Tristan Cazenave

Jacques Pitrat, l'Intelligence Artificielle et les Jeux

Volume 3, nº 1-2 (2022), p. 113-126.

<http://roia.centre-mersenne.org/item?id=ROIA_2022__3_1-2_113_0>

Revue Ouverte d'Intelligence Artificielle
Volume 3, nº 1-2, 2022, 113-126

Jacques Pitrat, l'Intelligence Artificielle et les Jeux

Tristan Cazenave[a]

[a] LAMSADE Université Paris-Dauphine, PSL, CNRS Paris, France
E-mail : Tristan.Cazenave@dauphine.psl.eu.

Résumé. — Cet article décrit les apports de Jacques Pitrat à la programmation d'intelligences artificielles pour les jeux ainsi que les travaux qui ont puisé leur inspiration dans ses recherches. L'article commence par évoquer le General Game Playing, puis viennent ensuite l'apprentissage par généralisation, l'amorçage, la dualité entre politique et évaluation et une évocation des thèses sur les jeux dirigées par Jacques Pitrat.

Mots-clés. — Intelligence Artificielle, Jeux, *General Game Playing*, Apprentissage Automatique, Amorçage.

1. Introduction

Jacques Pitrat fut le premier à explorer différentes facettes de l'intelligence artificielle pour les jeux. Il a proposé des approches innovantes qui ont été reprises par la suite par de nombreux chercheurs.

Nous détaillerons ses contributions liées aux jeux ainsi que leurs prolongements dans les recherches actuelles. Nous commencerons par le General Game Playing, puis nous poursuivrons avec l'apprentissage par généralisation, l'amorçage, la dualité politique versus évaluation et nous finirons par évoquer quelques thèses qu'il a encadrées sur les jeux.

2. Le General Game Playing

Jacques Pitrat publie en 1968 le premier papier sur un système général de jeux [39, 40] et fonde le General Game Playing (GGP) en écrivant le premier programme général de jeux. Ses travaux seront ensuite repris dans les années 1990 par Barney Pell [38], puis en 2005 Michael Genesereth à Stanford lancera les premières compétitions de GGP qui connaîtront un grand succès. Les premières compétitions sont gagnées par des programmes qui créent des heuristiques ensuite utilisées comme fonctions d'évaluation par un Alpha-Béta [17]. Puis à partir de 2007 et jusqu'à aujourd'hui, toutes les compétitions sont gagnées par des variantes de recherche Monte-Carlo [24, 29], comme le programme ARY [30, 31, 32, 55] que nous avions conçu avec Jean Méhat pour la compétition de 2007 et qui a gagné les compétitions à IJCAI 2009 et à AAAI

2010. Le langage de description des règles des jeux joués en GGP est le Game Description Language (GDL) [26]. Il utilise une variante de la logique des prédicats qui s'interprète facilement à l'aide d'un interpréteur Prolog.

L'article initiateur du General Game Playing est reproduit en annexe. On peut voir à la fin du papier page 1574 les questions qui ont été posées à Jacques Pitrat lors de la présentation de l'article ainsi que les réponses qu'il a données. Il indique, dans la réponse à la question de David Levy sur le comportement dans les positions non tactiques, l'heuristique utilisée par le programme général qui consiste à maximiser le nombre de coups possibles pour le programme et à minimiser le nombre de coups possibles pour l'adversaire.

Jacques Pitrat avait toujours en vue la généralité des algorithmes d'intelligence artificielle sur lesquels il travaillait. Cela se retrouvait aussi dans ses enseignements. Ainsi je me rappelle qu'il donnait comme projet à ses étudiants dont je faisais partie, un analyseur syntaxique général. Le générateur pouvait s'appliquer dans trois cas différents et j'avais fait un copier-coller de mon code pour chacun de ces trois cas. Jacques Pitrat m'avait fait remarquer lors de ma soutenance de projet qu'il aurait été plus judicieux de ne faire qu'un seul algorithme pour les trois cas plutôt que des copier-coller. J'ai mis en pratique par la suite ses conseils (notamment en GGP) mais il m'arrive encore d'avoir des bugs dans les codes que j'écris et cela est souvent dû à des copier-coller mal modifiés !

Une anecdote amusante qu'aimait raconter Jacques Pitrat, à propos des bugs dans nos programmes, était sa rencontre avec les méta bugs, les bugs dans les méta programmes qui généraient des programmes ainsi buggués avec d'innombrables bugs.

En 1998, quelque temps après la victoire de Deep Blue sur Kasparov, il a publié dans le journal de l'International Computer Games Association, dont il était membre, un article incitant à écrire des programmes généraux de jeux plutôt que des programmes ultra-spécialisés comme Deep Blue [48].

Si on considère la recherche en intelligence artificielle et jeux aujourd'hui, les idées de Jacques Pitrat sont en pleine actualité et même extrêmement populaires. Ainsi AlphaGo en 2016 [56] a donné l'année suivante AlphaGo Zero [58] puis Alpha Zero en 2018 [57] qui apprend seul à jouer au Go, aux Échecs et au Shogi avec le même algorithme et qui dépasse ce faisant les joueurs humains aussi bien que les algorithmes spécialisés. Plus récemment, en 2020, Polygames [12], un programme conçu principalement par Facebook FAIR associé à de nombreux autres chercheurs, apprend tout seul à partir de zéro à jouer à de très nombreux jeux. Il arrive de cette façon à des niveaux surhumains avec un algorithme général.

Un autre exemple actuel de système général de jeux est le système Ludii [3] issu d'un projet de recherche européen de Cameron Browne pour explorer les ludèmes, les composants réutilisables de différents jeux. Plus d'un millier de jeux sont disponibles de manière unifiée en Ludii. Il comporte des algorithmes généraux de jeux notamment UCT [28] et le plus récent GRAVE [9] qui est l'état de l'art en GGP [59] et qui permet de jouer à tous ces jeux. Ludii a été utilisé pour permettre aux programmes de jouer

entre eux lors des vingt-troisième et vingt-quatrième *Computer Olympiad* de 2020 et 2021. À propos de ces *Computer Olympiads* et des algorithmes généraux de jeux, il est remarquable qu'un programme général de jeux, de plus fondé sur des idées originales, ait gagné cinq médailles d'or en 2020 et onze médailles d'or en 2021, le plus grand nombre de médailles d'or gagnées par un même programme dans une même olympiade jusqu'ici. Ce programme nommé Athénan a été écrit par Quentin Cohen-Solal [18]. Il est original puisqu'il fait appel au Minimax et non à la recherche Monte-Carlo et qu'il n'apprend pas de politique mais seulement une fonction d'évaluation.

3. L'Apprentissage par Généralisation

Jacques Pitrat a aussi abordé dans les années 1970 la programmation des Échecs avec des programmes de planification qui trouvaient des combinaisons [43, 45, 44] et des programmes qui apprenaient à jouer en généralisant les explications trouvées par le système à partir des règles du jeu [42, 41]. Son approche de l'apprentissage reposait sur l'utilisation des règles du jeu pour raisonner sur les combinaisons nouvellement découvertes afin de les généraliser de façon sûre et de les réutiliser par la suite sous forme de règles du premier ordre. Il était partisan des programmes qui utilisaient des heuristiques et des connaissances pour réduire l'exploration de l'espace d'états plutôt que de la force brute et de l'exploration exhaustive [1]. Ses recherches ont été reprises par la suite notamment par Steven Minton [33] et ont donné naissance à l'*Explanation Based Learning* [19] et à l'*Explanation Based Generalization* [34]. Ce sont des méthodes d'apprentissage automatique très parcimonieuses en termes d'exemples utilisés pour apprendre et qui généralisent de façon sûre. La parcimonie et la généralisation sûre sont des propriétés recherchées par les systèmes actuels de Deep Learning qui reposent sur des millions, voire des centaines de millions d'exemples pour apprendre des représentations sous forme de réseaux de neurones.

Ma thèse avec Jacques Pitrat, soutenue en 1996, portait sur ces méthodes d'apprentissage à partir d'explications, appliquées au jeu de Go [4]. J'appréciais énormément nos rencontres mensuelles qu'il organisait dans son bureau avec Bruno Bouzy et Patrick Ricaud qui effectuaient aussi leurs thèses sur le jeu de Go [2, 52].

Jacques Pitrat ayant lui-même fait sa thèse sur la démonstration de théorèmes il n'est pas étonnant que la méthode d'apprentissage par généralisation soit en fait de la démonstration de théorèmes dans les jeux utilisant des exemples instanciés pour générer des règles générales sûres de pouvoir être utilisées.

4. L'Amorçage

Jacques Pitrat a aussi travaillé pendant de nombreuses années sur l'amorçage de l'intelligence artificielle. Il a écrit plusieurs systèmes d'amorçage à partir des années 1980, de Maciste [46, 47] qui utilisait des métaconnaissances, à CAIA [49], un Chercheur Artificiel en Intelligence Artificielle. Les idées d'amorçage utilisées dans ces systèmes m'ont inspiré pour améliorer la recherche Monte-Carlo. Ainsi ma motivation première lorsque j'ai conçu la recherche Monte-Carlo imbriquée [7] était d'utiliser

la recherche Monte-Carlo pour amorcer la recherche Monte-Carlo. Une autre façon d'amorcer la recherche Monte-Carlo plus liée à la découverte automatique d'algorithmes serait d'utiliser la génération d'expressions par la recherche Monte-Carlo [8] pour découvrir de nouveaux algorithmes de recherche Monte-Carlo [6]. La recherche Monte-Carlo imbriquée a par ailleurs aussi été utilisée pour les jeux [15].

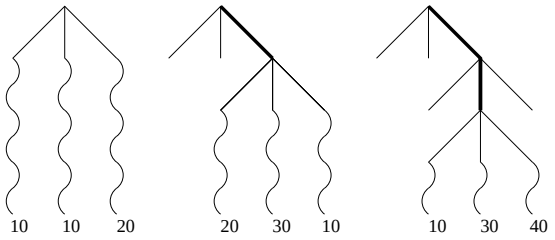


FIGURE 4.1. Recherche Monte-Carlo imbriquée.

La figure 4.1 donne une illustration de la recherche Monte-Carlo imbriquée. Les lignes ondulées symbolisent les *playouts* du niveau immédiatement inférieur. Sur l'arbre de gauche on voit le début d'une recherche Monte-Carlo imbriquée. Il y a trois coups possibles. Le principe de l'algorithme est d'essayer ces trois coups possibles puis de faire une recherche de niveau inférieur après chacun de ces coups. Dans l'exemple, la recherche qui renvoie le meilleur score de 20 est celle qui commence par le coup de droite. L'algorithme joue donc le coup de droite et continue son *playout* à partir de l'état où le coup a été joué. Il a de nouveau trois coups possibles qui sont symbolisés dans l'arbre du milieu. Le meilleur coup est le deuxième qui arrive à un score de 30 lorsqu'il est suivi par une recherche de niveau inférieur. Il est donc joué et la recherche continue ainsi comme montrée dans l'arbre de droite pour le coup suivant jusqu'à atteindre un état terminal.

Jacques Pitrat n'était pas un fanatique des méthodes de recherche Monte-Carlo. Il appelait cela le random Go. Il préférait les modélisations de la conscience réflexive et les systèmes à base de règles et de contraintes.

Les idées d'amorçage de systèmes d'intelligence artificielle se retrouvent dans les meilleurs algorithmes actuels pour les jeux. Que ce soient Alpha Zero, Polygames ou Athénan, ils s'améliorent tous automatiquement à partir de versions plus anciennes d'eux-mêmes en apprenant sur les traces des parties qu'ils jouent contre eux-mêmes.

5. Politique et Évaluation

L'équipe Métaconnaissance de ce qui était dans les années 1990 le LAFORIA (le Laboratoire Formes et IA qui a été intégré au LIP6 lors la fusion des laboratoires d'informatique de l'université Paris 6) était dirigée par Jacques Pitrat. Tous les ans, les membres de l'équipe et d'anciens doctorants de Jacques Pitrat devenus chercheurs ou enseignants-chercheurs, ainsi que d'autres chercheurs proches de l'équipe, se

réunissaient pendant quelques jours en septembre pour le colloque Métaconnaissance. Plusieurs colloques ont eu lieu sur l'île de Berder dans le Morbihan.

Lors du colloque de septembre 1999 à Berder, j'ai organisé un tournoi de programme de football des philosophes ou Phutball [5]. Lors du tournoi différentes équipes composées des participants au colloque ont été constituées. Un programme de Phutball de base était fourni à chaque équipe, qui devait l'améliorer et se monitorer en train de l'améliorer de façon à comprendre comment on découvrait des concepts utiles pour programmer le jeu. Chaque équipe a choisi un nom de philosophe pour se représenter et s'est isolée pour trouver de nouveaux concepts et les programmer. Je naviguais entre les équipes pour les aider à résoudre les problèmes de programmation qu'elles rencontraient. A la fin de l'après-midi chaque équipe a rendu son programme et j'ai fait jouer les programmes dans un tournoi. C'est l'équipe Platon constituée de Jean-Yves Lucas et d'Hélène Giroire qui a obtenu les meilleurs résultats. Le principe de leur programme était d'ignorer la fonction d'évaluation que j'avais proposée et de programmer une politique à la place. La politique consistait à placer un pion au bout de la ligne de pions la plus proche du bord adverse de façon à permettre des coups qui rapprochaient la balle du camp adverse. L'équipe Kant dirigée par Jacques Pitrat a essayé quant à elle de mettre des heuristiques dans la fonction d'évaluation, dont l'heuristique de mobilité du papier de 1968 sur le General Game Playing. Mais les résultats ont été bien moins bons que la modification de la politique proposée par l'équipe Platon.

On retrouve ici les concepts de politique et d'évaluation qui sont au centre des algorithmes actuels de jeux. Le débat n'est pas encore tranché entre les tenants des deux approches. Ainsi Alpha Zero et Polygames utilisent une politique combinée à une évaluation qui donne des bons résultats pour les jeux qui ont un grand nombre de coups possibles comme le Hex ou le Go. Quant à Athénan, il n'utilise pas de politique, mais seulement une évaluation, et a de meilleurs résultats pour les jeux ayant un plus petit facteur de branchement comme Breakthrough (*cf.* Figure 5.1).

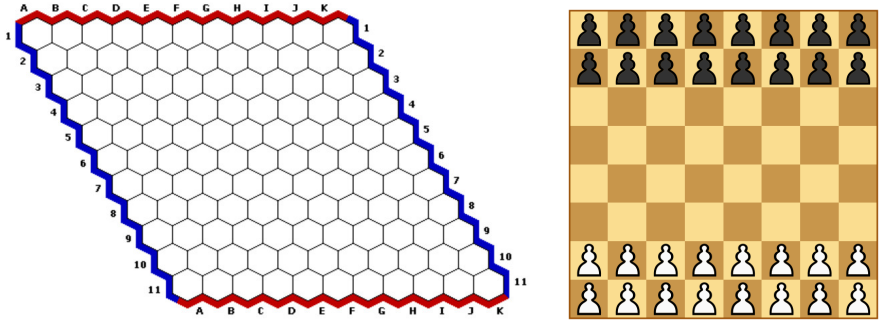


FIGURE 5.1. Un plateau de Hex et un plateau de Breakthrough. Il y a beaucoup plus de coups possibles à Hex.

6. Les Directions de Thèses sur les Jeux

Jacques Pitrat a encadré de nombreuses thèses sur l'intelligence artificielle dans les jeux. Il a ainsi encadré quatre thèses sur le jeu de Go, en commençant par celle de Bruno Bouzy [2] sur la modélisation d'un joueur de Go et sur Indigo le programme de Go correspondant, suivie par celle de Patrick Ricaud [52] sur l'abstraction du début de partie au jeu de Go. Vient ensuite ma propre thèse sur l'apprentissage par auto-observation au jeu de Go [4], et enfin la thèse de Régis Moneret [35] sur l'apprentissage de la dépendance entre sous-jeux. Les doctorants de Jacques Pitrat ont aussi abordé d'autres jeux complexes. Ainsi la thèse de Jean-Marc Nigro [36] portait sur la génération automatique de commentaires, avec une application au jeu de Tarot, et celle de Tristan Pannérec [37] sur des jeux de plateau de type *wargames*.

7. Conclusion

Jacques Pitrat avait une vision à long terme de l'évolution de l'intelligence artificielle et ses idées se retrouvent dans les systèmes d'intelligence artificielle les plus actuels, non seulement les systèmes de jeu mais aussi dans de nombreuses autres applications. Ainsi l'amorçage de la recherche Monte-Carlo est utilisée dans de nombreuses applications qui vont de la théorie des codes [27] à la conception de molécules d'ARN [50]. L'apprentissage imbriqué de politiques de simulations [54] a lui aussi de nombreuses applications qui vont des transports [14, 16, 21, 22] au génie logiciel [51] et à la bioinformatique [13, 23].

Les hommages suite à sa disparition ont été nombreux [11, 10, 20, 25, 53] et montrent le respect, l'admiration et l'attachement que de nombreux chercheurs avaient pour lui ainsi que l'importance de ses contributions à l'intelligence artificielle.

INFORMATION PROCESSING 68 - NORTH-HOLLAND PUBLISHING COMPANY - AMSTERDAM (1969)

REALIZATION OF A GENERAL GAME-PLAYING PROGRAM

JACQUES PITRAT
Institut Blaise Pascal, C.N.R.S.,
23, *Rue du Maroc*, 75, *Paris XIX*, *France*

We study some aspects of a general game-playing program. Such a program receives as data the rules of a game: an algorithm enumerating the moves and an algorithm indicating how to win. The program associates to each move the conditions necessary for this move to occur. It must find how to avoid a dangerous move.

We describe the part of the program playing the combinatorial game in order to win: how it can find the moves which lead to victory and what are the only opponent's moves with which he does not lose. This program has been tried with various games: chess, tic-tac-too, etc.

1. INTRODUCTION

My aim was to realize a program playing several games. The rules of the particular game which it must play are given as data. If we want to have a performing program, it must be capable of studying these rules.

The program is not completely general. It has limitations of three kinds:

a. It can only play games on a bidimensional board.
b. The rules of a game are written in a language which cannot describe every game, but which, however, covers a very large ground.
c. The more severe restrictions arise from heuristics which can be used in various games, but with very weak performances for some games.

I cannot describe the whole program, which is very large. I shall describe the combinatorial play which happens when we try to win, whatever the opponent may do.

The program can also play a positional game: this comes about when the opponent can play many moves without serious threats. We shall not discuss this part of the program.

2. LANGUAGE USED TO DESCRIBE THE RULES OF A GAME

There are two parameters for each square of the board:

a. One giving the occupation of the square: empty - friend - enemy,
b. One giving the type of man if the square is not empty.

For example, if the game is chess, the piece may be: king, rook, pawn ...

For some games, all men are of the same kind: tic-tac-toe, Go-Moku.

There are variables. They can represent a square or a number.

There are statements such as those of FORTRAN, ALGOL: arithmetic, test, go to statements.

Some are very specific to games:

a. Result statement. This statement gives information about winning in a particular state of the board. This may be: victory, loss, draw, no victory ...
b. Move statement. This statement describes a move, which can be made up of several parts (partial moves). The parts of a move fall into four types:
   i. The man in square A goes to square B
   ii. The man in square A is captured
   iii. A man of type T is put in square A
   iv. The man in square A becomes a new type T.

To sum up, the rules of a game are given as algorithms written in the language described above: an algorithm enumerating legal moves and an algorithm indicating how to win.

3. STUDY OF AN ALGORITHM

First, the program must find, for each move or for each indication of victory, the conditions necessary to obtain it. This is useful if we want to destroy an opponent's move or if we want to try to make a move possible, or to win, or to escape a danger.

These conditions are not given by algorithms,

which merely enumerate moves or indicate whether we win or not. To obtain these conditions, the program must study these algorithms.

An algorithm is put into the computer as a graph. It has many branch points, corresponding to tests, computed go to, loops. At each such point, there is a condition. For instance: "If a square K is empty, go to L1, otherwise to L2".

To each branch point, we associate its "conjugate". This is the first statement where we are sure to arrive, whatever branch we choose and whatever the answers of the following tests may be. There is always a conjugate: we gather in one statement all the possible terminal points of the algorithm. This is done in a first step: each branch point has its conjugate.

While executing an algorithm, at each branch point, we put the condition which has been satisfied in a push down. We remove it when we execute the conjugate: this condition now has no importance. Whether this condition is true or false, we are sure to arrive at this statement.

If we have a statement of result or of move, the necessary conditions are in the push down. Hence, we output the contents of the push down at each such statement.

For instance, if the game is chess, and if we have the state of the board shown in fig. 1, if we enumerate the opponent's moves we have:

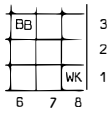


Fig. 1.

The man in the square (8,1) is captured. The man in (6,3) goes to (8,1).

With this move, the program gives the conditions:

a. Enemy man in (6,3)
b. Bishop in (6,3)
c. Square (7,2) empty
d. Friendly man in (8,1).

We see the interest of this method. If we want to avoid losing, we must destroy one of these conditions. The number of moves to consider is thus greatly reduced.

This method is not entirely exact. Under certain circumstances, some conditions are not taken into account. But this case does not arise in any of the games for which I have written rules. It is possible to write a more complex program which always gives all the conditions.

It is also useful to enumerate the moves or the wins that would be possible if the board were changed. If we force an empty square to be occupied by an enemy man, what are the new moves? If we force a square to be occupied by a friendly man, do we win? In these cases, we may say that we are forcing.

To see what happens with forcings, at each modifiable branch point, we store the state of the variables and the references of the statements to which the other branches lead. This is done only if this forcing is possible for the game.

When its normal work is finished, the program is reset to the state which has been stored and takes another branch. It stops when it arrives at the conjugate: the rest of the algorithm has already been done. At each result or move, it also outputs the conditions, but indicates those which have been forced.

In the figure shown above for chess, the following move should be recognized as a forcing: The man in (7,2) is captured. The man in (6,3) goes to (7,2).

Under the conditions:

a. The man in (6,3) is an enemy
b. A bishop is in (6,3)
c. The man in (7,2) is a friend. This is the forcing condition.

4. THE SEARCH FOR A WIN

First, let us show how we can destroy or fulfil a condition.

If the condition is:

the square A is empty

we can destroy it by bringing a man there. To destroy: the man in the square A is a friend, we can take away the man.

To destroy: the man in the square A is an enemy, we can capture the man in A.

The methods to fulfil a condition are similar.

This method is entirely general and good for every game. Of course, if a particular game has no capture move, a condition like: "enemy is in square A", cannot be destroyed by the player.

We count as a single condition two conditions on the same square.

Now, we can see how to win against all defences. In a first stage, we shall get pairs: move dangerous for the opponent - list of conditions one of which must be destroyed by the opponent. In a second stage, we shall get pairs: move dangerous for the opponent - list of the opponent's counterstrokes. Then we shall see how the program can choose among its moves.

Suppose the algorithm indicates when we win. In parenthesis, we write what to do when it indicates when we do not win.

There are two cases:

a. There is a win with one forcing (only one condition prevents us from winning). There are conditions $E_1, \ldots, E_p$, already fulfilled (for win only) and one condition k to be fulfilled (destroyed). If a move fulfils (destroys) k without destroying $E_1, \ldots, E_p$, we win. But if no move fulfils (destroys) k, we try to see if we can fulfil (destroy) k in two moves. We enumerate all the moves which can fulfil (destroy) it with one forcing, and we remove all those which destroy one or several of the E_j. Let $q_1, \ldots, q_n$ be these moves.
Let q_i be one of these. For the others, we will proceed in the same way. We know the conditions $C_1, \ldots, C_t$ necessary for this move, and there is a condition D to fulfil since there is one forcing. We look for the move fulfilling D and we remove those destroying one of the E_j or one of the C_j. Let $r_1, \ldots, r_s$ be these moves.
If we play one of these (which fulfils D), for instance r_m, the opponent is obliged to destroy the move q_i, if he does not, he loses after q_i, unless he destroys one already fulfilled condition for a win. Thus he must destroy one of the conditions C_j or D, or one of the E_j, winning conditions already fulfilled. We have a list of pairs:

$$r_m - E_1, \ldots, E_p, C_1, \ldots, C_t, D,$$

b. There is a win with two forcings (two conditions prevent us from winning).
Suppose we are in the first case. Conditions $E_1, \ldots, E_p$ are already fulfilled and two conditions, k_1 and k_2, must be fulfilled. There may be many possibilities of this type. We will proceed in the same way for each of them.
Let us take k_1 (when this is finished, we do it again, swapping k_1 and k_2). We enumerate the moves fulfilling k_1 and we eliminate those which destroy one or several E_j. Let $q_1, \ldots, q_n$ be these moves.
After playing q_i, if we fulfil the condition k_2, we win. Let $t_1, \ldots, t_s$ be the moves fulfilling k_2. Suppose for the sake of simplicity that there is only one move M and let $C_1, \ldots, C_t$ be the conditions necessary for this move.
If we play q_i, the opponent must prevent us from playing M or destroy one of the conditions already fulfilled; he must destroy one of the C_j to prevent us from playing M next, which thus enables us to win, or one of the E_j or k_1 to destroy a condition already fulfilled.
If we play M next, there will always be a condition to fulfil.
Thus we have the list of pairs:

$$q_i - C_1, \ldots, C_t, E_1, \ldots, E_p, k_1.$$

Now, for each condition to destroy, we look for the opponent's moves destroying it. Thus we have a new list of pairs, the first element being a player's move and the second, the list of the opponent's counterstrokes. He must choose among them if we play the first element and if he does not want to lose. If there is no move in the second element, we win.

Let us examine the application of this method to chess. The win algorithm tells us we do not win. By our method, the program sees that it is because there is an opponent's king in the square A. We are in the first case and $p = 0$. Condition k is: enemy king is in the square A. If there is a move which destroys it: a move which captures the man in A, we win. If not, we look for the moves which destroy it with one forcing. For instance, the condition to fulfil may be:

a. Friendly rook in square B, or
b. Square C empty (discovered check).

Let us take the first case: there must be a friendly rook in B. The win move: "Rook captures king" needs other conditions which are fulfilled, for instance:

C_1 : enemy man in A
C_2 : square E empty.

We look for the moves bringing a friendly rook in B. Let a move be r_i: Rook in F goes to B.

If we play this move, the condition: "Friendly rook is in square B" is then fulfilled; the opponent must destroy one of the conditions of "Rook captures king":

a. Friendly rook is in B
b. Enemy man is in A
c. Square E is empty.

For each condition, he looks for the moves destroying it. If, for instance, he cannot move his king, without a new check, nor capture the rook, he may have only two moves:

a. The man in H goes to E
b. The man in I goes to E.

Then we know that if we play:
Rook in F goes to B,
the opponent has only two counterstrokes:

a. The man in H goes to E
b. The man in I goes to E.

It is essential to see that this method is entirely general. I gave an instance for chess, but we can apply it to Go-Moku or to tic-tac-toe.

We can make two remarks:

a. A move may produce many threats (for instance double check). In this case, we will find twice or more the same move as first element of a pair. Then, we remove all these pairs and we create a new pair: its first element is the move, and the second a list of conditions obtained by a "and" of the lists of conditions. If there is no condition, we win.
b. When we play the move which is the first element of the pair, we must verify that the opponent does not win. If he does, we must remove the pair: it is useless for the opponent to destroy the threat, he wins before it occurs. We must also verify for each opponent's counterstrokes that there is no new threat for him. For instance, if the game is chess, it is useless to move his king from one check towards another check.

When the program has the following list of pairs: threatening moves - list of opponent's counterstrokes, it must choose one of the first elements. The opponent is then free to choose among the corresponding list of counterstrokes. We have a tree.

We must use heuristics in order to find quickly if we can win. One of them is to try first the moves where the opponent has few counterstrokes. We restrict his possibilities and we see more easily all the possible cases.

When we reach a win, it does not mean that we win, because the opponent may attempt a different move previously. We must prune the tree, working towards the beginning. If the opponent has only one possibility and loses, we climb up two levels higher: we choose the move which leads to a win. If there are more than one branch, we only cut the branch, if there is only one branch, we resume the procedure.

If we return to the beginning, we win whatever move the opponent chooses. We stop if we have no further possibility of threatening the opponent or if he has too many ways of escape. This measure is heuristic.

In chess, this method leads to a sequence of checks. It is very close to Mater I of Baylor and Simon [1].

5. SOME RESULTS OF THE PROGRAM

It has been written for the CDC 3600. We describe a move by:

a. A man is added in square A : A
b. A man is moved from A to B : A-B
c. The man in square A is captured : × A.

A square is characterized by its two coordinates.

Tic-tac-toe.

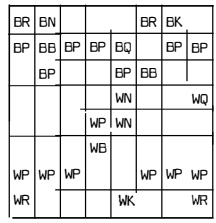
```
cross      noughts
 2,3         2,1
 3,1         1,3
 2,2 - 3,2  and victory after the next move.
```

```
cross      noughts
 3,3         1,1
 1,3         2,3
 3,3 - 3,2  and victory after the next move.
                                  Time: 6 seconds
```

Fig. 2.

Thus, the program wins if it is playing first and can put its first man in the center.

Chess. K = King; Q = Queen; R = Rook;
B = Bishop; N = Knight; P = Pawn;
W = White; B = Black.




```
x  8,7;8,5-8,7        x 8,7;7,8-8,7
x  6,6;5,4-6,6        if    8,7-8,8

                Then   5,5-7,6  and victory
                Thus   8,7-8,6

   5,5 - 7,4           8,6 - 7,5
   8,2 - 8,4           7,5 - 6,4
   7,2 - 7,3           6,4 - 6,3
   4,3 - 5,2           6,3 - 7,2
   8,1 - 8,2           7,2 - 7,1
   5,1 - 4,2  and win          Time: 47 seconds
```

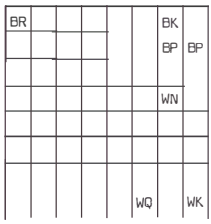
Fig. 3.

Edward Lasker played exactly the same sequence of moves. It is not the quickest mate. There is a mate in seven moves (see fig. 4).

The sequence found by the program is the sequence given by Tarrasch.

6. CONCLUSION

We must not compare the performance of a general program with that of a program playing




```
6,1 - 3,4     If  7,8 - 6,8
              Then  3,4   6,7 and win
              Thus  7,8 - 8,8
7,5 - 6,7           8,8 - 7,8
6,7 - 8,6     If  7,8 - 6,8
              Then  3,4 - 6,7 and win
              Thus  7,8 - 8-8
3,4 - 7,8       x 7,8   1,8 - 7,8
8,6 - 6,7   and win
              Time : 77 seconds
```

Fig. 4.

only one game. Artificial intelligence aims to get general programs. If they are too particular, we often have progress in the theory of the particular problem, but no progress in the theory of artificial intelligence. Also, a particular program cannot study new problems.

This program has been tried on games as varied as tic-tac-toe, Go-Moku, chess and cylindrical chess.

It is important to see that nowhere does it use the particular properties of the game. If it is playing chess, it does not know, for example, that it is advisable to protect its men.

REFERENCE

[1] Baylor and Simon, A chess mating combinations program, AFIPS, Vol. 28 p. 341.

DISCUSSION

Question by K. Paton

In your conclusion, you state that in playing chess, the program does not know that it is protecting men. Does the program ever find out that it is protecting men?

Answer

No, it cannot learn. The general method enables it to perform in each specific case, but it is not formally capable of discovering the notion of protection. Incidentally, the idea of protection in chess does not apply in all cases, and makes the formal discovery of the program difficult.

Question by D. Levy

How does the program decide what to do in a non-tactical position?

Answer

The program takes into account the possibility of winning and forcing, e.g. in Go-Molar, it takes into account relationships of moves and opponent's moves, and tries to increase its mobility (i.e. the number of possible moves) and lessen the mobility of the opponent. The definition of mobility is included in the program, and is completely general.

Question by P. Braffort

I can understand the advantage of a general approach to the problem of artificial intelligence treating games of a different nature on the same level and with the same formalism for the establishment of rules. But have you also made an independent evaluation or comparison of your program with others, and if so how do you find it?

Answer

The aim of the program is to play several board games, but not specifically any one game, and I would not expect it to be better than a specific program for a specific game.

Question by G. Nagy

Can you compare the performance of your program with the Simon-Baylor results?

Answer

A general purpose game playing program will obviously be less efficient than a special program for a game. The combinational program is finished, and I am now writing the positional program.

Bibliographie

[1] H. Berliner, R. Greenblatt, J. Pitrat, A. Samuel & D. Slate, « Panel on Computer Game Playing », *IJCAI* (1977), p. 975-982.

[2] B. Bouzy, « Modélisation cognitive du joueur de Go », Thèse, Université Paris 6, 1995.

[3] C. Browne, M. Stephenson, É. Piette & D. J. N. J. Soemers, « A Practical Introduction to the Ludii General Game System », in *Advances in Computer Games*, Springer, 2019, p. 167-179.

[4] T. Cazenave, « Système d'apprentissage par auto-observation. Application au jeu de Go », Thèse, Université Paris 6, 1996.

[5] ———, « Un tournoi de programmes de Phutball », in *Actes du Colloque de Berder*, 1999.

[6] ———, « Evolving Monte Carlo tree search algorithms », *Dept. Inf., Univ. Paris* **8** (2007).

[7] ———, « Nested Monte-Carlo Search », in *IJCAI*, 2009, p. 456-461.

[8] ———, « Nested Monte-Carlo Expression Discovery », in *ECAI*, 2010, p. 1057-1058.

[9] ———, « Generalized rapid action value estimation », in *24th International Joint Conference on Artificial Intelligence*, 2015, p. 754-760.

[10] ———, « Disparition de Jacques Pitrat », Site web du CNRS `https://ins2i.cnrs.fr/fr/cnrsinfo/disparition-de-jacques-pitrat`, 2019.

[11] ———, « Jacques Pitrat (1934-2019): An obituary », *ICGA Journal* **42** (2020), n° 1, p. 38-40.

[12] T. Cazenave, Y.-C. Chen, G.-W. Chen, S.-Y. Chen, X.-D. Chiu, J. Dehos, M. Elsa, Q. Gong, H. Hu, V. Khalidov, L. Cheng-Ling, H.-I. Lin, Y.-J. Lin, X. Martinet, V. Mella, J. Rapin, B. Roziere, G. Synnaeve, F. Teytaud, O. Teytaud, S.-C. Ye, Y.-J. Ye, S.-J. Yen & S. Zagoruyko, « Polygames: Improved Zero Learning », *ICGA Journal* **42** (2020), n° 4, p. 244-256.

[13] T. Cazenave & T. Fournier, « Monte Carlo Inverse Folding », in *Monte Search at IJCAI*, 2020.

[14] T. Cazenave, J. Lucas & H. Triboulet, Thomas andKim, « Policy Adaptation for Vehicle Routing », *AI Communications* **34** (2021), n° 1, p. 21-35.

[15] T. Cazenave, A. Saffidine, M. J. Schofield & M. Thielscher, « Nested Monte Carlo Search for Two-Player Games », in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, 2016, p. 687-693.

[16] T. Cazenave & F. Teytaud, « Application of the Nested Rollout Policy Adaptation Algorithm to the Traveling Salesman Problem with Time Windows », in *LION*, 2012, p. 42-54.

[17] J. E. Clune, « Heuristic evaluation functions for general game playing », *KI-Künstliche Intelligenz* **25** (2011), n° 1, p. 73-74.

[18] Q. Cohen-Solal & T. Cazenave, « Minimax Strikes Back », in *Workshop Reinforcement Learning in Games at AAAI*, 2021.

[19] G. DeJong & R. Mooney, « Explanation-based learning: An alternative view », *Machine learning* **1** (1986), n° 2, p. 145-176.

[20] Y. Demazeau, « Hommage à Jacques Pitrat », *Bulletin de l'AFIA* **107** (2020), p. 4.

[21] S. Edelkamp, M. Gath, T. Cazenave & F. Teytaud, « Algorithm and knowledge engineering for the TSPTW problem », in *Computational Intelligence in Scheduling (SCIS), 2013 IEEE Symposium on*, IEEE, 2013, p. 44-51.

[22] S. Edelkamp, M. Gath, C. Greulich, M. Humann, O. Herzog & M. Lawo, « Monte-Carlo Tree Search for Logistics », in *Commercial Transport*, Springer International Publishing, 2016, p. 427-440.

[23] S. Edelkamp & Z. Tang, « Monte-Carlo Tree Search for the Multiple Sequence Alignment Problem », in *Eighth Annual Symposium on Combinatorial Search*, 2015.

[24] H. Finnsson & Y. Björnsson, « Simulation-Based Approach to General Game Playing », in *AAAI*, 2008, p. 259-264.

[25] J.-G. Ganascia, « Hommage à Jacques Pitrat », *Bulletin de la Société Informatique de France* **15** (2020), p. 113-116.

[26] M. R. Genesereth, N. Love & B. Pell, « General Game Playing: Overview of the AAAI Competition », *AI Magazine* **26** (2005), n° 2, p. 62-72.

[27] D. Kinny, « A New Approach to the Snake-In-The-Box Problem », in *ECAI*, vol. 242, 2012, p. 462-467.

[28] L. Kocsis & C. Szepesvári, « Bandit based Monte-Carlo planning », in *17th European Conference on Machine Learning (ECML'06)*, LNCS, vol. 4212, Springer, 2006, p. 282-293.

[29] J. Méhat & T. Cazenave, « Monte-carlo tree search for general game playing », *Univ. Paris* **8** (2008).

[30] ———, « Ary, a general game playing program », in *Board games studies colloquium*, 2010.
[31] ———, « Combining UCT and Nested Monte Carlo Search for Single-Player General Game Playing », *IEEE Transactions on Computational Intelligence and AI in Games* **2** (2010), nº 4, p. 271-277.
[32] J. Méhat & T. Cazenave, « A Parallel General Game Player », *KI* **25** (2011), nº 1, p. 43-47.
[33] S. Minton, « Constraint-based generalization: Learning game-playing plans from single examples », in *Proceedings of the Fourth AAAI Conference on Artificial Intelligence*, 1984, p. 251-254.
[34] T. M. Mitchell, R. M. Keller & S. T. Kedar-Cabelli, « Explanation-based generalization: A unifying view », *Machine learning* **1** (1986), nº 1, p. 47-80.
[35] R. Moneret, « Strategos : un système multi-jeux utilisant la théorie combinatoire des jeux, capable d'apprendre automatiquement les dépendances entre sous-jeux locaux », Thèse, Université Paris 6, 2000.
[36] J.-M. Nigro, « La conception et la réalisation d'un générateur automatique de commentaires : le système GénéCom. Application au jeu du Tarot », Thèse, Université Paris 6, 1995.
[37] T. Pannérec, « Un système général avec un contrôle de la résolution à base de métaconnaissances pour des problèmes d'affectation optimale », Thèse, Université Pierre et Marie Curie, 2002.
[38] B. Pell, « Strategy generation and evaluation for meta-game playing », Thèse, Citeseer, 1993.
[39] J. Pitrat, « Realization of a general game-playing program », in *IFIP Congress (2)*, 1968, p. 1570-1574.
[40] ———, « A General Game Playing Program », *Artificial Intelligence and Heuristic Programming (eds. Findler and Meltzer)* (1971), p. 125-155.
[41] ———, « A Program to Learn to Play Chess », *Pattern Recognition and Artificial Intelligence* (1976), p. 399-419.
[42] ———, « Realization of a program learning to find combinations at chess », in *Computer oriented learning processes*, vol. 14, Noordhoff, 1976.
[43] ———, « A chess combination program which uses plans », *Artificial Intelligence* **8** (1977), nº 3, p. 275-321.
[44] ———, « The behaviour of a chess combination program using plans », *Advances in Computer Chess* **2** (1979).
[45] ———, « A Program which Uses Plans for Finding Combinations in Chess », *ICCA Newsletter* **2** (1979), nº 2.
[46] ———, « MACISTE ou comment utiliser un ordinateur sans écrire de programme », in *Colloque Intelligence Artificielle de Toulouse, publication 58, CNRS-LAFORIA, Université de Paris VI*, 1985, p. 223-240.
[47] ———, *Métaconnaissance, futur de l'intelligence artificielle*, Hermès, Paris, 1990.
[48] ———, « Games: The next challenge », *ICGA Journal* **21** (1998), nº 3, p. 147-156.
[49] ———, *Artificial beings: the conscience of a conscious machine*, John Wiley & Sons, 2013.
[50] F. Portela, « An unexpectedly effective Monte Carlo technique for the RNA inverse folding problem », *BioRxiv* (2018).
[51] S. M. Poulding & R. Feldt, « Generating structured test data with specific properties using nested Monte-Carlo search », in *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014*, 2014, p. 1279-1286.
[52] P. Ricaud, « Gobelin une approche pragmatique de l'abstraction appliquée à la modélisation de la stratégie élémentaire du jeu de Go », Thèse, Université Paris 6, 1995.
[53] C. Roche, « Jacques Pitrat (54) Pionnier Français de l'Intelligence Artificielle », *La Jaune et la Rouge* **756** (2020), nº 6, p. 22.
[54] C. D. Rosin, « Nested Rollout Policy Adaptation for Monte Carlo Tree Search », in *IJCAI*, 2011, p. 649-654.
[55] A. Saffidine, T. Cazenave & J. Méhat, « UCD: Upper Confidence Bound for Rooted Directed Acyclic Graphs », *Knowledge-Based Systems* **34** (2011), p. 26-33.
[56] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel & D. Hassabis, « Mastering the game of Go with deep neural networks and tree search », *Nature* **529** (2016), nº 7587, p. 484-489.

[57] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan & D. Hassabis, « A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play », *Science* **362** (2018), n° 6419, p. 1140-1144.
[58] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan & D. Hassabis, « Mastering the game of go without human knowledge », *Nature* **550** (2017), n° 7676, p. 354.
[59] C. F. Sironi, « Monte-Carlo Tree Search for Artificial General Intelligence in Games », Thèse, Maastricht University, 2019.

Abstract. — This paper describes the contributions of Jacques Pitrat to artificial intelligence applied to games as well as some works that were inspired from his research. The paper evocates General Game Playing, then Explanation Based Generalization, Bootstrap, duality between policy and evaluation as well as some PhD thesis on games advised by Jacques Pitrat.

Keywords. — Artificial Intelligence, Games, General Game Playing, Machine Learning, Bootstrap.

Manuscrit reçu le 30 mars 2021, révisé le 10 novembre 2021, accepté le 30 novembre 2021.