UNIVERSITE D'ANTANANARIVO

ECOLE SUPERIEURE POLYTECHNIQUE

DEPARTEMENTS : GENIE MECANIQUE ET PRODUCTIQUE

GENIE ELECTRIQUE

Filière : GENIE INDUSTRIEL

Mémoire de fin d'étude en vue de l'obtention du diplôme d'ingénieur en Génie Industriel

Numéro d'ordre :...../2013

CONCEPTION ET REALISATION D'UN LOGICIEL DE JEU D'ECHECS

Présentée par :

Mademoiselle RAHOLIDERANTSOA Fanja

Directeur de mémoire :

Monsieur ANDRIAMANOHISOA Hery Zo

Maître de conférences à l'ESPA

Promotion 2013

UNIVERSITE D'ANTANANARIVO

ECOLE SUPERIEURE POLYTECHNIQUE

DEPARTEMENTS : GENIE MECANIQUE ET PRODUCTIQUE

GENIE ELECTRIQUE

Filière : GENIE INDUSTRIEL

Mémoire de fin d'étude en vue de l'obtention du diplôme d'ingénieur en Génie Industriel

CONCEPTION ET REALISATION D'UN LOGICIEL DE JEU D'ECHECS

Présentée par : Mademoiselle RAHOLIDERANTSOA Fanja

Directeur de mémoire : Monsieur ANDRIAMANOHISOA Hery Zo, Maître de conférences à l'ESPA

Président du Jury : Monsieur RAKOTOMANANA Charles Rodin, Chef de Département Génie Mécanique et Productique

Examinateurs : Monsieur RANARIJAONA Jean Désiré, Maître de conférences à l'ESPA

Monsieur RAKOTONINDRINA Tahiry, Enseignant chercheur à l'ESPA

Soutenu le 17 décembre 2014 Promotion 2013

REMERCIEMENTS

Je rends grâce au Seigneur tout puissant pour m'avoir donné la force, la santé et le courage pendant la préparation de ce mémoire.

J'adresse mes vifs remerciements à :

- Monsieur ANDRIANARY Philippe, Directeur de l'Ecole Supérieure Polytechnique d'Antananarivo

- Monsieur RAKOTOMANANA Charles Rodin, Chef de Département Génie Mécanique et Productique et qui a accepté de présider ce présent mémoire.

J'adresse aussi mes remerciements à Monsieur ANDRIAMANOHISOA Hery Zo, Maître de conférences à l'ESPA, qui a suivi de près mon travail. Un énorme merci pour sa disponibilité, son soutien moral et pédagogique tout au long de la réalisation de ce mémoire.

Je tiens également à exprimer toute ma gratitude aux membres du jury :

- Monsieur RANARIJAONA Jean Désiré, Maître de conférences à l'ESPA

- Monsieur RAKOTONINDRINA Tahiry, Enseignant chercheur à l'ESPA

Je ne saurais oublier tous mes enseignants à l'ESPA, qui ont partagé leurs savoirs durant ces années.

Un grand merci à ma famille, à tous mes amis pour leurs soutiens et encouragements et à tous ceux qui ont contribué de près ou de loin à la réalisation de ce mémoire.

TABLES DES MATIERES

LISTE DES FIGURES

INTRODUCTION

Les jeux, activités futiles en apparence, ont toujours fasciné les informaticiens. Dès le début de l'intelligence artificielle, les premiers chercheurs se lancèrent dans la programmation de jeux. Ainsi un des tous premiers programmes de Simon, Newell, Sham fut un programme de jeu d'échecs.

Pourquoi cette fascination des informaticiens pour le domaine des jeux ? Tout d'abord, parce qu'il semble intuitivement (et inconsciemment) que la pratique de jeux intellectuels soit le propre de l'homme. De plus, chacun est astreint à respecter des règles strictes et précises, et ne peut l'emporter que grâce à sa valeur et à son habileté. La victoire et la défaite sont sanctionnées clairement et sans appel possible. Ainsi, réaliser une machine qui joue aussi bien qu'un homme, voire mieux, « prouverait » l'intelligence (ou la supériorité) des ordinateurs, et serait apparue comme une grande réussite pour l'intelligence artificielle. Sur le plan technique, les règles sont généralement simples et surtout, les interactions au monde extérieur sont parfaitement nulles : il s'agit d'un micro-monde totalement clos et donc aisément accessible à un programme d'ordinateur. D'autre part, selon les informaticiens, sa puissance de calcul devrait donner, dans un univers restreint, un avantage considérable à l'ordinateur. Cependant, les jeux complexes comme les dames et les échecs demandent des algorithmes spécifiques et des méthodes heuristiques de recherche, ou encore les jeux à information partielle comme le bridge ou le poker demandent en plus des raisonnements de type probabiliste.

Pour concrétiser ce mémoire, la première partie vous explique ce qu'est le jeu d'échecs, ensuite, la seconde partie introduit sur les principes importants de la théorie moderne de la programmation des jeux, et enfin la dernière partie est consacrée à la phase de conception de notre jeu d'échecs.

Partie 1 : Le Jeu d'Echecs

Tout d'abord, puisque ce projet porte sur un jeu de société, il est nécessaire de commencer par expliquer le jeu en question soit plus précisément, son origine historique, la logique du jeu afin d'avoir une idée de quoi programmer et enfin ses relations avec l'intelligence artificielle.

Chapitre I : Histoires et origines du jeu d'échecs

Il est bien difficile de dater la naissance du jeu d'échecs. De la Chine en Egypte, il a existé en orient de nombreux jeux de plateau représentant un combat de pions à déplacer sur une sorte de damier. Où, quand et comment l'un de ces jeux s'est-il progressivement transformé pour donner naissance au jeu d'échecs ?

Echecs et mat ou al-shah-mat – le roi est mort – comme diraient les arabes. Si l'origine des échecs est sujette à controverses, on s'accorde tout de même sur la finalité de ce jeu millénaire: capturer le roi adverse.

D'après les écrits mentionnant le jeu d'échecs, il semblerait que leur origine remonte au VI^{ème} siècle après J.C. Bien que la datation soit incertaine et que les traductions du sanskrit peuvent avoir été source d'interprétations, le Chaturanga, inventé par le brahman Sissa, donc, originaire du nord de l'Inde est présenté comme l'ancêtre le plus probable des échecs. Le Chaturanga, du sanskrit *chatu* (quatre) et *anga* (membres) fait référence aux quatre corps de l'armée indienne: l'infanterie, les chars, la cavalerie et les éléphants.

Le chaturanga aurait été introduit en Perse sous le nom de Chatrang ou Shatranj, sous le règne du roi Khosroe Nushirwan (531-579). Les arabes, ayant adopté le jeu lors de leur invasions, se chargeront de poursuivre la propagation du jeu d'échecs vers l'ouest jusqu'en Espagne, et vers l'est jusqu'en Chine et au Japon. C'est au cours des IX^{ème} et X^{ème} siècles qu'apparaissent les premiers champions et les premiers traités. Les pièces seront alors stylisées en raison de l'interdiction coranique de représenter des êtres animés.

Les pièces suivantes constituaient le jeu:

- Le roi (Shâh, c'est lui qui donne son nom au jeu) se déplace d'un pas dans toutes les directions
- Le conseiller (Farzin ou Vizir) dont le mouvement est limité à une seule case en diagonale
- L'éléphant (Fil, cf. sanskrit pīlu) avec un déplacement correspondant à un saut de deux cases en diagonale

- Le cheval (Faras), identique au cavalier moderne
- Le (Roukh), semblable à la tour actuelle
- Le soldat (Baidaq, cf. sanskrit padāti : "piéton, fantassin"), l'équivalent du pion, mais dépourvu du double pas initial

Figure 1 : Pièce de « San Genado » découvertes en Espagne au X^{ème} siècle

L'arrivée des échecs en Europe se fait sans doute par le sud de l'Europe (Espagne musulmane et Italie). Le jeu subit alors plusieurs modifications:

- Le plateau devient bicolore avec les cases rouges et noires (qui deviendront plus tard blanches et noires)
- Le vizir devient fierge (ou vierge), puis reine et/ou dame (il est difficile de déterminer lequel des deux termes prévalait — sans doutes étaient-ils utilisés indifféremment)
- L'éléphant (al fil en arabe, qui reste alfil en espagnol aujourd'hui) devient aufin, puis fou (bishop : "évêque" en anglais)
- Le roukh arabe devient roc (ce nom donnera rook en anglais, le verbe « roquer » en français et désignera la tour d'échecs en héraldique), puis tour vers la fin du XVIIe siècle (les tours de guet étant souvent placées en hauteur).

Vers la fin du moyen âge, les règles de jeu évoluent fort rapidement, avec notamment des mouvements amplifiés et renforcés pour la reine et le fou. Vers 1650, on peut considérer que les règles du jeu moderne sont à peu près établies. L'aspect des pièces le plus courant aujourd'hui, le style « Staunton », date de 1850.

Figure 2: Original Staunton Chess Set Piece

C'est également durant la seconde moitié du XIXe siècle qu'émergent les échecs modernes.

Ainsi, malgré leurs origines communes, les règles des jeux indiens, chinois et japonais diffèrent de ceux des jeux occidentaux.

Au XXe siècle, plusieurs pays, l'URSS étant pionnier dans ce domaine, en assurent une promotion très active le considérant comme un excellent outil de formation intellectuelle. L'importance des échecs dans le monde est telle que depuis le début de l'année 2008, on discute l'entrée de ce sport aux Jeux Olympiques.

Chapitre II : La logique du jeu d'échecs

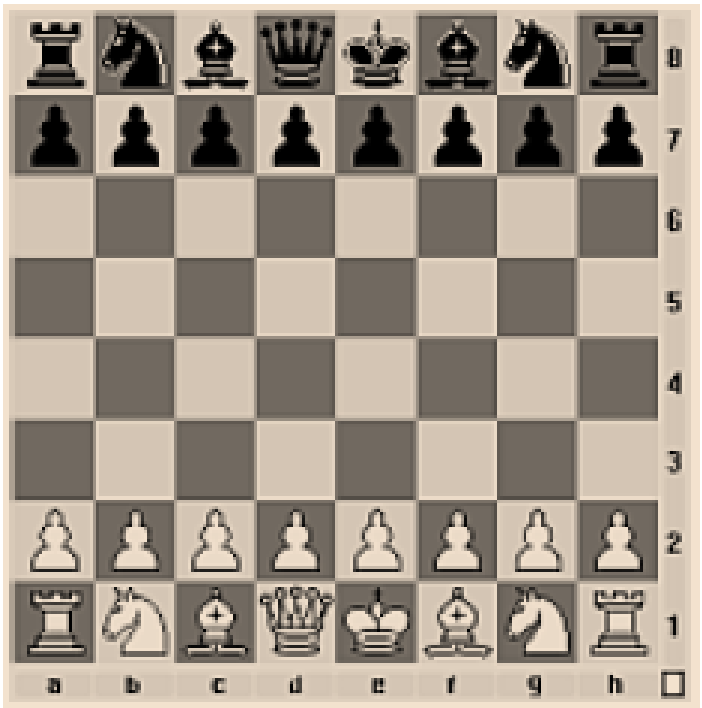
1. Connaissances sur le jeu :

1. 1 Notion sur les jeux de plateau :

Jeu de société qui se joue sur un plateau de jeu, représentant un parcours ou une carte, sur lequel se déplacent des pions, selon des règles ou selon le hasard, qui peuvent parfois s'affronter ou accumuler des points en collectant des ressources matérialisées par des marqueurs. Il s'agit généralement de jeux faisant appel à nos capacités de réflexion, d'observation ou de négociation, où l'on joue avec des pions (qui représentent les joueurs), des jetons et des figurines, et où l'on intègre une part de hasard par l'utilisation de cartes et de dés qui influent sur le déroulement des parties. Dans ce type de jeu, l'action se déroule sur une surface délimitée, le plateau de jeu, comportant des étapes à franchir afin d'arriver à la victoire. Ainsi, le premier joueur à avoir accompli le parcours, ou le dernier à avoir encore des pions, ou celui qui a accumulé le plus grand nombre de points, ou toute autre condition spécifique, est déclaré vainqueur. Parmi les jeux de plateau les plus connus, on trouve : Monopoly, Risk et Clue.

1. 2 L'échiquier :

Joueur 1



Joueur 2

Figure 3: Représentation de l'échiquier et représentation des pièces sur l'échiquier

L'échiquier est le tablier ou plateau du jeu d'échecs. C'est une grille carrée de 8 cases de côté, soit 64 cases en tout, en alternance sombres (appelées «noires») et claires (appelées « blanches»). L'échiquier est posé sur une table. Avant de jouer, chaque joueur doit s'assurer d'avoir une case blanche sur sa droite dans le coin devant lui.

- Les huit files de cases allant d'un bord de l'échiquier à l'autre, perpendiculairement aux colonnes, s'appellent «rangées» ou «traverses».
- Les huit files de cases, allant du bord de l'échiquier le plus proche de l'un des joueurs à celui le plus proche de l'autre joueur, s'appellent «colonnes».
- Les files de cases de même couleur se touchant par les angles s'appellent «diagonales»

Chaque case de l'échiquier possède un nom qui est la combinaison de sa position, sur la colonne et de sa position sur la rangée, de la forme (lettre, chiffre). Afin de repérer facilement chaque case, un échiquier possède très souvent un système de coordonnées.

Les lettres de a à h désignent les colonnes, les chiffres de 1 à 8 désignent les rangées. Une case sera donc repérée par un couple (lettre, chiffre) comme par exemple e4, a8, g7 etc. avec ce repérage, il sera facile de noter une partie. Les blancs partent donc toujours du « bas » de l'échiquier et « montent », tandis que les noirs « descendent ».

Le jeu d'échecs comporte 32 pièces dont 16 blanches et 16 noires. Chaque couleurs se composent de : 1 Roi, 1 Dame, 2 Tours, 2 Fous, 2 Cavaliers et de 8 pions.

Chaque pièce possède sa place sur l'échiquier :

Les pièces blanches (Roi, Dame, Tours, Cavaliers et Fous) sont placées sur la première rangée, les pions blancs sur la $2^{ème}$ rangée, tandis que les pièces noires sont sur la $8^{ème}$ rangée et les pions noirs sur la $7^{ème}$ rangée.

Les Tours se retrouvent aux extrémités de l'échiquier, les Cavaliers viennent se placer à leurs côtés, suivis ensuite par les Fous et enfin le Roi et la Dame dans les 2 cases centrales de la rangée en notant que la Dame doit toujours se située sur la case de sa couleur.

1. 3 Le déplacement des pièces :

Chaque pièce possède son propre déplacement. Une pièce ne peut jamais passer par-dessus une autre sauf le Cavalier

Le Roi : C'est la pièce centrale du jeu. La capture du Roi est en effet le but final de chaque partie. Il ne peut avancer que d'une case dans n'importe quelle direction, à condition que celle-ci ne soit pas menacée par une autre pièce. Le Roi n'est jamais battu, de sorte que sa mise en « échec et mat » signifie la *fin* de partie

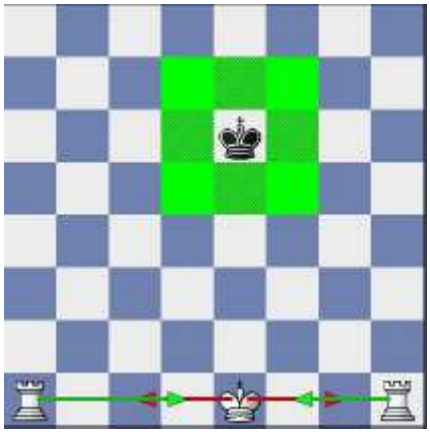


Figure 4: Déplacement du Roi

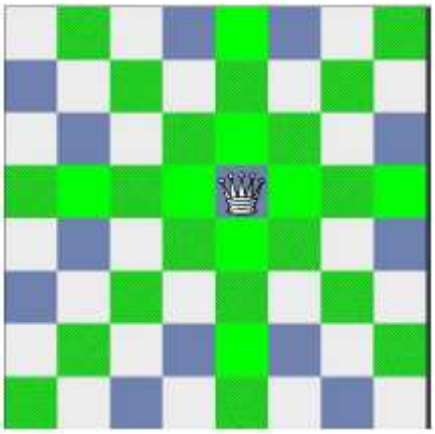


Figure 5 : Déplacement de la Dame

La Dame : La Dame est la pièce la plus importante du jeu. Elle peut se déplacer tant comme la Tour que comme le Fou. Soit ; sans restrictions sur les rangées, les colonnes, et les diagonales. En fin de partie, elle peut être une force déterminante pour l'obtention du mat.

Le Fou : Le Fou se déplace sur les 2 diagonales de la position qu'il occupe. Le Fou restant durant toute la partie sur la couleur de sa case de départ, il convient de préciser que chaque camp possède 2 Fous. Un sur les cases blanches et l'autre sur les cases noires.

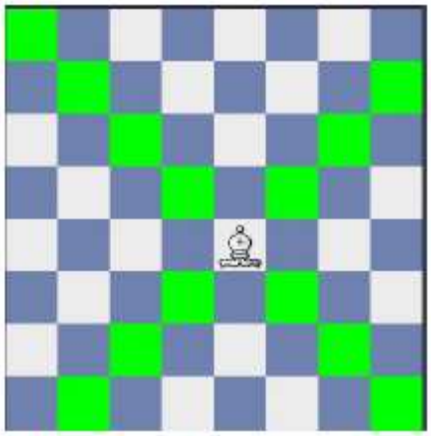


Figure 6 : Déplacement du Fou

Figure 7 : Déplacement du Cavalier

Le Cavalier : Le Cavalier est la pièce avec laquelle les débutants ont le plus de mal, tant il est compliqué de le déplacer sur l'échiquier. Son mouvement est particulier : 2 cases à la verticale ou à l'horizontale, puis une case en diagonale. Chaque camp possède 2 Cavaliers.

La Tour : La tour est placée aux angles de l'échiquier. Cette pièce se déplace sur les lignes et les colonnes de la position qu'elle occupe au départ. Il est souvent délicat de les activer et elles restent souvent en fin de partie.

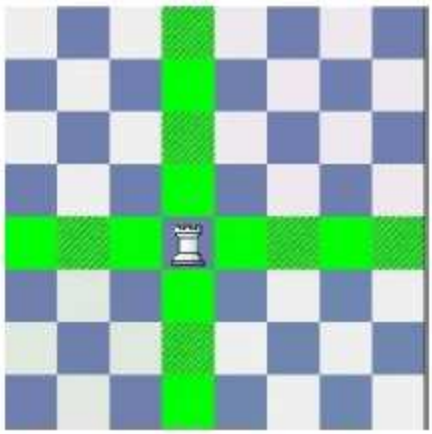


Figure 8 : Déplacement de la Tour

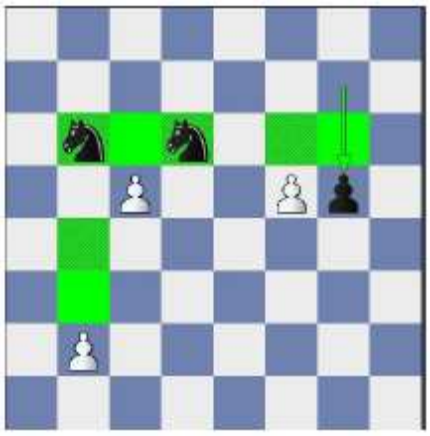


Figure 9 : Déplacement et prise du pion

Le Pion : Dans les échecs modernes, le pion est la pièce de plus faible valeur. Mais il bénéficie de propriétés que ne possède aucune autre pièce du jeu. Il ne peut revenir en arrière, ce qui fait que chacun de ces mouvements est irréversible. Depuis sa case, il peut se déplacer d'une ou de deux cases. Ensuite, il ne peut avancer que d'une case. Il prend les autres pièces uniquement en diagonales.

2. Règles et but du jeu :

Le but du jeu d'échecs est de capturer le Roi de l'adversaire. Il est alors « échec et mat » et la partie est finie.

Quand le Roi est attaqué, on dit qu'il est en échec et on prévient son adversaire en lui disant « échec ».

Dans la position ci-dessous, je viens de jouer mon Fou en b5 et j'attaque le Roi noir.

2. 1 L'échec :

S'il ne se défend pas, je le mangerais au prochain coup et la partie sera finie. Les Noirs sont OBLIGES de défendre leur Roi.

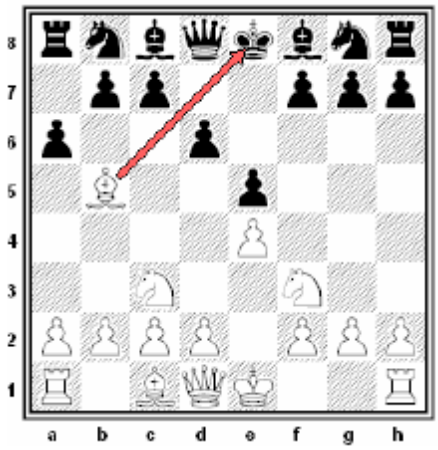


Figure 10 : Représentation de l'échec

Pour cela, il y a plusieurs solutions :

L'adversaire peut déplacer son Roi

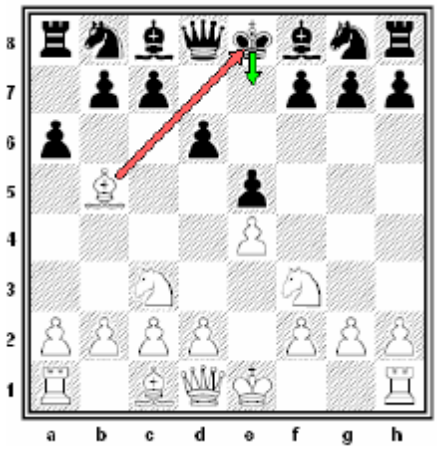


Figure 11 : Parer l'échec en déplaçant le Roi

- L'adversaire peut manger la pièce qui les attaque.

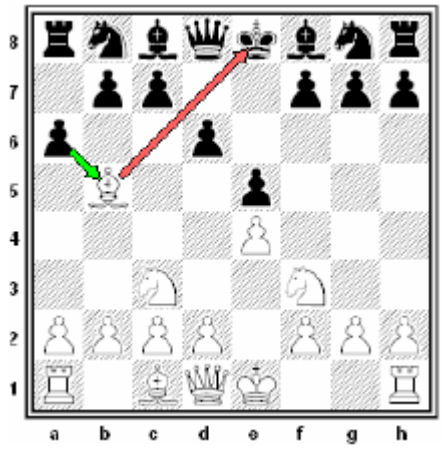


Figure 12 : Prendre l'adversaire pour parer l'échec

- L'adversaire peut déplacer une pièce entre le Fou et le Roi

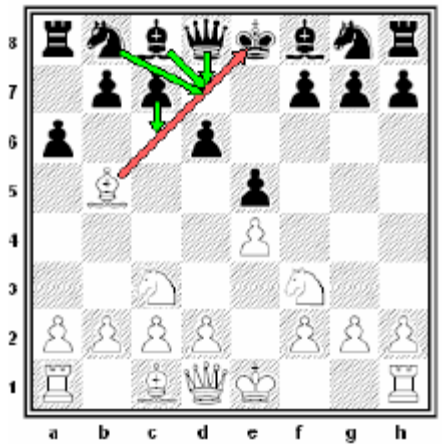


Figure 13 : Déplacer une pièce entre le Roi et le Fou

Comme nous somme en début de partie, les Noirs ont beaucoup de défenseurs. Mon échec avec le Fou ne va pas donc servir à grand-chose, j'aurais mieux fait de jouer un autre coup, j'ai perdu du temps avec cette attaque.

2. 2 L'échec et mat:

Le Roi est en échec et mat si :

- Une pièce adverse l'attaque
- Ni lui, ni aucune autre des pièces de sa couleur ne peut manger la pièce qui attaque
- Il ne peut se déplacer sur aucune autre case sans se faire prendre
- Aucune pièce ne peut se faire bouclier entre lui et la pièce qui l'attaque

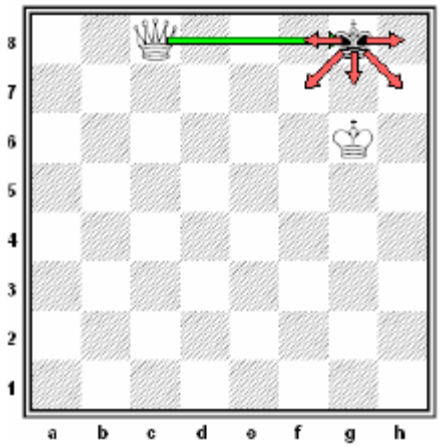


Figure 14 : Position du Roi en échec et mat

2. 3 Le pat :

Les blancs viennent de jouer leur Roi en h7, en espérant faire bientôt échecs et mat au Roi noir. Et pourtant, c'est une catastrophe!

Dans la position suivante, le trait est aux Noirs, or :

- Ils n'ont plus que leur Roi à bouger
- Le Roi n'est pas attaqué
- Le Roi ne peut pas bouger

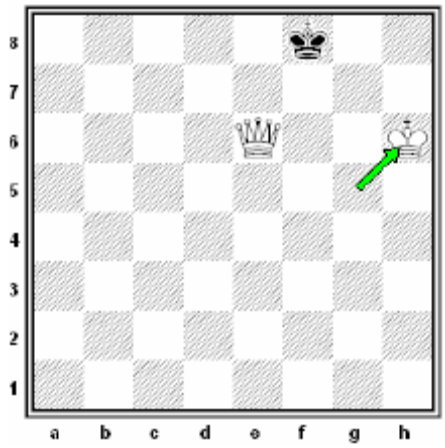


Figure 15 : Position du PAT

Comme il est interdit de mettre son Roi en danger, le Roi noir ne peut pas bouger. Il est donc PAT et c'est un match nul!!! Cela peut paraître bizarre ou injuste mais les Blancs ont encerclés leur adversaire sans le capturer. C'est rageant mais c'est la règle

3. Méthodes de réflexion :

3. 1 Observation :

La position sur l'échiquier se doit d'être observée et analysée minutieusement ;

3. 2 Questionnement :

La position des pièces va provoquer beaucoup de questions sur les coups à jouer, à prendre en compte, à anticiper, … ;

3. 3 Formulation d'hypothèse :

Le joueur devra, selon la position sur l'échiquier, émettre des hypothèses d'attaque ou de défense (à la fois pour lui et pour son adversaire) et valider les hypothèses émises par le choix du coup le plus approprié ;

3. 4 Argumentation :

Aux échecs, le joueur doit être capable de justifier et d'expliquer chacun de ses coups (il vaut mieux déplacer une pièce avec une idée, une intention, un plan, même mauvais, plutôt que de jouer sans but précis) ;

3. 5 Modélisation :

La mémorisation de modèles, de schémas, de tableaux de mat permettra au joueur de tenter, dans ses parties, de recréer, de modéliser ces situations mémorisées afin de lui permettre un gain de la partie plus aisé.

Chapitre III : Les jeux d'échecs et l'intelligence artificielle

Donald Michie dit :

« La recherche sur le jeu d'échecs est le champ le plus important de la recherche cognitive. Les échecs seront pour nous ce que la drosophile a été pour les généticiens : un moyen simple et pratique de développer de nouvelles techniques. »

1. L'évolution de l'intelligence artificielle dans le jeu d'échecs :

•"The Turk" –Baron Wolfgang von Kempelen – Avant le 19ème siècle

•Premier *real*algorithm – Turing (1947)

•1948 – le programme UNIVAC était supposé imbattable!

•1952 – Turing a posté que les ordinateurs pourraient éventuellement devenir plus puissant que les humains

•1958 – Première victoire d'une machine contre un humain

•1966 – Un programme Russe bat un programme Américain

•1970 – First ever all-computer tournament

•1977 – ICCA création

•1977 – GM Michael Stean perd dans une partie blitz contre un ordinateur

•1981 – Cray Blitz gagne Mississippi state championship un score parfait de 5.0

•1988 – DEEPTHOUGHT partage top place dans le US Chess Championship

•1992 – Fritz 2 perd contre le champion du monde Kasparov dans un jeu semi rapide

•Fev 1996 – Kasparov bat IBM Deep Blue (4 –2)

•Mai 1997 – Deep Blue gagne face à Kasparov (3.5 –2.5)

•Oct 2002 – Deep Fritz fait nulle le champion du monde Kramnik 4-4 in "Brains in Bahrain" match

•Jan 2004 – ChessBrain joue son premier match contre GM Nielsen

2. Analyse théorique :

Les échecs sont considérés comme le jeu des Rois et surtout le Roi des jeux. Dès le début de l'Intelligence artificielle (IA), c'est sur les échecs que s'est porté l'effort le plus important et ce sont les échecs qui ont donné lieu à des luttes épiques que se sont livrés les meilleures équipes universitaires comme HITECH, mené par Hans Berliner, ou DEEP THOUGHT, mené par Feng-Hsiung Hsu, devenus depuis DEEP BLUE avec le succès que l'on sait.

Il faut sans doute craindre que la victoire de DEEP BLUE face à Gary KASPAROV en mai 1997 n'ait des répercussions que l'on pourrait qualifier néfastes : le plus grand défi que l'on s'était lancé (la victoire d'un programme sur un champion du monde humain dans le cadre de partie à cadence de tournoi et sur six parties) a été réalisé et il est probable que les efforts se porteront davantage dans d'autres directions. D'autre part, même si la lutte entre les plus importantes firmes commerciales reste réelle, le niveau des programmes disponibles pour micro-ordinateurs est maintenant tellement élevé qu'ils battent systématiquement 99% des joueurs. On accordera donc plus l'importance à la qualité de l'interface avec l'utilisateur qu'à la force pure du programme. Mais les grandes avancées liées aux échecs demeureront.

Les pères fondateurs de l'IA, Simon et Newell, menèrent, ainsi que le psychologue hollandais De Groot et le psychologue et Grand Maître soviétique Nikolaï Kroguious une analyse de la méthode de réflexion des grands champions d'échecs. Nous allons rapidement parler des résultats de ces études avant d'aborder les techniques de programmation proprement dites.. Deux raisons à cela : tout d'abord les résultats mise en évidence par De Groot(en 1965) et Kroguious(en 1986) sont très intéressantes car ils sont applicables à la plupart des raisonnement humain dans tous les domaines ; enfin, ces résultants n'ont en rien influencé les programmes d'échecs (du moins les meilleurs) qui utilisent des méthodes de raisonnement totalement différentes aujourd'hui, preuve qu'il est bien difficile d'appliquer les techniques humaines pour faire résoudre des problèmes à des machines.

2.1 Les études des psychologues :

*N*ous allons ici évoquer les travaux effectuer par le psychologue hollandais De Groot et ceux du psychologue et GMI soviétiqiue Krouguious.

De Groot put travailler avec un échantillon tout à fait remarquable de six GMI (dont un champion du monde, Max Euwe), quatre MI, deux champions des Pays Bas et 10 experts. De par l'exceptionnelle qualité de ces joueurs, l'étude de De Groot est incomparable.

De Groot et ses élèves analysèrent la façon dont les joueurs d'échecs perçoivent une position. Pour ce faire, ils présentèrent à chaque joueur une position sur un échiquier pendant 5 secondes puis enlevèrent les pièces de l'échiquier et leur demandèrent de les replacer. Ils s'intéressèrent alors à 2 facteurs : le nombre de pièces replacées correctement et la façon de replacer les pièces. Ils remarquèrent un certain nombre de points intéressant :

- Les GMI parvenaient à replacer l'intégralité des pièces dans près de 40% des cas alors que les experts n'y sont jamais parvenus, tout en obtenant le score honorables
- L'ensemble des individus testés qui savaient jouer aux échecs replaçaient les pièces par groupe logique. Ainsi, on replaçait en même temps l'ensemble des pièces qui attaquaient le roque et celles qui le défendaient. Les individus qui ne savaient pas jouer aux échecs replaçaient les pièces de façon corrélée et obtenaient un score désastreux.
- Les grands maîtres avaient fait réaliser une mémorisation de la position qui était proche d'une analyse. Ils se la rappelaient comme un ensemble de lignes d'attaque et de défense, et non comme un ensemble de pièces.
- Les grands maîtres avaient reconstruisaient la position par référence à des positions de partie déjà connues qui ressemblaient à celle-ci. Ils étaient capables de dégager les grandes lignes du jeu qui avaient mené à la position considérée.
- Placés devant des positions aléatoires (non plus tirées de parties réelles, mais composées de pièces posées au hasard sur l'échiquier), tous les individus, joueurs, non joueurs et GMI obtenaient des scores comparables et très mauvais.

D'autres travaux furent menés pour étudier la façon dont les joueurs d'échecs « à l'aveugle » voient mentalement la position. Ainsi un article publié par Pina Morini en 1946 dans la revue italienne Minerva Medica estimait qu'il existait différents types de mémorisation de la position, visuelle pour certains et linguistique pour d'autres (qui relisent mentalement la liste des coups joués). En revanche, l'analyse d'une position se fait toujours par bloc logique et par analogie avec les positions connues.

De Groot s'intéressa ensuite à la façon de raisonner de chacun de ses cobayes. Il leur présenta des positions et leur demanda quel coup ils joueraient et pourquoi. Il en dégagea les enseignements suivants :

- La plupart des positions sont tout d'abord pensées par analogie avec des positions déjà rencontrées. Ces analogies permettent d'isoler les coups qui peuvent permettre un gain ou éviter une défaite, toujours en rapport avec des stratégies globales à long terme.

- La recherche du joueur est très sélective. L'aptitude principale des grands maîtres est leur capacité à identifier rapidement le ou les bons coups, à analyser sérieusement, précisément par analogie. Une fois ces coups identifiés, tous les autres coups sont ignorés après une analyse superficielle.

- La profondeur du raisonnement est faible. Aucun GMI n'examine une position à plus d'une dizaine de demi coups de profondeur (alors que les ordinateurs les plus puissants peuvent examiner jusqu'à quinze ou vingt demi coups).

- La fonction d'évaluation utilisée est floue ; elle contient des concepts difficiles à identifier clairement, comme la notion de mobilité utile, la pression d'attaque, ou de position active… ces critères remplacent l'analyse en profondeur qu'un joueur humain ne peut effectuer en raison de la limitation de ses capacités mentales. Très souvent l'évaluation est aussi effectuée par référence à des schémas connus, faciles à expliquer dans certains cas (position centralisée des Tours, Fous de la « bonne » couleur pour jouer la finale) mais parfois, plus indistincts, voire liés au joueur lui-même.

- Le joueur peut effectuer des changements de plan en fonction des analyses qu'il effectue. Ces changements de plan l'amènent alors à reconsidérer les coups à examiner en profondeur.

Les études de Kroguious sont lues et orientées vers une analyse utilitaire de la psychologie du joueur d'échecs. Capitaine de plusieurs équipes soviétiques, il cherche à dégager quels sont les paramètres qui interviennent dans le jugement du joueur d'échecs et renforcent ou dégradent la qualité de son analyse. Il met en évidence l'existence d'images résiduelles, à la fois à long terme et à court terme. L'image résiduelle à long terme permet de reconnaitre dans une position courante une référence à la position déjà vue, et d'appliquer ainsi immédiatement le même type de plan.

Les études de De Groot et de Kroguious sur les joueurs d'échecs sont donc particulièrement intéressantes car elles recouvrent en fait les techniques de raisonnement des humains face à la plupart des problèmes qui leur sont posés : la base de connaissances est fondamentale et l'essentiel de la résolution se fait par référence à des exemples déjà rencontrés.

Ceci est rendu possible par la formidable capacité du cerveau humain à stocker et trier les informations, et à retrouver instantanément (ou presque) l'ensemble des informations relatives à un objet donné, pourvu que l'on fournisse un identificateur suffisamment clair de l'objet.

Ce qu'il y a parfois de plus remarquable encore c'est que nombre d'informations peuvent disparaitre, mais certaines associations apparemment sans valeur subsistent. Il faut d'ailleurs noter la fantastique quantité d'informations que le cerveau emmagasine tout au long d'une existence. Il faut également souligner que la capacité à mémoriser les informations, si elle est importante, n'est pas primordiale.

Ce qui est réellement important c'est la capacité que l'on a à récupérer efficacement les informations quand on est confronté à certains indices. Tous les enseignants savent bien que faire comprendre quelque chose est important, mais qu'il existe certains élèves qui, bien ayant compris, restent incapables d'appliquer la méthode apprise dès que le problème est légèrement modifié dans sa présentation.

La technique est mémorisée, mais le cerveau est incapable de reconnaitre l'analogie des situations. Il nous est impossible, à l'heure actuelle, de modéliser comment le cerveau détecte les analogies. Ce fut la cause de l'échec du GPS de Newell, Simon et Shaw et reste le problème majeur de résoudre aujourd'hui.

Si on savait le faire, on serait non seulement capable de construire des ordinateurs reproduisant le raisonnement humain, mais on serait peut-être aussi capable de rendre les humains plus intelligents en développant les capacités de leur cerveau à raisonner correctement.

3. La programmation des jeux :

Depuis le 18ième siècle, des scientifiques et des chercheurs se concentrent sur la réalisation de machines et d'automates capables de jouer à des jeux tels que les échecs et les dames.

Depuis quelques années, certains chercheurs en informatique se concentrent sur la modélisation de jeux et la façon de faire des programmes efficaces capables de jouer à des jeux en ayant un temps de réponse acceptable.

Dans le cadre de ces recherches, certains programmes pouvant jouer à des jeux (échecs, tic-tac-toe, dominos, backgammon, etc.) ont été effectués et analysés.

La résolution de programmes qui jouent à des jeux est une excellente application de l'intelligence artificielle pour les raisons suivantes : la planche ou l'environnement jeu et les règles du jeu sont faciles à modéliser, contrairement à la modélisation de l'environnement et des règles du jeu, il est difficile à l'ordinateur de calculer les bons mouvements ou les bonnes actions à effectuer. L'ordinateur doit analyser le jeu de façon similaire à un joueur humain, il existe certains maîtres humains de ces jeux qui sont généralement heureux de donner leur avis sur les performances du programme de jeu. Les scientifiques ayant crée le programme vont pouvoir se servir de ces critiques afin d'améliorer le programme; et des tournois sont généralement organisés dans la plupart de ces jeux et les grands maîtres peuvent se mesurer avec l'ordinateur. Les performances de l'ordinateur peuvent ainsi se comparer directement avec l'expert humain.

4. Programmation automatique (génération de code) :

Écrire et déterminer un programme complexe est une tâche qui demande beaucoup de temps. Une grande équipe de programmeurs peut travailler plusieurs années pour produire un compilateur ou un système d'exploitation et le résultat n'est jamais sans erreurs.

Généralement, même si le programme est dans sa version finale, il va quand même contenir des erreurs appelées « bugs » qui vont nuire au bon fonctionnement du logiciel jusqu'à temps que ces erreurs soient trouvées et éliminées. Cette situation caractérise la production de logiciels comme victime de la « crise du logiciel ».

Un moyen d'éviter cette situation est de faire en sorte que l'ordinateur puisse générer du code à partir des énoncés du problème à résoudre. Une telle application génératrice de programmes serait une extension des compilateurs actuels et des générateurs de rapports. Ces applications produisent également des programmes, mais ils demandent au programmeur de décrire l'application à produire en plus grands détails.

5. La logique programmée :

Parfois, les scientifiques ont besoin de prouver qu'une série de faits sont une conséquence logique d'autres faits. Un avocat de la couronne, par exemple, doit prouver hors de tout doute et avec preuves à l'appui que l'accusé a commis le crime dont on l'accuse. La logique programmée est l'art de programmer un ordinateur afin qu'il puisse faire ces déductions.

Des applications évidentes de la logique programmée se retrouvent dans les mathématiques. Un mathématicien débute avec un petit ensemble de faits (appelés axiomes) et déduit de ces faits des conséquences intéressantes (appelées théorèmes). Le mathématicien garantit que dans toute situation où les axiomes sont vrais, les théorèmes seront également vrais. Nous pouvons donc appliquer ces théorèmes dans des cas scientifiques sans devoir répéter les déductions des mathématiciens.

L'ordinateur est ainsi capable d'effectuer des preuves mathématiques lorsque nous lui fournissons des théorèmes.

Une autre application de la logique programmée est de prouver qu'un programme est correct. Ceci est une alternative à la programmation automatique. Le programmeur humain écrit un programme. L'ordinateur essaie de prouver que le programme est correct; c'est à dire qu'il résous le problème qu'il a été crée pour résoudre. Si l'ordinateur réussit, nous pouvons utiliser le programme avec confiance, sinon le programmeur devra découvrir pourquoi la preuve a échoué, probablement dû à une erreur dans le programme.

D'autres programmes d'intelligence artificielle ont également besoin de la logique programmée, en particulier ceux impliqués par la recherche d'information et la représentation de connaissance. Certains faits sont entreposés dans l'ordinateur. D'autres faits sont recherchés par le programmeur ou le programme travaillant sur un problème. L'ordinateur doit essayer de différencier les informations demandées des informations disponibles.

Partie 2 : La Théorie des Jeux

Chapitre IV : L'intelligence artificielle et les jeux

L'analyse théorique qui portera sur notre application est construite autour de deux aspects principaux suivants : La théorie des jeux et son application à la programmation d'un jeu d'échecs. Mais avant tout cela, il est important de vous faire connaître la vraie nature de l'intelligence artificielle, ensuite, nous présenterons les principaux algorithmes de la théorie des jeux, et après une brève comparaison sur leur efficacité, nous en choisirons un pour notre projet. Quand tout ceci sera en place, nous aborderons les aspects plus propres au jeu comme la représentation de l'échiquier en machine et la fonction d'évaluation.

1. L'intelligence artificielle

Jusqu' à aujourd'hui, il n'existe pas vraiment de consensus sur la définition du terme « intelligence artificielle ». Voici quelques définitions tirées de la littérature :

- « L'étude des facultés mentales à l'aide des modèles du type calculatoires » *Charniak et McDermott, 1985*
- « conception d'agents intelligents » *Poole et al., 1998*
- « discipline étudiant la possibilité de faire exécuter par l'ordinateur des tâches pour lesquelles l'homme est aujourd'hui meilleur que la machine » *Rich et Knight,1990*
- « L'études des entités ayant un comportement intelligent » *Nilsson, 1998*

2. Les 2 approches de l' IA :

❖ Approche biologique de l'IA, IA forte :

On cherche à créer une IA reproduisant à l'identique le fonctionnement du système cognitif humain. Selon cette approche, la machine pense et éprouve de vrais sentiments. L'atout majeur de ce système est l'apprentissage. En effet, cette capacité fondamentale est de pouvoir emmagasiner de nouvelles informations et même dans certains cas, modifier son propre fonctionnement. Cependant, jusqu'à aujourd'hui, les ordinateurs actuels ne sont pas capables de créer une telle intelligence. Ils ne possèdent pas le langage approprié.

❖ Approche ingénieur, IA faible

Ici, au lieu de tenter de reproduire le système cognitif humain, on cherche plutôt à donner des comportements intelligents à des systèmes de calcules , ils agissent « comme si » ils étaient intelligents.

Mais comme précédemment, cette approche considère qu'un programme peut être capable de raisonner, d'apprendre et même de résoudre. Les différences avec l'IA forte se situe donc à ces niveaux : l'IA faible ne pense pas, elle donne « l'impression de ». Elle reflète donc les idées et la logique des humains. Enfin si une IA faible est suffisamment efficace pour donner l'impression de se trouver devant un comportement intelligent, elle a atteint son objectif.

3. Utilisations concrètes de l'IA :

Bien que nous ne nous en rendions peu compte, l'IA est déjà présente dans notre vie quotidienne. En entendant, « Intelligence Artificielle », nous nous attendons à des dispositifs futuristes et miraculeux à apparence humaine, dotés de jugements qui leurs sont propres, la réalité est tout autre.

Car jouer à un jeu vidéo, traduire automatiquement un texte à l'aide d'un logiciel, se servir d'un moteur de recherche et utiliser la reconnaissance faciale pour déverrouiller un ordinateur représentent des activités faisant appel à l'IA.

3.1 Les jeux vidéo :

Les jeux vidéo sont une mine à IA. D'ailleurs, ils ont permis des avancées importantes en ce qui concerne le développement de l'IA faible.

Les jeux vidéo tout comme l'IA, sont historiquement très récents. Mais leur évolution a été extrêmement rapide de sorte que, en l'espace de moins de 40 ans, on est passé de Pong à Starcraft. Ainsi l'industrie du jeu vidéo, bien que parfois oubliée, a grandement participé à l'amélioration de l'IA faible car pour rendre une expérience de jeu plus complète, rien de tel que de simuler « une vie » à l'intérieure de son ordinateur ou console préférée. Grace à cela, on est passé du jeu où il fallait être 02 à des jeux où l'expérience est « quasi » complète en étant seule.

3.2 L'évolution du jeu d'échecs :

De nombreux jeux de société sont aujourd'hui disponibles sur ordinateur, qu'il s'agisse du Monopoly, ou du jeu Risk.

Ces différents jeux proposent des défis qui sont différents, et le programmeur devra savoir quelle technique utiliser pour qu'une intelligence artificielle fournisse un bon rendement. Dans ce chapitre, nous exposerons les problèmes que propose le jeu d'échecs en IA. Nous décrirons un système de pointe et actuel.

Le jeu d'échecs tel qu'il est connu aujourd'hui a été inventé au XVI^{ème} siècle. Depuis longtemps, les hommes se sont intéressés à inventer des machines capables de jouer aux échecs. Outre les blagues où un homme se cachait dans une boîte pour jouer au jeu, les premières machines capables de jouer aux échecs datent du début du siècle précédent. En 1912, Torres construit un jeu d'échecs « Ajedrecista », utilisant l'électromagnétisme sous le plateau pour jouer les finales de Tour et Roi contre Roi. Il s'agit d'une simulation de fin de partie dans laquelle l'automate possède un Roi et un pion, tandis que le joueur perdant n'a qu'un Roi. L'automate gagnera peu importe les mouvements du joueur. Evidemment quand les ordinateurs modernes ont fait leur apparition, plusieurs systèmes informatiques ont été créés par différents programmeurs en utilisant différents techniques. Parmi les plus connus, on entend souvent parler de l'ordinateur Deep Blue d'IBM. Deep Blue est devenu célèbre en battant le champion de l'époque Garry Kasparov, en mai 1997.

Pour analyser un système d'intelligence artificielle qui joue à un jeu de société, il faut à la base parler de l'ordinateur qui exécutera le programme et de l'algorithme qui sera utilisé. Dans le cas de Deep Blue, l'ordinateur avait été créé sur mesure et avait une puissance de calcul de 300 millions de coup par seconde.

La philosophie d'IBM consistait à utiliser la puissance de calcul brute pour évaluer plus d'option possible. L'algorithme utilisé pour rendre le choix des mouvements était basé sur l'algorithme Min-Max. l'algorithme exact n'a pas été dévoilé, et Deep Blue a été démantelé peu de temps après sa victoire. L'algorithme Min-Max consiste à calculer toutes les options de jeu sur un nombre prédéterminé de tours. Cet algorithme est un classique des jeux de société, d'ailleurs dans les systèmes les plus efficaces d'aujourd'hui, il est encore utilisé. Il se joue un tournoi qui se nomme « World Computer Chess Championship ».

Le dernier champion a été le programme « Rybka ». Pour des raisons de concurrence, le code de ce programme n'est pas connu. Par contre, on sait que ce programme utilise, à la base, l'algorithme Min-Max.

3.3 Représentation du déroulement d'une partie :

Le jeu d'échecs est un jeu où les deux joueurs jouent séquentiellement. Le but des deux joueurs est opposé et incompatible. Le joueur blanc veut mettre le roi noir mat tandis que le joueur noir veut mettre le roi blanc mat. Tour à tour, un joueur puis l'autre choisit parmi tous les coups possibles celui qui lui paraît le meilleur pour parvenir à son objectif.

Ainsi il est possible de représenter une partie de jeu par un arbre. Chaque sommet représente une position possible et les arcs sont tour à tour les coups jouables par le joueur blanc puis par le joueur noir puis le blanc…

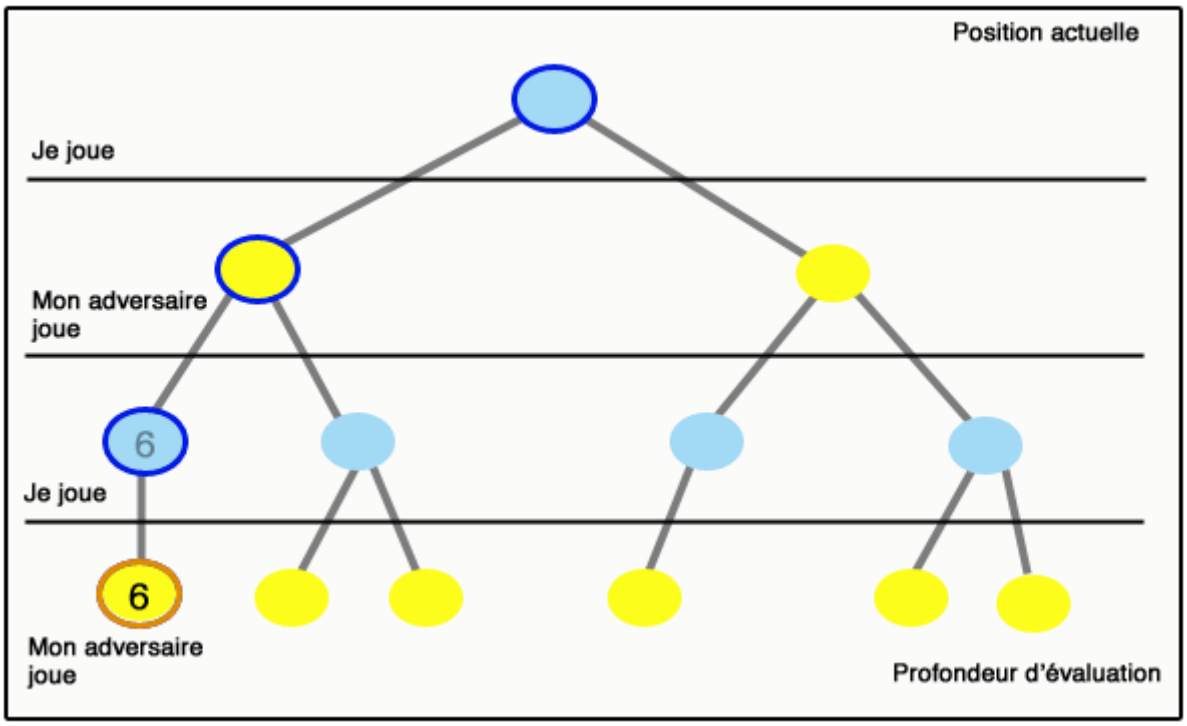


Figure 16 : Représentation en arbre de la partie

Il est donc possible de programmer une Intelligence Artificielle du jeu d'échecs qui va réfléchir à la conséquence de ses actes en regardant les mouvements possibles d'un joueur pour pouvoir jouer à sa place.

Chapitre V : Les algorithmes de recherches

1. Présentation de la théorie des jeux

La théorie des jeux est à la base de nombreux algorithmes utilisés, entre autre, dans les jeux vidéo. La théorie des jeux a pour but de décrire le comportement d'un individu (ou d'un agent de façon plus générale) lors d'un jeu en partant du principe qu'il souhaite gagner. Cette théorie vise donc à établir les stratégies optimales a l'aide de diverses modélisations, et d'établir « **l'équilibre du jeu** ». Elle est utilisée dans de nombreux domaines : l'économie, la politique ou encore la biologie. On trouve une autre application, de façon plus évidente, dans le domaine de l'intelligence artificielle.

Afin de rentrer dans les détails, nous allons donner quelques brèves définitions qui nous permettrons d'aborder des notions plus concrètes.

Pour appliquer la théorie des jeux, il est nécessaire de catégoriser les jeux. Pour cela, il existe de nombreux critères :

- Jeux simultanés : les joueurs jouent en même temps
- Jeux séquentiels : les joueurs jouent à tour de rôles
- Jeux finis : si les joueurs ont un nombre fini de stratégies
- Jeux à somme nulle : si les joueurs sont directement opposés
- Jeux à somme non nulle : si les deux joueurs peuvent trouver un intérêt commun
- Jeux à répétition : la stratégie à suivre varie considérablement si les joueurs ont une mémoire des parties précédentes
- Jeux à information complète si les joueurs connaissent : celles des autres joueurs ainsi que leurs motivations

Il existe plusieurs méthodes de modélisation de jeux :

- La forme stratégique (ou normale) : représentation sous forme d'une matrice
  Exemple pour le cas du jeu Pierre-Papier-Ciseau

		J_2		
		Pierre	Ciseaux	Papier
J_1	Pierre	(0,0)	(1,-1)	(-1,1)
	Ciseaux	(-1,1)	(0,0)	(1,-1)
	Papier	(1,-1)	(-1,1)	(0,0)

Figure 17 : Représentation sous forme de matrice

- La forme extensive : représentation sous forme d'un arbre

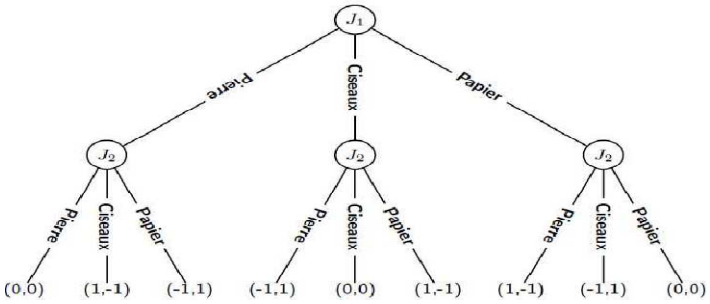


Figure 18 : Représentation en arbre

1.1 Concept de la solution :

C'est un processus qui permet déterminer les équilibres d'un jeu. Un concept de solution permettra donc de déterminer les stratégies les plus probables employées par les joueurs. Il existe de nombreux concepts de solution, la plupart sont des raffinements d'autres concepts (ils éliminent les équilibres peu plausibles).

On appelle meilleure réponse l'ensemble des stratégies permettant au joueur d'être dans la meilleure position au coup suivant sans aller bien loin

1.2 Equilibre de Nash :

L'équilibre de Nash repose sur le choix, à chaque tour et pour chaque joueur de la meilleure réponse. Par conséquent, la stratégie de chaque joueur est établie et aucun d'entre eux n'a intérêt à en sortir.

Le théorème de Nash stipule que parmi les stratégies possibles, il existe un équilibre parmi les stratégies mixtes.

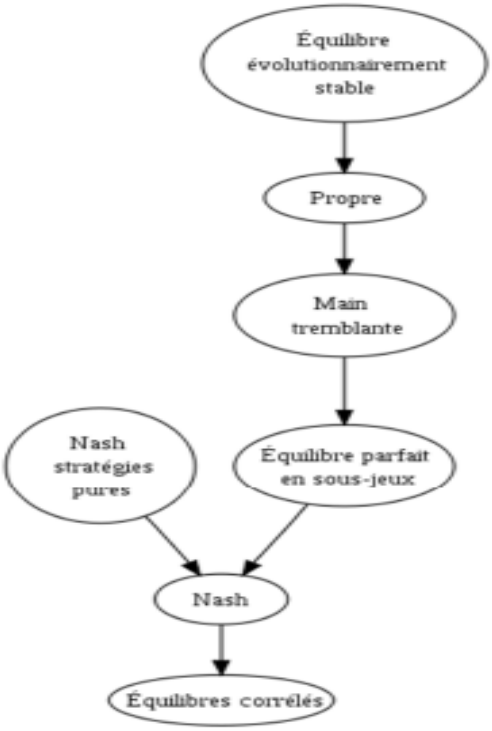


Figure 19 : Représentation de l'équilibre de Nash

2. Les méthodes algorithmiques de bases

Dans cette partie, nous allons présenter des algorithmes classiques utilises dans les jeux. Nous parlerons d'abord des arbres de recherches, qui constituent une branche de la théorie des jeux que nous venons d'introduire. Ils trouvent une application dans les jeux de plateau (Go, Echecs, etc.). Ensuite, nous nous attaquerons au problème de recherche d'un chemin « Pathfinding » et nous présenterons les algorithmes qui permettent de le résoudre. Les différents algorithmes que nous allons présenter restent couramment utilises dans les implémentations d'IA et constituent souvent des solutions privilégiées.

2.1 Arbre de recherche :

La présentation qui va suivre est directement liée a la partie concernant la théorie des jeux puisque les arbres de recherche sont une des façons les plus classiques de représenter les décisions possibles des joueurs. Pour rappel, en théorie des jeux, on appelle cette visualisation des possibilités la **forme extensive**.

Comme nous l'avons déjà évoqué, les arbres de recherches s'adaptent tout particulièrement aux jeux « classiques » (jeux de plateau se jouant au « tour par tour » : échecs, dames, go, cartes, etc.

Le principe de l'IA est simple :

1. Elle recherche les séquences de coup jouable d'une certaine longueur
2. Elle évalue la position résultante de la séquence
3. Elle choisit le coup optimal

Pour expliquer les algorithmes utilisés dans ces jeux, nous allons prendre un cas particulier qui nous intéresse, celui du jeu d'échecs.

2.2 Algorithme Min-Max :

Pour une position donnée des pièces sur l'échiquier, on va construire un arbre qui contient les séquences (d'une certaine longueur) de coup jouables. A partir de la, on évalue a l'aide d'une fonction d'évaluation les positions résultantes de chaque séquence.

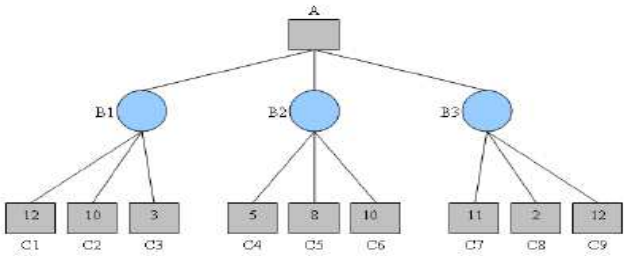


Figure 20 : Arbre de recherche - 1ère étape

On obtient l'arbre ci-contre :

On considère ici une séquence avec deux coups. Plus la valeur de la ligne C est haute, plus la position est intéressante pour l'IA. Par conséquent, l'adversaire n'a pas intérêt à les choisir. Cet algorithme se base donc sur une hypothèse fondamentale : l'adversaire joue pour gagner et va donc maximiser son score, c'est-a-dire minimiser le score de l'IA. On remplit de cette manière la ligne B (en choisissant la valeur minimale de chaque sous arbre) :

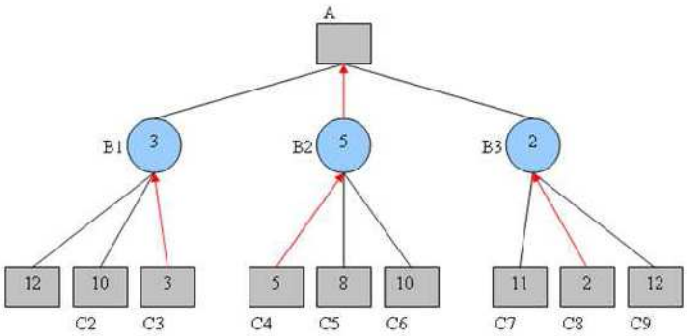


Figure 21 : Arbre de recherche - 2ème étape

Pour finir, on choisit la valeur maximale de la ligne B : 5. On obtient, avec cet exemple basique, l'idée directrice de l'algorithme MinMax : maximiser le minimum que peut nous offrir l'adversaire.

Le principal inconvénient de cette technique vient du fait que l'on explore l'arbre dans son intégralité alors que certaines branches pourraient être éliminées des le début (sachant que chaque nœud est de l'ordre de 35 fils). Ce problème est corrigé par l'algorithme d'élagage alpha-beta.

2.3 Alpha-Beta :

Le principe du MinMax est conserve. On l'améliore cependant en éliminant les séquences inintéressantes.

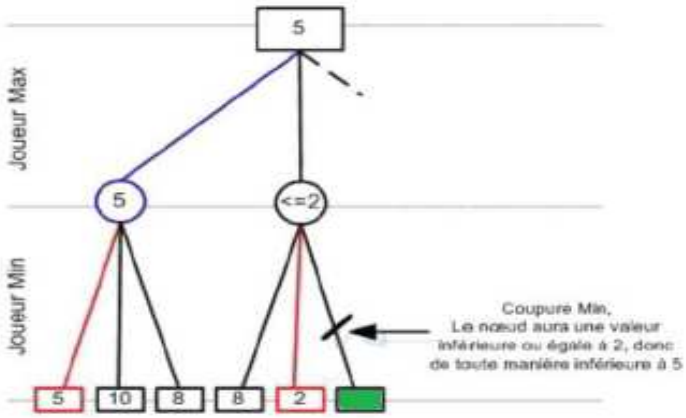


Figure 22 : Arbre de recherche - coupure Min

Ici, par exemple, on élimine la branche verte pour la raison suivante : la valeur « 2 » de la case précédente est inferieure la valeur « 5 » de la branche bleue ; étant déjà inférieure, il n'y a aucune raison de continuer d'espérer une stratégie plus intéressante dans ce sous arbre.

L'idée s'adapte évidemment au cas suivant :

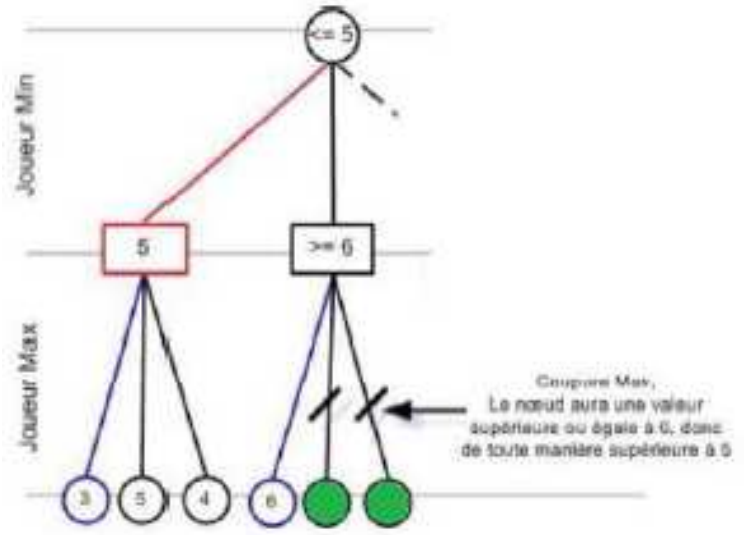


Figure 23 : Arbre de recherche - coupure Min

La valeur « 6 » (case précédent la case verte) est supérieure a « 5 » (case rouge).

2.4 La fonction d'évaluation :

Le rôle de cette fonction est de comparer deux positions. La difficulté de développer une IA performante repose en grande partie sur la construction d'une fonction d'évaluation pertinente et performante. La difficulté réside dans le fait que le nombre de possibilité est énorme. Les algorithmes précédents fournissent un grand nombre de positions.

Si la fonction d'évaluation réclame trop de calculs, cela risque de nécessiter des ressources mémoires colossales. Une autre difficulté réside dans le "calibrage" de la fonction (définir la gravite ou l'intérêt d'une position). Une position n'ayant pas la même valeur au cours de la partie.

Voici une fonction (partielle) permettant le développement d'une IA correcte :
Chaque pièce se voit attribuer une valeur. On somme les valeurs des différentes pièces de l'IA pour obtenir la force de l'IA (même opération pour l'adversaire). Ces grandeurs ne tiennent néanmoins pas compte des positions des pièces et notamment du contrôle ou de l'occupation du centre de l'échiquier !

Afin de prendre en compte ces éléments, des bonus/malus viennent moduler la force.

Par exemple une position dans laquelle une pièce majeure (pas les pions ni le roi) serait immobilisée donnera un malus. Tandis que le fait de positionné un Fou sur une diagonale ouverte donnera un bonus. Egalement dans le but de privilégier la zone centrale de l'échiquier, on peut attribuer une valeur a chaque case de telle façon que, plus en s'éloigne du centre, plus la valeur diminue. Au final, l'évaluation est donnée par la force de l'IA moins la force de l'adversaire auquel on ajoute les bonus et soustrait les malus.

De plus, dans le but de démarrer efficacement la partie, l'ordinateur doit être doté d'un stock d'ouvertures de telle manière que, si une position enregistrée est reconnue, alors l'ordinateur pourra lui associer le meilleur coup à jouer.

Enfin, pour terminer efficacement la fin de partie, il faut constater la complexité stratégique due au fait que les pièces sont sur un "plateau plus ouvert" et ou il n'est pas rare que les séquences de coup permettant le gain dépassent la profondeur de l'arbre. Actuellement, la solution utilisée consiste en des tables qui analysent toutes les configurations avec cinq pièces majeures ou moins. Il ne reste plus qu'a implémenter une analyse rétrograde pour trouver la meilleure solution.

2.5 Bilan :

Ces stratégies ont beau être les précurseurs de la plupart des stratégies d'IA dans les jeux vidéo, le temps de calcul très important lié au nombre de coups élèvé ne la rend adaptable qu'à certaines composantes des IA. On peut donc la retrouver, par exemple, dans certains jeux où il faut conquérir des zones.

2.6 Pathfinding :

Nous allons maintenant nous intéresser à un problème très courant dans les IA des jeux vidéo: la recherche de chemin ou « pathfinding ». Le problème est simple, comment se déplacer d'un point A à un point B avec, si possible, le « meilleur » itinéraire? L'efficacité de l'algorithme utilisé est primordiale au niveau de la jouabilité et du réalisme. Combien de fois a-t-on vu des PNJ traverser un mur ou bien rester bloques par une caisse ? Un exemple d'utilisation courant est celui du jeu de stratégie (tel que *Warcraft*), ou le joueur peut demander a une ou plusieurs unités de se déplacer vers un point pour qu'il défende sa citadelle au prix de sa vie.

Les joueurs étant parfois nerveux, l'IA ne peut se permettre de parcourir toutes les impasses de la carte avant d'arriver sur les ruines de son ancien campement. Le pathfinding constitue donc un plan fondamental de l'IA dans les jeux vidéo.

Le pathfinding trouve des applications dans de nombreux domaines autres que les jeux vidéo (robotique, routage, internet, etc.). De nombreux algorithmes permettent de résoudre ce problème et tous ont des particularités en matière de précision et de rapidité.

L'univers du jeu vidéo est extrêmement contraignant, ce qui limite les algorithmes utilisables. En effet, les solutions doivent utiliser un minimum de ressources et être les plus rapides possibles, au détriment parfois de la précision. Par ailleurs, il faut parfois prendre en compte des contraintes additionnelles telles que la gestion d'un groupe d'unité, ou le fait que l'environnement soit évolutif.

Les algorithmes les plus utilisés sont sans aucun doute celui de Dijkstra et l'algorithme A*. Même si ceux-ci trouvent une application idéale dans les jeux en 2D, nous verrons par la suite qu'ils sont également trés utilises dans les jeux en 3D en complément d'autres méthodes.

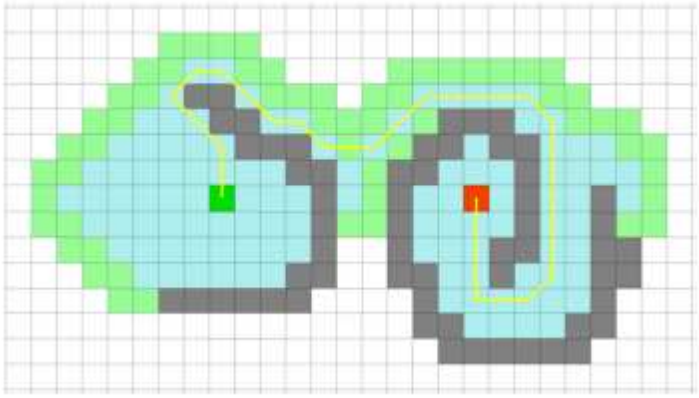


Figure 24 : Exemple de simulatin de Pathfinding

2.7 Algorithme de Dijkstra :

La théorie expliquant cet algorithme étant assez complexe, nous allons essayer de le simplifier et de donner les idées directrices.
Tout d'abord, cet algorithme demande des notions de base en théorie des graphes.
Un graphe se compose de nœuds (ici des villes), et d'arêtes (les jointures entre différents nœuds), ces dernières étant affectées d'un poids (ici le nombre de kilomètres séparant chaque ville l'une de l'autre).

Principe : il s'agit de trouver le chemin ayant le poids le plus faible (somme des poids des arêtes qui le composent) entre 2 nœuds *s1* et *s2*.

Résolution : on cherche a construire une succession de sous graphe *G'* de *G* tel que tout sommet *s* de *G'* ait une distance minimale avec *s1* dans *G*. Voici l'algorithme que l'on peut suivre :

- On initialise le premier sous-graphe a *s1* seul. Les autres sommets sont dits « non-marques » (on affecte la valeur $+\infty$ a leur poids).

- Tant qu'il existe un sommet non marque :

o On choisit le sommet non marque *a* de plus faible poids *P(a)*,

o On le marque,

o Pour chaque sommet *b* voisin de *a*, on lui attribue le poids

$$P(b) = min(P(b), P(a) + dist(a,b)).$$

L'algorithme se termine quand *G'* devient un arbre couvrant (contenant *s1* et *s2*) ou que tous les noeuds d'intérêt (noeuds qui ont plus d'un voisin) sont dans *G'*.

2.8 L'algorithme A* :

L'algorithme A* est de loin le plus utilise dans les jeux video car il répond bien aux contraintes imposées par ceux-ci.

Il constitue une amélioration de l'algorithme de Dijkstra. On dit de cet algorithme qu'il est informé, ce qui signifie qu'il utilise une fonction dite heuristique qui estime la distance à l'arrivée.

Contrairement a l'algorithme de Dijkstra dit « aveugle », A* oriente sa recherche au lieu d'explorer toutes les zones. Le gain de temps est conséquent.

La question réside donc dans la façon de calculer la distance heuristique pour orienter au mieux l'algorithme. La fonction heuristique évalue d'abord les sommets proches de l'arrivée ou du départ (plus proche du raisonnement humain).

La fonction heuristique est une caractéristique de la recherche « gloutonne ». Il existe deux principales heuristiques de distance dans les jeux vidéo : la distance euclidienne (vol d'oiseau) et la distance de Manhattan (jeux se déroulant sur un damier).

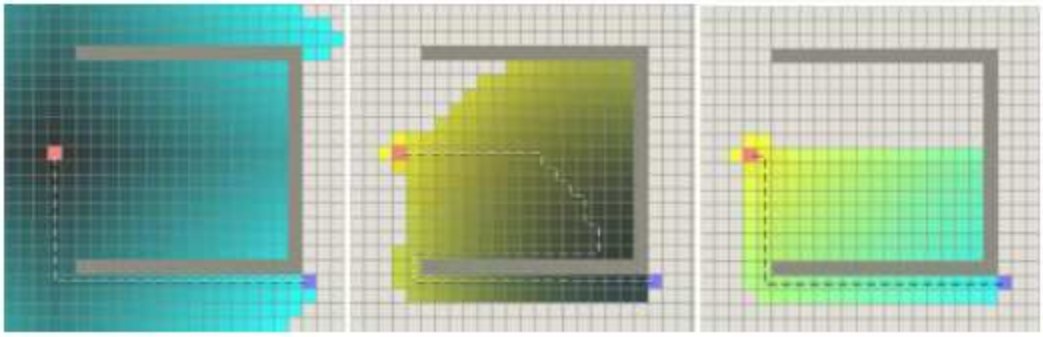


Figure 25 : De gauche à droite : Dijsktra, Glouton, A*

Dijkstra et A* retourne un chemin optimal contrairement a la recherche gloutonne. Cependant, Dijkstra explore beaucoup trop de cases.

2.9 Autres solutions :

A* est satisfaisant pour la plupart des jeux. Cependant, avec les contraintes toujours plus importantes imposées dans les jeux video (destruction de terrain par exemple), d'autres algorithmes ont été développé.

Voici une courte description des principaux algorithmes :

- D* permet de redéfinir le chemin si l'IA rencontre un obstacle
- IDA* n'explore qu'un seul chemin à la fois (ce qui permet une économie d'énergie), il arrive cependant que certains sommets soient revisites plusieurs fois
- HPA* établi une hiérarchie entre les différentes zones d'une carte

2.10 Cas des jeux 3D :

Ces algorithmes sont très adaptes aux graphes abstraits, mais on se voit mal proposer au joueur une carte sous forme de graphe vu dans l'exemple de Dijkstra (cf. ci-dessus).
Plus la résolution des cartes est importante, plus l'abstraction inhérente aux différents algorithmes vu précédemment semblent compromettante pour les adapter au pathfinding. La question est donc la suivante :

Comment adapter ces techniques aux cartes en 3D ?

Il existe deux principales méthodes :

- Le système des **waypoints** : on plaque un échiquier sur la carte et chaque case constitue un sommet du graphe. On applique alors les méthodes vues précédemment.

- Le **mesh de navigation** : on fabrique un graphe a partir de polygones. Les personnages peuvent se déplacer à l'intérieur de ceux-ci. Les polygones doivent être de taille suffisamment faible pour y utiliser la méthode A*.

Il existe évidemment d'autres méthodes de pathfinding qui ont plus ou moins succès.

Dans la dernière partie, nous verrons les **réseaux de neurones** qui sont utilisés dans *Black and White* et qui permettent de résoudre ce problème.
Les **algorithmes génétiques** sont également utilises dans *Creatures* à cet effet.

Chapitre VI : Les optimisations possibles

1. Optimisation de la fonction d'évaluation :

1.1 Mobilité et territoire :

Veiller à conserver une grande mobilité des pièces du jeu permet de conserver le maximum de possibilités pour les coups suivants et donc de préserver un maximum de chance de remporter la partie.

C'est pourquoi les positions suivantes devront être pénalisées :

- position où des pièces de valeur supérieure aux pions seraient bloquées dans leur mouvement ;
- position où le cavalier perd une partie de sa mobilité car il est trop près d'un bord.

Les positions suivantes se verront octroyées un bonus :

- position où les fous sont placés sur des diagonales ouvertes ;
- position où les tours sont placées sur des colonnes ouvertes ;
- position où les tours sont doublées sur les colonnes ouvertes.

Une autre priorité quand on joue aux échecs est de préserver le contrôle la zone centrale de l'échiquier. Une solution serait d'accorder un bonus de valeur à toute pièce se situant au centre. En outre, il faudrait aussi favoriser la « centralisation » des cavaliers. On pourrait étendre cette fonctionnalité en attribuant une valeur à chaque case de l'échiquier. Plus la case en question serait proche du centre, plus son bonus de valeur sera fort. On considèrera également qu'un joueur contrôle une case si le nombre de ses pièces attaquant cette case est plus grand que le nombre des pièces adverses attaquant cette même case.

1.2 Préservation des pièces dites importantes :

La victoire passe par la défense de son propre roi. Ainsi, les deux mouvements de roque du roi seront bonifiés en valeur tandis que toute intention de déplacer des pions visant à affaiblir le roc sera pénalisée.

Le tropisme représente la pression à laquelle est soumis le roi par les pièces adverses. Il peut s'agir par exemple d'une tour adverse qui mettrait en échec le roi si la pièce située entre le roi et la tour venait à bouger ou se faire prendre par la tour. Ainsi, plus la valeur du tropisme sera élevée, plus le roi se trouvera en danger.

Démarrant à zéro, toute valeur de tropisme non nul sera un malus. Il sera aussi possible d'adapter ce principe de calcul aux autres pièces.

Ainsi, il faudra commencer à faire le jeu et attaquer avec des pièces dites mineures (Cavalier, Fou, pions) afin de préserver les tours et la reine pour le reste du déroulement de la partie. Il sera néanmoins accordé des bonus aux combinaisons favorisant la survie des deux Fous ou la conservation du fou de la « bonne couleur » (un fou est dit de la bonne couleur s'il peut attaquer les cases où sont bloqués les pions de l'adversaire).

1.3 Formation des pions :

La maîtrise des pions, bien qu'ayant une valeur très faible individuellement, permet en général de s'approcher de la victoire. Certaines combinaisons de pions devront être pénalisées tandis que d'autres seront bonifiées.

Les positions pénalisées seront :
- pion doublé ou triplé : plusieurs pions sur la même colonne se gênent dans leur mouvement ;
- pions bloqués : deux pions se faisant bloquer face à face ;
- pions passés : un pion qui ne peut être que bloqué ou pris par l'adversaire constitue un grand danger car il menace la ligne arrière ;
- pion isolé : un pion qui n'a aucun pion ami sur ses cotés ;
- les 8 pions : avoir ses 8 pions sur la même ligne gène le déplacement.

Sera bonifié toute intention visant à rapprocher un pion de la dernière ligne arrière de l'ennemi afin d'y obtenir une dame. Le bonus sera d'autant plus grand que la distance sera faible.

1.4 Réduction du nombre de possibilités à évaluer :

Etant donné qu'on ne bouge qu'une pièce par coup, il paraît logique qu'un coup ne nécessite pas forcément d'évaluer une nouvelle fois les positions de toutes les autres pièces de l'échiquier. Une telle idée permettra ainsi de réduire le nombre de calculs de positions par coup calculé. Cependant, il faudra concevoir un module qui définira pour quelles pièces de l'échiquier il faudra recalculer ou non le coefficient.

Il est également possible de réduire le temps d'exécution de la fonction d'évaluation en réduisant le nombre de coups qu'elle a à évaluer.

Ainsi tous les mouvements inutiles (ex : aller-retour d'une même pièce) seront à identifier et à bannir. En outre, les coups qui ne viseront rien d'autre que d'immobiliser les pièces adverses seront également exclus. En effet, une telle tactique ne conduirait qu'à étaler ses forces et rendre sa propre défense vulnérable, si jamais le roi se retrouve en positon d'échec, les seuls coups évalués seront les coups permettant de sortir le roi de sa situation d'échec. Le programme sera ainsi allégé du calcul d'autres coups superflus.

Notons cependant un point capital : *la notion de plan stratégique d'ensemble est absente.* On peut, certes, introduire dans la fonction d'évaluation quelques éléments pseudo-stratégiques, comme l'attaque sur le roque adverse « à la baïonnette[47] », mais cela reste très limité. Il s'agit bien là du principal défaut des programmes d'ordinateurs.

Cela est particulièrement sensible lorsqu'ils passent de la séquence d'ouverture qu'ils jouent « par cœur » au milieu de partie. En effet, à chaque ouverture est associée une idée stratégique et la suite de la partie doit développer cette idée. Ainsi une sicilienne doit être jouée de façon agressive par les blancs qui doivent attaquer rapidement le petit roque noir, alors que la chance des noirs se situe en général à l'aile dame. Or un programme est incapable de saisir ces subtilités stratégiques et appliquera la même fonction d'évaluation quelle qu'ait été l'ouverture. Tous les programmes d'ordinateur, même les meilleurs, manquent de « liant » dans leur jeu, ce qui rend leur style facilement reconnaissable et les laisse encore vulnérables.

On ne connaît actuellement aucune méthode pour remédier à cet état. Simplement, la force tactique des programmes leur permet de compenser leur faiblesse stratégique.

La technique de recherche dans l'arbre de jeu est un algorithme-tel que nous l'avons décrit au paragraphe précédent. La technique de contrôle de temps de jeu et de reclassement des coups avant d'aborder des recherches plus profondes est également identique. Il faut cependant introduire certaines techniques indispensables aux échecs :

L'élagage de futilité : Nous savons que l'algorithme α-β élague une branche au niveau dès que l'évaluation partielle de cette branche est inférieure (ou supérieure si le niveau est pair) à l'évaluation définitive du niveau n-1. Supposons par exemple que l'évaluation du niveau n-1 soit 3 et qu'une évaluation partielle au niveau n soit 3.01. Certes, l'évaluation n'est pas inférieure, mais il est clair qu'elle ne sera pas non plus franchement meilleure. On peut donc élaguer cette branche. Cette technique est appelée *élagage de futilité.*

Le coup meurtrier : Supposons que lors de la recherche α-β, l'algorithme note que le coup de premier niveau ρ_1 est détruit par la riposte de d_1 (c'est-à-dire que si l'adversaire répond d_1 après ρ_1 le score de la position s'effondre). Cela signifie que d_1 est un *coup meurtrier*. Lors de l'examen du coup de premier niveau suivant ρ_2 la première riposte que l'algorithme α-β devra examiner sera précisément (si cela est possible)d_1 car si le coup d_1 a été meurtrier pour ρ_1 il a de fortes chances de l'être également pour ρ_2.

De tels coups sont liés, aux échecs, à la notion de menace et la même menace reste souvent valable face à de nombreux coups. En procédant de cette façon, on améliore encore l'efficacité de l'élagage de l'algorithme α-β.

Utilisation du temps de réflexion de l'adversaire : Après avoir joué un coup, il est bon de retenirl'intégralité de la branche. En effet, on peut relancer la recherche α-β *pendant le temps de réflexion de l'adversaire* sur la position telle qu'elle se produirait si l'adversaire joue la riposte que nous estimons la meilleure pour lui. De cette façon, si l'adversaire joue effectivement ce coup, nous aurons déjà effectué une grande partie de l'évaluation de la position.

Retour à l'équilibre : Si l'algorithme α-β travaille à une profondeur fixe, il s'expose à de sérieux ennuis. Supposons en effet que l'algorithme α-β descende à une profondeur 3 et qu'il trouve une séquence du genre : je déplace mon Cavalier (premier niveau), on me le prend (deuxième niveau), je prends la Dame adverse (troisième niveau). Bien qu'il enregistre la perte de son Cavalier, le résultat est globalement positif puisque le programme comptabilise la prise de la Dame adverse. Mais supposons que le quatrième coup (non examiné puisqu'il s'est arrêté à la profondeur 3) est la prise de sa Dame. Le bilan global de l'opération est là franchement négatif. Pour éviter ce type de comportement, tous les algorithmes utilisent une stratégie dite *de retour à l'équilibre*. Une fois atteint leur profondeur maximale d'évaluation, ils poursuivent le calcul de la position terminale en continuant à examiner toutes les positions (et seulement les positions) découlant de prises ou d'échecs, jusqu'à ce qu'il n'y en ait plus. Ils peuvent ainsi éviter ce type de mauvaise évaluation.

Recherche secondaire : Un autre problème qui guette les ordinateurs est *l'effet horizon*.
Le programme, pour trouver la solution correcte, devrait effectuer la recherche à la profondeur 7, ce qui est prohibitif pour un ordinateur moyen. Certains écrivaient, il y a quelques années, que ce type de situation était ingérable par un ordinateur. En fait, la plupart

des bons programmes de jeu utilisent aujourd'hui une technique appelée *recherche secondaire* qui leur permet de trouver la solution du problème précédent en quelques secondes. Il s'agit simplement de relancer la recherche α-β à une profondeur supérieure, mais uniquement sur la le coup sélectionné. Il s'agit, en quelque sorte, d'une vérification de la validité du coup choisi. Comme l'évaluation n'est faite que sur la position terminale sélectionnée, le temps nécessaire est raisonnable. Certes, ce mécanisme ne permet pas de faire disparaître l'effet horizon, mais il en limite considérablement les effets.

L'algorithme SEX : L'algorithme SEX (Search EXtension) (Levy *et al.* 1989) a aussi pour but de limiter les problèmes d'effet horizon. Dans un mécanisme de recherche minimax classique, la profondeur de recherche est fixée au départ à et à chaque descente d'un niveau dans l'arbre, on diminue de 1 la valeur de . Lorsque atteint 0, la recherche est terminée. Avec l'algorithme SEX, on utilise une variable *SX* qui prend une valeur nettement plus grande que la profondeur à laquelle on veut chercher (une valeur typique est dix fois la profondeur, si vaut 3, *SX* vaut 30), mais à chaque descente d'un niveau dans l'arbre, on diminue *SX* d'une quantité variable en fonction de l'intérêt du coup joué. Ainsi, un coup « moyen » soustrait 10 à *SX*, une prise ou un échec est un coup intéressant, et on diminue peu *SX* (2 ou 3) ; en revanche, une retraite de pièces est un coup peu intéressant et on diminue beaucoup *SX* (jusqu'à 35). La recherche est terminée quand *SX* devient négatif ou nul.

De cette façon, les coups intéressants sont étudiés de façon plus profonde, et les coups « peu intéressants » sont rapidement abandonnés.

L'heuristique du « coup nul » : Appelé *Null Move Heuristic* en anglais, cette technique consiste à supposer que, dans une situation donnée, un joueur exécute deux coups successifs[50] (son adversaire ne joue pas, d'où le nom de « coup nul »). Si la situation ainsi générée n'est pas clairement favorable, alors le premier coup est élagué de l'arbre de recherche. Cette technique permet d'améliorer l'efficacité de l'élagage mais provoque parfois des élagages intempestifs, comme un match du programme Mephisto au tournoi ACM de 1991 le montra...

Signalons, pour conclure, un des inconvénients de la méthode minimax. Si l'ordinateur se trouve dans une situation perdante et qu'il a le choix entre deux coups, dont le second est légèrement plus mauvais que le premier mais tend un piège à l'adversaire, il sera incapable de s'en apercevoir. Le principe minimax choisira obligatoirement le premier, alors qu'il garantit presque assurément la perte de la partie, alors qu'un joueur humain tentera le tout pour le tout

et jouera le second coup. La notion de piège est bien difficile à formaliser, cependant Donald Michie proposa, il y a quelques années, une intéressante méthode qu'il est bon de rapporter. Supposons que nous ayons un arbre à deux niveaux que nous parcourons par une méthode minimax. L'ordinateur cherche la valeur S_1 du premier coup de niveau 1.

À partir de la position ρ_1 correspondant à ce coup, son adversaire peut atteindre trois positions ρ_{11}, ρ_{12} et ρ_{13} dont les scores sont respectivement S_{11}, S_{12} et S_{13} Théoriquement, l'ordinateur devrait remonter comme valeur au niveau 1 le minimum de ces trois valeurs. Michie propose de raffiner la méthode en prenant comme valeur du niveau 1 le minimum, mais la combinaison linéaire :

$$\rho_{11} x\, S_{11} + \rho_{12} x\, S_{12} + \rho_{13} x\, S_{13}$$

ρ_{11}, ρ_{12} et ρ_{13} représentent les probabilités que l'adversaire a de jouer les coups S_{11}, S_{12}, S_{13} d'après l'ordinateur.

Comment évaluer ces probabilités ? Michie a mis au point une méthode pour les échecs, qu'il appelle *modèle de discernement*. Il estime que le discernement d'un joueur face à une position donnée est donné par la formule empirique :

$$d = (C/100+1)\, 3^{1/(n+0.01)}$$

C représente le classement ELO du joueur que l'on affronte et *n* nombre de coups dont on a calculé une évaluation. Pourquoi cette formule ? Il est clair qu'un joueur joue d'autant mieux qu'il est plus fort. D'autre part, nous savons que plus il y a de coups différents possibles et plus le choix est difficile, surtout s'ils ont des valeurs proches. Michie estime alors que la probabilité qu'un joueur de discernement *d* face à une position joue un coup menant à un score estimé v sera :

$$\rho = A d^{\,v}$$

où *A* est un coefficient bien choisi pour normer la probabilité. Il serait intéressant de tester la méthode de Michie, mais aucun programme ne semble l'utiliser aujourd'hui.

Enfin, et d'après Feng-Hsiung Hsu (responsable du projet DEEP THOUGHT), les programmes d'échecs à l'heure actuelle n'utilisent pas de technique d'apprentissage :

« Une grave erreur doit être évitée : DEEP THOUGHT n'apprend pas. La fonction d'évaluation que nous utilisons est linéaire par rapport à presque tous ses termes (de 100 à 150). Nous ne faisons qu'une optimisation du modèle de mérite dans un espace multi-dimensionnel, en utilisant un mécanisme de régression linéaire. Ce processus produit des résultats intéressants, même s'il n'est pas clair qu'il produise toujours les effets désirés. »

2. Optimisation possibles :

2.1 Approfondissement itératif :

Construire l'arbre avec le principe de l'approfondissement consiste à en créer tous les noeuds à chaque niveau de profondeur jusqu'à trouver la solution finale au problème. On calcule ainsi tous les noeuds de profondeur n avant de passer à ceux de profondeur n+1. Etant donné que le nombre de coups possible est proportionnel au niveau de profondeur de l'arbre, le nombre de noeuds explorés à une profondeur n sera négligeable face au nombre d'itération n+1.

2.2 Le coup qui tue :

L'ordre dans lequel on considère les coups a une grande influence sur les performances de l'algorithme. Une des heuristiques les plus utilisées est de considérer que si un coup est très supérieur aux autres dans une branche, il peut être intéressant de l'explorer en premier dans les autres branches. Ce coup est appelé "coup qui tue".

La plupart des jeux d'échecs, utilisent cette heuristique en mémorisant le meilleur coup à chaque profondeur. On peut généraliser cette heuristique en gardant tous les mouvements légaux rencontrés dans l'arbre de recherche et en leur accordant une note. Cette note est proportionnelle à la profondeur du sous arbre exploré. La note du meilleur coup est augmentée de 2 Profondeur Max − p où p est la profondeur à laquelle le noeud a été exploré. Ensuite, on explore le noeud suivant l'ordre de cette note.

2.3 La recherche aspirante :

Lorsqu'on utilise l'algorithme alpha-beta, au lieu de démarrer avec les valeurs de +oo et –oo à chaque parcours de niveau de profondeur, on initialise la recherche avec des valeurs issues de recherches précédentes. On réduit ainsi le champ de recherche pour gagner en vitesse d'exécution. La valeur retournée sera correcte si elle est comprise entre les valeurs d'initialisation.

2.4 Alpha Beta trié :

Cela consiste à donner une « pré-note » à chaque noeud parcouru. Le parcours devra alors se faire de manière à avoir le plus de coupures possibles en utilisant une méthode de parcours de type "meilleur d'abord". En choisissant ainsi quels noeuds on va parcourir, nous pouvons ainsi augmenter le niveau de profondeur parcouru sans avoir à augmenter le nombre de feuilles parcourues.

Chapitre VII : Les tables de transpositions

Dans le chapitre sur la recherche, nous avons parlé du problème qui arrivent quand l'arbre contient plusieurs fois le même état. Ce même problème arrive dans le arbres de jeu car il y a souvent plusieurs suites de coups qui amènent à la même configuration. Comme il est inefficace de recalculer à plusieurs reprises la valeur d'une configuration, il est souvent intéressant de garder en mémoire les valeurs déjà calculé. La table contenant les configurations et leur valeurs est connu sous le nom d'une table de transpositions. Bien entendu, nous aurions pas suffisamment de mémoire de tout sauvegarder, donc nous aurions besoin de faire un choix parmi les configurations à stocker. La bonne utilisation d'une table de transpositions peut avoir des effets très importants sur l'efficacité de la recherche.

Par exemple, en échec, il st parfois possible d'examiner les nœuds d'une profondeur deux fois plus grandes grâce à cette technique. Les tables de transpositions sont exploités par Deep Blue et Chinook.

L'évaluation d'un noeud de l'arbre de jeu repose sur une fonction d'évaluation complexe et donc coûteuse en temps de calcul. Les tables de transposition conservent la valeur de chacune des positions rencontrées au cours de l'évaluation d'un noeud afin de pouvoir les réutiliser lors de chaque évaluation successive de l'arbre. Le temps de calcul est ainsi largement amélioré. Les informations stockées pour une position sont la position, la valeur calculée par la fonction d'évaluation, le meilleur coup à jouer pour cette position et la profondeur à laquelle le noeud a été évalué.

Exemple :

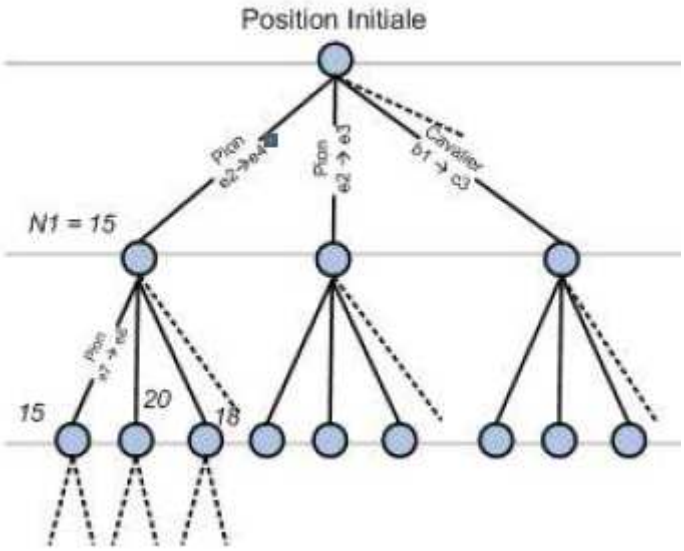


Figure 26: Exemple d'arbre pour évaluer une position

Soit le N1 dont les informations stockées par l'ordinateur sont :

Position	N1
Evaluation	15
Meilleur coup	Gauche
Profondeur	2

Tableau 1 : Informations stockées

Imaginons que lors d'une recherche future l'ordinateur rencontre de nouveau la position N1. Si la distance entre la racine et le noeud de l'arbre (profondeur) est plus grande que la profondeur stockée pour cette position dans la table de transposition, l'ordinateur n'évaluera pas ce noeud et utilisera directement la valeur calculée précédemment. Dans le cas contraire, l'ordinateur réévaluera la position. En effet la valeur de cette position peut être affinée par l'algorithme de recherche de type Min-Max vu que la distance entre le noeud et les feuilles de l'arbre est plus grande. L'ordinateur est donc capable de voir plus de coups en avance et de calculer un meilleur coup à jouer.

Pour stocker les tables de transposition, il est conseillé d'utiliser des tables de hachage permettant une recherche rapide. Les positions de l'échiquier sont ainsi indexées et accessibles rapidement en mémoire par l'ordinateur en évitant une recherche itérative longue et coûteuse en temps de calcul. On utilise donc un nombre (clef de hachage) caractérisant cette position.

Le problème des tables de hachage est qu'il impossible d'utiliser un index de trop grande taille. En effet une telle structure utiliserait beaucoup trop de mémoire dans le cas du jeu d'échecs. Il faut donc diminuer le nombre de positions stockées en mémoire. Généralement on utilise comme index de position la clef de hachage tronquée. C'est-à-dire que si on utilise des clefs de hachage sur 32 bits (un clef = une position unique), on utilise par exemple que les 16 derniers bits de ce nombres. En diminuant le nombre de position stockée, une case mémoire de la structure ne correspond plus uniquement à une position de l'échiquier. Il est possible de résoudre ce conflit en comparant la valeur de la position stockée avec la valeur de la position cherchée et en remplaçant la plus ancienne des deux valeurs par la plus récente pour la suite du programme.

Chapitre VIII : Etat de l'art

L'état de l'art est l'état des connaissances dans tout domaine donné (scientifique, technique, artistique, médical...) à un instant donné. C'est un des objectifs des grandes encyclopédies que de dresser un état de l'art sur les grands sujets culturels, scientifiques et techniques de leur époque, y compris par des illustrations .

- Jeu qui a suscité le plus d'attention et d'effort.
- Au début, les progrès ont été très lents.
- 1970 : Premier programme à gagner le championnat d'échecs informatiques. Il utilisait un élagage α-β, un répertoire d'ouvertures classiques et un répertoire de fins de partie.
- 1985 : Hitech (Berliner) se classe parmi les 800 meilleurs joueurs du monde. Il est capable d'analyser plus de 10 millions de positions par coup.
- Les dernières rencontres :
  - 1996 : Kasparov – Deep Blue : 4 – 2
  - 1997 : Kasparov – Deep Blue : 2.5 – 3.5
  - 2002 : Kramnik – Deep Fritz : 4 – 4
  - 2003 : Kaspaarov – Deep Junior : 3 – 3
  - 2003 : Kasparov – Fritz X3D : 2 – 2
- Deep Blue :
  - Analyse environ 126 millions de positions par secondes , avec des pointes à 330 millions de positions
  - Effectue une recherche sur 14 niveaux en moyenne
  - Peut étendre certaines recherches à plus de 40 niveaux…
  - Explore plus de 30 milliards de combinaisons par coup
  - Fonction d'évaluation comportant 8000 caractéristiques…
  - Répertoire d'ouvertures de 4000 positions

- Base de données de 700 000 parties de GMI
- Base de données de fins de parties comportant des positions résolues avec 5 pièces au plus et quelques unes avec 6 pièces.

- Deep Junior
  - Meilleure fonction d'évaluation par rapport à Deep Blue
  - 3 millions de coup par seconde mais aussi fort que Deep Blue
- Techniques utilisées pour les échecs :
  - Ouvertures :
    - ❖ Théorie des ouvertures très développée : l'encyclopédie des ouvertures dépasse les 2000 pages…
    - ❖ Indexation des positions favorables + coup(s) possible à jouer
  - Les finales
    - ❖ Problèmes
      - ➢ L'effet d'horizon de l'Alpha-Beta,
      - ➢ Incapacité d'établir un plan
    - ❖ Nécessité d'intégrer :
      - ➢ Règles particulières permettant de calculer instantanément la valeur de certaines pièces (pions passés…)
      - ➢ Savoir reconnaître des positions typiques
      - ➢ Savoir appliquer des plans précis adaptés à des cas précis
    - ❖ Solution : L'analyse rétrograde …
  - Le milieu de partie
    - ❖ Royaume de l'Alpha-Beta et de la fonction d'évaluation

- Fonction d'évaluation extrêmement complexe. Prend en compte :
  - Valeurs des pièces :
    - ✓ Dame : 9
    - ✓ Tour : 5
    - ✓ Cavalier, Fou : 3
    - ✓ Pions : 1
  - Sécurité du roi
  - Paire de fous
  - Domination du centre
  - Mobilité
  - Cavaliers centralisés
  - Tours placées sur des colonnes ouvertes
  - Fous placés sur des diagonales ouvertes
  - Qualité de la structure de pions
  - Pions passés soutenus…
- Alpha-Beta utilisé dans un programme d'échecs incorpore quelques améliorations
  - Elagage de futilité
  - Test prioritaire des coups meurtriers
  - Retour à l'équilibre
  - Heuristique du coup nul (Null Move Heuristic)

Partie 3 : La Réalisation du Jeu

Après toutes ses analyses théoriques, nous entrons enfin dans le vif du sujet, c'est-à-dire, la phase de conception de notre jeu d'échecs. Nous présenterons en premier lieu les outils et méthodes choisis pour ce projet, ensuite, nous observerons les étapes importantes du développement, comme les problèmes rencontrés et les solutions choisies, et enfin nous apprécierons ensemble la version finale du projet réalisé.

Chapitre IX : Outils de développement du jeu

1. Choix du développement:

1.1 Bibliothèque graphique :

Une fois le langage choisi, il convient de décider quelle bibliothèque graphique utiliser pour la réalisation de l'interface graphique de notre jeu. Trois possibilité s'offrent à nous : SDL, GTK et Qt. GTK et Qt sont les 2 bibliothèques graphiques les plus connus mais complexes. C'est ainsi que je choisi la SDL (Simple DirectMedia Layer), une bibliothèque de bonne qualité et puissante. La SDL est très utilisée dans le monde de la création d'applications multimédias en 2 dimensions comme les simulations ou les jeux vidéo.

La SDL est une bibliothèque tierce. Il faut savoir qu'il existe deux types de bibliothèques :

- La bibliothèque standard : C'est la bibliothèque de base qui fonctionne sur tous les OS et qui permet de faire des choses très basique comme des *printf*. Elle a été automatiquement installée lorsque vous avez téléchargé votre IDE et votre compilateur.
- La bibliothèque tierce : Ce sont des bibliothèques qui ne sont pas installées par défaut. If faut les télécharger et les installer sur son ordinateur pour en user. Il en existe des milliers écrites par d'autres programmeurs. Certaines sont bonnes, d'autres moins, certaines payantes, d'autres gratuites etc. L'idéal étant d'en trouver une de bonne qualité et gratuite à la fois.

1.2 Spécificité de la SDL :

C'est une bibliothèque écrite en C, elle peut donc être utilisée par des programmes en C et il est possible de l'utiliser en C++

C'est une bibliothèque libre et gratuite

On peut réaliser des programmes commerciaux et propriétaire avec.

C'est une bibliothèque multi plateforme : qui fonctionne sous Windows, Mac ou Linux mais elle peut aussi fonctionner sur Symbian, Dreamcast, etc.

Enfin c'est une bibliothèque qui permet de faire des choses amusantes.

Revenons à notre projet, en effet, la SDL n'est pas spécialement conçue pour créer des jeux vidéos, je l'admets, mais la plupart des programmes utilisant la SDL sont des jeux vidéos.

1.3 Langage de programmation :

Une multitude de langages de programmation permettent de concevoir des jeux vidéo, il n'a donc pas été facile d'en choisir un plus que les autres. Notre choix s'est porté sur un langage permettant la programmation orienté objet et qui est souvent plus simple lors de la réalisation de notre projet. Nous avons par la suite choisi d'utiliser le langage C++ car non seulement il est compatible avec la bibliothèque graphique que j'ai choisi mais en plus, il est facilite l'implémentation de l'intelligence artificielle dans notre programme. La conception d'un modèle orienté objet pour le développement d'un jeu est simple, elle permet de plus de faire évoluer simplement le jeu ou de réutiliser des classes dans un autre contexte.

1.4 L'environnement de développement (IDE = Integrated Development Environment) :

Le strict minimum pour un programmeur : un éditeur de texte pour écrire le code source, un compilateur pour transformer ou plus précisément « compiler » votre source en binaire, et un débogueur pour traquer les erreurs du programme.

Le programmeur peut utiliser les trois séparément, la méthode est plus compliquée, mais actuellement, il existe des programmes 3 en 1 appelés « IDE » ou « environnements de développement ». Il en existe plusieurs, mais ce qui est sûre, on peut réaliser n'importe quel type de programme, quel que soit l'IDE choisi.

Pour la réalisation de ce mémoire, j'ai opté pour un IDE parmi les plus connus, gratuit et qui fonctionne sur tous les systèmes d'exploitation (Windows, Mac, Linux). Ainsi j'ai donc choisi Code::Blocks avec compilateur MinGW.

2. Description de l'environnement:

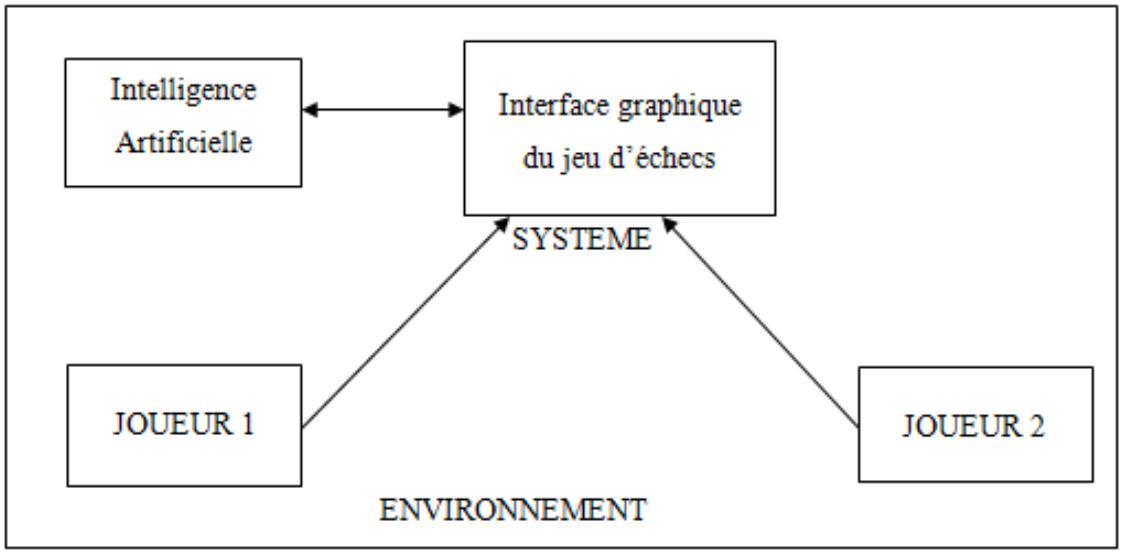


Figure 27 : Schéma de l'environnement à développer

On va concevoir un système qui gèrera l'interface graphique du jeu d'échecs avec la bibliothèque SDL 1.2. Il pourra interagir 1 ou 2 joueurs pour des parties sur un même ordinateur, ce qui constitue l'environnement du système. Le programme devra également interagir avec une intelligence artificielle qui est le moteur du jeu d'échecs dans le cas d'une partie contre l'ordinateur.

2.1 Description des fonctions à satisfaire:

Les fonctions principales que le logiciel devra satisfaire sont les suivantes :

- Permettre à deux joueurs de jouer aux échecs via une seule instance de logiciel sur la même machine : gestion du tour par tour
- Permettre à un joueur de jouer contre une intelligence artificielle : gestion de la communication entre l'interface de jeu et l'intelligence artificielle programmée.
- Gestion des règles du jeu d'échecs :
    - Contrôle de la validité des déplacements des pièces
    - Gestion des règles spéciales type « Roque »
    - Remise à l'état initial après un échec et mat

3. Schéma de l'interface:

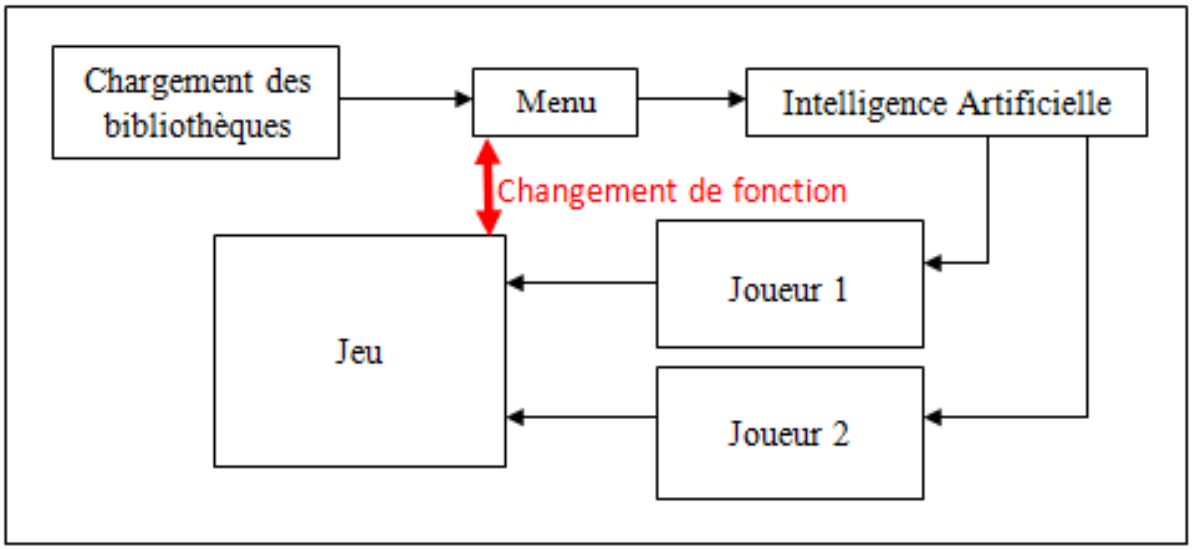


Figure 28 : Schématisation de l'interface

3.1 Explication du schéma :

1/ On lance l'application et les bibliothèques se chargent

2/ Une fois chargées, les animations d'ouverture sont lancées

3/ On arrive au Menu, on fait le choix du mode de jeu ou on quitte

Vous pouvez regarder les IA se mesurer entre eux

⇨ Joueur 1 IA (F/M/D) vs Joueur 2 IA (F/M/D)

Lorsqu'on joue seul, l'ordinateur active l'intelligence artificielle pour se mesurer contre vous

⇨ Joueur 1 IA (F/M/D) vs Joueur 2 humain

⇨ Joueur 1 humain vs Joueur 2 IA (F/M/D)

Où IA : Intelligence artificielle

F : Facile M : Moyen D : Difficile

Vous pouvez aussi choisir de jouer avec une autre personne

⇨ Joueur 1 humain vs Joueur 2 humain

4/Entre temps, il suffit de cliquer sur Echap pour revenir au Menu et la même chose pour revenir au Jeu

Chapitre X : Etape de programmation du jeu

1. Développement globale du jeu:

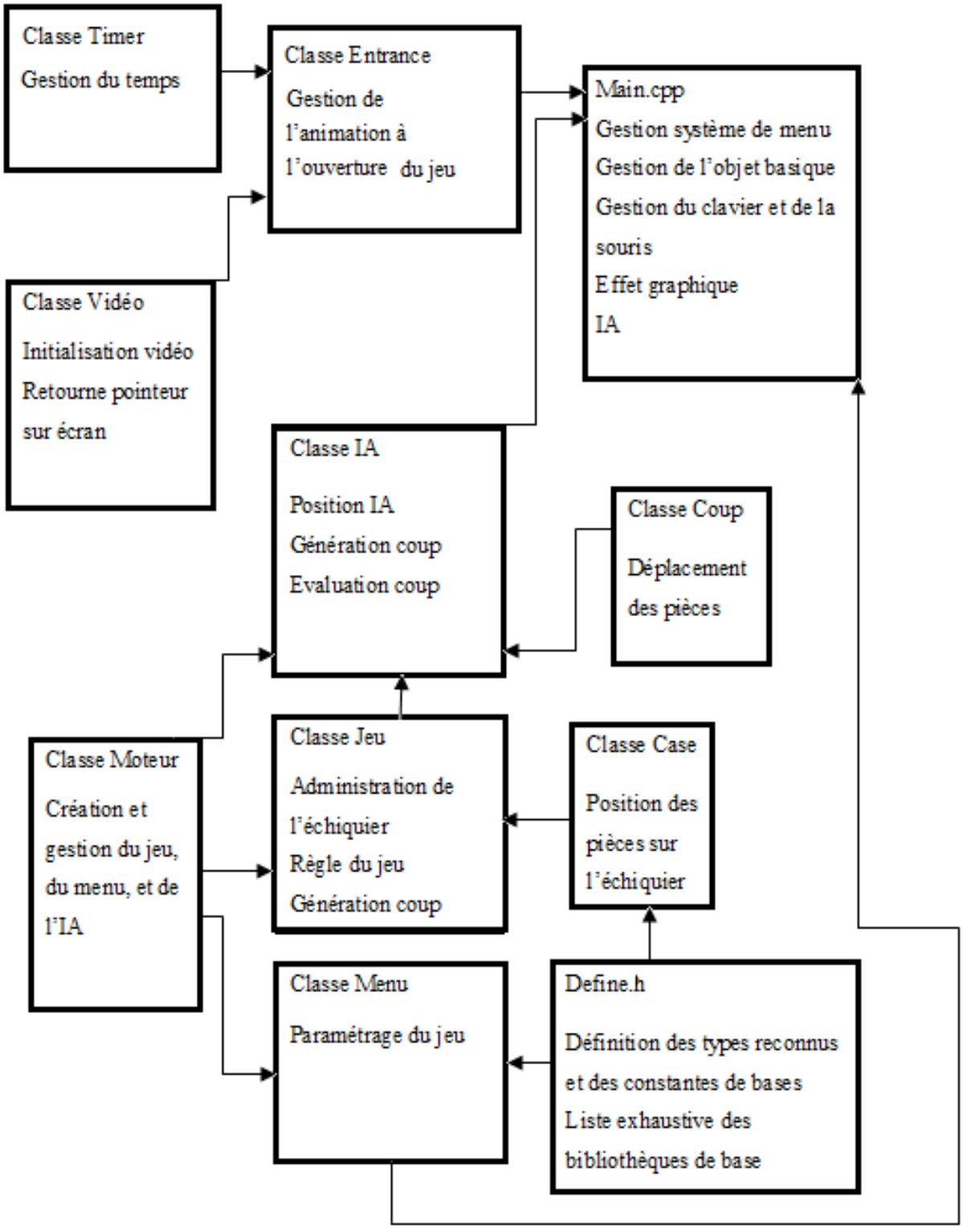


Figure 29 : Schématisation de la programmation

2. Architecture du programme :

L'algorithme choisi pour la réalisation de ce programme est l'algorithme min-max avec un élagage α-β dans un environnement crée par la bibliothèque SDL.

Peu importe la solution choisi, la création d'un système d'intelligence artificielle n'est pas un processus simple. Il a été choisi que dans ce travail, de considérer 2 modes de jeu ; le premier étant la programmation d'une partie entre 2 personnes réelles, tandis que le second est celui d'une personne contre l'ordinateur implémenté d'une intelligence artificielle que nous allons nous même programmé. Suite à cela, nous verrons si le système donne des résultats intéressants et corrigerons les erreurs qui entravent notre réalisation.

Le système est donc, d'un point de vue structure, est plutôt simple.

2.1 Système d'intelligence artificielle :

Le système étant créé sur le modèle d'algorithme min-max, voici les étapes de réalisation de cette étude :

- La représentation du plateau de jeu : Dans ce cas de système d'intelligence artificielle, il est question de représenter la position des pièces sur le plateau de jeu.
- La base de règles que le système exécute pour trouver les coups à jouer.

Les étapes dans le choix d'un coup dans les jeux a somme nulle (ici two-person zero-sum)

Soient 2 joueurs, MIN et MAX, jouant à tour de rôle.

MAX joue en premier :

- ❖ Construction de l'espace d'états (arbre de jeu), avec connaissance parfaite des états : état initial, état finaux, opérateur déterministe de génération des états successeurs
- ❖ Fonction éval, qui indique si un état terminal est gagnant, perdant ou de résultat nul pour le joueur MAX
- ❖ Choix du meilleur coup : algorithme MIN-MAX ou Alpha-Beta

2.2 Arbre de jeu :

Un arbre de jeu répertorie l'ensemble des coup réalisables au cours d'une partie, en partant du nœud racine, qui symbolise la configuration initiale, jusqu'aux nœuds terminaux : les positions finales.

Chaque nœud de l'arbre représente une position de jeu et chaque arc un coup possible d'un joueur permettant de passer d'une position à une autre.

A chaque nœud terminal est associé un score ternaire :

- ✓ +1 si MAX gagne
- ✓ -1 si MIN gagne
- ✓ 0 s'il y a nulle

} par exemple..

L'arbre de jeu est généralement gigantesque. Pour le jeu d'échecs :

- Nombre de choix par coup : ~35
- Nombre moyen de coups par jeu : 50
- Nombre total d'états : 35^{100}

Il est donc impossible d'explorer tout l'arbre de jeu pour trouver le meilleur coup à jouer.

2.3 Arbre de recherche :

C'est une partie de l'arbre de jeu. Il répertorie un sous-ensemble des coups réalisable au cours d'une partie, en partant d'un nœud atteint par un joueur jusqu'à des nœuds à un certain niveau de profondeur.

A chaque nœud du dernier niveau est associé un score dans l'intervalle [-1 ; 1] (en général) :

- ❖ +1 si MAX gagne
- ❖ >0 si la position est plus favorable pour MAX
- ❖ -1 si MIN gagne
- ❖ <0 si la position est plus favorable à MIN
- ❖ 0 s'il y a nulle

Fonction d'évaluation d'une position

Objectif

Associer à chaque position *p* une valeur estimant l'issue de la partie pour un des joueurs se trouvant en *p*.

Exemples

Aux dames : différences entre le nombre de pions des joueurs

Aux échecs : balance des pièces, roi en sécurité, structure de pions...

Propriétés d'une fonction élaborée

● Meilleur jeu

MAIS

● Plus coûteux en calcul,

● Limite ainsi l'exploration en profondeur

Pour être efficace, il est important d'avoir une fonction d'évaluation **precise**.

Généralement, **eval**(s) est une fonction linéaire :

$$eval(s) = w1\ f1(s) + w2\ f2(s) + ... + wn\ fn(s)$$

Prenant en compte un ensemble de paramètres, tels que : la balance des pièces, la sécurité du roi, la mobilité des pièces, la domination du centre, les possibilités d'attaque...

Elagage Alpha-Beta

Etendre l'arbre de jeu jusqu'à une profondeur h par une recherche en profondeur

● Ne plus générer les successeurs d'un noeud dès qu'il est évident que ce noeud ne sera pas choisi, compte tenu des noeuds déjà examinés

● Chaque **noeud MAX** conserve la trace d'une **α-valeur**, qui est la valeur de son **meilleur successeur** trouvé jusqu'à présent

● Chaque **noeud MIN** conserve la trace d'une **β-valeur**, qui est la valeur de son **pire successeur** trouvé jusqu'à présent

● Valeurs initiales : a = -α et b = +β

- Interrompre la recherche d'un noeud MAX si son **α-valeur** ≥ **β-valeur** de son nœud parent.

- Interrompre la recherche d'un noeud MIN si sa **β-valeur** ≤ **α-valeur** de son nœud parent.

Algorithme α-β (version minmax)

```
Fonction AlphaBeta( e ) Retourne une action
    v ← MaxValue( e, –∞, +∞ )
    Retourner action dont l'état successeur de e vaut v
FinFonction
Fonction MaxValue( e, α, β )
    Si e est terminal Alors Retourner eval( e )
    v ← –∞
    Pour chaque successeur s de e Faire
        v ← max( v, MinValue( s, α, β ) )
        Si v >= β Alors Retourner v
        α ← max( α, v )
    FinPour
    Retourner v
FinFonction

Fonction MinValue( e, α, β )
    Si e est terminal Alors Retourner eval( e )
    v ← +∞
    Pour chaque successeur s de e Faire
        v ← min( v, MaxValue( s, α, β ) )
        Si v <= α Alors Retourner v
        β ← min( β, v )
    FinPour
    Retourner v
FinFonction
```

L'élagage conserve la qualité du résultat. L'efficacité de l'élagage est fonction de l'ordre d'apparition des noeuds successeurs : à un niveau MAX, examiner d'abord les coups qui ont le plus de chance de générer des noeuds à forte valeur d'évaluation.

Dans le meilleur des cas, α–β évalue $2\sqrt{N}$ noeuds lorsque Min-Max en évalue N.

Complexité en temps

● meilleur des cas : **O(bm/2)**

→ permet de doubler la profondeur de recherche

● pire des cas : identique à *MinMax*

● La valeur α d'un **noeud MAX** est une limite inférieure sur la valeur remontée.

● La valeur β d'un **noeud MIN** est une limite supérieure sur la valeur remontée.

● Mettre à jour la valeur α ou β du parent d'un noeud N lorsque toute la recherche en dessous de N est terminée ou a été interrompue.

● Interrompre la recherche en dessous d'un noeud N de type MAX si sa valeur $\alpha > \beta$ d'un ancêtre de N de type MIN.

● Interrompre la recherche en dessous d'un noeud N de type MIN si sa valeur $\beta < \alpha$ d'un ancêtre de N de type MAX

Figure 30 : Schématisation de la programmation

Inconvenients de α–β

● Premier défaut : **manque total de stratégie**

- ● Programmes utilisant α–β ne sont pas guidés par un plan
- ● Jouent des positions totalement indépendantes les unes des autres
- ● Pas de vision à long terme de la partie

● Deuxième défaut : l'**effet d'horizon** (*Berliner*, 1974)

● Dû à la nécessité de fixer a priori une profondeur maximum

● Programme ne peut percevoir les effets d'un coup au delà de la profondeur choisie

● Tendance à repousser au delà de son horizon une position défavorable

2.4 Algorithme de recherche pour la résolution de problème :

Définition formelle d'un problème :

Le problème sera défini par les 5 éléments suivants : un état initial, un ensemble d'actions, une fonction successeur, qui définit l'état résultant de l'exécution d'une action dans un état, un ensemble d'état but, une fonction de cout associant à chaque action un nombre non-négative (le cout de l'action)

Nous pouvons voir un problème comme un graphe orienté où les nœuds sont des états accessibles depuis l'état initial et où les arcs sont des actions. Nous appellerons ce graphe l'espace des états. Une solution sera un chemin de l' état initial à un état but. On dit qu'une solution est optimal si la somme des couts des actions du chemin est minimal parmi toutes les solutions du problème.

Souvent les actions sont conditionnées par les actions des autres. C'est notamment le cas pour les jeux à plusieurs joueurs comme le cas du jeu d'échecs :

Les jeux à plusieurs joueurs sont plus compliqués que les problèmes que nous avons vu parce que l'agent ne peut pas choisir les actions des autres joueurs. Donc, au lieu de chercher un chemin qui relie l'état initial à un état but (qui contiendrait des actions des adversaires, dont nous n'avons pas le contrôle), nous allons chercher une stratégie, c'est-à-dire un choix d'action pour chaque état où pourrait se trouver l'agent. Voici une formalisation d'un jeu à plusieurs joueurs:

Etat : Des configurations du jeu plus le nom du joueur dont c'est le tour

Etat initial : La configuration initiale plus le nom du joueur qui commence

Actions : Les coups légaux pour le joueur dont c'est le tour

Fonction du successeur : La nouvelle configuration du jeu, et le nom de joueur dont c'est le tour

Test de but : Les configurations gagnantes pour le joueur

Cout des actions : Les configurations gagnantes pour le joueur

Pour le jeu d'échecs, nous avons la configuration suivante :

Etat : Des états sont composés d'une configuration de l'échiquier plus le nom de joueur dont c'est le tour

Etat initial : La configuration standard pour le début d'une partie.

Actions : Ce sont des coups qui peuvent être joués par le joueur dont c'est le tour dans la configuration actuelle du jeu (y compris l'action d'abondonner le jeu)

Fonction du successeur : La configuration du jeu évolue selon le coup choisi, et c'est maintenant à l'autre joueur de jouer

Test de but : Il y a beaucoup de façons de terminer une partie, la plus connue étant l'échec et mat

Cout des actions : Nous pouvons les mettre à 1 pour chercher les coups qui amènent le plus vite à une configuration gagnante

L'algorithme de recherche :

Avant de considérer les algorithmes spécifiques, voici le pseudo-code d'une algorithme de recherche générique :

```
Fonction RECHERCHE (état_initial, SUCCESSEURS, TEST_BUT, COUT)
    nœud_à_traiter ⟵ CREER_LISTE (CREER_NOEUD (état_initial, [], 0))
    boucle
      Si VIDE ? (nœud_à_traiter) alors renvoyer échec
          alors renvoyer CHEMIN (noeud), ETAT (noeud)
      pour tout (action, état) dans SUCCESSEURS (ETAT (noeud))
          chemin ⟵ [action, CHEMIN (noeud)]
          cout_du_chemin ⟵ COUT_DU_CHEMIN (noeud) + COUT (état)
          s ⟵ créer_noeud (état, chemin, cout_du_chemin)
          INSERER (s,nœud_à_traiter)
```

L'algorithme prend en entrée la description d'un problème : un état initial, une fonction de successeurs, un test de but, et une fonction de cout.

La première étape de l'algorithme est d'initialiser une liste de nœud à traiter, un nœud étant composé d'un état, d'un chemin et d'un cout du chemin. Nous commencerons avec un seul nœud correspondant à l'état initial.

Nous allons voir comment la recherche peut être utilisée dans les jeux. Nous commencerons par la description et une formalisation d'une certaine classe de jeux. Puis nous allons introduire des algorithmes qui permettent d'obtenir les stratégies optimales pour cette classe de jeux. Comme le temps de calcul nécessité par ces algorithmes les rend inutilisable en pratique, nous consacrerons la dernière partie aux modifications possibles afin de pouvoir prendre les décisions en temps réel.

Formalisation des jeux :

Il existe de nombreux type de jeux avec des propriétés très différentes. En intelligence artificielle, la plupart de la recherche s'est focalisé autour de la classe de jeux finis déterministes à deux joueurs, avec tours alternés, à information complète, et à somme nulle.

Fini : Un jeu est fini si le jeu termine toujours. Dans beaucoup de jeux, comme dans notre cas pour les échecs, les règles du jeu imposent des conditions pour rendre le jeu fini.

Déterministe : un jeu est déterministe si le déroulement du jeu est entièrement déterminé par le choix des joueurs. Des jeux utilisant les dés ou les cartes distribuées au hasard sont non déterministes.

Information complète : Dans les jeux à information complète, les joueurs connaissent parfaitement la configuration actuelle du jeu. Ce n'est pas le cas, par exemple, avec la plupart des jeux de cartes où les joueurs ne connaissent pas les cartes des autres. Ce sont des jeux à information partielle.

Somme nulle : Un jeu est dit à somme nulle si la valeur gagnée par l'un des joueur est la valeur perdue par l'autre. Le poker est un exemple de jeu à somme nulle, le montant qui et gagné par un joueur est égale à la somme des montants perdus par les autres joueurs. Les jeux où il y a toujours un gagnant et un perdant sont des jeux à somme nulle.

Le jeu d'échec est un bon exemple d'un jeu satisfaisant ces critères : il est fini parce que les règles imposent des conditions pour le rendre fini ; le hasard n'intervient pas, donc il est déterministe ; les deux joueurs connaissent l'état de l'échiquier donc il est à information complète, et enfin a somme nulle parce quand un joueur gagne, l'autre perd.

La formalisation de jeux de ce type :

Etat initial : C'est la configuration du jeu plus le nom du joueur

Fonction de successeur : La fonction définit quelles actions sont possibles pour un joueur dans une configuration donnée, ainsi que les configurations résultantes des différentes actions.

Test de terminalité : Ce test définit les configurations terminales du jeu

Fonction d'utilité : cette fonction associe une valeur à chaque configuration terminale. Souvent, les valeurs sont +1,-1 et

Qui correspond à un gain pour le premier joueur, une perte pour le premier joueur, et un match nul (parfois +inf et –inf sont utilisés au lieu de +11 et -1). Noter que comme nous ne considérons que les jeux à somme nulle, nous n'avons besoin que d'une seule valeur par configuration : la valeur attribuée au second joueur est exactement le négatif de la valeur reçue par le premier joueur.

La programmation du jeu d'échecs et tout à fait accessible à toutes personnes connaissant la programmation. Elle peut se découper en deux parties totalement distinctes, le moteur et l'interface graphique. La partie intéressante étant bien sur la réalisation du moteur. D'autant plus qu'il existe des interfaces graphiques prêtes à l'emploi

2.5 La programmation des interfaces :

C'est toute la partie graphique du logiciel : gestion des menus, déplacements des pièces etc. Vous pouvez bien sur écrire votre propre interface mais cela prend du temps alors qu'il en existe de nombreuses gratuites prêtes à l'emploi ! Il suffit pour cela que votre moteur implémente un protocole de communication.

Il existe aujourd'hui deux "protocoles" qui permettent de relier un moteur et une GUI.

- Winboard
- UCI : Universal Chess Interface

Pourtant, pour ce mémoire, nous n'en userons pas, nous programmerons tout de A à Z : c'est-à-dire n'utiliser que le langage de programmation et de la logique du jeu et sans aide d'un quelconque protocole.

La bibliothèque SDL étant facile d'usage : voici les sprites dont nous nous sommes munis pour créer l'interface du jeu.

Figure 31 : Les Sprites du jeu

Maintenant que nous avons le matériel nécessaire, il ne reste plus qu'à programmer !

Cependant, connaissant les règles du jeu, nous somme face à une contrainte, comment les reproduire dans le jeu.

La solution que j'ai utilisée est la création de plusieurs classes et de leurs fonctions propres. Ainsi, il sera plus facile d'implémenter les innombrables idées qui me passent par la tête.

La programmation est tout un art. Elle exige beaucoup de patience, mais surtout de l'attention. Il ne faut pas se lasser de corriger ses erreurs et de compiler.

Voici donc un extrait des fonctions de cet interface :

```
//Fonction d'initialisation (chargement des surfaces)
bool init(string file);

//Fonction qui vide l'échiquer
      void videJeu();

      //fonction qui initialise la place des pièces au début du jeu
void nouveauJeu();

   //Gestion du jeu lors d'un clic
   void clic(int x, int y);



//Fonction qui définit la situation d'échec du joueur courant
bool estEchec();

//Fonction qui définit la situation d'échec est Mat du joueur courant
bool echecEstMat();

   //Fonction d'affichage
   void aff(SDL_Surface *screen);

      //Fonction qui retourne la valeur de fini
      bool getFini();

      //fonction qui détermine si un roque est disponible
      bool getRoque(int i);

      //Fonction qui retourne la couleur du joueur courant
      Couleur getTour();

//Fonction qui retourne qui a gagné
Couleur getCouleurGagne();

//Fonction qui retourne la couleur de la pièce sur la case
Couleur getCouleur(int i, int j);

//Fonction qui retourne le type de la pièce sur la case
int getType(int i, int j);

//Fonction qui retourne si le joueur courant est un humain
bool estHumain();

//Fonction qui retourne le type du joueur courant
TypeJoueur getTypeJoueur();
```

```
//Fonction qui retourne l'état de la case
bool estVide(int i, int j);

//Fonction qui joue: déplace la pièce dep vers arr
//Après cette action tour change
void joue(Coup coup);

//Fonction qui silmule un déplacement
void simule(Coup coup);

//Fonction qui annule le dernier déplacement
void annule(Coup coup);

    //Fonction qui met en place le type des joueurs
    void setTypeJoueur(int i, TypeJoueur type);
```

```
529  //Fonction qui met en place le type des joueurs
530  void Jeu::setTypeJoueur(int i, TypeJoueur type)
531  {
532      if(i==1)
533      {
534          j1 = type;
535      }
536      else
537      {
538          j2 = type;
539      }
540  }
```

Figure 32 : Exemple du codage d'une fonction

2.6 La programmation du moteur (engine) :

C'est le cœur d'un programme d'échecs. Le principe est simple : on calcule des suites de coups possibles à partir de la position courante et en suivant des critères d'évaluation, le moteur choisit le meilleur coup.

1) La force brute : c'est-à-dire essayer de calculer un maximum de coups possibles.

2) Optimiser la fonction d'évaluer : le programme calcule moins de coups mais il évalue mieux la position

Quand l'interface est OK, on implémente l'IA

Voici un aperçu de l'IA :

```
#ifndef H_IA
#define H_IA
#include "Jeu.h"
//Ici j'utilise une file pour stoquer les coups trouvées
#define MINEVAL -100000
#define MAXEVAL 100000

//Classe qui gère l'intelligence artificielle
class IA
{
   private:
      //Compteur du nombre d'évaluations
      long nbEval;
       //Fonction qui compte le nombre de pieces
      int comptePieces(Jeu *jeu);
      //Fonctions pour le calcul
      int calcMin(Jeu *j, int prof, int alpha, int beta);
      int calcMax(Jeu *j, int prof, int alpha, int beta);
        //Fonction permettant de vérifier si les coordonnées (i, j) sont correctes
      bool estCorrect(int i, int j);
        //Génération des coups
      void genereCoupPion(Jeu *jeu, queue<Coup> *q, int i, int j);
      void genereCoupCavalier(Jeu *jeu, queue<Coup> *q, int i, int j);
      void genereCoupFou(Jeu *jeu, queue<Coup> *q, int i, int j);
      void genereCoupTour(Jeu *jeu, queue<Coup> *q, int i, int j);
      void genereCoupRoi(Jeu *jeu, queue<Coup> *q, int i, int j);
   public:
      IA();
      ~IA();
```

```
 //Fonction qui évalue le jeu
      int evalue(Jeu *j);
                  //Fonction qui calcule le prochain coup*/
                   void calcIA(Jeu *j, int prof);
                   //Fonction qui génère tous les coups possibles de la pièce se trouvant en (i, j)
      void genereCoup(Jeu *jeu, queue<Coup> *q, int i, int j);
  };
#endif
```

Et ainsi de suite pour chaque fonction de chaque classe.

CHAPITRE XI : La simulation

La simulation est la partie de ce projet qui nous intéresse le plus : non seulement il démarre le programme, mais c'est la preuve des lignes des programmes que j'ai saisies.

Pour lancer le programme, il suffit de double –cliquer dessus et apprécions ensemble le jeu

Vous verrez ceci , sur ceux, bon jeu !!!

Figure 33 : Le menu du jeu en question

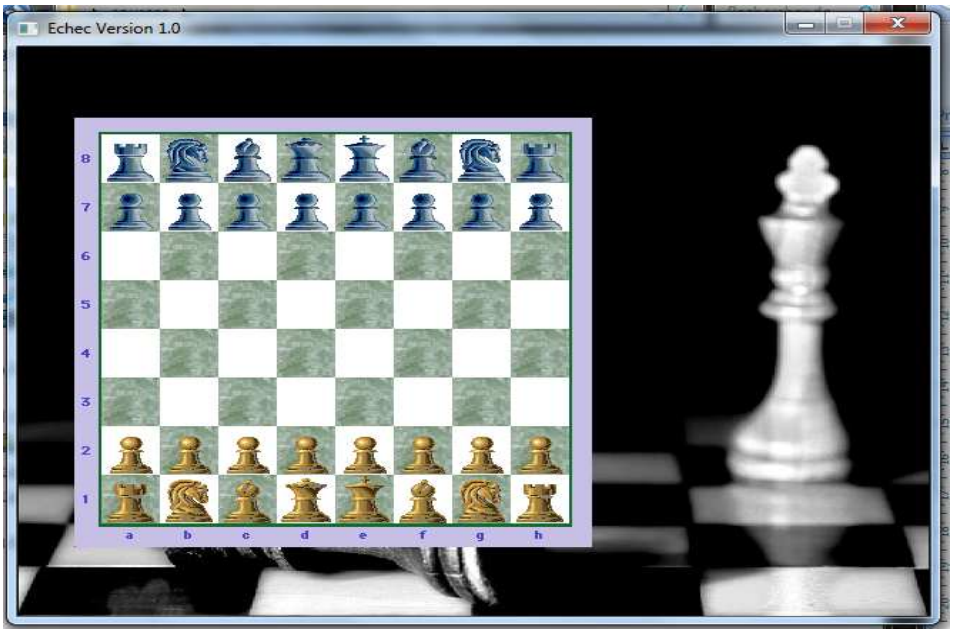


Figure 34 : L'interface de Jeu

CONCLUSION

Jouer aux jeux d'échecs est un vrai plaisir, mais la programmation du jeu est tout simplement attachante. Après chaque étape franchie avec succès, l'enthousiaste va en croissant. Quand son propre programme effectue pour la première fois des coups légaux et la première partie est surmontée sans crash, c'est une euphorie de succès à peine descriptible. Si de plus quelques coups sensés en sortent, il n'y a définitivement plus d'échappatoire. Il convient de souligner que le principe minimax est exactement similaire à la programmation dynamique utilisée abondamment dans l'industrie ; et que l'élagage alpha-beta est un exemple parmi d'autres d'élagages très utiles en parcours d'arbre et en programmation. Enfin, les méthodes présentées s'appliquent aussi pour la planification adversariale, lorsque l'on cherche à optimiser des choix futurs au pire cas sur diverses incertitudes.

Enfin, et surtout, les jeux fournissent un fascinant défi pour l'intelligence artificielle. Il est critique de comprendre ce que nous savons programmer, ou non, des activités humaines. Les jeux sont un outil parfait pour tester nos outils d'intelligence artificielle dans un cadre où la compétition avec nos collègues du monde entier est claire et nette. C'est pour ce rôle qu'à côté d'applications plus industrielles, nous nous intéressons aux jeux, notamment des jeux d'échecs.

Annexes

ANNEXES

A1 – LEXIQUES

Quelques mots élémentaires des échecs.

Nous allons commencer par expliquer des termes souvent utilisés avec leur signification exacte.

ABANDON : Une façon de terminer des patries amicales ou officielles est l'ABANDON. L'un des joueurs abandonne le combat. Parfois par manque de combativité mais le plus souvent car il sait, et son adversaire aussi, que le Mat est inéluctable, à court ou à moyen terme, et que son adversaire est d'une valeur telle qu'il ne le laissera pas échapper. Le fair-play et la correction de joueurs de bons niveaux exigent de ne pas continuer à jouer si l'on est clairement perdant et de préférer l'abandon à des coups sans signification.

ADOUBER : Le joueur ayant le trait peut rectifier la position d'une ou de plusieurs pièces sur leur case en disant « j'adoube » au préalable.

AILE-DAME : Portion de l'échiquier comprenant les cases des colonnes a, b, c et d. Elle est à la gauche du joueur qui a les Blancs et à la droite de celui qui a les Noirs.

AILE-ROI : Portion de l'échiquier comprenant les cases des colonnes e, f, g et h. Elle est à droite du joueur qui a les Blancs et à la gauche de celui qui a les Noirs.

ALGÉBRIQUE : Se dit de la notation échiquéenne communément adoptée où chaque case est représentée par la lettre de sa rangée suivie du chiffre de sa colonne.

ANALYSE : Étude détaillée d'une ou de plusieurs positions pour trouver le meilleur coup pour chaque camp. De même il est d'usage et utile, entre deux amateurs d'analyser la partie que l'on vient de jouer, où de faire analyser sa partie par un Maître ou enseignant d'échecs.

ANNULER : Ne gagner ni perdre, mais faire partie nulle avec le partage du point, soit par technique (Échec perpétuel, répétition de la même position trois fois, le Pat, etc.,.) soit par accord mutuel entre les deux joueurs.

ARRIÉRÉ : Un pion est dit " arriéré " quand les pions amis des colonnes voisines ont été poussés et ne peuvent plus le protéger. Ainsi fragilisé, un pion " arriéré " constitue généralement une faiblesse pour le camp qui le possède.

ATTAQUE : Ensemble de coups constitué de menaces, ou de préparatifs de menaces, sur une ou plusieurs cibles adverses.

ATTENTE (coup d') : Coup dont la seule intention est de passer le trait au camp adverse, estimant que celui-ci n'a pas de bons coups, ou alors attendant qu'il tombe dans un piège.

AVANTAGE : Supériorité de position, de développement de pièces. Il existe autant de sortes d'avantages que de critères de supériorité : du point de vue de l'espace occupé, du matériel, du contrôle de cases-clefs, etc.

AVEUGLE (jeu à l') : Forme de jeu où la vision physique d'un échiquier est absente. Le joueur

ayant les yeux bandés se fait une représentation mentale, par la mémoire, de l'échiquier ou des échiquiers suivant le nombre de ses opposants. Il annonce ses coups oralement au moyen de la notation algébrique (La dame va de e2 en e7).

BLITZ : De l'Allemand « blitz » qui signifie « éclair ». Forme de jeu où chaque joueur n'a que très peu de temps (en général 5mn) de réflexion pour toute la partie. Rapidité de décision et bon coup d'œil sont requis, plutôt qu'une stratégie bien échafaudée.

BLOQUEUR: Nom donné à une pièce, qui placée devant un pion, l'empêche d'avancer. On parle ainsi parfois de Cavalier bloqueur.

CADENCE : Désigne le temps et éventuellement le nombre de coups à jouer pour une partie. ex. 5min (cadence blitz), 25min (cadence semi-rapide), 2heures/40coups (partie longue, il faudra jouer 40 coups en deux heures)

CENTRAL (pion): Se dit des pions qui se trouvent sur les colonnes « d » ou « e ».

CENTRE: le centre de l'échiquier constitué des quatre cases, d4, d5, e4, e5.

CHAINE DE PIONS: Groupe de pions d'un même camp liés les uns aux autres et formant une chaîne.

CLOUAGE: Action effectuée par une pièce à longue portée (Dame, Tour ou Fou) qui vise une pièce ennemie derrière laquelle se trouve une autre, de valeur plus importante. La première pièce est donc immobilisée sous peine de perte matérielle plus importante que ce qu'elle vaut. Le clouage est dit « absolu » quand la deuxième pièce est le Roi car la pièce clouée n'a pas le droit de bouger.

CLOUÉE (pièce): Pièce immobilisée totalement ou en partie à cause d'un clouage (Cf.: CLOUAGE).

COIN: Chacune des cases situées dans l'angle de l'échiquier, a1, a8, h1, h8.

COLONNE: Ligne verticale sur l'échiquier (exemple : la colonne a comprend toutes les cases de a1 à a8).

COMBINAISON: Suite de coups d'un camp provoquant des réponses forcées ou quasi forcées, en vue de la réalisation d'un plan concret. Une combinaison bien pensée et réalisée peut ainsi amener un petit avantage voire l'échec et mat.

CONTRE-GAMBIT: Contre sacrifice (Voir GAMBIT).

CONTREJEU: Défense pouvant déboucher sur une contre-attaque. Trop de contre-jeu peut dissuader l'adversaire d'attaquer voire de prendre une pièce.

CONTRÔLE : On dit qu'une pièce contrôle une case si elle peut s'y rendre en un coup. Le contrôle est appelé « protection » si la case est occupée par une pièce amie.

COULOIR (Mat du): Cas d'échec et mat donné par la Dame ou la Tour, le long d'une rangée ou d'une colonne, et où le roi est bloquée par ses propres pièces. Il ne peut sortir de son couloir...

DÉCOUVERTE (échec à la) : Échec au Roi effectué par une pièce que l'on démasque.

DÉNUDE (Roi) : Un Roi est ainsi lorsque plus aucun pion de son propre camp ne le protège.

DÉROQUER : Enlever à l'adversaire la possibilité d'effectuer le roque en forçant à bouger son Roi ou une Tour

DÉVIATION : Action consistant à forcer une pièce adverse à se déplacer, et créer ainsi un dommage pour le camp adverse.

DOUBLE (attaque) : Action de menacer deux objectifs différents avec un seul coup. Genre d'attaque très difficile à parer, qui est donc plus efficace qu'une menace simple.

DOUBLE (ECHECS) : Attaque avec deux pièces adverses différentes et en même temps du roi opposé. Seul le déplacement du roi constitue une parade à un échec double, arme très efficace.

ELO : Système de classement des joueurs d'échecs mis au point par le physicien américain d'origine hongroise Arpad ELO

ESPACE : L'avantage d'espace pour un camp signifie le contrôle d'une plus grande portion de l'échiquier par ses unités que l'adversaire n'en contrôle lui-même avec son matériel. L'avantage d'espace peut-être déterminant si aucune compensation (matériel, tempo, développement) n'est acquise par l'adversaire.

FIANCHETTO : Développement du fou d'une case devant le Cavalier sur une des grandes diagonales a1-h8, h1-a8. Les fianchettos blancs seront en b2 et g2, les fianchettos noirs en b7 et g7.

FIDE : Fédération Internationale Des Echecs

FOURCHETTE : Attaque simultanée de deux pièces adverses (attaque double).

GAIN : Terme souvent employé à la place du mot victoire. Ce mot revient dans de nombreuses phrases : « Les blancs jouent pour le gain », « forcer le gain »...

GAMBIT : Sacrifice de pion ou de pièces en vue d'une attaque. La compensation obtenue n'est donc pas concrète et la foi dans un gambit reflète généralement le style offensif d'un joueur. Le contre-gambit est un gambit en réplique à un autre gambit.

G.M.I. (Grand Maître International) : Titre décerné par la FIDE.

ISOLÉ (Pion) : Pion dépourvu de pions de son camp sur ses deux colonnes adjacentes. Un pion isolé ne peut donc plus être protégé par un autre pion et se révèle encore général plus faible que la moyenne.

LIBÉRATION (coup de) : Se dit généralement d'un coup de pion qui permet de gagner de l'espace et de sortir ainsi d'une position resserrée.

LIQUIDATION : Échange de plusieurs pièces en milieu de partie. Une liquidation permet souvent de réduire les tensions et d'aplanir une position complexe. Le camp en difficulté à tout intérêt à provoquer une liquidation.

MAJEURES (pièces) : La Dame et les Tours sont des pièces majeures. Quand on perd une tour contre un Fou ou un Cavalier, on perd la qualité (cf. : qualité).

MAJORITÉ (de pions) : Nombre de pions supérieurs d'un camp sur une aile ou au centre. Une majorité sera susceptible de créer un pion passé.

M.I. (Maître International) : Titre décerné par la FIDE.

MINEURES (pièces) : Les Fous et les Cavaliers sont des pièces mineures au contraire des Tours et de la Dame qui sont des pièces majeures ou pièces lourdes.

NULLE (partie) : Partie sans vainqueur. Une partie nulle peut être le résultat d'un commun accord entre les joueurs ou la conséquence des cas de parties nulles (voir pat, échec perpétuel, répétition de position, règles des 50 coups)

OPPOSITION : Quand deux Rois se font face à une case de distance, on dit qu'ils sont en opposition. S'ils sont sur la même diagonale, on dit « opposition diagonale ».

OUVERTURE : Désigne les premiers coups d'une partie.

PAT : Quand un joueur ne peut pas jouer il est pat et la partie est nulle.

PRISE EN PASSANT (P.E.P.) : Un Pion, attaquant la case traversée par un Pion adverse qui a été avancé de deux cases à partir de sa position initiale, peut prendre ce pion comme s'il n'avait été avancé que d'une case. Cette prise ne peut avoir lieu qu'en réponse immédiate à l'avance de deux cases du pion adverse. Elle s'appelle la prise en passant.

QUALITÉ : Différence de force entre la Tour d'une part, et le Fou ou le Cavalier d'autre part. On estime à deux pions l'équivalent matériel de la qualité en faveur de la tour.

RÉFUTATION : Suite de coups forcés montrant que l'adversaire a commis une erreur. (Ex : réfuter un sacrifice)

TEMPO : Unité temporelle représentant un demi-coup, soit un coup d'un camp. Par exemple, pour qu'un pion arrive à dame (promotion du pion) il faut, à ce moment-là, cinq tempos.

SACRIFICE : Perte volontaire de matériel (pion, pièce) en vue d'obtenir un avantage décisif.

TOMBER : Perdre une partie au temps. Ce terme vient des pendules à aiguilles sur lesquels un petit drapeau se levait et tombait quand il ne restait plus de temps. L'adversaire annonce alors « Tombé ».

TRAIT : Quand c'est aux Blancs de jouer, on dit que les Blancs ont le trait. Idem quand c'est aux noirs de jouer.

VARIANTE (Line) : Différentes lignes de jeu possibles à partir d'une ligne principale.

ZEITNOT Quand il reste peu de temps à un joueur on dit qu'il est en zeitnot. Le zeitnot peut conduire à commettre des gaffes.

ZUGZWANG : Situation curieuse mais assez fréquente où le meilleur coup pour un camp serait de ne pas jouer, tous les coups possibles entraînant un dommage, un danger ou une perte de matériel. Cela étant interdit, il faut donc jouer et causer sa propre perte

A2 – CHAMPIONS

Les échecs comptent de nombreux grands champions, personnalités diverses aux caractères fascinants.

LES CHAMPIONS DU MONDE OFFICIELS

1		Wilhelm STEINITZ	1886-1894	Américain d'origine autrichienne
2		Emmanuel LASKER	1894-1921	Allemand
3		José-Raoul CAPABLANCA	1921-1927	Cubain
4		Alexandre ALEKHINE	1927-1935	Français d'origine russe
5		Max EUWE	1935-1937	Néerlandais
		Alexandre ALEKHINE	1937-1946	Français d'origine russe
6		Mikhaïl BOTVINNIK	1948-1957	Russe
7		Vassili SMYSLOV	1957-1958	Russe
		Mikhaïl BOTVINNIK	1958-1960	Russe
8		Mikhaïl TAL	1960-1961	Russe (Letton)
		Mikhaïl BOTVINNIK	1961-1963	Russe
9		Tigran PETROSSIAN	1963-1969	Russe
10		Boris SPASSKY	1969-1972	Russe
11		Robert James FISCHER (dit Bobby)	1972-1975	Américain
12		Anatoli KARPOV	1975-1985	Russe
13		Garry KASPAROV	1985-1993	Russe
		Anatoli KARPOV	1993-1999	Russe
14		Alexander KHALIFMAN	1999-2000	Russe
15		Wiswanathan ANAND	2000-2002	Indien
16		Ruslan PONOMARIOV	2002-2004	Ukrainien
17		Rustam KASIMDZHANOV	2004-2005	Ouzbek
18		Veselin TOPALOV	2005-2006	Bulgare
19		Vladimir KRAMNIK	2006-2007	Russe
		Wiswanathan ANAND	2007-2013	Indien
20		Magnus CARLSEN	2013	Norvégien

A3 – Classement des moteurs de jeu

#	Name	Elo	1 Komod	2 Stock	3 Houdi	4 Gull	5 Equin	6 Critt	7 Fire	8 Rybka	9 Black	10 Strel	11 Naum	12 Chiro
1	Komodo 8 64-bit 4CPU	3303		52.5 – 41.5 +19–8=67 **+11**	53.5 – 40.5 +30–17=47 **+12**	35 – 15 +21–1=28 **+9**	33 – 17 +17–1=32 **–30**	35.5 – 14.5 +22–1=27 **–7**	33 – 17 +19–3=28 **–47**	35 – 15 +20–0=30 **–28**	40.5 – 11.5 +30–1=21 **+12**	38.5 – 10.5 +31–3=15 **+13**	42 – 8 +35–1=14 **+48**	45 – 7 +38–0=14 **+68**
2	Stockfish 5 64-bit 4CPU	3283	41.5 – 52.5 +8–19=67 **–11**		42 – 35 +20–13=44 **+17**	21 – 11 +11–1=20 **0**	31.5 – 18.5 +14–1=35 **–28**	54.5 – 22.5 +35–3=39 **+17**	57 – 20 +39–2=36 **+30**	19.5 – 10.5 +12–3=15 **–31**	21 – 9 +12–0=18 **–36**	18 – 2 +16–0=4 **+160**	36.5 – 13.5 +26–3=21 **–31**	40 – 10 +31–1=18 **+17**
3	Houdini 4 64-bit 4CPU	3275	40.5 – 53.5 +17–30=47 **–12**	35 – 42 +13–20=44 **–17**		20 – 14 +13–7=14 **–19**	21 – 11 +11–1=20 **–2**	37 – 14 +25–2=24 **+38**	24 – 10 +15–1=18 **+12**	23 – 11 +16–4=14 **0**	34 – 13 +23–2=22 **–2**		44.5 – 13.5 +32–1=25 **–4**	39 – 12 +30–3=18 **+2**
4	Gull 2.8b 64-bit 4CPU	3199	15 – 35 +1–21=28 **–9**	11 – 21 +1–11=20 **0**	14 – 20 +7–13=14 **+19**			18.5 – 15.5 +9–6=19 **+1**	1 – 1 +0–0=2 **–35**		18.5 – 11.5 +11–4=15 **+6**			21.5 – 11.5 +16–6=11 **0**
5	Equinox 3.20 64-bit 4CPU	3186	17 – 33 +1–17=32 **+30**	18.5 – 31.5 +1–14=35 **+28**	11 – 21 +1–11=20 **+2**			15 – 17 +3–5=24 **–24**	18 – 14 +6–2=24 **+8**		20.5 – 11.5 +10–1=21 **+26**		21.5 – 10.5 +11–0=21 **+8**	17.5 – 14.5 +11–8=13 **–59**
6	Critter 1.6a 64-bit 4CPU	3175	14.5 – 35.5 +1–22=27 **+7**	22.5 – 54.5 +3–35=39 **–17**	14 – 37 +2–25=24 **–38**	15.5 – 18.5 +6–9=19 **–1**	17 – 15 +5–3=24 **+24**		17 – 17 +4–4=26 **–9**	24.5 – 25.5 +9–10=31 **–18**	20 – 12 +9–1=22 **+27**	32.5 – 22.5 +13–3=39 **–13**	19.5 – 12.5 +9–2=21 **–13**	30 – 24 +32–6=36 **+31**
7	Fire 3.0 64-bit 4CPU	3164	17 – 33 +3–19=28 **+47**	20 – 37 +2–39=36 **–30**	10 – 24 +1–15=18 **–12**	1 – 1 +0–0=2 **+35**	14 – 18 +2–6=24 **–8**	17 – 17 +4–4=26 **+9**			18.5 – 13.5 +6–1=25 **+9**		18 – 14 +9–5=18 **–29**	19 – 13 +10–4=18 **–12**
8	Rybka 4 64-bit 4CPU	3162	15 – 35 +0–20=30 **+28**	10.5 – 19.5 +3–12=15 **+31**	11 – 23 +4–16=14 **0**			25.5 – 24.5 +10–9=31 **+18**			16.5 – 11.5 +10–5=13 **+27**			
9	BlackMamba 2.0 64-bit 4CPU	3136	11.5 – 40.5 +1–30=21 **–12**	9 – 21 +0–12=18 **+36**	13 – 34 +2–23=22 **+2**	11.5 – 18.5 +4–11=15 **–6**	11.5 – 20.5 +1–10=21 **–26**	12 – 20 +1–9=22 **–27**	13.5 – 18.5 +1–6=25 **–9**	11.5 – 16.5 +5–10=13 **–27**			20.5 – 11.5 +11–2=19 **+46**	18 – 14 +8–4=20 **–2**
10	Strelka 5.5 64-bit	3114	10.5 – 38.5 +3–31=15 **–13**	2 – 18 +0–16=4 **–160**				22.5 – 32.5 +3–13=39 **+13**						
11	Naum 4.6 64-bit 4CPU	3103	8 – 42 +1–35=14 **–48**	13.5 – 36.5 +3–26=21 **+31**	13.5 – 44.5 +1–32=25 **+4**		10.5 – 21.5 +0–11=21 **–8**	12.5 – 19.5 +2–9=21 **+13**	14 – 18 +5–9=18 **+29**		11.5 – 20.5 +2–11=19 **–46**			19 – 13 +11–5=16 **+52**
12	Chiron 2 64-bit 4CPU	3098	7 – 45 +0–38=14 **–68**	10 – 40 +1–31=18 **–17**	12 – 39 +3–30=18 **–2**	11.5 – 21.5 +6–16=11 **0**	14.5 – 17.5 +8–11=13 **+59**	24 – 50 +6–32=36 **–31**	13 – 19 +4–10=18 **+12**		14 – 18 +4–8=20 **+2**		13 – 19 +5–11=16 **–52**	

Performance Legend Color : (Only pairs with at least 30 games)

BIBLIOGRAPHIE

[1].Bjarne STROUPSTRUP :

« The C++ Programming Language »

4th edition , 2013

[2]. Jean-Marc Alliot & Thomas Schiex

«Intelligence Artificielle et Informatique Théorique»

2ème édition Broché – Mars 2002

[3].Mathieu NEBRA

«Apprener à programmer en C»

OpenClassrooms, 23013

[4]. Stuart RUSSELL & Peter NORVIG :

« ARTIFICIAL INTELLIGENCE A MODERN APPROACH »

–3rd Edition, 2010

[5].Vincent T'kindt

«Programmation en C++ et Génie Logiciel »

Dunod, 2006

WEBOGRAPHIE

[6]. http://chess.verhelst.org/search.html

[7]. http://www.chessopolis.com/cchess.htm#info

[8]. http://www.ffothello.org/info/algos.htm

[9]. http://www.gamedev.net/reference/list.asp?categoryid=18

[10]. http://wannabe.guru.org/scott/hobbies/chess

[11]. http://history.chess.free.fr

[12]. http://www.ifrance.com/jeudechecs/pageechecs.htm

[13].http://www.meruvia.fr/index.php?option=com_content&view=category&id=10:big-tuto&Itemid=5&layout=default

[14]. http://www.paginus.com/lesechecs/a.html

[15]. http://www.wired.com/wired/archive/12.01/stuntbots.html?pg=1&topic=&topic_set

Auteur : **Mademoiselle RAHOLIDERANTSOA Fanja**
Adresse : **II V 88 TER AMPANDRANA BESARETY**
Téléphone : **0327889910**
E-mail : **hifanja@yahoo.fr**

Titre du mémoire :

« CONCEPTION ET REALISATION D'UN LOGICIEL DE JEU D'ECHECS »

Encadreur : **Monsieur ANDRIAMANOHISOA Hery Zo**

Nombre de pages : **68**
Nombre de figures : **34**
Nombre de tableau :1

Résumé.

Cet ouvrage est consacré au développement d'un jeu d'échecs. Le but est de pouvoir gérer non seulement les paramètres graphiques mais aussi s'assurer du fonctionnement de l'intelligence artificielle dans le jeu dans une partie contre l'ordinateur.

En se basant sur les algorithmes de recherches les plus utilisés dans l'industrie, soient le principe min_max et l'élagage α – β, nous avons réussi à créer un moteur de jeu capable de jouer contre quelqu'un.

L'amélioration continuelle de la programmation dans l'écriture du code a permis d'atteindre l'objectif de ce mémoire.

Abstract.

This book is devoted to the development of chess. The goal is to manage not only the graphics settings but also to ensure the functioning of artificial intelligence in the game in a game against the computer.

Based on the algorithms of the most widely used research in the industry, are the principle min_max and pruning α - β, we have managed to create a game engine capable of playing against someone.

The continuous improvement of programming in writing code has achieved the objective of this thesis

Mots clés : jeu, intelligence artificielle, moteur, interface

Promotion 2013