appelée « concaténation ». On a donc là une structure algébrique typique.

D'autre part, lorsqu'on utilise une représentation syntaxique des langages comme celle de la figure 6 ou également une représentation de programme sous forme de bloc diagramme comme ceux de la page 44, on utilise un autre type de structure qui est celle des *graphes* qui sont la représentation sagittale de relations binaires. C'est ainsi que la relation binaire définie par l'ensemble des couples *(a, b), (b, c), (a, c), (c, e), (e, a), (a, d), (d, e)* est l'équivalent du graphe de la figure i6.

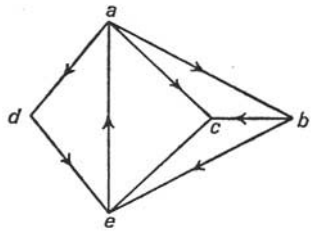

FIG. I6
Représentation « sagittale » d'un graphe

C'est encore là un type bien connu de structure algébrique.

Une seconde remarque est liée à la dualité que l'on peut apercevoir dès à présent entre le langage ou le système formel considéré comme réalisé dans leur totalité et l'automate muni d'un programme qui engendre les expressions, les formules de ce langage, au fur et à mesure de son fonctionnement. Nous aurons l'occasion de revenir à plusieurs reprises sur cette distinction qui est celle de *l'extension* et de *l'intension.*

CHAPITRE III

Les jeux

Dans le chapitre VIII au cours duquel nous rassemblerons un certain nombre d'informations d'ordre historique, on verra que, même chez les lointains précurseurs de l'intelligence artificielle, la simulation des jeux a été considérée comme un exemple ou un test significatif.

Cela n'est pas étonnant si l'on adopte, pour aborder le problème de l'intelligence, un point de vue génétique analogue à celui de Jean Piaget, comme nous avons tenté de le faire au cours du premier chapitre.

L'activité ludique commence avant même l'activité linguistique et se mélange intimement avec elle. Elle met en oeuvre les six stades que nous avons énumérés en analysant les étapes de l'apparition de l'intelligence, en particulier la troisième (procédés destinés à faire durer les spectacles intéressants) et la cinquième (découverte des moyens nouveaux par expérimentation active).

C'est donc dans le jeu que s'exerce l'intelligence naissante. Non contente de l'apprentissage que la rencontre du monde lui impose, elle se donne ainsi la possibilité d'une expérimentation accélérée où les problèmes de l'univers réel sont posés sous une forme concentrée.

La simulation des jeux est ainsi un exercice privilégié pour la recherche des méthodes et pour l'estimation des difficultés en intelligence artificielle.

D'ailleurs si l'automate ne peut manipuler que des informations préalablement codées, et ceci en fonction d'un programme lui-même rédigé de façon convenue, il y a évidemment intérêt à se proposer tout d'abord des « textes » tirés d'un langage assez pauvre, que ce soit du point de vue du vocabulaire ou de celui de la syntaxe.

Ici encore, les jeux nous fournissent une entrée en matières commode. Tout comme dans le cas d'un système formel de type mathématique,

les jeux - au moins ceux que nous considérerons ici - se définissent à l'aide d'un système de règles complètement explicitées. Mais leur conduite, contrairement aux mathématiques usuelles, se fait en énonçant complètement chaque pas, sans aucun raccourci, sous-entendu ou «abus de langage ».

Ou encore, si l'on considère une partie comme une phrase où les mots définissent les coups joués - on a donc ici un dialogue -, les règles de grammaire sont les règles du jeu et le point final le constat du gain ou de l'échec ; on voit qu'on dispose là d'une langue fort simple malgré la grande variété de ce qu'elle permet de dire !

Pour illustrer cette remarque nous allons en développer les conséquences dans le cas d'un jeu de cartes particulièrement simple.

Imaginons un jeu de trente-deux cartes usuel. Pour simplifier nous dénoterons les couleurs par les signes K (pour carreau), Q (pour coeur), P (pour pique), T (pour trèfle) et les valeurs par R (roi), S (reine), V (valet), I0, 9, 8 et 7. Les joueurs seront désignés respectivement par A et B. Supposons que chaque joueur ait reçu sept cartes et qu'on ait retourné une des cartes restantes pour désigner l'atout.

Dès lors une partie pourra être considérée comme un texte composé de sept « paragraphes ». Chaque paragraphe comprend trois « phrases » ; la première phrase comprend trois « mots » : un symbole de couleur, un symbole de valeur et un nom de joueur. La phrase suivante est de même nature, mais le nom du joueur doit être différent. Enfin la troisième phrase ne comprend qu'un seul mot : le nom du joueur qui emporte le pli. Dès lors le nom du joueur figurant dans la première phrase du paragraphe suivant devra être le même.

Ces règles assurent ce qu'on pourrait appeler la cohérence *syntaxique* du texte.

Il semble naturel de rattacher au niveau *sémantique* la règle qui désigne le joueur emportant le pli en tenant compte d'une relation d'ordre entre valeurs à l'intérieur d'une même couleur, et de la priorité accordée soit à l'atout, soit, en l'absence d'atout, au joueur ayant déposé la première carte dans le pli.

Enfin le calcul des plis à la fin de la partie, en vue de la proclamation du vainqueur, appartient évidemment au niveau *pragmatique.*

Bien entendu les « textes » ainsi rédigés semblent fort différents de notre langage naturel, mais au fond certaines formes de poésies le

sont également. Par contre syntaxe et sémantique sont ici d'une définition autrement aisée que dans le cas du langage naturel.

L'image linguistique est donc particulièrement commode pour exprimer la structure du jeu, notamment en ce qui concerne l'énoncé des règles.

Par contre la découverte d'une *stratégie* propre à assurer le succès, c'est-à-dire le gain de la partie, si elle peut être aidée par l'utilisation d'un modèle linguistique, c'est-à-dire d'une formalisation, nous oblige à mettre en oeuvre d'autres concepts, situés à un niveau différent mais qui, bien entendu, sont, eux aussi, susceptibles d'être formalisés.

C'est ainsi que l'une des notions qui se présentent le plus naturellement à l'esprit d'un joueur est celle de la *distance* qui le sépare encore de la victoire.

Dans les jeux où l'habileté est essentiellement liée à un effort physique, cette distance est facile à définir : distance (au sens géométrique) restant à parcourir, écart à combler entre les buts marqués, etc.

Dans les jeux « intellectuels », les plus élémentaires, la définition de la distance reste transparente : nombre de dominos restant à caser, nombre de pièces d'un puzzle restant à assembler, distance restant à parcourir sur le « chemin » du jeu de l'oie, etc.

Pour les jeux intellectuels plus compliqués comme le jeu des échecs, il n'existe pas de définition naturelle de la distance, et il en va de même pour des tâches intelligentes que nous examinerons par la suite : démonstration des théorèmes, traduction automatique.

Définir une distance entre « situations » au cours d'un jeu et, grâce à cette définition, déterminer les conditions d'un chemin optimal (ici, un chemin conduisant au gain) est donc une contribution importante à l'ensemble des rubriques de l'intelligence artificielle. On peut même dire que c'est là l'essence de cette « heuristique », notion mal définie mais très suggestive, que l'on évoque bien souvent dans les tentatives qui appartiennent à notre domaine.

En résumé, il existe deux voies d'accès principales à la simulation des jeux intellectuels : un point de vue « géométrique », où la notion de « situation » et de « distance » joue un rôle essentiel, et un point de vue « linguistique » où l'on met l'accent sur l'énoncé des situations et des actions. L'automatisation établit naturellement un lien entre ces deux points de vue puisque le cheminement dans l'espace des situations

est accompli par des actions dont l'énoncé définit une macro-instruction, au sens que nous avons défini dans le chapitre précédent.
Les jeux les plus simples (cache-tampon, quatre coins, etc.) font appel souvent à des éléments géométriques et physiques de l'univers extérieur que les automates actuels, meubles rigides et fixes, ne peuvent évidemment pas atteindre directement. Certes, une représentation symbolique des lieux et des déplacements n'est nullement exclue, mais nous nous limitons pour le moment aux jeux dont la définition même ne comporte que l'échange d'informations, mots, lettres, chiffres ou autres symboles aisément codifiables.
Enfin une distinction s'impose entre les jeux qui font intervenir un élément de hasard (jeux de cartes en particulier) et ceux dont le développement est entièrement déterminé par la seule initiative des joueurs (le jeu des échecs est l'exemple le plus couramment utilisé).
Les deux types de jeux peuvent être abordés par les automates, mais il est clair que l'intervention du hasard n'apporte rien de significatif. Aussi nous concentrerons-nous sur l'étude de jeux appartenant au deuxième type, en remarquant toutefois qu'il existe des programmes ou des fragments de programmes relatifs aux jeux de cartes et notamment au bridge.
Les jeux dans lesquels le hasard n'intervient pas sont appelés « jeux à information complète ».

ELEMENTS D'UNE THEORIE DES JEUX

Si nous avons invoqué, au début de ce chapitre sur les jeux, l'importance qu'ils possèdent dans l'apprentissage de l'intelligence naturelle, nous n'avons pas l'intention d'aborder leur simulation par les machines à calculer sous l'angle de l'apprentissage. Certes il y a eu - et il y a encore - de nombreux travaux relatifs à la simulation de l'apprentissage par les machines. Ces travaux sont même rangés par certains auteurs (notamment par Minsky) dans le cadre de l'intelligence artificielle.
Il nous semble que les jeux ne sont simulés de façon intéressante que si l'on introduit d'emblée dans le programme une information théorique préalable (nous renonçons donc délibérément, encore une fois, au point de vue anthropomorphique).
C'est cette information théorique que nous allons présenter maintenant.

Pour essayer d'analyser, au point de vue théorique, ce qui se produit au cours d'un jeu « à deux personnes » (personnes que nous appellerons respectivement « blanc » et « noir »), nous devons tout d'abord définir une « position». On appellera « position» le couple constitué par un *diagramme* et par le *trait.* Le diagramme sera la place des pièces sur l'échiquier, des pions sur le damier, etc., à un instant donné. Le trait désigne celui des joueurs qui doit jouer à cet instant.
On a donc deux ensembles : E_b = ensemble de toutes les positions possibles avec trait aux « blancs » ; E_n = ensemble de toutes les positions possibles avec trait aux « noirs ». Évidemment, le trait passe alternativement d'un joueur à l'autre.
En utilisant les notations ensemblistes classiques (que nous avons rappelées au chapitre précédent), on pourra formaliser l'algèbre des situations et la logique du succès et de l'échec (I).

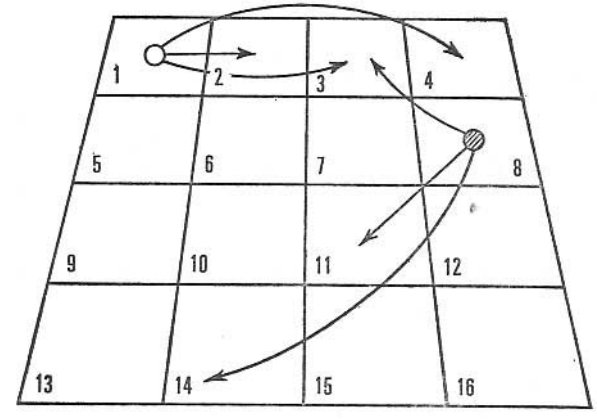

FIG. 17. - Représentation géométrique d'un jeu à deux personnes

Pour fixer les idées, imaginons une représentation géométrique du jeu comme celle de la figure t7.

(I) Pour ce paragraphe on se reportera avec fruit au petit livre de C. BERGE, *Théorie générale des jeux* à n *personnes,* Gauthier-Villars, 1960.

Les flèches représentent les mouvements possibles du pion blanc et du pion noir. On suppose qu'un pion n'a pas le droit d'en chevaucher ou d'en sauter un autre.
A la position initiale on peut associer l'expression :

$$x = \{ (b,I) ; (n,8) \}$$

puis les positions possibles suivantes :

$$y_1 = \{ (b,2) ; (n,8) \}$$
$$y_2 = \{ (b,3) ; (n,8) \}$$
$$y_3 = \{ (b,4) ; (n,8) \}, \text{ etc.}$$

ce qui se représente aisément par le graphe de la figure 18.

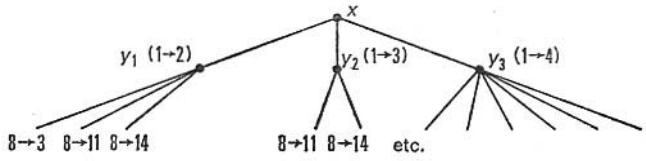

Fig. 18. - Développement arborescent des coups possibles à partir de la position représentée par la figure 17

Une position x étant donnée, on désigne par Γx l'ensemble de toutes les positions qui sont des *successeurs* possibles pour x ; la loi Γ est une « application multivoque » :

$$\text{si } x \in E_b \qquad \text{on a } \Gamma x \subset E_n$$
$$\text{si } x \in E_n \qquad \text{on a } \Gamma x \subset E_b .$$

Une *position de gain aux blancs* est une position $z \in E_b$ telle que $\Gamma z = \emptyset$. Leur ensemble est $M_b \subset E_b$.
On définira de même *l'ensemble de gain aux noirs* M, *et l'ensemble de jeu nul* $P = \{ z / \Gamma z = \emptyset \} - (M_b \cup M_n)$.
Le jeu se termine si la position $x \in P \cup M_b \cup M_n$, mais dans ce modèle, il peut aussi ne jamais se terminer.
La théorie mathématique du jeu se divise alors en deux : une théorie locale et une théorie globale.

I° La théorie locale repose sur l'idée de *tactique :* à toute position $x \in E_n$, les blancs attachent une valeur numérique $f(x)$ qui est d'autant plus grande que la position x est jugée bonne pour les blancs. f est la *fonction de préférence.* On prendra :

$$0 \leq f(x) \leq I$$
$$f(x) = I$$

et on peut définir :

- une tactique d'ordre I - si $x \in E_b$, les blancs choisissent dans Γx la position y telle que :

$$f(y) = \max_{z \in \Gamma x} f(z) ;$$

- une tactique d'ordre 2 - si $x \in E_b$, et si au coup suivant la position est $x' \in E_b$, les blancs peuvent garantir :

$$\max_{y' \in \Gamma x'} f(y').$$

Si les blancs choisissent dans Γx une position $y \in E_n$, les noirs vont choisir dans Γy une position x' telle que l'expression ci-dessus soit aussi petite que possible ; ils peuvent garantir :

$$\min_{x' \in \Gamma y} \max_{y' \in \Gamma x'} f(y') = f^{(1)}(y).$$

Une tactique d'ordre 2 consistera pour les blancs à choisir dans Px une position y telle que :

$$f^{(1)}(y) = \max_{z \in \Gamma x} f^{(1)}(z).$$

La théorie locale est donc relative au choix de la fonction de préférence f. Or, dans le cas général, il n'existe pas de déduction simple de f à partir des règles du jeu. Le choix de f résulte d'un ensemble d'intuition et de tâtonnements. Il est « heuristique » !
2° La théorie globale repose sur l'idée de stratégie ; pour les blancs, *la stratégie,* d'après von Neumann, est une loi qui fait correspondre à toute position $x \in E_b$ une position $y \in \Gamma x$ que l'on note $y = \sigma(x)$. σ est alors une application univoque.

On peut donner une description ensembliste de l'ensemble des positions gagnantes.
Si A c E_b par exemple, on pose :

$$\bar{\Gamma}A = \{\ x/\Gamma\, x \cap A \neq \varnothing\ \}$$
$$\text{et}\quad \overset{+}{\Gamma}A = \{\ x/\Gamma\, x \neq \varnothing\ \Gamma\, x \subset A\ \}.$$

L'ensemble des positions gagnantes d'ordre i pour les blancs peut donc s'écrire :

$$G\,(I) = \bar{\Gamma}\, M_b.$$

L'ensemble des positions qui donnent le gain en deux coups, ou position gagnante d'ordre *2* pour les blancs, peut donc s'écrire :

$$G\,(2) = G\,(I) \cup \bar{\Gamma}\, \overset{+}{\Gamma}\, G\,(I).$$

Cette méthode consiste, en somme, à supposer le problème résolu on se donne les positions gagnantes possibles, puis celles qui, compte tenu des règles, pouvaient les précéder, et on remonte en arrière.
En général, l'ensemble des positions qui amènent à la victoire en *n* coups peut se définir par l'expression :

$$G\,(n) = G\,(n\text{-}1) \cup \bar{\Gamma}\, \overset{+}{\Gamma}\, G\,(n\text{-}1).$$

ou plus symboliquement encore :

$$G\,(n) = (I \cup \bar{\Gamma}\, \overset{+}{\Gamma})\, G\,(n\text{-}1).$$

Par induction, on voit donc que l'ensemble des positions qui donnent le gain en *n* coups est :

$$G\,(n) = (I \cup \bar{\Gamma}\, \overset{+}{\Gamma})^{(n-1)}\, \bar{\Gamma}\, M_b.$$

Si l'on suppose que le jeu a une durée limitée, l'ensemble des positions gagnantes sera la réunion de tous les $G\,(n)$ (pour $n = 1, 2, \ldots$).
(En réalité la formule correcte pour l'ensemble des positions gagnantes est la réunion de tous les G (α) pour α = nombre ordinal transfini,

mais nous n'entrerons pas ici dans une analyse qui deviendrait vite trop abstraite.)
De cette formule qui, elle, est rigoureuse, et ne dépend d'aucune fonction f à déterminer intuitivement, on ne peut pas, malheureusement, déduire une construction *effective* de l'ensemble des positions gagnantes. par contre, dans le cas du jeu des échecs, on en déduit aisément des théorèmes, par exemple le théorème de Zermelo-Kalmar-von Neumann, qui s'énonce : « Ou bien il existe une stratégie gagnante pour les blancs, ou bien il existe une stratégie gagnante pour les noirs, ou bien chacun des deux joueurs a une stratégie qui lui garantit la nullité. »
Ce théorème n'est pas trivial comme il semble à première vue, car il suffit de changer très légèrement la règle du jeu des échecs pour que l'on ait la situation suivante : les blancs peuvent garantir la nullité par une stratégie mais les noirs ne peuvent rien garantir du tout à l'aide d'une stratégie.

ESSAI DE PROGRAMMATION D'UN AUTOMATE POUR SIMULER UN JOUEUR

Les théorèmes du type de ceux que nous venons de rencontrer sont des théorèmes d'existence. Avoir démontré qu'il existe une stratégie gagnante peut satisfaire le mathématicien - ou le philosophe -, mais sûrement pas le joueur. Cela ne satisfait pas non plus celui qui veut simuler un joueur à l'aide d'une machine à calculer, car sa préoccupation est d'amener l'automate à développer un jeu correct et, si possible, gagnant, mais le gain n'est pas vraiment essentiel.
La nécessité de préparer un programme réel qui permette à l'automate de jouer, et de jouer aussi bien que possible, permet de donner un éclairage nouveau à la distinction présentée ci-dessus entre théorie locale et théorie globale. Mais il nous faut d'abord préciser en quels termes nous allons décrire ce programme.
Soit B et N les camps en présence. Nommons b_i et n_j les coups joués respectivement par B et N ; une partie sera équivalente à une suite $b_1, n_1; b_2, n_2; \ldots x_p$.
Suivant que x_p est un b ou un n, B ou N aura gagné la partie en *p* coups.

Supposons que le joueur N soit remplacé par un automate, le programme d'ensemble pour le jeu sera le résultat d'une combinaison de programmes partiels que nous allons décrire brièvement ci-dessous.

L'organisation générale de ces programmes partiels est indiquée dans le schéma de la figure 19.

Pour décrire chaque sous-programme nous utiliserons un format standard donnant le nom du sous-programme et son abréviation, puis, sous la rubrique « algorithme », l'énoncé de ce qu'il doit faire ; sous la rubrique « documentation », l'ensemble des informations permanentes qu'il doit contenir ; sous la rubrique « données » les informa-

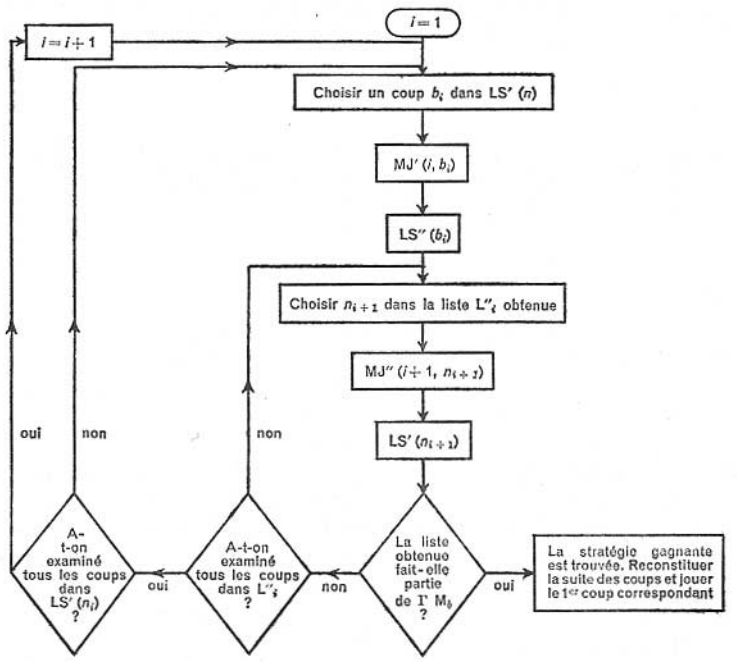

FIG. 19. - Schéma de principe du programme de simulation d'un joueur

tions qui se renouvellent à chaque exécution du programme ; et enfin sous la rubrique « résultat », ce qui sort de l'automate lorsque le programme est exécuté.

On peut ainsi définir les programmes suivants :

PJ
Programme général de jeu

algorithme : choix du coup n_i ;
documentation de base : règles du jeu + tactique ;
données : coup de l'adversaire b_i + situation du jeu au temps i - 1 ;
résultat : symbole représentant le coup n_i.

Le programme PJ se construira à l'aide des sous-programmes suivants :

MJ' (*i*, b_i)
Mise à jour de la situation au temps *i*

algorithme : transformation des tableaux de situation à la suite du coup b_i ;
documentation de base : définition des tableaux ;
données : T'_{i-1} et b_i ;
résultat : T' (1).

On construira symétriquement le programme MJ" *(i, n_i).*

LS" (b_i)
Recherche de l'ensemble des coups possibles par N après b_i

algorithme : examen successif des « pièces » du coup N ;
documentation de base : règles du jeu;
données : T'_i ;
résultat : L''_i .

On construira symétriquement le programme LS' (*n*).

On pourrait, en principe, à l'aide de ces sous-programmes, effectuer le choix du coup à jouer, au sein de la liste L''_i fournie par le couple MJ', LS" à chaque pas.

PREMIERE RENCONTRE AVEC LE « MUR DU COMBINATOIRE » ·

Supposons - ce qui n'est jamais vrai, mais qui nous permettra une estimation grossière des ordres de grandeur en présence - que chaque joueur ait, à chaque coup, le choix entre 10 coups possibles.

(1) Les T'_i et T''_j représentent l'état du jeu pour les jours B et N aux temps *i* et *j*.

Il faudra donc explorer au premier niveau io possibilités, au second niveau $10^2 = 100$ possibilités, etc.
Au ne niveau il faudra explorer 10^n possibilités.
Si la vitesse et la capacité de mémoire de l'automate sont illimitées, on peut de la sorte atteindre les positions gagnantes de N de toutes les façons possibles. Et, au fond, le théorème de Zermelo-von Neumann, dont nous avons présenté plus haut une variante, ne fait qu'exprimer cette possibilité.
D'après le théorème, on sait qu'à tout moment il est *théoriquement* possible de désigner le vainqueur. Par contre, on ne peut pas, en général, décrire explicitement les démarches qui entraînent cette victoire. C'est là une situation bien connue en mathématique où l'on distingue les définitions existentielles et les définitions constructivistes.
C'est ainsi que dans le cas des jeux très simples, comme le jeu de Nim ou le Tic-Tac-Dou, il est possible d'expliciter la méthode victorieuse (c'est l'algorithme du jeu), alors que pour un jeu complexe comme celui des échecs on ne possède pas d'algorithme. Dans ce cas, il faut développer des méthodes d'approche que l'on nomme «heuristiques », et qui, sans assurer la victoire, la rendent relativement probable.
Ce qui distingue au premier coup d'oeil un jeu comme le Nim (jeu rendu célèbre par un épisode du film *L'année dernière à Marienbad),* des dames ou des échecs, c'est évidemment le nombre des «pièces » en présence. Par la même occasion le nombre maximum de coups possibles est diminué et ceci de façon exponentielle.
Cette caractéristique de tels problèmes, c'est-à-dire cette dépendance vis-à-vis de fonctions de type exponentiel, conduisant, lors de certaines évaluations, à des chiffres astronomiques, c'est celle de tous les problèmes *combinatoires.* Le problème des jeux est l'un d'entre eux.
Une première conclusion se dégage cependant de ce qui précède même pour la démonstration de théorèmes généraux relatifs aux jeux, il n'est pas inutile d'utiliser la notion d'automate, en exploitant évidemment un modèle purement théorique d'automate, dont la capacité de mémoire est illimitée et dont la vitesse peut également être rendue aussi grande que l'on veut.
Mais, bien entendu, pour tous les jeux présentant quelque intérêt, il n'existe pas d'automate répondant à de telles exigences. Il faut donc, pour sélectionner le meilleur coup, faire appel à des critères moins

évidents et moins généraux. Chaque jeu pose ainsi un problème nouveau, une fonction *f* nouvelle à choisir.
Cette fonction *f* permet, à chaque fois, de remplacer l'examen de tous les coups suivants possibles par celui d'un seul coup (ou en tout cas d'un très petit nombre de coups). Dans ces conditions, l'examen de toutes les ramifications est considérablement réduit.
Dans certains cas la réduction est telle (ou encore le pouvoir *anti-combinatoire* de la fonction de préférence est tel) que l'examen de l'arborescence peut être effectivement accompli par un automate.
C'est ce qu'on verra mieux en examinant en détail un jeu particulier, ou plutôt une classe de jeux voisins les uns des autres. Nous pourrons alors détailler les programmes MJ et LS décrits plus haut, et définir des programmes de sélection plus fins. Pour un cas particulier, la programmation a été réellement effectuée et l'expérience conduite sur machine. Elle semble instructive et permet de dégager des perspectives générales qui s'offrent au développement de la simulation des jeux par les machines.

ÉTUDE D'UN EXEMPLE

Les jeux que nous utiliserons se rangent dans la classe de ce que nous proposons de nommer « les jeux de grille ».
Chaque jeu appartenant à la classe est caractérisé par deux nombres entiers a et b. Chaque jeu particulier sera appelé « le jeu de grille *(a, b) »* où a et b sont fixés. Le premier entier a détermine les dimensions de la grille (il s'agit donc d'une grille carrée à a lignes et à a colonnes). Chaque joueur dispose à son tour un pion à son emblème au croisement d'une ligne et d'une colonne. Le diagramme est ici l'image de la grille et des pions disposés à ses nceuds. Le vainqueur est celui qui réussit à disposer b de ses propres pions en alignement.
On doit avoir évidemment $a \leq b$.
Si $a = b = 3$, il s'agit du Tic-Tac-Dou.
Si $a = 19$, $b = 5$, il s'agit du Go-Bang.
Si a est très grand, $b = 5$, il s'agit du « Morpion » bien connu des lycéens.

Remarquons d'abord que si $a = b = 2$, le jeu n'a aucun intérêt car, quelle que soit la tactique adoptée, le premier joueur gagne.

Pour $a = b = 3$, une règle simple permet d'obtenir automatiquement le succès, ou au moins le jeu nul pour le premier joueur.

Lorsque $b > 3$, il faut que a devienne beaucoup plus grand que b pour que le jeu présente un intérêt. La raison en sera analysée en détail au chapitre VI relatif à la notion de « complexité ».

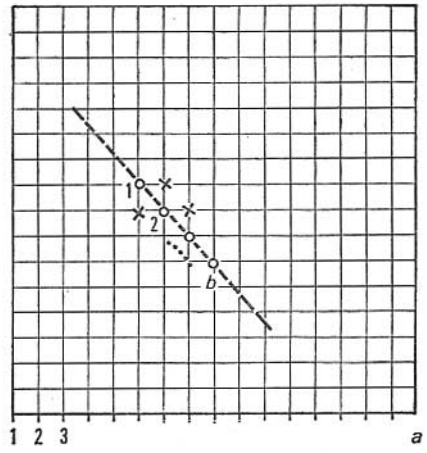

FIG. 20. — Exemple d'un jeu de grille de paramètres a et b

Restant pour le moment au niveau du jeu, nous nous limiterons à étudier la programmation d'un automate qui, placé devant une situation donnée de la grille, doit choisir le coup suivant. Il n'y aurait évidemment aucune difficulté de principe à traiter le problème d'un jeu complet.

Nous décomposerons ce programme en un ensemble de programmes simples, aux fonctions bien déterminées, ensemble dont l'articulation sera précisée par la suite.

Nous allons surtout insister sur le traitement de ce que nous avons appelé un peu plus haut les données, c'est-à-dire les informations relatives à la situation des pions déposés par les joueurs sur la grille.

Le programme correspondant est le suivant :

LG Lecture de la grille
algorithme : une liste de a^2 registres est associée à la grille ; chaque registre reçoit le contenu 0, 1 ou 2, suivant que le noeud de la grille est inoccupé, a reçu un pion du joueur A ou du joueur B ;
documentation de base : état de la grille au coup précédent;
données : deux coups joués par l'adversaire ;
résultat : grille mise à jour.

Ce programme donne en quelque sorte à l'automate la perception de la grille. Il faut maintenant pouvoir introduire quelques « concepts » rudimentaires. Ceci se fait au niveau du programme.

Mais pour que le programme soit efficace, il est important de choisir avec soin le format sous lequel les données y sont introduites.

Dans l'exemple actuel, la «vision» que l'automate peut avoir du jeu se situe sur un plan *statique :* la vision du diagramme lui-même, et sur un plan *dynamnique,* la vision des combinaisons déjà réalisées (2, 3 pions déjà alignés, etc.).

a) Nous appellerons TJ un tableau de 361 cases qui sont des « mots » de 36 bits, c'est-à-dire des suites de 36 chiffres formées de o et de i uniquement.

2 bits définissent l'occupation de la case

- 00 : la case est libre ;
- 01 : la case est occupée par un pion de la machine;
- 10 : la case est occupée par un pion de l'adversaire ;
- 11 : une erreur a été commise dans la construction du tableau.

Les 34 bits restants comprennent

- 17 bits exprimant la grandeur de la fonction d'évaluation relative à la machine en ce point ;
- 17 bits exprimant la grandeur de la fonction d'évaluation relative à l'adversaire en ce point.

b) Nous appellerons *n*-uplet un ensemble de n points alignés et occupés par un même joueur avec la condition que l'adversaire ne possède aucun point dans ce domaine ainsi défini, et que, d'autre

part, ce domaine n'excède pas 5 cases. Il peut y avoir plusieurs types de *n*-uplets et ils peuvent posséder différentes orientations. On a les types suivants :

singulets : ce sont les pions isolés déposés par chaque joueur ;
doublets : ce sont les couples de pions déposés de la façon suivante :

X X . . . X . X . . X . . X .
X . . . X

triplets :

X X X . . X X . X . X X . . X
X . X . X

quadruplets :

X X X X . X X X . X X X . X X

quintuplets :

X X X X X

... et les orientations suivantes :

H : horizontal ;
D : diagonale montant vers la droite ;
V : vertical ;
G : diagonale montant vers la gauche.

Compte tenu de ces conventions, il faut s'efforcer *d'organiser* les données de façon à encombrer le moins possible la mémoire de l'automate tout en conservant un accès aussi rapide que possible.

Il existe une organisation de ce genre qui est particulièrement commode, c'est l'organisation en *listes.* Les listes sont construites de la façon suivante : chaque information est divisée en deux parties la première est un nom (ou un titre), la seconde une adresse (le pointeur) ou une donnée (ici une position dans le tableau TJ). Lorsqu'on a ensemble de telles informations on peut en déduire une représentation graphique naturelle en reliant par une flèche la partie « adresse » d'une information à l'information contenue dans cette adresse. Sans entrer dans les détails de cette organisation nous en présentons, dans la figure 21, deux applications particulières :

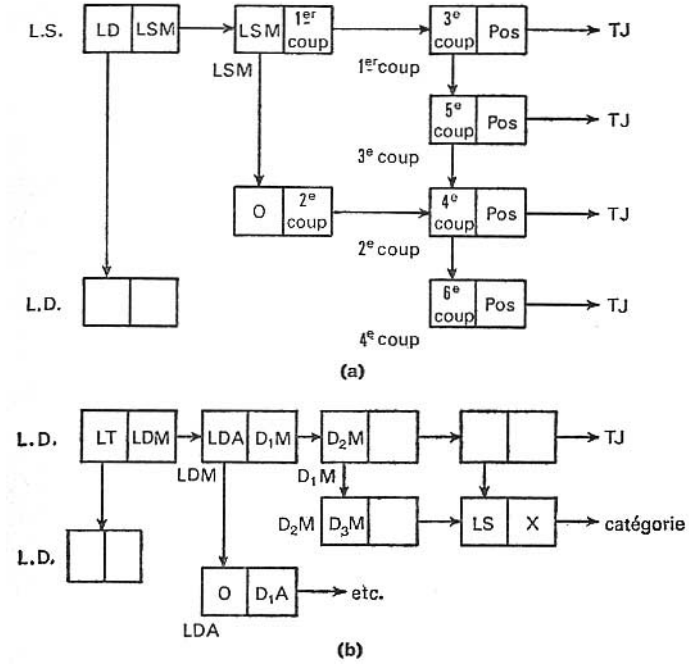

FIG. 21. - Modèle d'organisation en « liste » des singulets et des doublets

L'indication « catégorie » est une information de 9 bits qui spécifie d'une part le *type* topologique du doublet, d'autre part l'*orientation* de ce doublet. 5 bits sont utilisés pour le type :

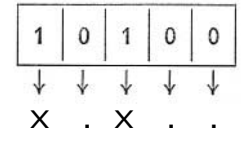

FIG. 22. — Correspondance entre l'arrangement topologique d'un *n*-uplet et le symbole binaire de son « type »

Enfin 2 bits expriment l'orientation en *n*-uplet :

- 00 : horizontal (H) ;
- 01 : diagonale montant vers la droite (DD) ;
- 10 : vertical (V) ;
- 11 : diagonale montant vers la gauche (DG).

Enfin, 2 bits dénotent la fonction de fermeture, c'est-à-dire l'occupation des positions voisines du doublet par l'adversaire :

- 00 : libre aux deux extrémités ;
- 01 : fermé à droite ;
- 10 : fermé à gauche ;
- 11 : fermé aux deux extrémités.

Les autres listes (des triplets, quadruplets, quintuplets) sont évidemment structurées de la même façon.

c) Calcul des listes. — Chaque fois qu'un joueur annonce le coup qu'il joue, la machine doit mettre à jour le tableau et les listes. Pour cela elle dispose d'un tableau spécial TOE des opérations élémentaires nécessaires au balayage d'une file de 5 points alignés. Ces instructions sont les suivantes

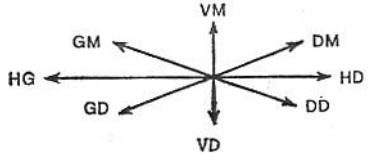

Fig. 23. - Les divers déplacements susceptibles d'assurer le balayage de la grille.

— o(1)	HD	ajouter	+ 1	à la 1re	coordonnée et	0 à la deuxième
— 0(5)	HG	—	— 1	—	0	—
— 0(2)	DM	—	+ 1	—	+ 1	—
— 0(6)	GD	—	— 1	—	– 1	—
— 0(3)	VM	—	0	—	+ 1	—
— 0(7)	VD	—	0	—	— 1	—
— 0(4)	GM	—	— 1	—	+ 1	—
— 0(8)	DD	—	+ 1	—	– 1	—

Ceci étant posé, le programme se construira comme suit :

1) on inscrit dans le tableau TJ à l'endroit où le coup vient d'être joué l'information relative à la machine du nouvel occupant ;
2) on ajoute à la liste LS l'adresse de cette case dans TJ ;
3) on complète les listes LD, LT, etc.

Nous allons maintenant pouvoir aborder, sur cet exemple, la recherche et le calcul d'une fonction d'évaluation.

a) Principe. — Dans un jeu relativement simple comme le Go-Bang, il est possible de faire jouer à la fonction d'évaluation un double rôle

- un rôle *stratégique* en déterminant les positions dominantes (au sens de la théorie des jeux) ;
- un rôle *tactique* en contribuant à la définition d'un minimax, lorsque aucune position dominante ne peut être atteinte.

C'est pourquoi les fonctions que nous avons considérées sont basées sur la combinaison de plusieurs informations :

- un tableau de valeurs associé à chacune des configurations de 1, 2, 3, 4, ou 5 pions identiques sur un même alignement de 5 cases ;
- un ou plusieurs filtres qui limitent le développement de l'arborescence aux branches conduisant à certaines de ces configurations;
- une règle d'arrêt qui limite la profondeur de l'arborescence à un nombre de coups donnés ;
- une formule qui donne l'évaluation pour la position en fonction du nombre des configurations obtenues aux extrémités de l'arborescence, de leurs valeurs et de la longueur de la branche qui la relie au sommet.

On a évidemment une certaine souplesse dans la construction de la fonction d'évaluation. Il convient de noter cependant que si l'on utilise cette fonction pour l'élaboration d'une stratégie de minimax il ne serait pas raisonnable que la fonction elle-même réclame, pour être calculée, l'examen d'une arborescence.

C'est pourquoi les deux premières fonctions que nous avons essayées correspondent respectivement à l'examen d'un ou de deux coups en avant. Une fois la valuation obtenue, il y a alors encore un sens à l'utiliser pour une exploration en profondeur et une stratégie minimax.

b) Description de notre première fonction. — Supposons que le jeu soit arrivé à un certain stade, on va alors construire les deux fonctions

$$f_{\text{ami}} \text{ et } f_{\text{ennemi}}$$

Pour les points où l'on a déjà joué

$$f_{\text{ami}} = f_{\text{ennemi}} = 0$$

Pour les autres, on calculera le poids des configurations nouvelles créées respectivement par le dépôt d'un pion ami ou ennemi (fig. 24).

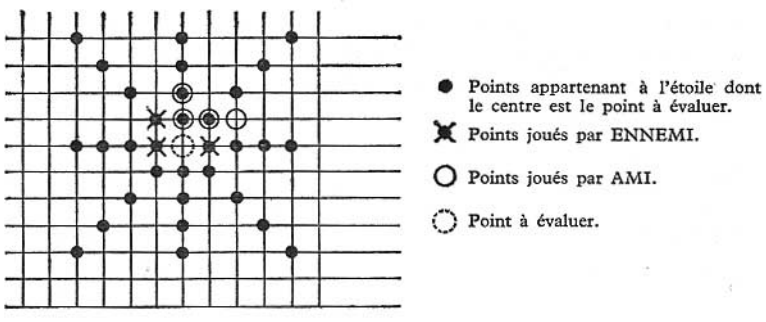

FIG. 24. — Utilisation d'une "étoile virtuelle" lors de l'évaluation d'une position

Le poids d'une configuration doit être tel que 5 pions alignés donnent la victoire et que l'absence de pions entraîne l'absence de valuation.

Une solution simple consiste à attribuer à un n-uplet $(n < 5)$ la valeur

$$k \frac{n}{5-n}$$

Lors de l'implantation du programme sur une calculatrice, les mémoires contiennent

– un tableau du jeu, entouré de ≠ dans lequel s'inscriront tous les coups ;

- une table des évaluations contenant
  a) le nom décodé de chaque case (exemple : la case B 07 s'écrit 427077 ; B = 42, 0 = 70, 7 = 77)
  b) l'adresse correspondante du tableau du jeu ;
  c) l'évaluation AMI pour la case correspondante ;
  d) l'évaluation ENNEMI pour la case correspondante ;
  au départ les évaluations sont toutes nulles ; de même dès qu'une case est occupée, ces évaluations deviennent nulles ;
- une table d'équivalence entre la situation du tableau et l'évaluation : par exemple si 5 cases consécutives d'une radiale ou d'une diagonale contiennent 10110, le tableau d'équivalence donne l'évaluation correspondante ;
- une zone de comparaison dans laquelle s'inscrit le contenu des 9 cases d'une diagonale, la position centrale étant celle de la case dont on calcule l'évaluation ;
- une table des coups joués, contenant, dans l'ordre, l'adresse des cases qui ont été occupées successivement.

Après chaque coup, seules les cases vides se trouvant sur l'étoile d'un diamètre de 9 cases (le centre de l'étoile étant la case du coup joué) seront réévaluées.

Pour chaque case à réévaluer, l'évaluation est d'abord remise à o. Ensuite chaque diagonale de 9 cases est mise dans la zone de comparaison.

Calcul de l'évaluation AMI. - Dans cette zone de comparaison, on examine chacun des 4 groupes de 5 cases successives. Si une de ces 5 cases contient un ≠ ou un coup ENNEMI, son évaluation est nulle. Sinon le dessin est comparé au tableau d'équivalence et on obtient ainsi l'évaluation pour ce groupe. Cette évaluation est ajoutée (!!!) à l'évaluation AMI de la case.

Calcul de l'évaluation ENNEMI. - Dans la zone de comparaison i est remplacé par i et réciproquement. On calcule alors comme pour l'évaluation AMI.

Lorsque des coups sont annulés, la machine fait successivement et dans l'ordre inverse des coups précédemment joués

1) remise à zéro de la case correspondante dans le tableau du jeu;
2) réévaluation de toutes les cases o se trouvant sur l'étoile.

Il peut être intéressant de donner quelques résultats expérimentaux relatifs à ce cas particulier.

Nous disposions tout d'abord d'un programme de calcul de fonction d'évaluation sur I.B.M.-I620; il s'agit d'un programme comportant 8oo instructions. Le calcul d'une nouvelle valuation, après qu'un coup vient d'être joué, prend en moyenne 50 secondes.

Nous avons tout d'abord effectué une série d'expériences semi-automatisées dans lesquelles la machine donne les deux évaluations AMI et ENNEMI de toutes les cases non occupées. Un opérateur humain utilise ces évaluations selon une tactique déterminée (joueur maximum AMI, c'est-à-dire «attaque»; joueur maximum ENNEMI, c'est-à-dire « défense », ou maximum de la somme des deux évaluations, c'est-à-dire « efficacité »). L'adversaire est un être humain qui n'a pas accès au listing de la machine. Les expériences ont montré que le programme succombait contre un joueur humain bien entraîné, mais après une défense honorable (de 15 à 20 coups). Par contre, un joueur humain débutant ou distrait est presque toujours battu.

Le programme a alors été réécrit pour la machine I.B.M.-7090, mais ici, pour épargner du temps machine, les deux adversaires sont des programmes. La tactique étant la suivante

Si le maximum de la fonction d'évaluation correspondant à l'attaque est supérieur à un certain seuil, on joue ce coup, sinon on joue le coup de défense. Si le coup de défense lui-même n'atteint pas ce seuil, on joue le coup d'efficacité. Il est alors facile d'opposer deux programmes identiques, sauf en ce qui concerne la valeur du seuil. Nous l'avons divisé en trois catégories : BAS, MOYEN, ÉLEVÉ et joué 9 parties qui ont donné les résultats suivants (fig. 25).

Ces résultats indiquent bien que la recherche en est à un stade transitoire : il existe des programmes qui jouent assez bien, mais qui ne gagnent pas à tout coup. Cependant, il est probable que la construction d'un tel programme «champion du monde » ne rencontrerait pas de difficultés insurmontables.

De même pour le jeu de dames (ou sa version anglo-saxonne dénommée *checkers) ;* des programmes ont été écrits qui donnent des performances excellentes.

Cet exemple est particulièrement intéressant, car en plus d'un programme de jeu proprement dit, comprenant un programme de lecture

1er joueur \ 2e joueur	B	M	E
B	nulle	nulle	B 73e coup
M	M 71e coup	M (2) 172e coup	M 128e coup
E	B 58e coup	M 82e coup	E (1) 103e coup

FIG. 25. — Quelques résultats expérimentaux relatifs à la simulation du jeu de Go-Bang

du damier analogue à notre LG ci-dessus et un programme d'évaluation de la position tenant compte du bilan des pions pris par les adversaires en présence, de l'avantage en position, etc., le programme a été muni de possibilités *d'apprentissage* (ce programme a été décrit dans plusieurs publications de A. Samuel, dont la plus importante est reproduite dans l'ouvrage édité par Feigenbaum et Feldman et cité dans notre bibliographie).

On se souvient qu'au début de ce chapitre nous avons écarté de notre enquête la simulation du phénomène psychologique de l'apprentissage. Mais l'apprentissage de l'automate simulant le joueur, tel qu'il est pratiqué dans les expériences de Samuel, n'a aucun rapport avec celui que les psychologues s'efforcent d'analyser chez l'homme. Il s'agit simplement de l'utilisation de la capacité de mémoire de l'automate pour le stockage des parties jouées et le décompte des

performances de l'automate. La fonction de préférence à laquelle nous avons fait allusion plus haut contient certains paramètres numériques qui peuvent être changés et pour lesquels on peut prévoir une procédure d'adaptation en fonction des résultats acquis.

Par contre, dans le domaine plus spectaculaire des échecs, malgré des efforts considérables, de nombreuses équipes (Shannon, Turing, Ulam, Bernstein, Newell-Shaw-Simon, Shura-Bura, Euwe), aucun résultat vraiment significatif n'a été atteint, du point de vue du jeu lui-même (1). Par contre, les difficultés rencontrées, les moyens mis en oeuvre pour les attaquer, sont une contribution fondamentale au développement de l'intelligence artificielle. Nous aurons l'occasion d'y revenir au chapitre VI (2).

(1) Tout récemment le soviétique Adelson-Vilsky a développé un programme efficace qui a battu, après une partie honnête, le programme américain dû à l'équipe de J. McCarthy (cf. l'article « Automates » dans le récent ouvrage de F. La LIONNAIS et E. MAGOT, *Dictionnaire des échecs,* Paris, Presses Universitaires de France, 1967, P. 25).

(2) Depuis que ces lignes ont été écrites nous avons abordé la simulation de jeux nouveaux sur machine IBM 360/50. Nous avons été amenés d'ailleurs *à inventer* des jeux spécialement adaptés à notre enquête. (Cf. notre article : The gaine of FIB an experiment in artificial intelligence, en collaboration avec H VAN HEDEL, soumis pour publication).

CHAPITRE IV

La raison

Dans le chapitre II, nous avons montré que la nécessité de codifier les informations émises par le monde extérieur, nécessité commune aux être organisés et aux automates, permettait de formuler les problèmes de l'intelligence dans un langage relativement simple et bien défini, et surtout protégé contre le danger de spéculations psychologiques ou plutôt pseudo-psychologiques.

Au cours du chapitre III, nous avons illustré cette thèse en analysant quelques jeux, dont la simulation a été tentée sur calculatrice (et en particulier l'un d'entre eux, le jeu de Go-Bang). Nous avons observé alors que la simulation des jeux nécessitait tout d'abord une double codification : celle des règles du jeu, et celle des états successifs du jeu (position des pions sur un damier, par exemple) ; puis le choix d'une stratégie et sa mise en oeuvre sous forme de programme.

Dès ce niveau, il est donc clair qu'on a dépassé le stade d'une simple succession de codifications.

En fait, dès l'exemple du chapitre II, figure 7, on se trouvait en présence de deux attitudes possibles, pour construire une traduction automatique d'un code dans l'autre.

C'est ainsi que la transformation de l'alphabet romain dans le code octal BCD pourrait s'effectuer par le simple programme