Language
4. http ://cui.unige.ch/langlist

1.4.1 Programmation fonctionnelle

La programmation fonctionnelle est issue de la théorie mathématique du λ-calcul et l'on peut donc faire remonter son origine à celui de cette théorie et des machines (théoriques) de Turing, dans les années 1930. Dans la pratique, c'est John McCarthy qui a été le premier à créer un langage de programmation de ce type avec $LISP$ (LISt Processing).

1.4.2 Programmation impérative

La programmation fonctionnelle consiste à voir l'ordinateur comme un état-mémoire que l'on fait varier. Historiquement, cette programmation est la programmation naturelle des ordinateurs, du moins de ceux à architecture von Neumann qui représentent la quasi-totalité des ordinateurs existants, leurs langages machines étant impératifs.

1.4.3 Programmation orientée objet (POO)

La programmation orientée objet est plus récente et peut se présenter comme une surcouche ajoutée au-dessus des styles impératif et fonctionnel. Elle permet de représenter les "choses" du monde réel et leurs interactions par des *classes* et les relations définies entre elles, notamment la notion d'*héritage*. Ainsi, par exemple, on pourrait définir une classe "voiture" dérivée (i.e. qui hérite) d'une classe "véhicule", la classe véhicule possédant des données membres comme "année de fabrication", "vitesse maximale" et la classe voiture possédant, en plus des données de véhicule, des informations comme "marque" ou "immatriculation".

Chapitre 2

Représentation interne

Comme pour la majorité des jeux, il nous faut connaître l'état de la partie à un moment donné, cet état se modifiant selon les coups joués par les deux camps. Pour cela, il faut que l'ordinateur ait une représentation interne du jeu. L'Orientation Objet de Caml va particulièrement nous servir ici.

Matériellement, un jeu de dames *physique* peut se décomposer en deux parties : le damier, constitué de cases, et les pièces. Il faut noter ici que si on représente les dames par un empilement de deux pions, leur comportement sensiblement différent va nous conduire à séparer la représentation des pions de celle des dames.

2.1 les pièces : pions et dames

Nous définissons donc une classe virtuelle *piece*. Le fait que cette classe soit virtuelle rend impossible son instanciation (*i.e.* la création d'objets de type *piece*). Le seul intérêt de cette classe est qu'elle nous permet de définir deux classes dérivées *pion* et *dame*. La classe *piece* regroupe toutes les données communes aux pions et aux dames.

- La couleur. Cette donnée est de type Graphics.color et n'est pas modifiable, une pièce ne pouvant changer de couleur au cours du jeu.
- L'abscisse de la pièce. C'est un entier qui doit être compris entre 0 et 10. Il est modifiable et évolue au cours du jeu.
- L'ordonnée. Idem
- La déclaration d'une fonction virtuelle *dessine* qui devra être définie dans les classes dérivées. Comme son nom l'indique, cette fonction a pour mission de dessiner la pièce à l'endroit indiqué par l'abscisse et l'ordonnée.
- Une fonction permettant d'effacer la pièce d'un endroit donné, par

exemple lors d'un déplacement ou lorsque cette pièce tombe sur le champ d'honneur.
— Une fonction permettant de déplacer la pièce, c'est à dire de changer l'abscisse et l'ordonnée, de l'effacer de son ancienne position et de la dessiner à la nouvelle.
— Une fonction renvoyant la couleur de la pièce.
— A cela il faut ajouter quelques données d'ordre strictement technique comme l'emplacement du damier dans la fenêtre graphique.

Les classes *pion* et *dame* se ressemblent beaucoup. Elles sont définies comme héritant de *piece* et ne comprennent guère que la fonction *dessine* annoncée précédemment et une méthode particulière, appelée automatiquement lors de l'instanciation, qui dessine la pièce si elle est définie comme étant visible.

2.2 Les cases et le damier

Maintenant que nous avons de quoi faire deux armées, il faut un champ de bataille.
Nous créons pour cela une classe *case*. Les données membres de cette classe sont notamment la couleur et surtout l'occupant de la case. Un occupant, vous l'aurez deviné, peut être un pion ou une dame. Cependant une case peut aussi être vide. Ne pouvant laisser une donnée sans valeur, nous avons construit un nouveau type regroupant pion, dame et Nil (qui correspond à une case vide) par

```
type occ = Nil | Pion of pion | Dame of dame;;
```

Chaque case comprend donc une donnée modifiable de ce type.

Les cases comprennent également un certain nombre de méthodes permettant de les sélectionner et les désélectionner, d'accéder à leur occupant, de lire leur état.

Nous avons les cases, reste à les assembler pour avoir notre damier. Nous utilisons pour cela une matrice de ces objets que sont les cases, cette matrice étant crée et initialisé par

```
let cm = Array.create_matrix 10 10
(new case Graphics.black 1 1 300 300 40 false) ;;
cree_damier 10 300 300 40 cm true ;;
```

la fonction *cree_ damier* se chargeant de placer les cases en respectant l'alternance des couleurs. La dernière étape

```
init_pions Graphics.black 300 300 cm 'b' true;;
init_pions Graphics.white 300 300 cm 'h' true;;
```

crée et place les pions, noirs puis blancs.

Chapitre 3

L'interface graphique

Ici aussi l'utilisation d'objets nous simplifie la tâche : les boutons et les barres d'attente [1] sont des objets, de même que, nous l'avons vu, les pièces et les cases. Nous avons implémenté la fonction *selectsave* sans utiliser d'objet : on peut la comparer avec *sauve* qui à un rôle plus complexe mais utilise des objets...

Il n'y a pas grand-chose à dire sur la partie graphique du programme : c'est une utilisation classique du module *Graphics* de Caml. Notons simplement l'existence d'une fonction *draw_ rect* dans le module *Graphics* non documenté dont nous avons découvert l'existence seulement après avoir écrit une fonction équivalente. On peut également noter l'utilisation de la bibliothèque *Sys*, pour déterminer le système d'exploitation utilisé : en effet, selon l'OS, les chaînes de caractères ne sont pas affichées de la même façon. Les couleurs du programme sous Unix s'inspirent donc vaguement du film *Matrix*, alors que sous un autre système les couleurs sont plus classiques. Il est cependant facilement possible de changer ces couleurs puisqu'elles sont définies au début du programme, un développement ultérieur pourrait même les charger depuis un fichier de configuration externe.

1. les barres d'attente s'inspirent de celles du programme d'installation de Netscape 6

Chapitre 4

L'intelligence artificielle

Dans notre cas, le terme d'intelligence artificielle est un bien grand mot, le programme se contentant d'élaguer un peu l'arbre de recherche lors de son parcours. L'algorithme que nous utilisons est un grand classique des jeux à deux joueurs : il s'agit du $minimax(\alpha, \beta)$.

4.1 L'algorithme de recherche d'un coup : Minimax-$\alpha\beta$

4.1.1 MiniMax

Les origines de l'algorithme sont floues. La théorie sous-jacente date, elle, de 1928. C'est une partie de la *Théorie des jeux* de John von Neumann. C'est ce dernier qui est donc souvent considéré comme le père de cet algorithme, bien que l'on retrouve également fréquemment des références à Claude Shannon comme dans [1].

Cet algorithme utilise une fonction de génération des coups possibles à partir d'une position donnée et une fonction lui permettant d'évaluer une position pour explorer l'arbre de jeu jusqu'à une certaine profondeur à laquelle il évalue alors toute les feuilles. Le meilleur coup sera celui se dirigeant vers la feuille au score le plus élevé en minimisant les possibilités de score de l'adversaire. Pour plus de détails, voir [5].

4.1.2 MiniMax avec coupures Alpha-Bêta

L'historique de cette version améliorée du minimax est encore plus flou que celui de la version standard. On cite souvent Alan Turing comme étant son créateur, cependant tout au long des années 1950 de nombreux articles sur l'élagage d'un arbre de jeu sont publiés par Samuel, Shaw, Simon, McCarthy, Hart ou Edwars mais le minimax et ses différentes versions sont

souvent confondues. Il semble que ce ne soit qu'en 1969 que Slagle et Dixon publient pour la première fois une description précise de l'algorithme, dans *Experiments with some programs that search game trees.*

L'idée de cet algorithme est qu'il est inutile d'explorer toutes les branches de l'arbre comme le fait le minimax. Sans entrer dans les détails notons que l'on utilise, pour éliminer des branches, les bornes inférieures des noeuds maximisants et les bornes supérieures des noeuds minimisants. Le lecteur pourra se reporter à [2] et [3].

Voici notre algorithme en pseudo-code. Cette version a été faite à partir de deux programmes en *C* téléchargés l'un sur le site du MIT, l'autre sur celui de CalTech.

$état \leftarrow$ (état actuel du jeu)
$noeudmax \leftarrow$ faux
$reste\_à\_voir \leftarrow$ (profondeur de l'arbre à explorer)
$alpha \leftarrow -\infty$ [1]
$bêta \leftarrow +\infty$ [2]

Si $(reste\_à\_voir = 0) \vee$ plus_de_coup_possible
    Alors évaluer *état*
        Si $noeudmax \wedge (score > alpha)$
            Alors $score \leftarrow alpha$
        Si $(\neg noeudmax) \wedge (score < bêta)$
            Alors $score \leftarrow bêta$
Sinon pour chaque coup possible
    Générer le coup
    mettre à jour *état*
    Appeler *minimax* récursivement avec $\neg noeudmax$ et $reste\_à\_voir - 1$
    Si $noeudmax \wedge (score\_retourné > alpha)$
        Alors $alpha \leftarrow score\_retourné$
    Si $(\neg noeudmax) \wedge (score\_retourné < bêta)$
        Alors $bêta \leftarrow score\_retourné$
    Si $alpha \geq bêta$
        Alors retourner *score* (et sauter le reste de la boucle et de la fonction [3])

Si *noeudmax*
    Alors retourner *alpha*
    Sinon retourner *bêta*

4.2 Fonction de génération des coups possibles

C'est peut-être la partie la plus complexe de notre programme, notamment à cause des difficultés posée par le "souffler n'est pas jouer".

Elle est en fait décomposée en plusieurs fonctions. L'une propose les coups directs que peut faire une pièce. Une autre prend ce coup, l'exécute dans un damier temporaire caché et regarde si cette pièce peut encore bouger et génère le cas échéant les rafles. Une autre fonction utilise la précédente pour générer l'ensemble des coups possibles pour un joueur. Il est difficile de décrire ces fonctions, aussi vous conseillons-nous de regarder le code-source pour plus de détails.

4.3 Fonction d'évaluation

La fonction d'évaluation est simple, par rapport à la précédente du moins : elle compte le nombre de pions de chaque couleur dans la matrice et fait la différence. Les dames comptent comme cinq pions. On peut imaginer une fonction plus évoluée qui modulerait la valeur attribuée à une pièce selon son emplacement sur le damier, mais il se poserait alors le problème d'attribution des coefficients, un peu à la manière de ce qui se passe pour les réseaux de neurones.

1. dans la pratique, nous avons utilisé *min_ int*
2. dans la pratique, nous avons utilisé *max_ int*
3. en Caml, nous avons utilisé une exception

Chapitre 5

Conclusion (et regrets)

La réalisation de ce programme s'est révélée instructive, les difficultés n'étant pas toujours là où nous les attendions. Comme nous le désirions, notre programme fonctionne sur Unix comme sur Windows (nous n'avons pas testé le programme avec MacOS, mais il ne devrait pas y avoir de problèmes particuliers. Si jamais il s'averait qu'il y en ait, il suffirait sans doute d'attendre MacOS X qui est un Unix), mais notre volonté de préserver cette compatibilité a coûté (provisoirement) aux utilisateurs d'Unix plusieurs fonctions, comme la possibilité de jouer à deux en réseau (TCP/IP) [1], ou celle de chiffrer les sauvegardes, le module *Unix* de Caml n'étant qu'imparfaitement implémenté sous Windows, et c'est la mort dans l'âme que nous avons supprimé notre implémentation de l'algorithme de cryptage RC4 [2] qui, si elle fonctionnait parfaitement sous Linux, ne provoquait avec Windows que le crash du jeu avec l'oraison funèbre "Failure (Unix.error : Not Implemented)", le temps nous manquant pour contourner les insuffisances de l'implantation Windows de Caml...

Nous aurions également pu imaginer d'inclure quelques animations, notamment pour les déplacements et les prises de pièces, mais le temps nous a aussi manqué pour ce genre de détails esthétiques, somme toute secondaires. Dans la même voie, on peut également imaginer de futures versions incluant une horloge indiquant les temps respectifs des deux adversaires, ou la possibilité de tirer au sort le joueur qui commence la partie en utilisant les fonctions de génération de nombre pseudo-aléatoire de Caml ou en en demandant un (sans doute "plus" aléatoire) à RC4.

> Notre vision de l'avenir est limitée, mais du moins nous voyons qu'il nous reste bien des choses à faire
>
> Alan Turing

1. Cela aurait aussi permis de faire des essais de distribution du minimax
2. réalisée à partir de la description donnée dans [6]

Annexe A

Mode d'emploi

A.1 Configuration requise

— Système d'exploitation : Unix avec Xwindow, Windows (MacOS et tout autre système sur lequel fonctionne Caml devraient également convenir). Le programme a été testé avec Linux (kernel 2.2) et Xfree 4.0, Windows 9x. Nous recommandons Unix pour une meilleure esthétique.
— Hardware : Ce programme a été testé et fonctionne correctement sur plusieurs PC, munis de processeur Pentium, K6-2 et Pentium 2.

A.2 Installation et lancement

Pour utiliser la version interprétée du jeu, vous devez avoir une distribution d'Objective Caml installée sur votre machine. Consultez la documentation fournie avec la distribution pour l'installer. Le code-source de différentes versions d'Objective Caml, ainsi que les exécutables correspondants pour la plupart des plateformes sont disponibles à l'adresse *http ://caml.inria.fr*.
Si vous avez une plateforme sous Unix, vous devez créer un *toplevel* incluant les bibliothèques *Graphics* et *Unix* en utilisant la commande

```
ocamlmktop -custom -o montoplevel graphics.cma unix.cma -cclib -lX11
```

ou si votre système place les bibliothèques X11 dans un autre répertoire, comme Linux,

```
ocamlmktop -custom -o montoplevel graphics.cma unix.cma -cclib \
           -L/usr/X11/lib -cclib -lX11
```

Vous devez alors lancer l'interpréteur puis copier-coller le fichier source dans sa fenêtre ou utiliser la commande

```
#use "chemin_et_nom_du_fichier" ;;
```

pour lancer l'interprétation.

La plupart des versions compilées nécessitent simplement d'avoir Objective Caml installé sur le système. Pour un fonctionnement correct, le programme doit pouvoir écrire dans son répertoire.

A.3 utilisation

Lors du lancement apparaît une page de présentation comprenant en bas une série de boutons.

Le premier permet de charger une partie sauvegardée. Vous êtes alors renvoyé à une autre page qui permet de choisir l'un des dix emplacements de sauvegarde. Notez que le choix d'un emplacement ne comprenant pas de sauvegarde provoquera l'arrêt du programme.

Le second bouton permet de jouer à deux sur l'ordinateur.

Le troisième permet de jouer contre l'ordinateur. Vous devez alors choisir votre couleur.

Le dernier bouton permet de quitter le jeu sans jouer.

Comment jouer ?

Quand c'est votre tour de jouer, vous devez cliquer sur le pion que vous désirez bouger puis successivement sur les cases sur lesquelles il devra se poser. En cas d'erreur, cliquez sur *Annuler* et rejouez. Quand vous voulez exécuter ce coup, cliquez sur *Fin de Tour*. Si vous voulez souffler un pion, cliquez dessus puis cliquez sur *Souffler*. Notez que si vous désirez souffler, vous devez le faire avant de jouer. Vous pouvez quitter le programme avec le bouton *Abandonner/Quitter* ; pensez éventuellement à sauvegarder la partie avant.

Sauvegarder une partie

Vous pouvez sauvegarder une partie lorsque c'est à vous de jouer en cliquant sur *Sauver* puis en sélectionnant un emplacement et en confirmant.

Attention ! Il est possible mais déconseillé de charger des sauvegarde faites avec le programme sous Unix dans la version Windows et *vice-versa*.

Annexe B

Code-source

Avertissement : ce programme est une version préliminaire. Il n'a pas été optimisé, ni pour la longueur du code, ni pour sa vitesse d'exécution, la complexité des algorithmes n'est sans doute pas la meilleure, l'entropie du format de sauvegarde est désastreuse et je ne parle même pas des bugs qui doivent rester ! Il y a peut-être même des fonctions qui ne servent plus. **L'AUTEUR NE POURRA EN AUCUN CAS ÊTRE CONSIDÉRÉ COMME RESPONSABLE DES DÉGATS QUE CE PROGRAMME POURRAIT OCCASIONNER.**

Annexe C

Bugs connus

Comme pour tout logiciel, il n'y en a pas jusqu'à ce que l'on en trouve un. Si vous en trouvez un, vous pouvez le signalez à l'auteur, vous pouvez également (surtout si vous êtes pressé de le voir corrigé) vous plonger dans le code et le résoudre vous-même et envoyer alors le descriptif du bug avec sa correction à l'auteur qui vous en sera éternellement reconnaissant :-)).

Annexe D

Exemple de partie typique

L'ordinateur entame la partie avec les Noirs, l'humain joue donc avec les Blancs

Après un début classique, on se dirige vers une immobilisation du front dans un scénario de "Guerre des Tranchées".

Le front s'est stabilisé dans le nodraughtsland initial. Comme lors de la première guerre mondiale, les premières troupes envoyées ayant été décimées, les deux camps ont appelé les réserves en première ligne. Les deux camps sont encore à égalité. On voit sur l'image une manoeuvre des blancs destinée, en sacrifiant un pion, à leur faire prendre l'avantage, mais une erreur stratégique qui aurait été minime en d'autre circonstance va faire basculer l'issue du conflit, comme le montre la dernière photo-satellite du champ de bataille.

La Garde Blanche (humain) meurt ou se rend (abandonne)...

Annexe E

Frequently Asked Questions

Pourquoi le jeu n'est-il pas en 3D ?

Le module Graphics d'Objective Caml est en 2D. Faire de la pseudo-3D avec un module 2D est, pour reprendre l'expression d'un de mes professeurs "long, ch... et pénible". Peut-être y aura-t-il une mise à jour en 3D lorsque Caml intégrera OpenGL :-).

Pourquoi faut-il cliquer sur "fin de tour" après avoir joué ? Ce n'est pas pratique.

Il y a effectivement de nombreux jeux de dames qui exécutent le coup après que l'on a cliqué sur la case. Ces jeux ne sont pas conformes aux règles polonaises des dames et ne prennent pas en compte le "souffler n'est pas jouer".

Peut-on copier et distribuer librement ce programme ?

Oui, vous êtes simplement prié de tenir l'auteur au courant de toutes modifications que vous pourriez effectuer.

Bibliographie

[1] Philip Ball. Quand darwin joue aux dames avec des ordinateurs neuronaux. *Le Monde*, 14/01/2000.

[2] Frédéric Bayard. Complexité de l'algorithme alpha-béta. Master's thesis, Ecole Normale Supérieure de Cachan, 1996-1997.

[3] Emmanuel Chailloux, Pascal Manoury, and Bruno Pagano. *Développement d'applications avec Objective Caml*. O'Reilly, 2000.

[4] Daniel Crevier. *A la recherche de l'intelligence artificielle*. Champs. Flammarion, 1997.

[5] Jean-Paul Haton and Marie-Christine Haton. *L'intelligence artificielle*. Que sais-je ? Presses Universitaires de France, troisième edition, 1993.

[6] Bruce Schneier. *Cryptographie Appliquée*. International Thomson Publishing France, deuxième edition, 1996.

[7] Alan Turing. Computing machinery and intelligence. *Mind*, 1950. Publié en français dans [8], sour le titre Les ordinateurs et l'intelligence.

[8] Alan Turing and Jean-Yves Girard. *La machine de Turing*. Points Sciences. Seuil, 1995.