M.Tech (Computer Science) Dissertation Series

GAME PLAYING USING EXPERT SYSTEMS

a dissertation submitted in partial fulfilment of the requirements for the M.Tech (Computer Science) degree of the Indian Statistical Institute

by

Vuppala Sreenivas

under the supervision of

Mr. Debashis Sarkar

INDIAN STATISTICAL INSTITUTE
203 B.T. ROAD
CALCUTTA - 700 035

ACKNOWLEDGEMENTS

I take this oppurtunity to thank my supervisor Mr. Debashis Sarkar, for offering me this project, and giving me constant encouragement during the work.

I also thank all my friends for playing the game and giving valuable suggestions for improvements in the game.

CONTENTS

1. INTRODUCTION

Expert systems are computer programs that solve problems in a way that would be considered *intelligent* if done by a human being. These programs are different from ordinary conventional programs. Conventional programs work on database according to given procedure and produce accurate results whereas Expert systems works on knowledge base producing results and even make mistakes as an expert do. Expert systems are mainly needed because of lack of human experts and their consistency over human expert . Expert systems can be developed for such problems where some knowledge regarding the problem is available and the solution can be found by some *thinking* over the available knowledge. The Science of developing expert systems is called *Knowledge Engineering* and the person involved in such task is called *Knowledge Engineer*. Expert systems are helpful in each and every field. Game playing is one such application.

The game selected is called *OTHELLO*. This contains a board which can be displayed on computer screen easily with minor modifications. The knowledge base includes mainly the board configuration i.e., the colour of each cell of the board. This can be easily represented by using predicate logic. The tool (or the programming language) which supports predicate logic and the required screen facilities is *Prolog*. Hence the language is selected for building, the Expert system.

2. DEFINITIONS

EXPERT SYSTEMS :- Systems that solve problems based on some available knowledge base as a human expert do.

Expert system is called a system rather than just a program because it contains both a problem solving component and a supporting component.

KNOWLEDGE ENGINEERING :- The process of building an expert system is called knowledge engineering.

KNOWLEDGE ENGINEER :- Person involved in building expert systems. The knowledge engineer extracts from the human experts their procedures, strategies, and rules of thumb for problem solving, and builds this knowledge into the expert system. The result is a computer program that solves the problems in much the same manner as the human experts.

DOMAIN EXPERT :- The person who, because of training and experience, is able to do the things the rest of human beings cannot. Experts are not only proficient but also smooth and efficient in the actions they take. But they generally have little knowledge of computers.

EXPERT SYSTEM BUILDING TOOL :- The Expert system building tool is the programming lanugage used by the knowledge Engineer or program to build the expert system. The term tool usually refers both to the programing language and to the support environment used to build the expert system.

<u>SUPPORT</u> <u>ENVIRONMENT</u> :- These are the facilities associated with an expert system building tool that keep the user interact with the expert system. These may include sophisticated debugging aids, friendly editing programs and advanced graphic devices.

The collection of domain knowledge is caled knowledge base , while the general problem solving knowledge is called the *inference engine*. A program with knowledge organised in this way is called a knowledge based system.

<u>FACT</u> :- It is a part of the knowledge base such as *"cell <5,6> has white colour"*. It will be represented in predicate logic as *color(5,6,white)* or *white(5,6)* etc.

<u>RULE</u> :- A list of predicates assigned to a clause.

<u>UNIFICATION</u> :- It is a equivalent to assignment operation in conventional programming.

<u>CLAUSE</u> :- A clause is a *fact* or a *rule*.

<u>PREDICATE</u> :- A set of clauses with same name representing similar facts.

<u>PREDICATE</u> <u>LOGIC</u> :- One way of representing knowledge. Knowledge will be represented in the form of *clauses*.

3. EXPERT SYSTEMS & THEIR DEVELOPMENT

3.1 FEATURES OF EXPERT SYSTEMS

The heart of an expert system is the powerful corpus of knowledge that accumulates during Expert System building. The knowledge is explicit and organied to simplify decision making. The accumulation and codification of knowledge is one of the most important aspects of an expert system.

The most useful feature of an expert system is the high-level *expertise* it provides to aid in problem solving. This expertise can represent the best thinking of the top experts in the field, leading to problem solutions that are imaginative, accurate and efficient. It is the high level expertise together with skill at applying it that makes the system cost-effective, able to earn its own way in the commercial market place.

The corpus of knowledge that defines the proficiency of an expert system can also provide an additional feature, an institutional memory. If the knowledge base was developed through interactions with key personnel, it represents the current policy of that group. This compilation of knowledge is a permanent record of best strategies and methods used by the staff.

A final feature of an Expert System is its ability to provide a training facility for key personnel and important staff members. The expert systems can be designed to provide such training, since they contain the necessary knowledge.

3.2 ADVANTAGES OF EXPERT SYSTEMS OVER HUMAN EXPERTS

1. One advantage of artificial expertise is its performance. Human expertise can quickly fade, regardless of whether it involves mental or physical activity. An expert must constantly practice and rehearse to maintain proficiency in some problem area. However, artificial expertise once acquired will stay forever.

2. Another advantage of artificial expertise is the ease with which it can be transfered or reproduced. Transfering knowledge from one human to another is laborious, lengthy and expensive process called education. Transfering artificial expertise is the trivial process of copying a program or data file.

3. Documenting a human expertise is extreemly difficult and time consuming. Documenting artificial expertise is relatively easy. There is a straight forward mapping between the way in which the expertise is represented in the system and the natural language description of that representation.

4. Artificial expertise produces more consistent, reproducible results than does human expertise. A human expert may make different decisions in identical situations because of emotional factors. An expert system is not susceptible to these distractions.

5. A final advantage of artificial expertise is its low cost. Human experts are very scarse and hence are very expensive.

3.3 DISADVANTAGES OF EXPERT SYSTEMS

1. People are much more creative and innovative than even the smartest programs. A human expert can reorganise information and use it to synthasize new knowledge, while an expert system tends to behave in a somewhat *uninspired*, routine manner. Human experts handle unexpected events by using imaginative and novel approaches to problem solving. Programs have had little success doing this.

2. Human experts adapt to changing conditions. They adjust their strategies to conform to new situations. Expert systems are not particularly adept at *learning* new concepts or rules.

3. Human experts can make direct use of complex *sensory* input whether it be visual, auditory, tactile or olfactory. But Expert systems manipulate symbols that represent ideas and concepts. So sensory data must be transformed into symbols that can be understood by the system.

4. Human experts can look at the big picture, examine all aspects of problem and see how they relate to the central issue. Expert systems, on the other hand, tend to focus on the problem itself ignoring issues relevant to, but separate from the basic problem.

5. Human experts and nonexperts alike have commonsense knowledge. Because of the enormous quantity of commonsense knowledge, there is no easy way to build it into an intelligent program, particularly a specialist like an expert system.

3.4 IMPORTANT ROLES IN BUILDING EXPERT SYSTEM

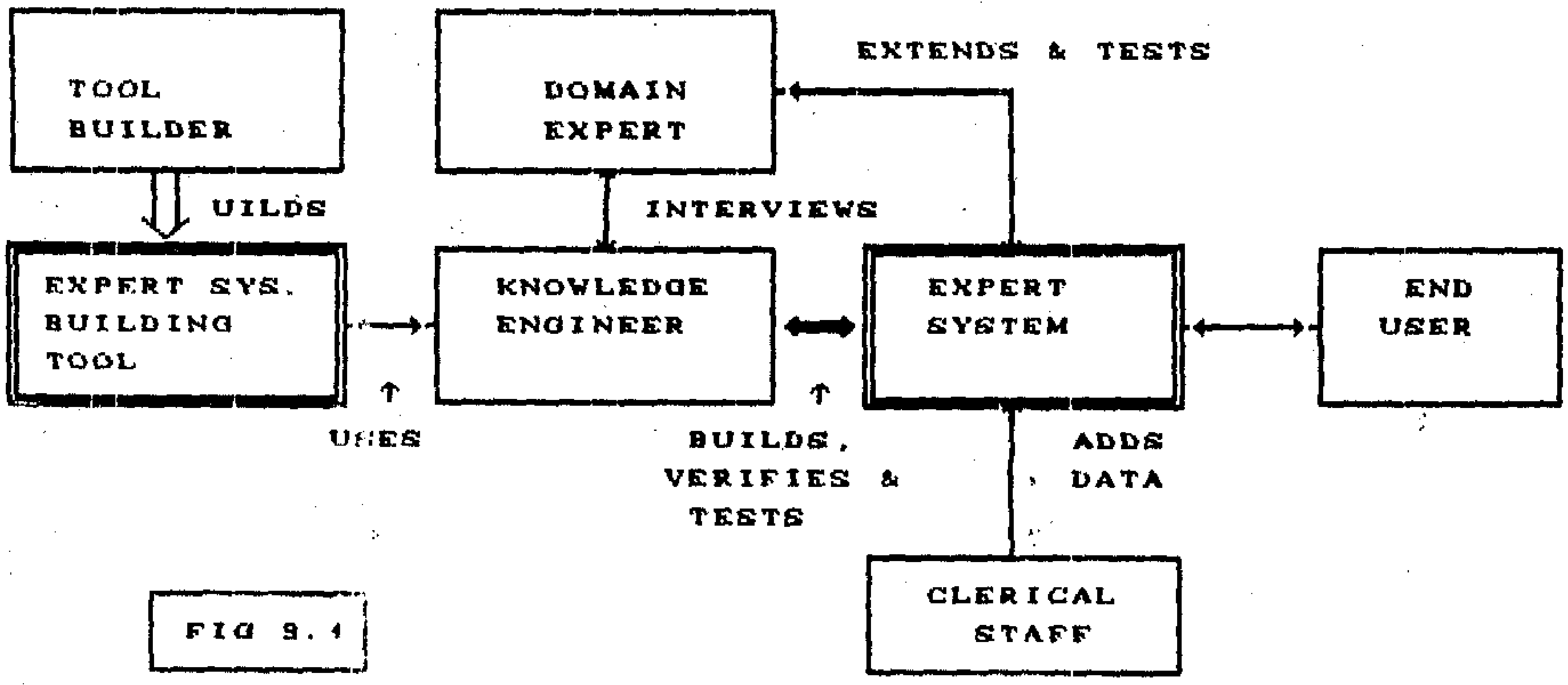


FIG 3.4

The main personnel and systems involved in building an expert system are , *the domain or area expert*, *the knowledge engineer* and *the expert system building tool*.

The domain or area expert is an articulate, knowledgeable person with a reputation and for producing good solutions to problems in a particular field. The expert uses tricks and shortcuts to make the search for a solution more efficient, and the expert system models these problem solving strategies.

The knowledge engineer is a person with a background in computer science who knows how to build expert systems. The knowledge engineer interviews the experts, organize the knowledge decides how it should be represented in the expert system, and may help programers to write the code.

The expert system building tool is the programming language used by the knowledge engineer or programer to build the expert system. These tools differ from conventional programming languages in that they provide convenient ways to represent complex, high level concepts. The term tool usually refers both to the programming language and to the supporting environment useed to build the expert system.

3.5 DIFFERENCES BETWEEN CONVENTIONAL PROGRAMS AND EXPERT SYSTEMS

Conventional program	Expert system
1. Representations and use of data	1. Representation and use of knowledge.
2. Algorithmic	2. Heuristic
3. Repetitive process	3. Inferential process
4. Effective manipulation of large databases	4. Effective manipulation of large knowledgebases
5. Accurate	5. Might make mistakes

TABLE 3.1

3.6 PHASES IN BUILDING AN EXPERT SYSTEM

Expert system development can be viewed as five highly interdependent and overlapping phases : identification, conceptu-alization, formalization, implementation and testing (fig 3.2)

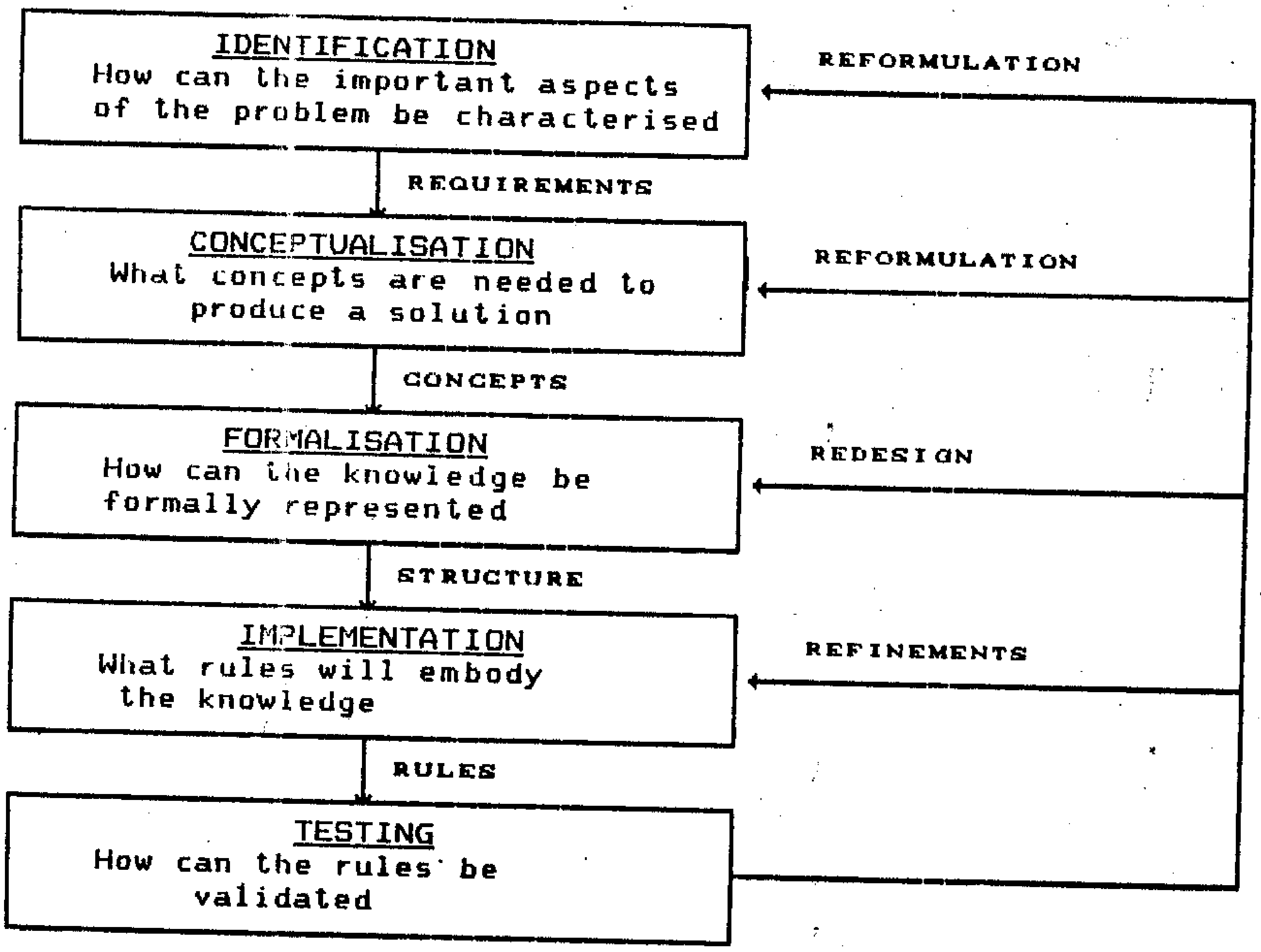

IDENTIFICATION
How can the important aspects of the problem be characterised
REQUIREMENTS
CONCEPTUALISATION
What concepts are needed to produce a solution
CONCEPTS
FORMALISATION
How can the knowledge be formally represented
STRUCTURE
IMPLEMENTATION
What rules will embody the knowledge
RULES
TESTING
How can the rules be validated
REFORMULATION
REFORMULATION
REDESIGN
REFINEMENTS

FIG 3.2

During *identification* , the knowledg engineer and expert determine the important features of the problem. This includes identifying the problem itself (e.g. scope and type) , the participants in the development process (e.g., additional experts), the required resources (e.g., time and computing facilities) , and the goals or objectives of building the expert systems (e.g., improve performance or distribute scarse expertice). Of these activities, identifying the problem and its scope gives developers the most trouble. Often the problem first considered is too large or complex and must be scaled down to manageable size. The knowledge engineer may obtain a quick measure of this complexity by focussing on a small but interesting sub problem and implementing routines to solve it.

During *conceptualization* the knowledge engineer and expert decide what concepts, relations, and control mechanisms are needed to describe problem solving in the domain. Sub tasks, strategies and constraints related to the problem solving activity are also explored. At this time the issue of granularity is usually addressed .

Formalization involves expressing the key concepts and relations in some formal way usually within framework suggested by an expert system building language. Then the knowledge engineer should have some ideas about appropriate tools by the time formalization begins.

During *implementation* , the knowledge engineer turns the formalized knowledge into a working computer program. Constructing a program requires content, form and integration. The content comes from the domain knowledge made explicit during formalization, i.e., the data structures, inference rules, and control strategies necessary for problem solving. The form is specified by the language chosen for system development. Integration involves combining and reorganizing various pieces of knowledge to eliminate global mismatch between data structure and rule or control specifications. Implementation should proceed rapidly because one of the reasons for implementing the initial prototype is to check the effectiveness of the design decisions made during the earlier phases of development. This means that there is a high probability that the initial code will be revised or discarded during development.

Finally, *testing* involves evluating the performance and utility of the prototype program and revising it as necessary. The domain expert typically evaluates the prototype and helps the knowledge engineer to revise it.

3.7 FACTORS REQUIRED FOR DEVELOPING EXPERT SYSTEM

Expert system development is possible when

1. Task does not require common sense
2. Task requires only cognitive skills
3. Experts can articulate their methods
4. Genuine experts exists
5. Experts agree on solutions
6. Task is not too difficult
7. Task is not poorly understood

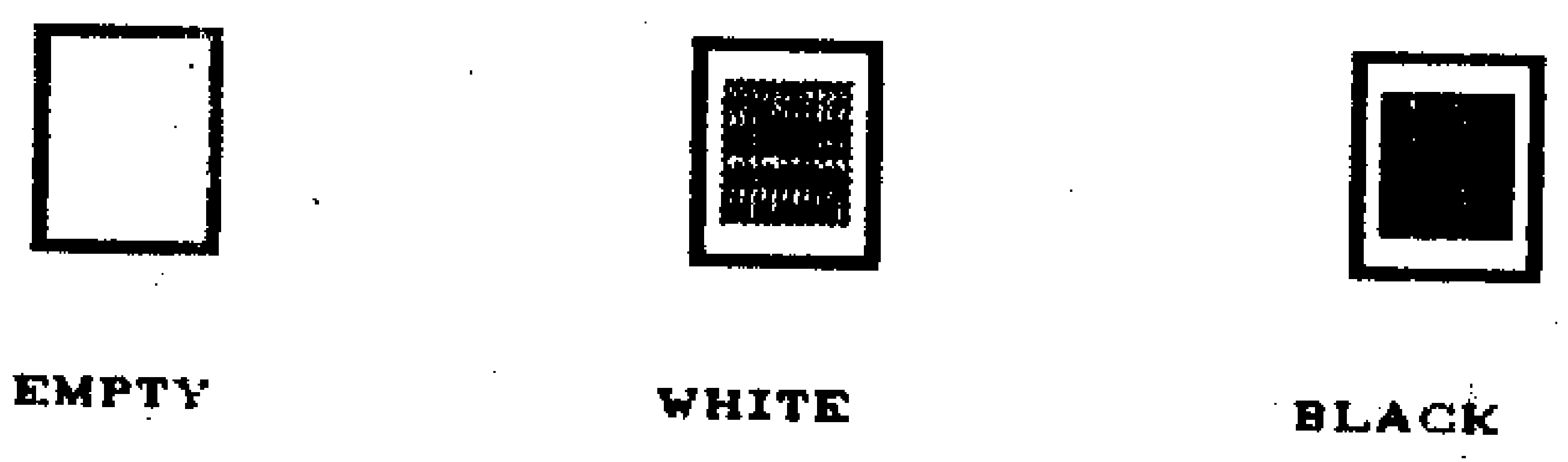
3.8 CHOOSING THE TOOL FOR BUILDING EXPERT SYSTEMS

Knowledge Engineer selects a particular tool for one or more of the following reasons.

1. He was already familiar with that tool.
2. The tool was the most efficient one available that ran on the developer hardware.
3. The tool was fully developed and then applications were found to test it.
4. The tool decreases the effort to be done by knowledge engineer.

4. THE GAME

OTHELLO, also called REVERSEE, is a board game to be played by two players. It consists of a sqrare board with 64 small sqrare cells arranged in 8 rows and 8 columns as in a CHESS board, but uniquely coloured. The coins, called COUNTERS or CHIPS, are also 64 in number and are circular. For the ease of programming, they are assumed to be of square shape. These counters are coloured with two colours on either side (white and black in general), the colours being the same for all the counters. Each counter is of size less than that of a cell on the board so that a counter can be placed in the cell without intersecting the other cells.For programming purpose, the counter is assumed to occupy the entire cell. The following diagrams shows how the cells appear when they are empty, occupied by white counter and black counter respectively. These are useful for understanding the illustrative diagrams.



A white counter is a counter which shows white colour up. A black counter shows black colour up. An empty cell is the cell that is not yet filled. A white cell is the cell containing a white counter and similar is the black cell.

The board before starting the game contains two counters of each colour at its center in a criss-cross manner such that each pair of counters of same colour are diagonally adjacent to each other.*(fig.1)* Each player will have to select a distinct colour (one of the two colours of the counter) before starting the game. Any player can start the game. With out loss of generality it is assumed that white plays first. The game goes on keeping new counters in the blank cells. A counter once placed can neither be taken back nor can be moved to a new cell. The only allowed operation on a placed counter is to turn it back (called *flipping*) to give the other colour. *Opposite* colour is the term used to refer to the other colour.

A counter can be placed in emptycell only if there exists atleast one counter on the board with the same colour as that of the counter being placed such that all the counters between that counter and the newly placed counter in any direction are all opposite coloured and does not include any blank cells in between them in the same direction. By any direction we mean one of the horizontal (left or right) , vertical (top or bottom) and diagonal (northeast, northwest, southeast or southwest) directions.

Placing a counter is called a *move* and a move satisfying the above condition is called a *valid move*. When a player plays a valid move, then all the counters in between the currently placed counter and already existing one of the counter, defined by the above condition are turned back (*flipped*) to give the colour of the currently placed counter, there by increasing the current players counters and decreasing the opponents. This flipping operation must be done in all the possible directions. *(fig.2)*

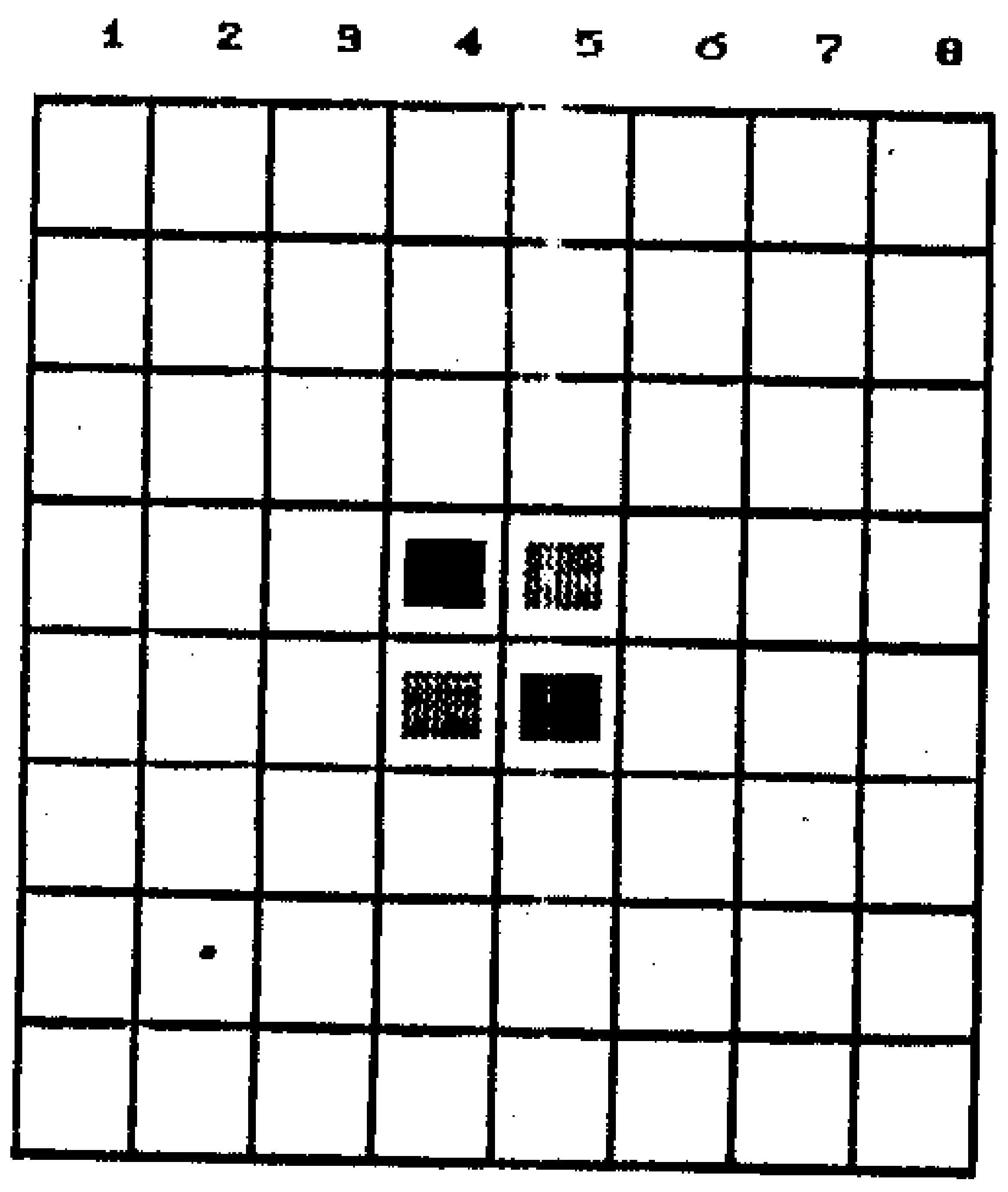


FIG 4.1

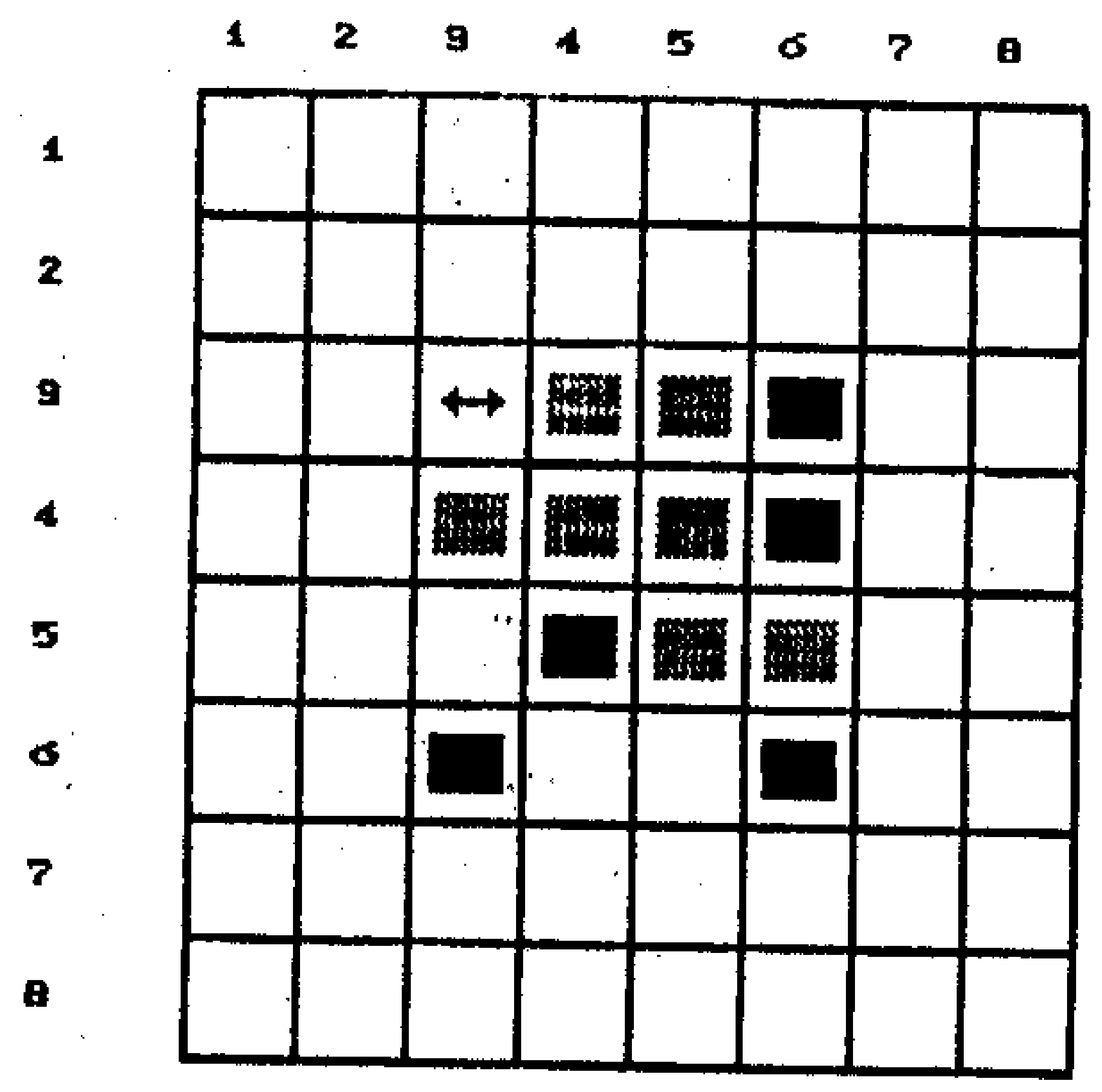


FIG 4.2

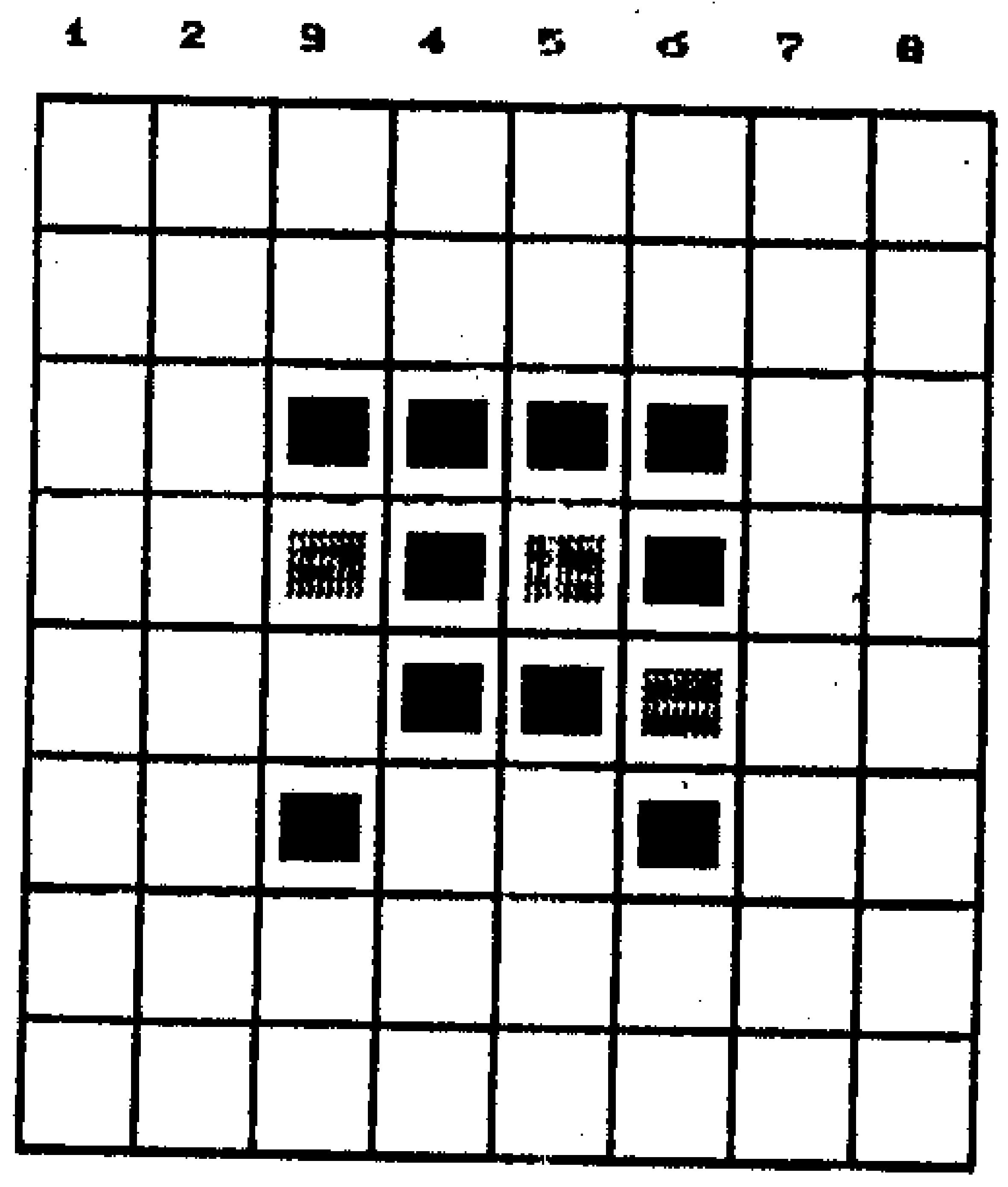


FIG 4.3

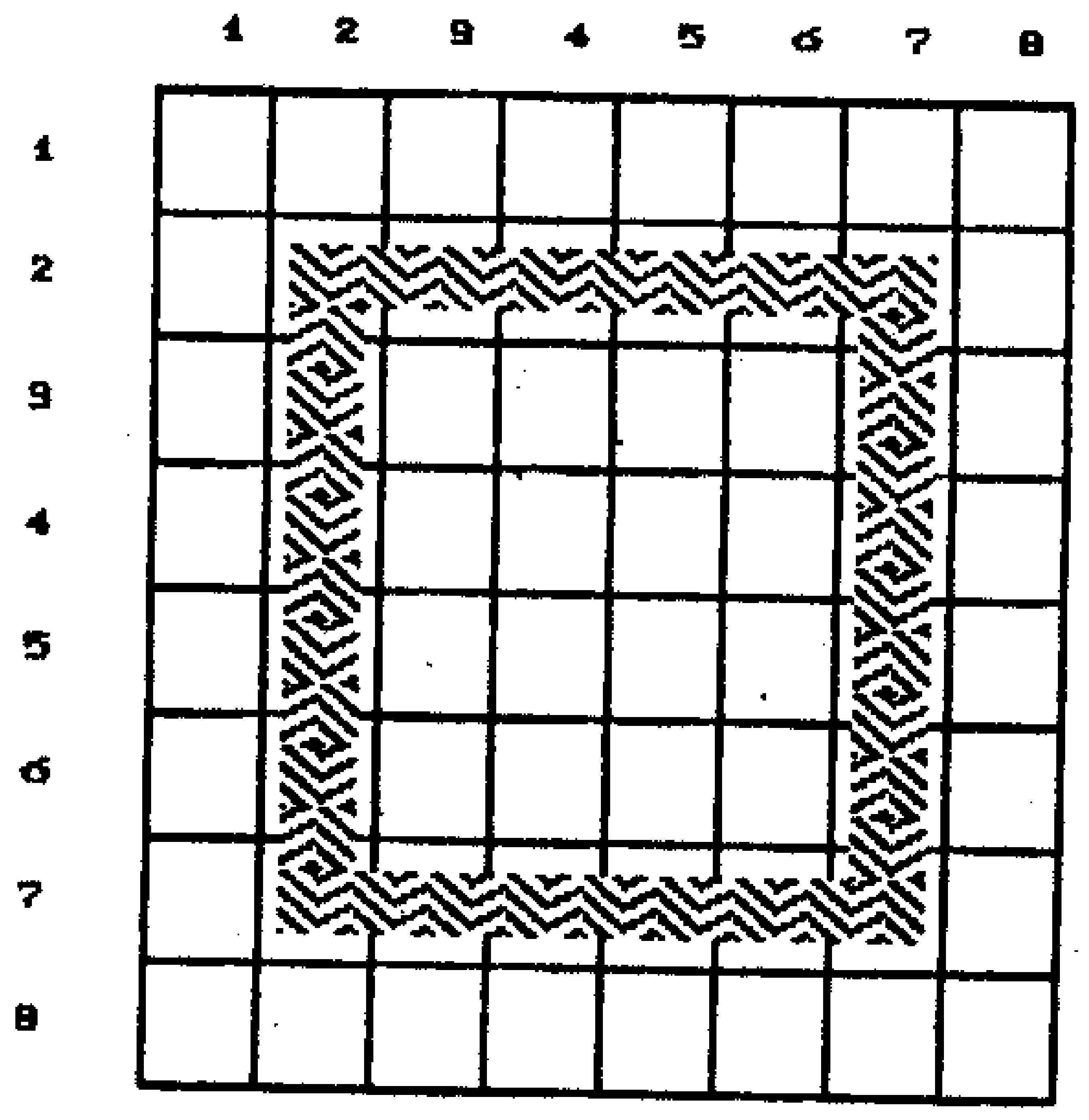


FIG 4.4

For example black now wants to place his counter in the cell marked ↔ (The cell in third row & third column). Since there is a black counter two cells to the right of it and the two cells in between are white, the move is valid. After placing this counter the two white cells will now become black. The same happens in the southeast direction where two more counters turns black.The board also contains a black two cells to the down of the current cell. But the white counter in one of the cells between them cannot be turned black because of the other cell which is blank. In this process, there may exist a situation in which there exist one or more white cells between a flipped black cell and a previously existing black cell. The counters in these cells, however, cannot change the colour. The resulting configuration will be as shown in *fig 4.3*.

After the player flips all the possible counters, then the second player makes a move. If it is not possible for any player to make a valid move, then the player must *pass*, that is, the player looses one chance. The game ends if any of the following conditions occurs at any time of the game.

1. All the cells on the board are filled by counters.
2. One player does not have any counter left on the board
3. There does not exist any move for both the players.

In any case, the player with more number of counters of his colour on the board at the end of the game will be the winner.

Any one , after playing a few games, can observe that the move at any time depends only on the current configuration, as the board configuration will be rapidly changing after each move. However, there are a few strategies, that will help winning the game. A counter placed in the four corner cells of the board will never be changed. Similarly, the counters placed in the outermost rows or columns can be changed only by placing counters in the respective rows and columns thereby reducing the possibility of the counter being flipped. If a player wishes to keep a counter in these *safe cells* , he should not keep a counter in any of the the preceeding cells (hashed in *fig.4.4*) These cells are called *dangerous* , because keeping a counter in these cells is nothing but giving a chance to the opponent to keep his counter in the outermost zone. All such strategies can be learned by experience or by consulting an expert. The configuration of the board is nothing but, the colour of each of the 64 cells (empty, white or black). The *knowledge* needed to make a move is this colour information and the information of which move has most advantage than the other.

5. IMPLEMENTATION & ALGORITHM

5.1 KNOWLEDGE REPRESENTATION

For building an expert system for this game, we need to code the knowledge by any of the knowledge representation schemes. The knowledge base contains two types of information. One information is regarding the colour of each cell on the board. This information will be changing for more than two cells after each move. The second information in the knowledge base is regarding the *priority* of each cell empty cell. Priority is defined over the cells as the future advantage. Any player can find which cell is having more priority than the other by playing a few games. For example, the four corner cells are having top priority over the other cells because, a counter once placed in this cell can never be changed and so on. The priority can be ranged to any number of levels from 1 to 64 but for ease, only 6 levels of priorities are considered. The priority of the cells (except for the corner cells) will be changing according to opponent's moves.

Advantage is defined as the number of counters being gained by placing the counter in that particular cell (This will be zero if it is not a valid move) including the currently placed cell. If there is a situation such that both priority and advantage are equal for different cells, then *randomness* is included in order to avoid repetition of game if played repeatedly.

This knowledge can be easily represented using *predicate logic*. The configuration of the board is coded using the precicate *color(I,J,Clr)* where *<I,J>* is the position of the cell on the board and *Clr* is its present colour. It is obvious that this predicate has 64 clauses one for each cell. The priority of each empty cell also coded into a predicate *priority(I,J,P)* where *<I,J>* is the position of the cell and *P* its priority. This predicate has 64 - 4 = 60 clauses at the beginning of the game and will be decreasing, as the number of blank cells will be decreasing, as the game proceeds.

5.2 IMPLEMENTATION

PROLOG (abbrivation for *PRO*gramming with *LOG*ic) is one language which directly supports the Predicate logic. The program in prolog is a predicate, i.e., a set of clauses of the form $A :- A_1 , A_2 , A_3, \cdots\cdots A_N$. where each A_i again is a predicate. A succeeds if all of the predicates $A_1 , A_2 , A_3, \cdots\cdots A_N$, succeeds. If atleast one of these predicates fails, then A fails.

In order to succeed one predicate, prolog attempts to unify with the clauses, satisfying its predicates. If any of the predicate fails, then the system backtracks, and makes the clause false. Then it tries with the next clause. If all the clauses fails, then the predicate fails, and hence the clause whose rule contains the current predicate A. Thus prolog program execution goes on two principles, *unification & backtracking.*

5.3 ALGORITHM

The game playing strategy of the expert system is as follows. Since this is not sequential, it is difficult to express it as a conventional algorithm. But, the following steps gives the main idea of the program.

1. Get user choice of colour.

2. Create the priorities of each of the cell and make the colour of each of the cell empty. Change the middle four blocks to appropriate colours (two black and two white diagonally) and update the knowledge base and draw the corresponding board.3. If the user's choice is white then simulate *user move & computer move* one after the other till the end of game.

4. If the user's choice is black then simulate *computer move & user move* one after the other till the end of game

5. The simulation of computer move is as follows.

    5.1 Check in the decreasing order of priority, &

    5.2 find all the cells and their advantages, i.e.,

        5.2.1 Check whether the cell is empty, &

        5.2.2 find its priority,

        5.2.3 count the number of possible advantages in all the eight directions (this is the advantage)

    5.3 if this advantage is zero, then backtrack to find next possible empty cell,

    5.4 collect all such possible cells and find the maximum advantaged cells out of them

    5.5 if more than one such cell exist then find one of them randomly, then change the cells , updating the knowledgebase.

6. The user move simulation is as follows.

6.1 Count the advantage of each of empty cell till the advantage is greater than zero for atleast one cell. If no such cell exist, then user has no valid move and hence take your chance again.

6.2 Accept user choice and check whether it is a valid move.

6.3 If it is valid , then change all the appropriate cells to user's colour .

6.4 Update the priorities of the cells surrounding the current cell occupied by the user.

7. The end of game condition is satisfied if one of the three conditions,

i . the board fill,

ii. one player has no counters on the board, or

iii. no player can make a move

All these conditions can be checked with one predicate.

5.4 TRAINING :-

One good reason why expert systems are widely used is their ability to train people of how to use them. The current expert system othello also provides such facility to train beginers of how to play the game. For this the system provides two facilities, that is,

1. Computer versus computer play &
2. User versus User play

Implementation of these two facilities is quite similar to that of the original game of Computer versus User play simulation. For implimenting these two facilities, instead of repeating Computer move and User move repeatedly, Computer move and computer move (one for black and the other for white) are repeated executed in the first case and User move and User move are repeated in the second case, the rest of the algorithm being the same.

5.5 DEVELOPMENTS :

This expert system works entirely on the priorities of the cells. This system creates initial priorities as shown in *fig.5.1*.

1	6	2	2	2	2	6	1
6	6	5	5	5	5	6	6
2	5	3	4	4	3	5	2
2	5	4	x	x	4	5	2
2	5	4	x	x	4	5	2
2	5	3	4	4	3	5	2
6	6	5	5	5	5	6	6
1	6	2	2	2	2	6	1

fig 5.1

If a cell with priority 1 is occupied by the expert system, immediately the cells adjacent to it (with priority 6) will be given priority 1 because occupying those cells gets more advantage after occupying the corner cell. This was implemented in the program.

Most of the cells on the outermost cells have priority 2. If one such cell is occupied by the opponent's counter, then its adjacent cells' priority must be decreased. The reason is that, even if we place counter in those cells, they will be immediately captured by the opponent. However, this is not a possibility if the next cell also contains an opponents counter (it is, we are placing a counter between two opponents counters). This is also one development of priority modification, which can be implemented easily.

In the program, the next move will be performed based on the current board configuration. It can further be developed considering what will be the board configuration after each possible move and thus guessing opponents moves. A further development to this concept is to extend this to n steps ahead. In each case, the resulting step is to make a most advantageous move and to update the priorities accordingly. The priority updating procedure will be increasing proportional to the expertise obtained regarding the play.

REFERENCES

1. Arity prolog programming - User's Manual
2. Building Expert Systems - D. A . Waterman
3. Artificial Intelligence through Prolog - Neil C. Rowe
4. Introduction to Artificial Intelligence - Elaine Rich