To the Graduate Council:

I am submitting herewith a thesis written by John O. F. Long entitled "The Use of Imperfect Information Concepts from Game Theory in the Solution of Games of Simultaneous Moves in Artificial Intelligence." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Michael G. Thomason

Michael G. Thomason, Major Professor

We have read this thesis and recommend its acceptance:

Lawrence S. Husch

David Snyder

Craig W. Thompson

Accepted for the Council:

L. Evans Roth

Vice Chancellor
Graduate Studies and Research

THE USE OF IMPERFECT INFORMATION CONCEPTS FROM GAME THEORY IN THE SOLUTION OF GAMES OF SIMULTANEOUS MOVES IN ARTIFICIAL INTELLIGENCE

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

John O. F. Long

March 1981

3049638

ABSTRACT

Game-playing programs in artificial intelligence have largely dealt with two-player games with sequential moves. Many other classes of games have not been examined closely. The present study develops solution methods for two-player games with simultaneous moves.

The first portion of the thesis is aimed at the artificial intelligence researcher interested in game playing and reviews some results from game theory which are important for extending the artificial intelligence perspective to classes of games other than two-person zero-sum games. The concept of a player's rationality is discussed and is used to justify a best-reply algorithm which extends the minimax solution for perfect information zero-sum games from two-person to n-person games. Then, classification criteria for games are reviewed and solution methods for the different classes are described.

The main result of this thesis is a simultaneous move algorithm (ASMA), which was developed from game theoretic concepts as a basic method for solving games of simultaneous moves. In a modular way, heuristics can be used with ASMA to increase its efficiency in game-playing programs.

A two-player version of a DIPLOMACY-like game called REDIPT is described, and a game-playing program of the same name is developed. ASMA, the main subroutine of the program REDIPT, is used to calculate a solution for the game.

TABLE OF CONTENTS

CHAPTER	PAGE
1. THE RELATIONSHIP BETWEEN GAME-PLAYING IN ARTIFICIAL INTELLIGENCE AND GAME THEORY	1
Introduction	1
Introduction to Games	2
How Artificial Intelligence Approaches Game-Playing	5
Extending Concepts to New Game Classes	13
The Usefulness of Game Theory to Artificial Intelligence	22
2. INTRODUCTION TO GAME THEORETIC CONCEPTS	24
A Player's Motivation: Rationality	24
The Definition of a Solution as an Equilibrium Situation	26
Classifications of Games	30
3. SOLUTIONS TO VARIOUS CLASSES OF GAMES	36
Two-Person Zero-Sum Games	36
Two-Person Nonzero-Sum Games	43
New Considerations for N-Person Games	45
Solutions for N-Person Games	49
4. SIMULTANEOUS MOVEMENT GAMES	53
Description of Simultaneous Movement Games	53
Algorithm for Solving Simultaneous Movement Games	55
A Description of the Game REDIPT	61
The Implementation of REDIPT	65
Experience with REDIPT	70
5. CONCLUSIONS	77
LIST OF REFERENCES	80
APPENDICES	83
A. EXAMPLE OF THE SIMPLEX METHOD	84
B. A LISTING OF THE PROGRAM REDIPT	90
VITA	120

LIST OF FIGURES

FIGURE		PAGE
1.	A Sample Game Tree	6
2.	Minimax Algorithm	10
3.	Game Tree for Minimax Example	12
4.	Comparison of Minimax Choice versus Best-Reply Choice	14
5.	Comparison of Minimax Solution and Best-Reply Solution	17
6.	Best-Reply Algorithm	18
7.	A Self-Policing Prison System	27
8.	Imperfect Information Game	34
9.	Imperfect Information without Simultaneous Moves	34
10.	Game Represented in Extensive and Normal Form	35
11.	Game with One Saddle Point	37
12.	Game with Two Saddle Points	38
13.	Game Reduced through Domination of Strategies	39
14.	Game without a Saddle Point	40
15.	Two-Person Nonzero-Sum Game	44
16.	Bargaining Set Solution of a Game	46
17.	Game in Characteristic Function Form	46
18.	Divide the Money Game	49
19.	Two-Level Sequential Movement Game	54
20.	Two-Level Simultaneous Movement Game	54
21.	Normal Form of Figure 20	54
22.	2K-Level Simultaneous Movement Game	56

FIGURE PAGE

23. ASMA 57

24. Game to Demonstrate ASMA 60

25. Sample Game Board for REDIPT 67

26. Representations of a Move 69

27. Possible First Moves with and without Heuristic 73

28. REDIPT Output without Heuristic 75

29. REDIPT Output with Heuristic 76

30. Simplex Game Matrix 86

31. Step 1 86

32. Step 2 86

33. Step 3 87

34. Step 6 87

35. Step 7 87

36. Step 8 88

37. Step 9 First 88

38. Step 9 Last 88

CHAPTER 1

THE RELATIONSHIP BETWEEN GAME-PLAYING IN ARTIFICIAL INTELLIGENCE AND GAME THEORY

Introduction

Perhaps one of the most entertaining aspects of artificial intelligence is the development of game-playing programs. Techniques used in these programs are important for artificial intelligence, yet are also valuable in modelling real world social situations. Social situations involve a number of persons who must make decisions until a given conclusion is reached, and the field of game theory provides a mathematical description of how to deal with such situations. The goal of artificial intelligence can be described as that of modelling intelligent behavior, including intelligent human behavior, and we will be concerned with intelligent behavior that deals with social situations. Traditionally, game-playing programs in artificial intelligence have involved heuristic search techniques for finding a (possibly nonoptimal) solution path in a solution space that is typically too large to search exhaustively.

Artificial intelligence game-playing programs typically deal with 2-person zero-sum sequential games such as checkers and chess. Little theory exists in artificial intelligence to generalize the results of these programs to other classes of games. Consequently, this work draws upon game theory for the necessary theory to extend game-playing

results to a different class of games: two-player zero-sum simultaneous movement games. Because this work is directed toward the field of artificial intelligence, solution methods are developed only after the necessary basic game theory background is presented. Hereafter, all references to artificial intelligence will refer to the subarea of artificial intelligence concerned with game-playing.

Introduction to Games

A game can be thought of as a series of moves, either simultaneous or sequential, which occur until the game comes to an end. The end of the game usually takes place when at least one player meets certain game termination conditions or a turn limit is reached.

Generally, games can be defined by five descriptors (10):

1. The players involved in the game.
2. The choices available to each player when it is his/her move.
3. The amount of information available to each player about the moves made previously.
4. A method to determine the end of the game.
5. A function specifying the amount of reward or payoff (positive or negative) given to each player at a given termination of the game.

Many games are models or simulations of real events, although other games represent very abstract situations with little resemblance to reality. The players of a game may represent individuals, groups of

individuals, or organizations. A move is a particular choice from the set of actions available to a player on his/her turn. In making a move, a player may need to know which moves have already been made. This is known as the information available in the game. Players make their moves each turn until the game ends, which is specified by game termination conditions. At each termination of the game, there is a particular payoff given to each player. Generally, players who receive positive payoffs are winners, and the other players are losers.

Most games can be described mathematically. From the previously given descriptors, a game can be reduced mathematically to a three-tuple

$$G = \langle I, \{\Sigma_i | i \varepsilon I\}, \{P_i | i \varepsilon I\} \rangle,$$

where

I = the set of players = $\{1, 2, \ldots, n\}$,

Σ_i = the strategy space for player i, and

P_i = the payoff function for player i (5, 17).

The set of players I includes n players.

In a game, each player i possesses K_i pure strategy choices $(a_i^1, a_i^2, \ldots, a_i^{K_i})$, which are the possible actions player i may make. If this player is certain that he wants to use strategy choice a_i^j for the game, we can represent this as a vector σ_i of the form

$$\sigma_i = (\sigma_i^1, \sigma_i^2, \ldots, \sigma_i^{K_i},$$

where σ_i^K is the probability that a_i^K will be chosen, and

$$\sum_{j=1}^{K_i} \sigma_i^j = 1 .$$

We can indicate that a_i^j is the only strategy choice used by giving σ_i the value

$$\sigma_i = (\sigma_i^1, \sigma_i^2, \ldots, \sigma_i^{j-1}, \sigma_i^j, \sigma_i^{j+1}, \ldots, \sigma_i^{K_i})$$
$$= (0,0, \ldots, 0,1,0, \ldots, 0) .$$

If, however, a player is not certain which strategy choice should be employed, he may decide to assign probabilities to his strategy choices to help him decide. This situation can also be represented by the vector σ_i. If, for instance, player i had two strategy choices a_i^1 and a_i^2 and wanted to assign equal probabilities to them, the vector σ_i would have the value

$$\sigma_i = (\sigma_i^1, \sigma_i^2) = (0.5, 0.5) .$$

By using a vector of probabilities to decide which strategy choice to use, a player may use different strategy choices in repeated plays of the same game. A player's strategy for a game, then, is not the scalar strategy choice a_i^j which is used for the game, but the probability vector σ_i used for picking the strategy choice a_i^j for player i. If a player's strategy

$$\sigma_i = (\sigma_i^1, \sigma_i^2, \ldots, \sigma_i^{K_i})$$

possesses only one nonzero probability $\sigma_i^j = 1$, then the strategy σ_i is said to be a pure strategy. A strategy σ_i which possesses more than one probability σ_i^j such that

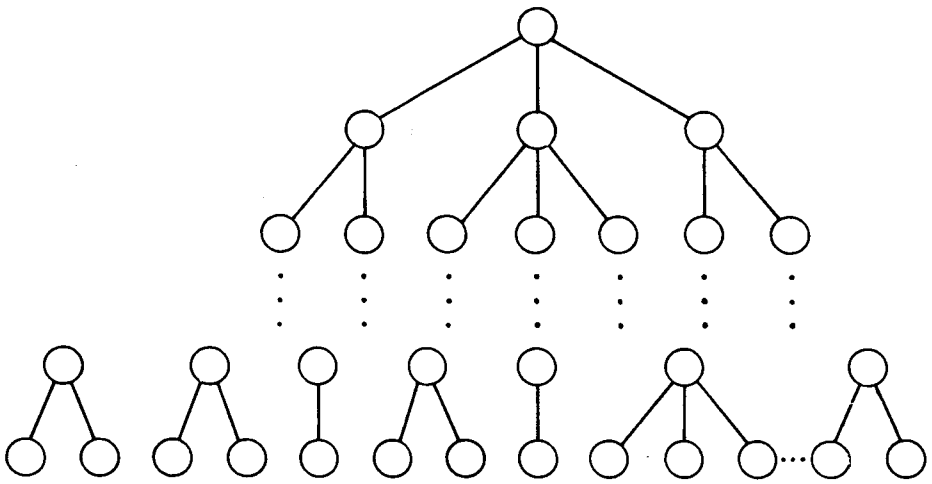
$$0 < \sigma_i^j < 1$$

is said to be a mixed strategy. A player may possess a finite number of pure strategies for a game, but he may possess infinitely many mixed strategies.

The preceding mathematical description of games has been useful in game theory to find solutions to games. A discussion of how artificial intelligence approaches game-playing follows.

How Artificial Intelligence Approaches Game-Playing

Most game-playing programs are heuristic in nature. Instead of solving a problem by exhaustively searching for a solution, a heuristic program uses short-cuts or guesses to find a solution. Typically, game-playing programs solve games by generating game trees which represent the alternatives available to each player at each turn. The tree in Figure 1 is an example of a game tree. A node represents a unique occurrence of the player's moves in reaching that point in the game. The branches emanating from each node are the choices or moves which



Level	Player's Move	Turn
1	1	1
2	2	
3	1	2
⋮	⋮	⋮

← leaf nodes

Figure 1. A sample game tree.

a player can make from a particular occurrence of moves in the game. When a node has no branches coming from it, the game has come to an end, and a reward or payoff is given to each player at that leaf node.

Since many game trees are very large, heuristic programs try to limit the size of the tree with heuristics to reduce the time and space spent searching the tree for a solution. Frequently used heuristics include:

1. Limiting the depth of the search and estimating the cost of a path from a leaf node to a goal node.
2. Limiting the breadth of the game tree by only considering some options.
3. Abandoning the direction of a search when a player has reached irreversible disaster.
4. Excluding choices which are trivially different from a current choice (14, 19).

Heuristic programs do not always give optimal solutions because they do not explore all possibilities, but they often provide quicker answers than programs doing an exhaustive search. Some heuristic game-playing programs can only be used for a specific game, although others are of a more general nature. Also, potentially inaccurate heuristics are used to estimate payoffs for intermediate states in the game.

Traditionally, these heuristic game-playing programs have dealt with two-player zero-sum sequential games. In a zero-sum game, the addition of the players' payoffs in any situation totals zero. This characteristic causes these games to be strictly competitive between

the players. For a two-player zero-sum game involving players X and Y, a payoff of p given to player X means player Y will receive a payoff of -p.

Because player X is playing against player Y, he wants to know the value of the lowest payoff he can obtain because this is Y's highest payoff, which is Y's objective. Using a strategy σ_x, this lowest payoff is given by

$$\overline{P}_x(\sigma_x) = \min_{\sigma_y \varepsilon \Sigma_y} P_x(\sigma_x, \sigma_y) ,$$

where P_i is the payoff given to player i and Σ_i is the set of all strategies of player i (5). The value $\overline{P}_x(\sigma_x)$ is designated the security level of player X. Player X's objective is to maximize his security level, and the highest payoff X can obtain is given by

$$\overset{v}{P}_x = \max_{\sigma_x \varepsilon \Sigma_x} \overline{P}_x(\sigma_x) = \max_{\sigma_x \varepsilon \Sigma_x} \min_{\sigma_y \varepsilon \Sigma_y} P_x(\sigma_x, \sigma_y) .$$

The value $\overset{v}{P}_x$ is X's maximin payoff for a game. Similarly, Y's maximin payoff is given by

$$\overset{v}{P}_y = \max_{\sigma_y \varepsilon \Sigma_y} \overline{P}_y(\sigma_y) = \max_{\sigma_y \varepsilon \Sigma_y} \min_{\sigma_x \varepsilon \Sigma_x} P_y(\sigma_x, \sigma_y) ,$$

and

$$\max_{\sigma_y \varepsilon \Sigma_y} \min_{\sigma_x \varepsilon \Sigma_x} P_y(\sigma_x, \sigma_y) = -\min_{\sigma_y \varepsilon \Sigma_y} \max_{\sigma_x \varepsilon \Sigma_x} P_x(\sigma_x, \sigma_y) .$$

From this, we notice that

$$\overset{v}{P}_y = \max_{\sigma_y \varepsilon \Sigma_y} \overline{P}_y(\sigma_y) = \max_{\sigma_y \varepsilon \Sigma_y} \min_{\sigma_x \varepsilon \Sigma_x} P_y(\sigma_x, \sigma_y)$$

$$= -\min_{\sigma_y \varepsilon \Sigma_y} \max_{\sigma_x \varepsilon \Sigma_x} P_x(\sigma_x, \sigma_y) = -\max_{\sigma_x \varepsilon \Sigma_x} \overline{P}_x(\sigma_x) = -\overset{v}{P}_x \quad (7) ,$$

due to the definition of zero-sum payoffs. Maximin and minimax will be used interchangeably because of the above.

In relation to a game tree, minimax implies that a player can choose his move based on the expectation that the player who will move after him will try to maximize the second player's payoff (or minimize the first player's payoff). At each level, either player X will try to maximize his own payoff, or player Y will try to minimize X's payoff. To use these payoffs, we must go to the leaf nodes of the tree to determine which payoffs are chosen and send these values up the tree at each level.

A minimax algorithm for two-player zero-sum sequential game trees is described in Figure 2 using the PL/C programming language. To demonstrate this algorithm, Figure 3 shows the path taken by the algorithm through the example game tree. From this, we see that player 1 can ensure for himself a payoff of 1, meaning player 2 will receive a payoff of -1.

The alpha-beta heuristic can be used with minimax to improve the efficiency of the latter (19). This heuristic eliminates the consideration of strategies which cannot be better than the strategies

```
MAIN:  PROCEDURE OPTIONS(MAIN);
       DECLARE (LEVEL,I,ROOT,PAYOFF) FIXED;
       LEVEL=1;  /* LEVEL=1 IF THIS IS X'S LEVEL  */
                 /*      =-1 IF THIS IS Y'S LEVEL */
       CALL READTRE (ROOT);  /* READTRE INPUTS THE GAME TREE */
       I=0;  /* I WILL BE THE BEST NEXT MOVE.  */
       CALL MINIMAX(ROOT,LEVEL,I,PAYOFF);
                      /* BEGIN AT THE ROOT OF THE TREE. */
       PUT SKIP LIST ('BEST MOVE IS ',I);
       PUT SKIP LIST ('PAYOFF IS',PAYOFF);
                      /* PRINT OUT BEST NEXT MOVE. */
       STOP;
       END MAIN;
```

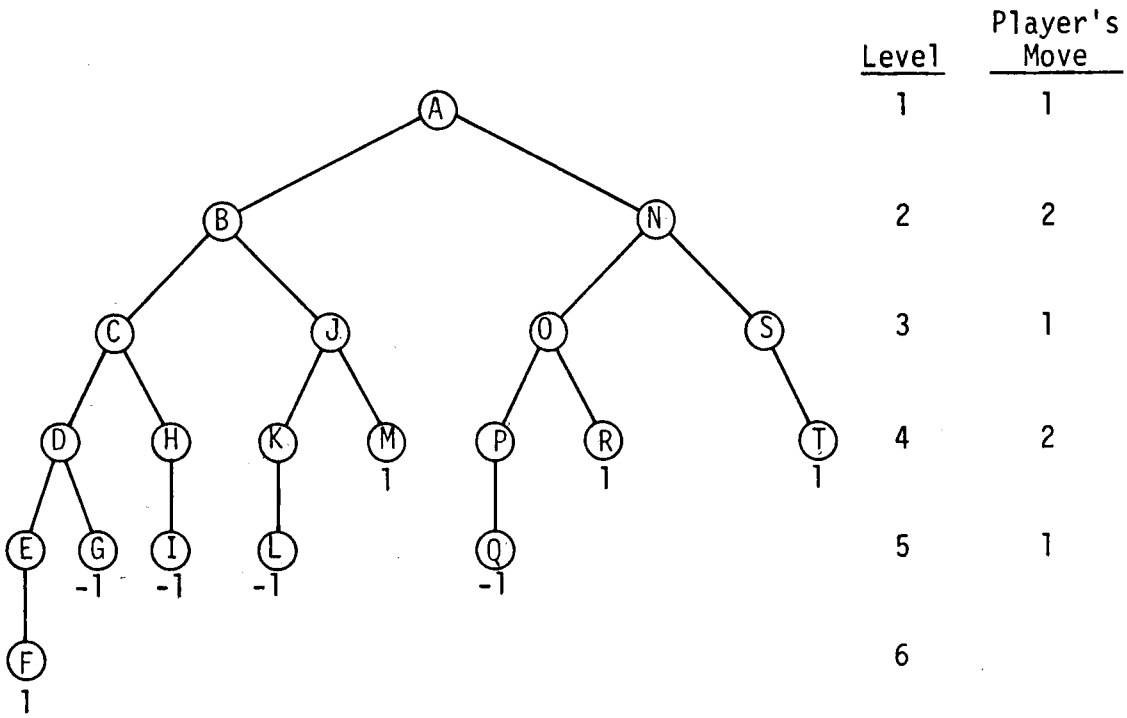
(a) Main program.

Figure 2. Minimax algorithm.

```
MINIMAX:  PROCEDURE(NODE,LEVEL,I,PAYOFF) RECURSIVE;
          DECLARE (NODE,LEVEL,I,PAYOFF,J,BEST) FIXED;
          /* LEAF RETURNS "TRUE" IF CURRENT NODE IS A LEAF NODE */
          /* AND "FALSE" OTHERWISE.                             */
          DECLARE LEAF ENTRY (FIXED) RETURNS(BIT(1));
          /* VALUE RETURNS THE PAYOFF OF A LEAF NODE. */
          DECLARE VALUE ENTRY (FIXED,FIXED) RETURNS(FIXED);
          /* NCHILD RETURNS THE NUMBER OF CHILDREN OF THE CURRENT NODE. */
          DECLARE NCHILD ENTRY(FIXED) RETURNS(FIXED);
          /* CHILD RETURNS THE JTH CHILD OF A GIVEN NODE. */
          DECLARE CHILD ENTRY(FIXED,FIXED) RETURNS(FIXED);
          IF (LEAF(NODE))
          THEN DO;  /* IF THIS IS A LEAF NODE, THE END OF THE GAME, */
                    /* RETURN THE VALUE OF THE NODE FOR THE CURRENT */
                    /* PLAYER.                                      */
                    PAYOFF=VALUE(NODE,LEVEL);
                    END;
          ELSE DO;  /* OTHERWISE, GET THE BEST PAYOFF FOR */
                    /* THE GAME BELOW THIS NODE.          */
                    BEST=-10000;
                    DO J=1 TO NCHILD(NODE);
                         K=J;
                         CALL MINIMAX(CHILD(NODE,J),-LEVEL,K,PAYOFF);
                         IF (-PAYOFF>BEST)
                         THEN DO;
                              BEST=-PAYOFF;
                              I=K;
                              END;
                         END;
                    PAYOFF=BEST;
                    END;
          RETURN;
          END MINIMAX;
```

(b) Subroutine.

Figure 2 (continued)



1. Path through tree: A,B,C,D,E,F,E,D,G,D; Player 2 sees he should choose the -1 payoff, which is a payoff of 1 for him.
2. C,H,I,H,C; Player 1 can get a payoff of -1.
3. B,J,K,L,K,J,M,J; Player 1 can get a payoff of 1.
4. B; Player 2 can get a payoff of -1.
5. A,N,O,P,Q,P,O,R,O; Player 1 can get a payoff of 1.
6. N,S,T,S,N; Player 2 can do no better than to choose a payoff of 1.
7. A; Player 1's best choice is a payoff of 1.

Figure 3. Game tree for minimax example.

already considered. Both alpha-beta and minimax will give the optimal solutions.

Minimax is a technique used by artificial intelligence to solve two-person zero-sum sequential games for which an entire game tree can be generated. Other heuristics can be used in conjunction with this technique to limit the size of the game tree searched. Alpha-beta or minimax can be used to level k if a heuristic evaluation function is used to estimate the payoffs at the leaves of the game tree.

Extending Concepts to New Game Classes

If we try to extend this result to an n-person zero-sum sequential game where n > 2, problems arise. Attempting to generalize the minimax technique will not always lead to a correct solution, because, with n players, attempting to minimize one player's payoff is not necessarily the same as maximizing another player's payoff.

For a general n-person zero-sum game, a minimax strategy can be defined, although this may not be an optimal strategy. A minimax strategy for player i expects the worst from every other player. In other words, every other player chooses strategies which will minimize the payoff of player i, while player i attempts to maximize his own payoff (5). Player i's payoff can be stated as

$$\overset{v}{P}_i = \max_{\sigma_i \varepsilon \Sigma_i} \overline{P}_i(\sigma_i) = \max_{\sigma_i \varepsilon \Sigma_i} \min_{\sigma^i \varepsilon \Sigma^i} P_i(\sigma_i, \sigma^i) ,$$

where

$$\sigma^i = (\sigma_1, \sigma_2, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n)$$

and

$$\Sigma^i = \Sigma_1 X \Sigma_2 \ldots X \Sigma_{i-1} X \Sigma_{i+1} \ldots X \Sigma_n \quad (5) \ .$$

As an example, consider the trivial game in Figure 4, which involves three players. The payoffs at the leaf nodes in the form (V_1, V_2, V_3) represent the payoffs of each player, where V_i is the payoff for player i. From this example, if player 1 was trying to minimize player 2's payoff, he would choose node C, yet, if player 1's motive was to maximize his own payoff, he would choose node B. The reason minimax works in a two-player game is that the two concepts, minimizing the other player's payoff and maximizing the payoff of player i, are equal. Thus, a "minimax" solution for an n-person game is not always the best.

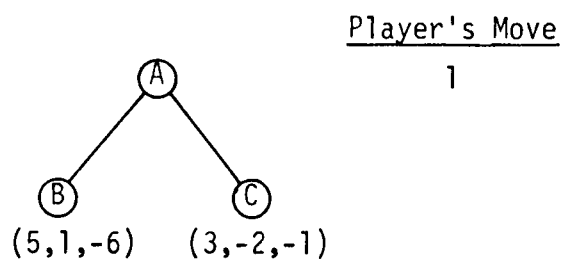


Figure 4. Comparison of minimax choice versus best-reply choice.

To find a useful generalization of the two-person minimax solution concept, the motivation for a player's actions in a game must be categorized. In game theory, this motivation is known as rationality and will be discussed later in detail. But, briefly, a player is said to

be acting rationally if he is acting to maximize his payoff in a game. With this definition, we can now develop a player's best-reply strategy as a strategy which, for a given set of alternatives or choices, will always choose an alternative which maximizes the player's payoff. It is possible that more than one of these payoffs exists in a game, but each of these "best" alternatives have the same payoff for the player.

Formally, given a set of alternatives for player i, he will choose his strategy $\sigma_i \varepsilon \Sigma_i$ yielding a payoff of

$$p_i(\sigma^i) = \max_{\sigma_i \varepsilon \Sigma_i} P_i(\sigma_i, \sigma^i) ,$$

where

$$\sigma^i = (\sigma_1, \sigma_2, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n) ,$$

or the strategies of every player excluding player i. This strategy σ_i is a best-reply strategy which can be used to rewrite p_i as

$$p_i(\sigma^i) = \max_{\sigma_i \varepsilon \Sigma_i} P_i(\sigma_i, \sigma^i) = P_i(\sigma_i^*, \sigma^i) ,$$

where σ_i^* is the optimal strategy of player i against the other players' strategies σ^i. A situation σ, which contains a strategy σ_i for every player i which is a best-reply strategy for the strategies σ^i of the other players, is a best-reply solution. A best-reply solution is equivalent to an equilibrium point, which will be discussed later. One might also describe a best-reply solution as

$$\overline{p}_i = P_i(\sigma^*) ,$$

where

$$\sigma^* = (\sigma_1^*, \sigma_2^*, \ldots, \sigma_n^*) .$$

As a further demonstration of the difference between minimax solutions and best-reply solutions, Figure 5 shows the minimax and best-reply payoffs for each player in a given three-player game.

A best-reply algorithm for sequential games is shown in Figure 6.

In a two-player game, then, the minimax solution can be expressed as

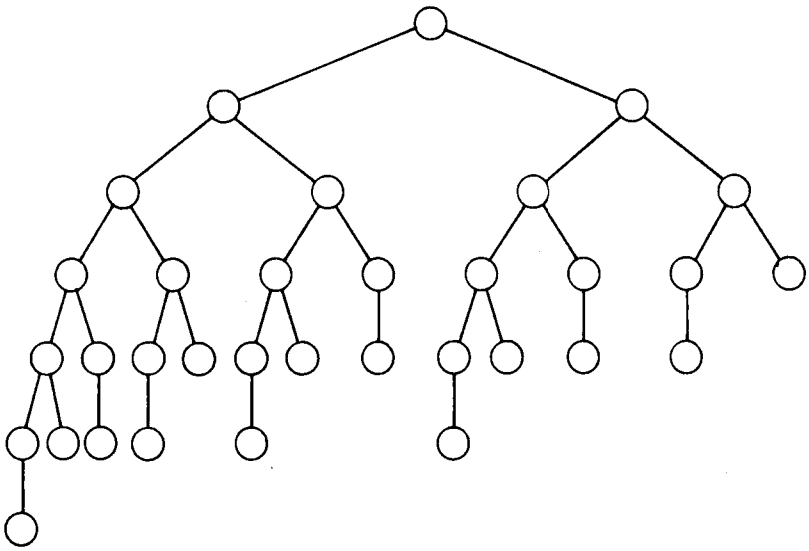
$$\overset{v}{P}_x = \max_{\sigma_x \varepsilon \Sigma_x} \min_{\sigma_y \varepsilon \Sigma_y} P_x(\sigma_x, \sigma_y) .$$

We can also express this as

$$\overset{v}{P}_x = \max_{\sigma_x \varepsilon \Sigma_x} \min_{\sigma^x \varepsilon \Sigma^x} P_x(\sigma_x, \sigma^x) ,$$

because the set of strategies Σ^x of all players except player X is equivalent to the set of strategies Σ_y of player Y. Simple inspection proves this, for there is only one player other than player X. Also, we are sure that Y wants to minimize X's payoff, since

$$\min_{\sigma_y \varepsilon \Sigma_y} P_x(\sigma) = -\max_{\sigma_y \varepsilon \Sigma_y} P_y(\sigma) ,$$



Player's Move	Payoff for Leaf Node at This Level
1	
2	
3	
1	(-1,0,1)
2	(1,-1,0)
3	(0,1,-1)
	(-1,0,1)

Player	Payoff Using Minimax	Payoff Using Best-Reply
1	0	1
2	-1	-1
3	-1	0

Figure 5. Comparison of minimax solution and best-reply solution.

```
MAIN:  PROCEDURE OPTIONS(MAIN);
       DECLARE N FIXED EXTERNAL STATIC;
       GET LIST(N);
       BEGIN;
            DECLARE (LEVEL,I,ROOT,PAYOFF(N)) FIXED;
            LEVEL=1;  /* LEVEL=1,2,... OR N ACCORDING TO WHICH */
                      /* PLAYER IS MOVING.                     */
            CALL READTRE(ROOT);
            I=0;  /* I WILL BE THE BEST NEXT MOVE.  */
            CALL BEST_REPLY(ROOT,LEVEL,I,PAYOFF,N);
                           /* BEGIN AT THE ROOT OF THE TREE.  */
            PUT SKIP LIST('BEST MOVE IS ',I);
            PUT SKIP LIST('PAYOFFS ARE ',PAYOFF);
            STOP;
            END;
       END MAIN;
```

(a) Main program.

Figure 6. Best-reply algorithm.

```
BEST_REPLY:  PROCEDURE(NODE,LEVEL,I,PAYOFF,N) RECURSIVE;
/* BEST_REPLY FINDS THE BEST MOVE FOR PLAYER I AT THE GIVEN NODE AND */
/* THE CORRESPONDING PAYOFF VECTOR FOR THE MOVE.                     */
            DECLARE (NODE,LEVEL,I,PAYOFF(*),J,BEST(N),K) FIXED;
            /* LEAF RETURNS 1 IF CURRENT NODE IS A LEAF NODE AND 0  */
            /* OTHERWISE.                                           */
            DECLARE LEAF ENTRY(FIXED) RETURNS(BIT(1));
            DECLARE N FIXED EXTERNAL STATIC;
            /* NCHILD RETURNS THE NUMBER OF CHILDREN OF THE CURRENT */
            /* NODE.                                                */
            DECLARE NCHILD ENTRY(FIXED) RETURNS(FIXED);
            /* CHILD RETURNS THE JTH CHILD OF A GIVEN NODE.         */
            DECLARE CHILD ENTRY(FIXED,FIXED) RETURNS(FIXED);
            /* NEXT_LEVEL RETURNS THE NEXT PLAYER TO MOVE.  */
            DECLARE NEXT_LEVEL ENTRY(FIXED) RETURNS(FIXED);
            IF (LEAF(NODE))
            THEN DO;  /* IF THIS IS A LEAF NODE, RETURN THE PAYOFF */
                      /* VECTOR OF THE MOVING PLAYER.              */
                      CALL VALUE(NODE,LEVEL,PAYOFF);
                      /* PAYOFF IS AN OUTPUT PARAMETER. */
                      END;
            ELSE DO;  /* OTHERWISE, GET THE MOVING PLAYER'S BEST CHOICE */
                      /* BY EVALUATING THE GAMES BELOW THE CURRENT NODE.*/
                      BEST=-10000;  /* ARRAY ASSIGNMENT. */
                      DO J=1 TO NCHILD(NODE);
                           CALL BEST_REPLY(CHILD(NODE,J),NEXT_LEVEL(LEVEL),
                                           K,PAYOFF);
                           IF (PAYOFF(LEVEL)>BEST(LEVEL))
                           THEN DO;
                                I=K;
                                BEST=PAYOFF; /* ARRAY ASSIGNMENT */
                                END;
                           END;
                      PAYOFF=BEST;
                      END;
            RETURN;
            END BEST_REPLY;
```

(b) Subroutine.

Figure 6 (continued)

or Y's payoff in a situation is the negative of X's payoff in the same situation.

In a game larger than two players, the minimax payoff for X is given as

$$\overset{v}{p}_x = \max_{\sigma_x \varepsilon \Sigma_x} \min_{\sigma^x \varepsilon \Sigma^x} P_x(\sigma_x, \sigma^x) .$$

Let us consider a three-player game. In finding a minimax strategy for player X, we must consider that, because of the minimax payoffs for more than two players, as defined above, player X may assume that the other two players will want to choose a $\bar{\sigma}^x$ such that

$$P_x(\sigma_x, \bar{\sigma}^x) = \min_{\sigma^x \varepsilon \Sigma^x} P_x(\sigma_x, \sigma^x) .$$

This would mean that

$$P_y(\sigma_x, \bar{\sigma}^x) + P_z(\sigma_x, \bar{\sigma}^x) = C ,$$

while

$$P_x(\sigma_x, \bar{\sigma}^x) = -C ,$$

where C is a constant. In other words, the payoff vector

$$(P_x(\sigma_x, \bar{\sigma}^x), P_y(\sigma_x, \bar{\sigma}^x), P_z(\sigma_x, \bar{\sigma}^x))$$

could be in one of the following forms:

$$
\begin{array}{ccc}
(-C\ , & 0\ , & C\ ) \\
(-C\ , & 1\ , & C-1\ ) \\
 & \vdots & \\
(-C\ , & C-1\ , & 1\ ) \\
(-C\ , & C\ , & 0\ )
\end{array}
$$

Player Y would prefer to choose $\tilde{\sigma}_y \varepsilon \Sigma_y$ such that

$$\tilde{\sigma}^x = (\tilde{\sigma}_y, \sigma_z)$$

and

$$P_x(\sigma_x, \tilde{\sigma}^x) = -C,\ P_y(\sigma_x, \tilde{\sigma}^x) = C,\ P_z(\sigma_x, \tilde{\sigma}^x) = 0\ .$$

Similarly, player Z would prefer to choose $\check{\sigma}_z \varepsilon \Sigma_z$ such that

$$\check{\sigma}^x = (\sigma_y, \check{\sigma}_z)$$

and

$$P_x(\sigma_x, \check{\sigma}^x) = -C,\ P_y(\sigma_x, \check{\sigma}^x) = 0,\ \text{and}\ P_z(\sigma_x, \check{\sigma}^x) = C\ .$$

Hence, the ideal situation for Y and the ideal situation for Z may not be the same, since, if C is nonzero,

$$P_y(\sigma_x, \tilde{\sigma}^x) \neq P_y(\sigma_x, \check{\sigma}^x)\ ,$$

and

$$P_z(\sigma_x, \tilde{\sigma}^x) \neq P_z(\sigma_x, \overset{v}{\sigma}{}^x) \ .$$

Thus, the minimax strategies will not necessarily yield suitable solutions for every player, but best-reply strategies will do so in the class of n-player zero-sum sequential games in which no agreements can be made between players (5). We will later briefly discuss agreements.

The Usefulness of Game Theory to Artificial Intelligence

The above discussion has shown that it is naive to extend the minimax solution to n-person games without first understanding the notion of a player's motivation in a game. Rather, it is necessary to first understand why the minimax strategy is an optimal strategy for two-person zero-sum sequential games and not for n-person zero-sum sequential games before considering possible optimal strategies for other classes of games. Otherwise, an optimal solution to a particular class of games can lead to a suboptimal solution for a different class of games.

In the development of game-playing programs, some programs have proven to be useful for only specific games. Such programs depend upon rules-of-thumb developed by good human game-players. Whereas these techniques may be useful for programs of a specific game, they may not be of value for programs of a different game. A general framework and rationale for game-playing would be valuable for programs dealing with various types of games, and each program could build upon this framework with its own particular set of heuristics for a game.

There has been a continuing effort toward the creation of general game-playing programs. One of these programs which models the human ability to learn good techniques for playing a game could develop its own rules-of-thumb and become a game-player which was as good as a program written for a particular game. At this time, however, these general game-playing programs typically require too much computer time and memory to be of much use for playing a specific game.

Nevertheless, a general game-playing rationale is necessary, as has been noted by Uhr:

> We want to be developing general principles and theories, moving toward more and more powerful, rather than more and more specific, mechanisms. (16)

If game-specific principles are developed without regard to this general rationale, it is possible that every new game to be modelled will require new methods (16).

The concept of rationality developed by game theory can provide a useful general framework for artificial intelligence game-playing. As will be seen, game theory has classified various types of games and developed not only solutions to these games, but an underlying motivation or rationale for game-playing.

CHAPTER 2

INTRODUCTION TO GAME THEORETIC CONCEPTS

A Player's Motivation: Rationality

Having discussed the need to examine game theory, we need to look at its basic concepts. The task of game theory is to understand and describe the process of rational behavior in any social situation.

According to game theory, a player is acting rationally if he is attempting to ensure for himself the highest possible payoff. All of a player's moves are oriented in this direction. This does not necessarily mean that he will make the move which has an expected payoff at least as great as that of any other payoff. Even if a player acts in cooperation with other players, he will never act irrationally, meaning that he will never cooperate to the point that he could achieve a better payoff if he acted individually.

We have already seen how this concept of rationality causes us to choose minimax strategies for two-person zero-sum sequential games, because minimax strategies are the "best" strategies for both players, and this same concept causes us to choose best-reply strategies for n-person zero-sum sequential games.

Philosophers have long debated whether any explanation of rational behavior can be valid. Shubik (12) and Harsanyi (5) point out that it is difficult to say human behavior can be adequately defined by a rationalistic theory. Many human tendencies seem to prevent this

behavior from being rational in a game theoretic sense. Error and what can be termed our "limited information-processing ability" often account for suboptimal actions. Additionally, insanity can cause very erratic behavior.

Although we may not be able to completely define all social behavior by a single rationalistic theory, we can try to understand irrational behavior as some deviation from rational behavior. For a reasonably acceptable theory of rationality, many observed behaviors will conform to normative rational behavior in at least some aspects. From this, one can obtain a heuristic measure with which to explain and predict some nonrational behavior (5).

For instance, in Figure 5 (page 17), if player 1 always chose his first (leftmost) move and the other players chose their second moves, they would be awarded their best possible payoffs of (1,-1,0). With the payoffs shown for each leaf node of the tree, we know that each player would be acting rationally in obtaining such a result. If player 3, however, felt that the payoffs at the leaf nodes were actually different from the given payoffs, he might choose a different strategy, yielding different payoff results, which would surprise players 1 and 2 because they expected player 3 to act differently. The first two players might feel player 3 acted irrationally, but, because of the context in which player 3 viewed the game, it can be argued that player 3 acted rationally in his context.

This theory of rational behavior, then, does not state how a person will act, but how a person should act, in a social situation.

However, in some fields, such as economics, political science, international relations, and some areas of sociology, game theory accurately describes and predicts desirable human behavior (5). Originally, game theory was organized as an attempt by John von Neumann and Oscar Morgenstern to describe economic behavior in Theory of Games and Economic Behavior.

Having defined rationality and assumed that every player will act rationally, we now have a basis for predicting the outcome of rational behavior.

The Definition of a Solution as an Equilibrium Situation

From the concept of rationality, it is clear that a player i wants a situation σ which will give him a payoff that is at least as large as the payoff for any other situation in the game, or

$$P_i(\sigma) \geqslant P_i(\sigma_i', \sigma^i) ,$$

where σ_i' is any strategy other than the one used in σ. This situation σ is said to be admissible for player i because of the above condition. A situation which is admissible for every player in the game is known as an equilibrium situation or equilibrium point, and the strategies involved in this situation are equilibrium strategies (17).

A game may possess many equilibrium situations, and it is at equilibria that solutions to games may be found, for, at an equilibrium point, no player can achieve a greater reward by deviating from his

equilibrium strategy. This self-enforcing characteristic of equilibrium situations tends to stabilize actions in a game toward an equilibrium point (12).

Shubik presents an intuitive view of equilibrium situations (12). His example is a self-policing system for a prison, shown in Figure 7. In this prison, there are no guards. Instead, there is a single blockhouse B in the middle of the prison which can open the only exit from the prison, a gate G. The blockhouse is surrounded by a barrier W which can only be opened by n buttons for the n prisoners. These n buttons are on the barrier, and each button can only be activated by one specific prisoner. When each prisoner pushes his specific button, the barrier is opened, and every prisoner is able to reach the blockhouse. However, only one prisoner is able to be in the blockhouse at a time. A prisoner in the blockhouse can press only one of two buttons inside. The first button opens the gate for ten seconds, allowing the rest of

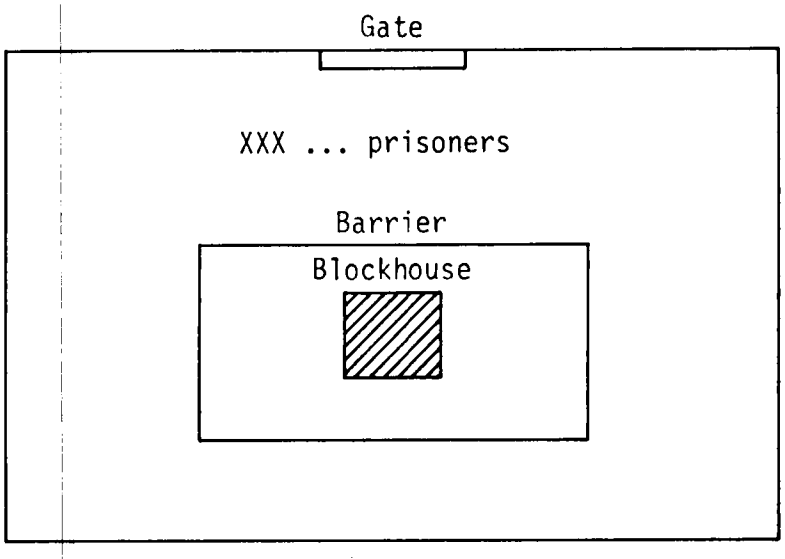


Figure 7. A self-policing prison system.

the prisoners to go free, but not the single prisoner, who is too far from the main gate to reach it in time. The second button activates gyros under the blockhouse, lifting it over the prison walls, but leaving the gate closed. If every prisoner is acting rationally, and his only goal is to escape from prison, then it will never be the case that all the prisoners will act together to allow one prisoner to escape, and no one prisoner will remain in prison while allowing the other prisoners to escape. The first button provides the prisoners with a motivation for allowing one prisoner to enter the blockhouse, while the second button motivates the single prisoner to enter the blockhouse. Thus, we have an equilibrium situation, at least in theory.

A few properties of equilibrium points must be given. These properties are true for all games. First, the payoff to a player at an equilibrium situation is at least as great as the payoff he would receive using a best-reply strategy, or

$$P_i(\overline{\sigma}) \geqslant P_i(\sigma_i^*, \sigma^i) ,$$

for any $\sigma^i \varepsilon \Sigma^i$, and $\overline{\sigma}$ being an equilibrium situation, and σ_i^* being a best-reply strategy. A best-reply strategy for a player yields the best payoff that a player can achieve for himself without the help of another player. However, if in choosing his strategy, a player employs a strategy yielding payoffs for the other players which are as good or better than their best-reply payoffs (the admissibility condition), and every player chooses such a strategy, then an equilibrium situation occurs.

Next, there may be more than one equilibrium point in a game. This is easy to see by symmetry in some unbiased three-person game, where players 1 and 2 play against player 3, or players 2 and 3 compete against player 1, etc.

The use of an equilibrium strategy does not guarantee the occurrence of an equilibrium situation (10). Consider the equilibrium situation

$$\bar{\sigma} = (\bar{\sigma}_1, \bar{\sigma}_2, \ldots, \bar{\sigma}_i, \ldots, \bar{\sigma}_n) .$$

In another situation

$$\tilde{\sigma} = (\tilde{\sigma}_1, \tilde{\sigma}_2, \ldots, \tilde{\sigma}_i, \ldots, \tilde{\sigma}_n) ,$$

it happens that

$$\tilde{\sigma}_i = \bar{\sigma}_i ,$$

although

$$\tilde{\sigma}^i \neq \bar{\sigma}^i$$

then it would also be true that

$$\tilde{\sigma} = (\tilde{\sigma}_i, \tilde{\sigma}^i) \neq (\bar{\sigma}_i, \bar{\sigma}^i) = \bar{\sigma} .$$

This last characteristic illustrates the fact that using an equilibrium strategy involves a risk.

Before proceeding to discuss equilibrium situations for different types of games, we will observe various ways of classifying games.

Classifications of Games

N-person and two-person games. There are a number of different ways to classify games. One of the most important distinctions is between two-person games and n-person games, where $n \geqslant 2$, although we usually consider $n > 2$. In general, two-person games are competitive, in that each player is competing against the other. However, n-person games can involve coalitions and the coordination of separate player's moves. Also, n-person game theory tends to be more complex, as will be seen when we discuss players' interests.

Constant-sum and nonconstant-sum games. Games can also be classified according to the nature of payoffs, or, more specifically, between constant-sum games and nonconstant-sum games. The addition of every player's payoff for a situation is equal to that of every other situation in a constant-sum game, which can be written as

$$P_1(\sigma) + P_2(\sigma) + \ldots P_n(\sigma) = c ,$$

where c is some constant, and the condition is true for every $\sigma \varepsilon S$. An important special case of constant-sum games is the set of zero-sum games, which is the above condition, where $c = 0$. Any constant-sum game can be converted into a zero-sum game by subtracting c/n from each player's payoff for every situation (1). In a nonconstant-sum game, the addition of every player's payoff for a situation is not the same for every situation. Nonconstant-sum games can be transformed into zero-sum games by the addition of a fictitious and nonparticipating player

whose payoff for a situation is equal to the negative of the sum of the other players' payoffs, or

$$P_{n+1}(\sigma) = -\sum_{i=1}^{n} P_i(\sigma) .$$

Cooperative and noncooperative games. Games can be classified as either cooperative or noncooperative. To define cooperation, one must look at the interests of the players because it is the similarity of interests that causes cooperation. Games in which the players have strictly identical interests occur when each player's payoff is strictly increasing with any other player's payoff, or, in other words, when a move beneficial to one player is beneficial to the rest. Cooperative games include those with strictly identical interests. The converse of this condition is the condition of strictly opposite interests. If all the players in a game have strictly opposite interests, then what is good for one player is bad for the rest of the players, and this is true for every player. Notice that only two-person zero-sum games can have this characteristic. In a two-person game, whatever payoff one person likes will be disliked by the other person, since a higher payoff for the first person will be a lower payoff for the second person. If a third person were added to the zero-sum game, even though the first two players may express an opposite ordering of two preferences, the third player may be indifferent to the two preferences already given.

Games of strictly opposite interests are noncooperative games. Should the interests of all the players in a game be neither strictly

identical nor strictly opposing, the interests are said to be mixed. With mixed interests, there is still the chance of agreements between players, and players who make such agreements can form coalitions. When agreements can be enforced, either through the rules of the game or the nature of the payoffs, then agreements can continue, and cooperation is encouraged. However, when agreements are not enforceable, then agreements are made only as long as they are profitable to the players involved. The nature of these games tends to be noncooperative. Thus, cooperative games include:

1. Games with strictly identical interests.
2. Games having mixed interests and enforceable agreements.

Noncooperative games include:

1. Games with strictly opposite interests.
2. Games having mixed interests and nonenforceable agreements (5).

Complete and incomplete information. If all the rules of a game are completely known by every player, the game is a game of complete information. Otherwise, the game is one of incomplete information. This latter characteristic is difficult to find solutions for, and most games dealt with by game theory are games of complete information. For our purposes, all the games we will discuss will be games of complete information.

Perfect and imperfect information. We have already observed that games can be represented by game trees. At a given node in the game tree, a particular player must choose his move. If, at every node in

the game tree a player can make his choice based on full information of all players' prior moves, or he can precisely trace the path from the root to the current node, then this is a perfect information game. The above is not true in an imperfect information game, such as the example game in Figure 8.

The nodes in level 2 are in an information set, meaning that, at that turn, the nodes are not distinguishable and player 2 cannot tell at which node he is located. Because of this, player 2 cannot make his move based on the knowledge of player 1's move. Player 1 and player 2 can be viewed as moving simultaneously. A simultaneous move is one in which all the immediate child nodes are in the same information set. A simultaneous movement game has only simultaneous moves. Perfect information games, however, have sequential moves and are all sequential games.

Not all imperfect information games are games of simultaneous movement. An example of this is Figure 9. Notice that not all the nodes on the second level are in the same information set. Thus, this is not a simultaneous move.

Perfect information games are generally represented as a sequence of moves over the entire length of the game, such as a game tree. A game in this form is said to be in extensive form (10).

In a game of imperfect information, a player cannot always make his move based on the last move of the last player. Instead of depicting this game in extensive form, it can be represented in normal form, where each player's strategies are listed, and, for each situation in the game,

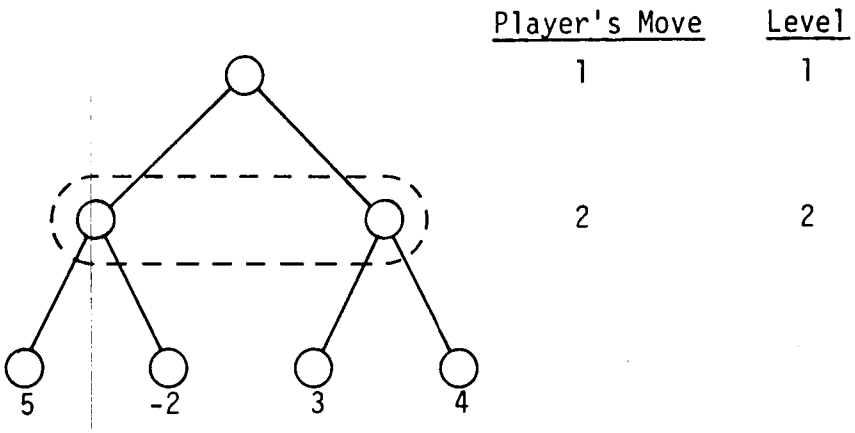


Figure 8. Imperfect information game.

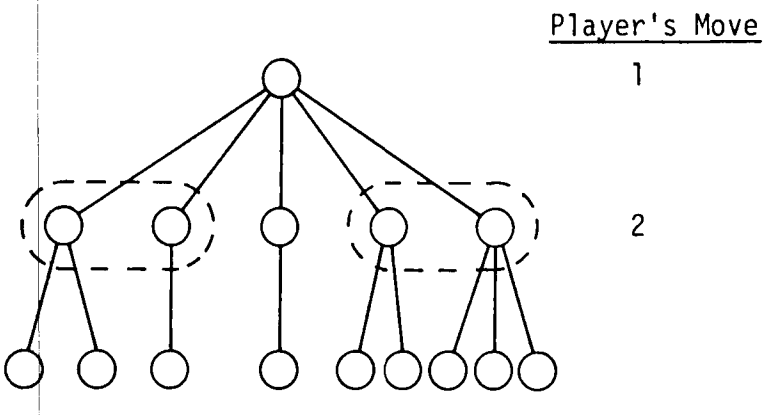
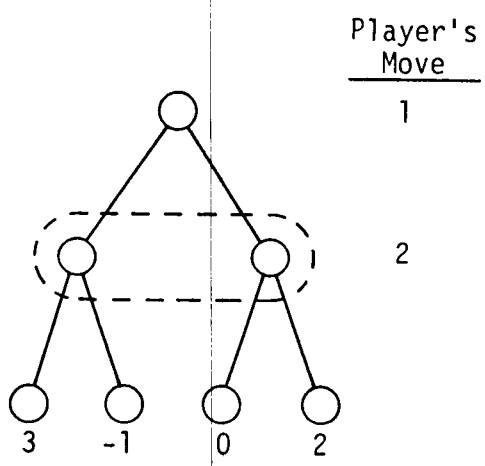


Figure 9. Imperfect information without simultaneous moves.

a resulting payoff vector is given. In Figure 10, a game is shown in extensive form and normal form. This is a two-person zero-sum imperfect information game.

We can number the levels in a game tree as 1st, 2nd, ..., rth, which are the r levels at which a particular player can make a move. The numbers x_1, x_2, ..., x_r are the serial numbers of the branch chosen by this player at each of the r levels. The sequence X_1, X_2, ..., X_r is, then, a unique strategy for the player.

It is necessary to classify games in these various ways to be able to investigate solution concepts for different game classifications. Often, we can only understand solution concepts for one class at a time, considering solutions in a case-by-case basis. Following is an introduction to solutions of games.



Extensive form

Player 2's strategies

Player 1's strategies	1	2
1	3	-1
2	0	2

Normal form

Figure 10. Game represented in extensive and normal form.

CHAPTER 3

SOLUTIONS TO VARIOUS CLASSES OF GAMES

For the different game classes, game theory has developed solutions, all of which are based upon the rationality assumption previously described. One way of fully describing rationality is by listing the solutions to the various game classes. Because game theory is concerned with understanding decision-making, finding optimal solutions is not its primary concern, but optimal solutions are often a byproduct.

Two-Person Zero-Sum Games

In examining solutions to various classes of games, the simpler classes should be considered first. The simplest class of games is the set of two-person zero-sum games, which are games of strictly opposite interests.

We have already seen that the minimax technique is sufficient for providing solutions to two-person zero-sum perfect information games, as discussed in Chapter 1. For two-person zero-sum imperfect information games, the minimax technique is still necessary, but will be applied in a different way.

First, two-person zero-sum imperfect information games must be represented in normal form. In normal form, it may be clear to one of the players which strategy he should choose. For example, consider the game in Figure 11. Player 2 sees that he should always choose strategy 2. If player 1 chose strategy 1, player 2 would get his best payoff with

		Player 2's strategies		
		1	2	3
Player 1's strategies	1	3	-1	0
	2	4	0	1
	3	1	-3	2

Figure 11. Game with one saddle point.

strategy 2. This is also true if player 1 chose strategy 2 or 3. Knowing that player 2 will always want to choose strategy 2, player 1 has a choice of choosing his strategy 1, 2, or 3 with payoffs of -1, 0, -3, respectively. Obviously, strategy 2, with a payoff of 0 is 1's best choice. Thus, we know that both players will choose their second strategies, and the expected payoff, or game value is 0. The value of a game, or the game value, is the expected payoff of a player in the given game.

This is a saddle point solution, meaning that it can be rationally assumed which pure strategies will be chosen. A quick way to determine whether an entry in a game matrix is a saddle point is to check whether it is the smallest entry in the row and the largest entry in the column. A game may have more than one saddle point. Each of these saddle points will have the same payoff. In Figure 12, there are two saddle points, both with a value of -1.

		Y's strategies		
		1	2	3
X's strategies	1	-2	1	-3
	2	-1	2	-1

Figure 12. Game with two saddle points.

In a game which possesses a saddle point solution, it is obvious to each player which strategy should be chosen. If a game possesses more than one saddle point solution, the players may have a choice as to which strategy each should choose. According to game theory, it will not make any difference which of these strategies are chosen.

Many games do not possess saddle point solutions. In these games, although it may not be clear to each player which strategies they want to choose, it may be that some strategies will never be chosen because of their relatively low payoffs. These unwanted strategies are referred to as dominated strategies, because other strategies are preferred over them. A strategy σ_i' is dominated by a strategy σ_i'' if the payoff for player i using σ_i'' is at least as much as he would receive in any situation using σ_i', or

$$P_i(\sigma_i'', \sigma^i) \geqslant P_i(\sigma_i', \sigma^i)$$

for any $\sigma^i \varepsilon \Sigma^i$.

If a strategy is dominated by at least one other strategy, the dominated strategy can be excluded from the set of strategies. The

search for dominated strategies can then continue. When this process is completed, only undominated strategies should exist in the player's strategy set.

Strategies 1 and 3 are dominated by strategy 2 for player X in Figure 13. Similarly, player Y's second and fourth strategies are dominated by his third strategy. The resultant game involves a choice of two strategies for both players.

Y's strategies

X's strategies

	1	2	3	4
1	2	2	-1	1
2	4	2	0	3
3	1	-2	-3	2
4	-1	4	1	2

→

	1	2	3	4
2	4	2	0	3
4	-1	4	1	2

→

	1	3
2	4	0
4	-1	1

Figure 13. Game reduced through domination of strategies.

Removing dominated strategies from a game reduces the number of potential situations which may be equilibrium solutions. It is often possible that a saddle point exists for this reduced game. Artificial intelligence may be able to reduce the number of considered strategies even further by using heuristics which can recognize certain strategies that will "probably" not be included in an equilibrium situation.

After it is determined that a game does not have a saddle point and after all unwanted strategies have been eliminated through domination, then we have a game in which each player wants to implement each strategy only when his opponents are using other particular strategies.

Consider the game in Figure 14. Player X wants to use strategy 1 when player Y is using his second strategy. However, when X uses strategy 1, Y wants to use strategy 1. Alternately, player X wants to use his first strategy when player Y is using his first strategy, but, when X is using his first strategy, Y wants to use his second strategy.

		Y's strategies	
		1	2
X's strategies	1	1	2
	2	2	1

Figure 14. Game without a saddle point.

Thus, there is no particular strategy that either X or Y always wants to play. Instead, X wants to play strategy 1 with probability Q and strategy 2 with probability 1 - Q.

Earlier, it was discussed that a strategy σ_i for player i is a probability vector of the form

$$\sigma_i = (\sigma_i^1, \sigma_i^2, \ldots, \sigma_i^{K_i}) ,$$

where σ_i^j is the percentage of the time that player i wants to play pure

strategy a_i^j. The saddle point solution for Figure 11, page 34, can then be represented as

$$\sigma_1 = (0, 1, 0) \ , \quad \text{and} \quad \sigma_2 = (0, 1, 0) \ ,$$

meaning that player 1 always will play his second strategy, and player 2 always will play his second strategy.

For the game in Figure 14, X will choose a strategy given by

$$\sigma_x = (Q, 1 - Q) \ , \text{ where } \ 0 < Q < 1 \ .$$

Mixed strategies, such as this, are also known as randomized strategies because a random number generator distributed according to the probability for each pure strategy can be used to choose which pure strategy should be used.

When a game does not possess a solution in pure strategies, the solution in mixed strategies can be obtained from the simplex method of linear programming. Given a game matrix with the following entries

$$\begin{matrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{m1} & a_{m2} & & a_{mn} \end{matrix} \ ,$$

we can express player Y's expectations of winning by

$$\begin{array}{l} a_{11}Y_1 + a_{12}Y_2 + \dots a_{1n}Y_n \leqslant V \\ a_{21}Y_1 + a_{22}Y_2 + \dots a_{2n}Y_n \leqslant V \\ \quad\vdots \\ a_{m1}Y_1 + a_{m2}Y_2 + \dots a_{mn}Y_n \leqslant V \end{array},$$

where V is the value of the game and Y_1, Y_2, ..., Y_n is the probability of using strategy 1, strategy 2, ..., strategy n, respectively (1). Dividing each equation by V, we get

$$\begin{array}{l} \frac{a_{11}Y_1}{V} + \frac{a_{12}Y_2}{V} + \dots \frac{a_{1n}Y_n}{V} \leqslant 1 \\ \frac{a_{21}Y_1}{V} + \frac{a_{22}Y_2}{V} + \dots \frac{a_{2n}Y_n}{V} \leqslant 1 \\ \quad\vdots \\ \frac{a_{m1}Y_1}{V} + \frac{a_{m2}Y_2}{V} + \dots \frac{a_{mn}Y_n}{V} \leqslant 1 \end{array}$$

or

$$\begin{array}{l} a_{11}\overline{Y}_1 + a_{12}\overline{Y}_2 + \dots a_{1n}\overline{Y}_n \leqslant 1 \\ a_{21}\overline{Y}_1 + a_{22}\overline{Y}_2 + \dots a_{2n}\overline{Y}_n \leqslant 1 \\ \quad\vdots \\ a_{m1}\overline{Y}_1 + a_{m2}\overline{Y}_2 + \dots a_{mn}\overline{Y}_n \leqslant 1 \end{array}$$

where $\overline{Y}_i = Y_i/V$.

Since Y_1, Y_2, ..., Y_n are all probabilities which must add up to 1, we can express this as

$$Y_1 + Y_2 + \dots Y_n = 1 ,$$

or

$$\overline{Y}_1 + \overline{Y}_2 + \dots \overline{Y}_n = \frac{1}{V} .$$

Because we arbitrarily chose to express V, the game value, as the average payoff given to player X, player Y's goal is to minimize V, or maximize 1/V. The result is a linear programming problem, stated (1):

$$\text{maximize} \quad \overline{Y}_1 + \overline{Y}_2 + \dots \overline{Y}_n$$

$$\begin{aligned}
\text{subject to} \quad & a_{11}\overline{Y}_1 + a_{12}\overline{Y}_2 + \dots a_{1n}\overline{Y}_n + S_1 + 0S_2 + \dots 0S_n = 1 \\
& a_{21}\overline{Y}_1 + a_{22}\overline{Y}_2 + \dots a_{2n}\overline{Y}_n + 0S_1 + S_2 + \dots 0S_n = 1 \\
& \vdots \qquad \vdots \qquad \vdots \qquad \vdots \\
& a_{m1}\overline{Y}_1 + a_{m2}\overline{Y}_2 + \dots a_{mn}\overline{Y}_n + 0S_1 + 0S_2 + \dots S_n = 1
\end{aligned}$$

This problem can be solved by the simplex method of linear programming, which is demonstrated in Appendix A. Through this method, a probability vector for each strategy is calculated for both players.

It should be noted that, in 1979, an alternative method, faster than the simplex method, was developed by L. G. Khachian, a Russian mathematician. This method utilizes Euclidean geometry in n-dimensional space to converge upon a solution (6).

Two-Person Nonzero-Sum Games

Thus far, we have considered only two-person zero-sum games, which we know are noncooperative games. Next, let us consider two-person nonzero-sum games.

Consider the game in Figure 15. Each entry in the game matrix is a vector (a, b), where a is X's payoff and b is Y's payoff. X prefers situation (3, 1), giving him a payoff of 9, while Y prefers situation (1, 3), giving him a payoff of 9. However, if both players chose the strategies which gave them their highest payoffs, the resulting situation (3, 3) would pay each player 0 (11).

		Y's strategies		
		1	2	3
X's strategies	1	0,0	0,0	1,9
	2	0,0	5,5	0,0
	3	9,1	0,0	0,0

Figure 15. Two-person nonzero-sum game.

If both players were to communicate, they could both bargain back and forth until they finally settled on the mutually acceptable situation (2, 2), with a payoff of 5 for each player. Many different methods of dealing with bargaining have been presented. However, there does not appear to be a dominant one. Threats are a large part of the bargaining process.

Each bargaining solution method graphs the payoffs of each entry as a point in n-dimensional space and forms a convex hull of these points. Among these points is a status quo point, as specified by the method, from which the players can begin negotiations (1).

The set of points on the perimeter of this convex hull is designated the negotiation set in which all rational players will be able to find a negotiable solution. Obtaining a solution is then the process of moving along a line, determined by the bargaining method, from the status quo point to a point in the negotiation set, and the point arrived at is the solution. One solution of the game in Figure 15 is shown in Figure 16.

If a game of this class does not have communication among the players, it is often the case that the negotiation set will be smaller, because the number of believed threats is reduced. From this reduced negotiation set, players may or may not reach the same solution.

New Considerations for N-Person Games

Before continuing into the class of n-person games, some concepts must be explained. Cooperative games are usually represented in characteristic function form, which represents each possible coalition and the corresponding payoff paid to the entire coalition. A division of the payoff among the players in the coalition is termed an imputation and there may be an infinite number of imputations for a coalition. It is assumed that the payoff is the total of the payoffs of each individual, and this collective payoff is shared.

Figure 17 represents a three-person game in characteristic function form, where $V(a_1, a_2, \ldots, a_k) = U_{12 \cdots k}$ is the payoff $U_{12 \cdots k}$ given to the players $a_1, a_2, \ldots, a_k$ who are members of a coalition. Notice that no player can receive any payoff gain without player 2.

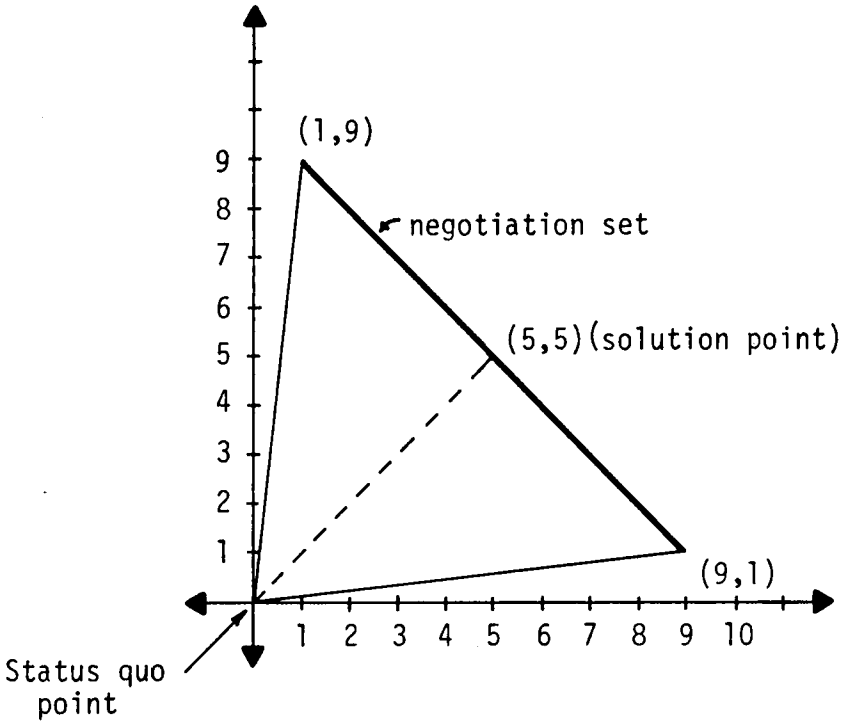


Figure 16. Bargaining set solution of a game.

$$V(\emptyset) = 0$$

$$V(1) = 0 \qquad V(2) = 1 \qquad V(3) = 0$$

$$V(1,2) = 3 \qquad V(2,3) = 3 \qquad V(1,3) = 0$$

$$V(1,2,3) = 7$$

Figure 17. Game in characteristic function form.

We must also introduce the concept of group rationality, for, when players form a coalition that coalition tends to act as one player, having its own goal, until it breaks up. A coalition has group rationality when it exhibits rational behavior similar to an individual player.

The characteristic function form has two properties (5). First, the empty coalition receives no payoff, and, second, if two groups of players form a coalition, then the players in each group will receive at least as much as they would if they did not form the coalition. The second property ensures individual rationality, because it would not be individually rational for a player to join a coalition if it put him in a worse position. These two properties can be expressed as

$$V(\emptyset) = 0 ,$$

and

$$V(i,j) \geqslant V(i) + V(j) ,$$

where $i \cap j = \emptyset$.

Because group rationality must preserve individual rationality, an individual will remain in a coalition only as long as it is profitable for him to do so. A player outside of a coalition may bargain with a member of the coalition to entice him to leave the coalition and form a new coalition between these two players. Rational players are aware of their most profitable action, whether it is joining a coalition or leaving one (1).

A two-person nonzero-sum game, as well as a nonzero-sum game with more players, can be converted to a zero-sum game by adding a fictitious

nonparticipating player who, for every situation, receives the negative of the summation of the other player's payoffs in that situation (11). This transformation eliminates the problem of finding a solution using the convex hull of payoffs. Coalitions are possible, however, and the addition of another player brings the added difficulty of determining whether a coalition could form including the fictitious player. Of course, the fictitious player cannot actually join a coalition, but the real players in the coalition must share the payoffs in such a way that the fictitious player would want to join.

The two important questions in n-person game theory are (11):

1. Which are the most likely coalitions to form?
2. How will coalition formation affect the outcome of a game?

Players can form coalitions if they can agree upon an imputation. Although there is the possibility of an infinite number of imputations, many imputations will not be considered because they are dominated by another imputation. An imputation dominates another if the majority of players prefer the first imputation over the second. As an example, the imputation (50, 10, -60) dominates the imputation (-10, -20, 30) because both player 1 and player 2 prefer the first imputation. Agreements are enforceable if threats are strong enough to force an agreement. A threat is the possibility that a play will receive an undesirable payoff due to the actions of another player or group of players. In the above example, player 1 may threaten player 2 by saying that player 1 will form a coalition with player 3 using the imputation (-10, -20, 30), unless player 2 joins him in a coalition using the imputation (50, 10, -60).

Only if players make concessions to each other can agreements be made. Concessions are made if the refusal to accept a concession is a risk that the involved players are unwilling to take. If one of these players wanted to forego such a concession in hope that he could achieve better terms, the risk of encountering a conflict from such a refusal is even greater.

Solutions for N-Person Games

Von Neumann and Morgenstern defined a solution of an n-person game as a set of imputations with internal and external stability. The set has internal stability if no imputation in this set dominates another imputation in the set, and has external stability if every imputation not included in the solution set is dominated by at least one imputation in the set (10).

This solution concept has some problems, however. Consider the game of dividing one hundred and twenty dollars by a majority vote of three players. Dollars cannot be divided further. If a majority of the three players can agree on how to divide the money, they can keep the money (1). The characteristic function form is given in Figure 18.

$$V(\emptyset) = 0$$

$$V(A) = 0 \qquad V(B) = 0 \qquad V(C) = 0$$

$$V(AB) = 120 \qquad V(BC) = 120 \qquad V(AC) = 120$$

$$V(ABC) = 120$$

Figure 18. Divide the money game.

One solution for this game is the set of imputations (60, 60, 0), (60, 0, 60), and (0, 60, 60). In each of these imputations, two players agree to divide the money equally among themselves and exclude the remaining player (11). One of these imputations will be chosen for the game. According to von Neumann and Morgenstern, the imputation chosen will reflect some social norm. However, it is not known how to find this norm. Thus, this solution concept does not necessarily choose a unique imputation for a game.

There may also be a number of other solutions for a game, as is the case for the last game (1). Such a solution is the set of imputations

(32, 0, 88)
(32, 1, 87)
(32, 2, 86)

(32, 86, 2)
(32, 87, 1)
(32, 88, 0)

Hence, it is also not possible to choose one solution from the others with the present concepts.

Further attempts have been made to define solution concepts which will improve the selection of a game solution. Luce's theory of stability is a solution concept which determines all stable imputations. The imputations which can be considered, termed permissible imputations, are those which involve only a slight change from another imputation, reflecting Luce's assertion that changes of coalition patterns occur only in small steps because alliances tend to continue. Out of these permissible imputations, stable imputations are determined. An

imputation is stable if no other permissible imputation is preferred by the majority of players. There may be many stable imputations in a game, and, frequently, this set of imputations is more restricted than the von Neumann and Morgenstern solution sets.

The strong solution, described by Vickrey, restricts the number of imputations which could be included in a von Neumann and Morgenstern solution, with the added restriction that a player who attempts to divert from a solution and then attempts to return to that solution will be penalized in his payoff. For many games, this has proved to greatly restrict the possible set of solutions.

Shapley's concept of power is another approach toward minimizing the number of acceptable solutions in a game. Shapley's observation was that some players are more sought after than others in coalition formation. The measure of this is known as a player's power. A group of players could form a coalition if the sum of their power indexes is not less than a given coalition formation power level. However, if their sum is less than this power level, there exists some motivation for the coalition not to form. Various methods exist for the determination of power indices and the coalition formation level.

One of the most recent solutions was developed by Harsanyi and Selten (4), who claim that their solution is for any cooperative or noncooperative game, whether it is a game of complete or incomplete information. To find a solution to a given game, this method divides the game into a number of component games, each, of which, is also divided into a number of component games, etc. There is a limited

number of these component games, however, which are created by the agent normal form, a form of the game between extensive form and normal form.

Each game or component game reaches a solution in two steps. First, for each player i, all other players express their initial expectations about the pure strategies of player i in a probability vector E_i. Using these vectors E_i, a tracing procedure converges upon one specific equilibrium point, tracing the solution from a best-reply solution to the unique solution. This solution method not only solves the problem of finding a determinate solution, but also helps to explain one behavior (rational) as a deviation of another behavior (best-reply).

In summation, here is a listing of solution concepts for various classes of games.

1. Two-person zero-sum games: saddle point, domination, or simplex solution.
2. Two-person nonzero-sum games: bargaining method on a convex hull.
3. N-person games: the von Neumann and Morgenstern solution, Luce's stable solution, Vickrey's strong solution, Shapley's power index solution, or Harsanyi and Selten's solution.

CHAPTER 4

SIMULTANEOUS MOVEMENT GAMES

Description of Simultaneous Movement Games

Games of perfect information, including the two-person zero-sum games used in artificial intelligence, can best be thought of as games in which each player moves sequentially and each player's move is based on all moves prior to his. Such is the case with the trivial game represented as a tree in Figure 19. Using the minimax concept, we know that both player 1 and 2 will use their first strategies, resulting in a zero-sum payoff of 1.

If the above game was a game of imperfect information, player 2 could not base his moves on what the other player did and, in effect, would be moving simultaneously with player 1. This game could be shown with all of the nodes at level 2 in an information set, which is illustrated in Figure 20. However, this type of game is best depicted as a game matrix which helps to eliminate the suggestion of sequential movement (Figure 21). Here, a player's strategy represents an individual move because of the short length of the game. In actuality, a strategy represents a method for selecting a player's move. From the game in Figure 3, page 12, we can obtain a single value representing the expected zero-sum payoff, as discussed earlier.

There are games which contain a combination of perfect and imperfect information. An important subset of these games are those

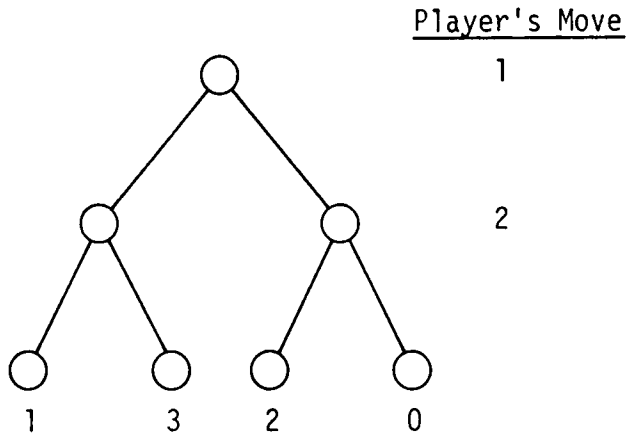


Figure 19. Two-level sequential movement game.

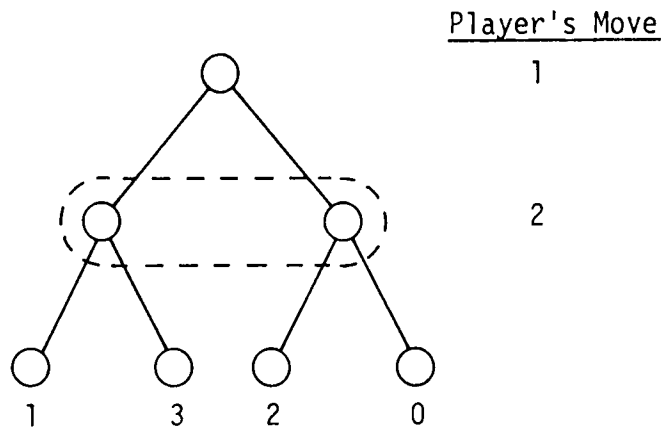


Figure 20. Two-level simultaneous movement game.

Player 2's strategies

Player 1's strategies

1	3
2	0

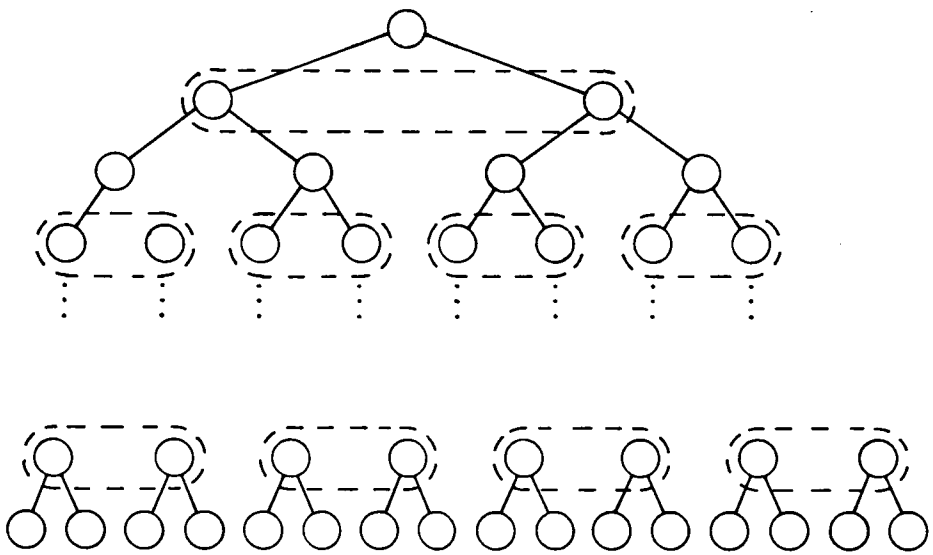
Figure 21. Normal form of Figure 20.

games in which there is imperfect information during each turn, but perfect information between turns, a turn being the event of each player making one move. In these games, the players move simultaneously each turn, but, after each turn, every player knows the outcome of the moves. A game tree for this type of game is depicted in Figure 22. Notice that every two levels represent a turn up to k turns, and between each turn, all the nodes are in the same information set.

The first two levels represent a game of the type shown in Figure 21. However, there are no payoffs at the nodes in level 2 because these are not leaf nodes. Instead, we obtain a payoff at one of these nodes by evaluating the two-level games below its second level nodes, etc. This process continues until the second level nodes are leaf nodes, at which we have payoff values to evaluate the games at the last levels. When a game value is obtained for one of these games, this game value is used as the payoff value of one of the second order nodes in the game above it. Thus, the game values are sent back up the tree until we can evaluate the original two-level game, and we have a value for the entire game.

Algorithm for Solving Simultaneous Movement Games

We can describe this process with an algorithm, named ASMA (A Simultaneous Movement Algorithm) shown in Figure 23. Minimax for perfect information cannot be applied after a turn because the players are moving simultaneously each turn, and there is no single player who can choose the best move. However, we can apply minimax for imperfect information and other techniques during the simultaneous moves.



Player's Move	Turn	Level
1	1	1
2		2
1	2	3
2		4
⋮	⋮	⋮
1	K	2K-1
2		2K

Figure 22. 2K-level simultaneous movement game.

```
MAIN:  PROCEDURE OPTIONS(MAIN);
       DECLARE K FIXED STATIC EXTERNAL;  /* Number of board regions. */
       GET LIST(K);
       BEGIN;
             DECLARE BOARD(K) FIXED;
             CALL CREATE(BOARD);
             PUT SKIP LIST('GAME VALUE IS ',ASMA(BOARD));
             /* SOLVE THE GAME. */
             STOP;
             END;
       END MAIN;
```

(a) Main program.

Figure 23. ASMA.

```
/* A SIMULTANEOUS MOVEMENT ALGORITHM.                          */
/* ASMA EVALUATES A GAME BY                                    */
/*   1.DETERMINING ALL OF A PLAYER'S POSSIBLE NEXT MOVES. */
/*   2.CREATING A GAME MATRIX FROM THESE MOVES                 */
/*   3.FILLING IN THE MATRIX BY EVALUATING THE SUBGAMES        */
/*     REPRESENTED BY EACH PAIR OF MOVES                       */
/*   4.OBTAINING A GAME VALUE FROM THE GAME MATRIX             */
ASMA: PROCEDURE(BOARD) RECURSIVE RETURNS(PAYOFF);
     /* FUNCTIONS USED:                                          */
     /* ESTIMATE - RETURNS AN OVERESTIMATE OF THE NUMBER OF      */
     /* POSSIBLE MOVES FOR A PLAYER FROM A GIVEN BOARD SITUATION. */
     /* EVALUATE - RETURNS THE GAME VALUE FOR THE GIVEN MATRIX.  */
     /* GAME_END - RETURNS 'TRUE' IF NEW_BOARD IS AN END         */
     /* OF THE GAME.                                             */
     /* GENERATE - RETURNS A VECTOR OF MOVES FOR THE             */
     /* GIVEN PLAYER.                                            */
     /* PAYOFF - RETURNS THE ZERO-SUM PAYOFFS FOR A BOARD        */
     /* REPRESENTING A GAME ENDING.                              */
     DECLARE ESTIMATE ENTRY(FIXED,CHAR) RETURNS(FIXED);
     DECLARE EVALUATE ENTRY ((*)FIXED,FIXED,FIXED) RETURNS(FLOAT);
     DECLARE GAME END ENTRY (FIXED) RETURNS(BIT(1));
     DECLARE GENERATE ENTRY ((*)FIXED,CHAR,(*)FIXED,FIXED) RETURNS(FIXED);
     DECLARE PAYOFF ENTRY((*)FIXED) RETURNS(FIXED);
     DECLARE GVALUE FLOAT; /* THE GAME VALUE. */
     DECLARE K FIXED STATIC EXTERNAL;
     DECLARE (BOARD(*),I,J,
             N1,N2, /* THE ESTIMATED NO. OF POSSIBLE MOVES FOR X AND Y */
             NY,NY) /* THE ACTUAL NO. OF POSSIBLE MOVES FOR X AND Y    */
             FIXED;
     N1=ESTIMATE(BOARD,'X'); /* APPROX. NO. OF X'S MOVES */
     N2=ESTIMATE(BOARD,'Y'); /* APPROX. NO. OF Y'S MOVES */
     BEGIN; /* 1.DETERMINE ALL OF X & Y'S POSSIBLE NEXT MOVES. */
             DECLARE (XMOVES(N1), YMOVES(N2))CHAR(20) VARYING:
             CALL GENERATE(BOARD,'X',XMOVES,NX); /* GENERATE X'S MOVES */
             CALL GENERATE(BOARD,'Y',YMOVES,NY); /* GENERATE Y'S MOVES */
             BEGIN;
                 DECLARE NEW_BOARD(K) FIXED;
                 /* 2.CREATE A GAME MATRIX FROM THESE MOVES. */
                 DECLARE GMATRIX(NX,NY) FLOAT;
                 /* 3.FILL IN THE GAME MATRIX BY EVALUATING THE */
                 /* SUBGAMES REPRESENTED BY EACH PAIR OF        */
                 /* MOVES.                                      */
                 DO I=1 TO NX;
                    DO J=1 TO NY;
                       CALL MAKE_MOVE(XMOVES(I),YMOVES(J),
                                      BOARD,NEW_BOARD);
                       IF (GAME_END(NEW_BOARD))
                       THEN DO; /* IF THIS IS A LEAF NODE, */
                                /* THEN RETURN THE VALUE   */
                                /* OF THIS BOARD SITUATION. */
                                GMATRIX(I,J)=PAYOFF(NEW_BOARD);
                                END;
                    ELSE DO; /* OTHERWISE, EVALUATE THE */
                             /* SUBGAME BELOW THIS      */
                             /* NODE.                   */
                             GMATRIX(I,J)=ASMA(NEW_BOARD)
                             END;
                    END;
                 END;
             /* 4.OBTAIN A GAME VALUE FROM THE */
             /* GAME MATRIX.                   */
             GVALUE=EVALUATE(GMATRIX,NX,NY);
             END;
         END;
   RETURN(GVALUE); /* RETURN THE VALUE OF THIS GAME. */
   END ASMA;
```

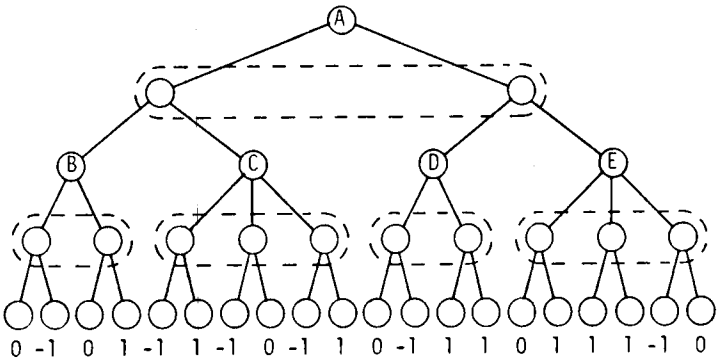
(b) Subroutine.

Figure 23 (continued)

ASMA can be used with either an implicit or explicit game tree. For an implicit game tree, the main program reads in a description of the game board. Then ASMA generates both player's moves each turn and evaluates the game matrix at that point in the game tree. The procedure is almost the same for an explicit game tree, except that Create reads in the game tree, and Generate only retrieves the proper nodes from the existing game tree.

As an example, consider the game given by the game tree in Figure 24(a). The game ends after two turns. After inputting the game tree in the main program, we call ASMA for the first time, telling the algorithm to evaluate the game below the root of the tree, node A. This gives us the game matrix in Figure 24(b). To obtain values for each entry in this matrix, ASMA is recursively called for each subgame represented by an entry, yielding the separate game matrices in Figure 24(c). Again, ASMA is called for each entry in these matrices, but, because the entries all represent leaf nodes, the algorithm returns the zero-sum payoff of each leaf. When the games in each of these four matrices have been evaluated, the game value for each matrix is returned to the initial occurrence of ASMA. Each of the returned game values fills in an entry in Figure 24(b). Lastly, the original game matrix is evaluated, and its game value is returned to the main program, yielding the game value of the entire game tree.

Each of the games in Figure 24(c) have saddle points yielding game values which can be used to fill in the original game matrix in Figure 24(b), which gives us Figure 24(d). We have now returned to



Player's Move	Turn	Level
1	1	1
2		2
1	2	3
2		4
		5

(a)

Player 2's move

Player 1's move	Game value of game below node B	Game value of game below node C
	Game value of game below node D	Game value of game below node E

(b)

Player 2

Player 1

0	-1	-1	1	0	-1	0	1
0	1	-1	0	1	1	1	1
		-1	1			-1	0

(c)

Player 2's move

Player 1's move

0	-1
1	1

(d)

Figure 24. Game to demonstrate ASMA.

the first call of step 2. Having filled in the original game matrix, the game in part a can be evaluated to yield a game value of 1, and, thus, we return from the original call of the algorithm providing the entire game with a value of 1.

The agent normal form of a game, developed by Harsanyi and Selten, is also useful in describing ASMA's solution procedure (4). In this form, each two-level subgame is thought of as being played by agents of the two players, each having payoffs and rationalities in common with the players they represent. As each pair of agents plays the subgame they are concerned with, they send the game value up the tree to the next subgame, so that two more agents can play their subgame. Each subgame can be played when all the subgames under this subgame are played. When there is only one two-level subgame left, this is the last game evaluated.

Now that we have defined an algorithm for the solution of two-person zero-sum simultaneous movement games, we can consider an application.

A Description of the Game REDIPT

The application of simultaneous movement games with which we will be concerned is a variant of the game DIPLOMACY (2), which we will call REDIPT (Revision of DIPLOMACY for Two-Persons). The simultaneous movement algorithm ASMA is incorporated in the game.

Basically, REDIPT involves two players, X and Y, in control of two respective countries, which are each divided up into a number of

regions. Some of these regions contain economic centers, which are large cities with economic resources of significant importance. Also controlled by each player are a number of armies, no more than one per region, which allow a player to control a region. By moving armies from region to region, each player tries to control as many economic centers as possible.

The game board environment is as follows:

1. The game board is divided into a number of regions, each labeled with a number, some of which make up one country, with the remainder constituting the other country.
2. Some regions are designated as possessing economic centers which remain for the length of the game.
3. A region belongs to a player until an enemy army occupies the region, at which time the region now belongs to the other player. Because of this, the collection of regions owned by a player may be constantly changing.
4. Both players begin the game with a specific number of armies. This number cannot increase, although it may decrease as a result of a conflict. All, some, or none of these armies may be moved by the owning player each turn. There can be no more than one army per region.

The player who has a net gain in the number of economic centers owned at the end of the game is the winner. Since there is never an unowned economic center, REDIPT is a zero-sum game.

REDIPT is played in turns, and the game ends after a specific number of turns, which is known at the start of the game. A turn is described in the following way.

1. Both players record their moves secretly, and then reveal their move orders to each other when both are finished. These orders are executed simultaneously.
2. A player's move consists of moving all, some, or none of his armies.
3. An army which is moved can be moved to any one region which is adjacent to the region it occupies and cannot move any more that turn.
4. A conflict involves more than one army trying to occupy a region. Any conflict encountered after the execution of both move orders, must be resolved according to the conflict order, to be described later.
5. An army which does not move out of the region it occupies is holding in its region. A hold must also be recorded.
6. Instead of holding or moving, an army may support another army which is holding or moving into another region. Supports must be recorded. An army may support another army if the supporting army is adjacent to the region that is being moved into by the other army, or if the supporting army is adjacent to the region in which the other army is holding. An army cannot support another army doing a support.

7. When an army is moving from one region to another, the move is recorded in the form r_1 - r_2, where r_1 and r_2 are the regions the army is moving from and to, respectively. A hold is recorded as r_1 - r_1. A support is recorded as r_1Sm_s, where r_1 is the region which is occupied by the supporting army, and m_s is the move being supported. Examples include:
   (a) 2-3; an army moves from region 2 to 3.
   (b) 4-4; an army holds in region 4.
   (c) 7S1-5; an army in region 7 is supporting the army in region 1 moving to region 5.

Conflicts are resolved according to the following rules.

1. An army moves or holds with the strength of itself and all uncut supports. For example, an army which is moving and has two supporting armies has a strength of three.
2. A support is cut if it is attacked from a region other than the one it is supporting a move or hold into.
3. If two or more armies of equal strength move into the same region, each of these moves fails.
4. If two armies of equal strength attempt to move into each other's regions, both moves fail. This is a "mirror move."
5. If one of the armies in the two preceding rules is, instead, greater in strength than all opposing armies moving into the same region, then the stronger army succeeds in its move, and all other armies fail in their moves.

6. An army remaining in its own region through a hold or a failed move, if opposed by a stronger army, is dislodged from this region and must retreat.
7. If an army is ordered to move into a region in which an army is holding or has failed in its move, the former army fails in its move unless it has greater strength than the latter army, causing the latter army to be dislodged.
8. An army can never succeed in dislodging another friendly army.
9. Two or more armies of the same strength moving into the same region from other regions will all fail in their moves, and an army inside this target region is not affected.
10. If two armies are moving into a region and one is dislodged by a move from that region.
11. After all moves have been executed, all dislodged armies are retreated into an adjacent region chosen by the owning player. Armies may not retreat into occupied regions, regions into which a move failed, or the region from which the attack came.
12. An army which cannot retreat is, instead, eliminated from the game.

The Implementation of REDIPT

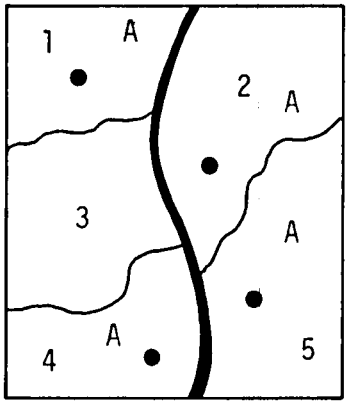
The game REDIPT has been modeled by a game-playing program of the same name. Both players are automated, or controlled by the program,

and are rational, in a game theoretic sense. The inputs to the program are a game board, including initial positions, and the number of turns in the game, and the game is analyzed according to ASMA, yielding a solution.

The adjacent regions in the game board are represented as an adjacency graph in the two-dimensional array BORDERS. Economic centers, armies, and region ownership are information stored in the arrays ECON_CENTR, OLD_BORD, and COUNTRY. Figure 25 represents a sample game board in array representation.

After the creation of the game board environment, the program enters ASMA which explores every movement possibility. For each possibility, ASMA has three distinct stages:

1. Generation of all possible next moves for both players.
2. Generation of a game matrix from the lists of moves. For each combination of a move for X and a move for Y:
   (a) Resolve any conflicts.
   (b) Find the expected zero-sum payoff for this pair of moves. If we have reached the turn limit, this is done by finding the net gain in regions owned by X. Otherwise, we call ASMA for the game below this pair of moves.
3. Evaluation of the game matrix, resulting in a game value. First, dominated strategies are eliminated. Then, a saddle point is found, if it exists. Should none exist, the simplex method is used to find a solution in mixed strategies.



Sample game board

1	1	1	0	0
1	1	1	1	1
1	1	1	1	0
0	1	1	1	1
0	1	0	1	1

BORDERS

1
1
0
1
1

ECON_CENTR

1	0
0	1
0	0
1	0
0	1

OLD_BORD

1
0
1
1
0

COUNTRY

KEY: 3 = region number

A = army

• = economic center

~ = region border

~ = country border

Figure 25. Sample game board for REDIPT.

Given a list of the board positions of a player's armies, the subroutine STRATS will generate all the possible next moves for that player. The principal routine for this is to generate all combinations of move, holds, and supports for every army into any region, subject to the following restrictions.

1. An army can only move into an adjacent region.
2. No more than one army will be moved into a region, and no "mirror moves" are allowed among a player's own armies. This restriction assumes that a player will not want to move against himself, although a player in an actual game could attempt such a move.
3. Illegal supports are prohibited.

A player's moves are stored in the array MOVMAKER, a two-dimensional array, with the rows representing the regions from which an order is being made and the columns representing the regions into which an order is being made. Player X's move in Figure 26(a) is shown in MOVMAKER in Figure 26(b). A player's move stored in MOVMAKER is then translated into a character string representation of the ordered moves and stored in an array of all possible moves.

MOVMAKER is also used in resolving conflicts, which is done in subroutine CNFLCT. Each possible pair of moves for player X and player Y are stored in the array, although Y's move is stored using negative numbers to distinguish from X's move.

An army ordered to move r_1 - r_2 can check for opposing armies by searching column r_2 of MOVMAKER. In the same column we also can find both friendly and enemy supports.

1-2, 3-3, 4S1-2

X's move

(a)

0	1	0	0	0
0	0	0	0	0
0	0	1	0	0
0	2	0	0	0
0	0	0	0	0

MOVMAKER

(b)

Figure 26. Representations of a move.

A "mirror move" would be an occurrence of one player's army moving r_1 - r_2 and the other player's army moving r_2 - r_1. Such moves, for example, could be stored as MOVMAKER (r_1, r_2 = 1 and MOVMAKER (r_2, r_1) = -1. Thus, we could detect a "mirror move" if MOVMAKER (r_1, r_2) = -MOVMAKER (r_2, r_1) and |MOVMAKER (r_1, r_2)| = 1.

The array MTYPE records the type of move each army is attempting. If MTYPE (r_1) = 1, then the army in r is attempting movement or a hold. If MTYPE (r_1) = 2, then it is a support. Should a movement, a hold, or a support fail, then its corresponding MTYPE is set to 3.

During the resolution of a conflict, if an army is dislodged, it is recorded in the array DISLDG. After all conflicts have been resolved, dislodged armies are retreated into the first available region using subroutine RETREAT. In this respect, the program does not accurately

model the game REDIPT, because the time spent in using ASMA to find the best retreat move was felt unjustified at this point. Thus, the programmed version of ASMA presently needs improvement in its retreat routine, which may best be done with heuristics.

One of the most difficult cases to deal with are "circle moves." In a "circle move," there are several movements of the form $r_m - r_n$ dependent on a movement $r_i - r_j$. The last movement in the circle is a movement into r_i. Thus, each movement is dependent on every other move. The series of movements 2-3, 3-5, 5-2 is an example of a "circle move." Circle moves are resolved by using an array DEPEND. When trying to resolve a move $r_i - r_j$ dependent on a move $r_j - r_k$, DEPEND (r_i) is set equal to 1, and when DEPEND $(r_k) = 1$, then we execute the move $r_j - r_k$, ignoring the original move. Then, each dependent move is made in turn.

The function EVALUATE used in the algorithm ASMA corresponds to the use of subroutines DOMINAT, SADDLE, and SIMPLEX in finding the game value of a game matrix. First, DOMINAT eliminates all dominated moves for the current game. Then, SADDLE finds a saddle point, if one or more exists. Should there be no saddle point, SIMPLEX finds the mixed strategy solution to the game, using the simplex method.

Experience with REDIPT

The program REDIPT was implemented in the PL/C programming language on an IBM 360/65. A program listing is given in Appendix B. The program output includes: (1) the original game board, (2) for each player, a list of possible first turn moves and their probabilities, and (3) the game value.

At present, only very simple games can be run because of the huge branching factor involved. To explain this, let Z = the regions in which player Z has an army = $\{Z_1, Z_2, \ldots, Z_k\}$ and Legal(Z) = the number of legal moves that can be made from Z. Then we can estimate the number of leaf nodes, L, as

$$L \approx \theta[(\text{Legal}(X))(\text{Legal}(Y))]^n ,$$

where n = the number of turns in the game, and X and Y are the two players. The value Legal(Z) can be estimated as

$$\text{Legal}(Z) \approx \text{Adj}(Z_1) + \text{Adj}(Z_2) + \ldots \text{Adj}(Z_k) ,$$

where

$\text{Adj}(Z_i)$ = the number of regions adjacent to region i, including region i.

Thus, for the game board given earlier, in Figure 25 (page 67), L can be estimated as

$$L \approx \theta[(\text{Legal}(X))(\text{Legal}(Y))]^n$$

$$\approx \theta[(3 + 4)(5)]^n \approx \theta[35]^n$$

Clearly, L grows logarithmically, and, even for this simple game, the branching factor is large.

To help reduce L, a heuristic was introduced which eliminated moves which did not seem promising. When moves are being generated from a given board position using STRATS, every region into which a friendly

army can move is given a point value based on the following point distribution criteria:

1. +1 point for every adjacent region.
2. +1 point if it contains an economic center.
3. +1 point if the region which contains an economic center is unoccupied.
4. -1 point if it is occupied by a friendly army.
5. +1 point if the economic center is unfriendly.

The points added are cumulative. Then, an average, REGNORM, is calculated from all regions considered and is rounded to the nearest integer. As each move is generated, the sum of every region being moved into, held in, or supported is found. If this sum is less than (REGNORM) (the number of friendly armies), then the current move is not considered.

In almost every game of interest, this heuristic reduces the game tree. To see why, notice that the above point distribution criteria tends to make enemy-owned regions more valuable and friendly regions less valuable. An average of these values will frequently exclude moves in a player's own regions. For instance, the number of moves X can make on the first move is 14, whereas the heuristic reduces this to 8 possible first moves, as shown by Figure 27.

In several test runs, the algorithm with and without the heuristic has returned identical results indicating that the heuristic may be pruning dominated strategies from consideration without pruning potential solution strategies.

X's moves	Y's moves
1-1, 4-2	2-1
1-2, 4-3	2-2
1-2, 4-4	2-4
1-2, 4-5	2-5
1-2, 4S1-2	
1-3, 4-2	
1-3, 4-5	
1S4-2, 4-2	

(a) With heuristic.

X's moves	Y's moves
1-1, 4-2	2-1
1-1, 4-3	2-2
1-1, 4-4	2-3
1-1, 4-5	2-4
1-2, 4-3	2-5
1-2, 4-4	
1-2, 4-5	
1-2, 4S1-2	
1-3, 4-2	
1-3, 4-4	
1-3, 4-5	
1-3, 4S1-3	
1S4-2, 4-2	
1S4-3, 4-3	

(b) Without heuristic.

Figure 27. Possible first moves with and without heuristic.

To demonstrate the program, Figure 28 shows the program output for the given game without the heuristic, and Figure 29 shows the output using the heuristic.

We have seen how game theoretic concepts concerning imperfect information have been used to create and implement an algorithm to solve two-person zero-sum simultaneous movement games, a class of games that has not been studied in detail in artificial intelligence.

REGION	ADJACENT REGIONS			
1	2	3		
2	1	3	4	5
3	1	2	4	
4	2	3	5	
5	2	4		

ORIGINAL BOARD DISPOSITIONS

REGION	COUNTRY	ECONOMIC CENTER	ARMY
1	X	1	1
2	Y	1	1
3	X		
4	X	1	1
5	Y	1	

PLAYER X"S MOVE	PROBABILITY
1 -1 ,4 -2	0.0
1 -1 ,4 -3	0.0
1 -1 ,4 -4	0.0
1 -1 ,4 -5	0.0
1 -2 ,4 -3	0.0
1 -2 ,4 -4	1.00000E+00
1 -2 ,4 -5	0.0
1 -2 ,4 S1 -2	0.0
1 -3 ,4 -2	0.0
1 -3 ,4 -4	0.0
1 -3 ,4 -5	0.0
1 -3 ,4 S1 -3	0.0
1 S4 -2 ,4 -2	0.0
1 S4 -3 ,4 -3	0.0

PLAYER Y"S MOVE	PROBABILITY
2 -1	1.00000E+00
2 -2	0.0
2 -3	0.0
2 -4	0.0
2 -5	0.0
GAME VALUE IS	1.00000E+00

Figure 28. REDIPT output without heuristic.

REGION	ADJACENT REGIONS			
1	2	3		
2	1	3	4	5
3	1	2	4	
4	2	3	5	
5	2	4		

ORIGINAL BOARD DISPOSITIONS

REGION	COUNTRY	ECONOMIC CENTER	ARMY
1	X	1	1
2	Y	1	1
3	X		
4	X	1	1
5	Y	1	

TURN LIMIT IS 2

PLAYER X"S MOVE	PROBABILITY
1 -1 ,4 -2	0.0
1 -2 ,4 -3	0.0
1 -2 ,4 -4	1.00000E+00
1 -2 ,4 -5	0.0
1 -2 ,4 S1 -2	0.0
1 -3 ,4 -2	0.0
1 -3 ,4 -5	0.0
1 S4 -2 ,4 -2	0.0

PLAYER Y"S MOVE	PROBABILITY
2 -1	1.00000E+00
2 -2	0.0
2 -4	0.0
2 -5	0.0

GAME VALUE IS 1.00000E+00

Figure 29. REDIPT output with heuristic.

CHAPTER 5

CONCLUSIONS

The original goal of this thesis was to write a program to play DIPLOMACY based on artificial intelligence techniques. These techniques were found to be unequal to the task, and a search for better tools led to studying game theoretic solutions to classes of games. REDIPT was developed as a two-player simplification of DIPLOMACY, a seven-player game.

There are a number of problems with extending the REDIPT solution to DIPLOMACY besides the number of players. First, some economic centers are neutral in DIPLOMACY. Because DIPLOMACY is nonzero-sum and n-person, it is possible for coalitions to form. The most important part of the game is negotiation and coalition formation, although, ultimately, only one player can win. Finally, DIPLOMACY also has different conditions than REDIPT has for winning the game. In DIPLOMACY, the winning player must own 18 economic centers on a board of 34 economic centers. There are a number of different methods for determining a player's payoff in DIPLOMACY, as noted by WALKER (18). Some of these methods yield zero-sum payoffs. One of these, the Calhamer method, gives a payoff of 1 to the winner, and 0 to the other players. This could be expressed as

$$P_i(\sigma) = 1$$

if σ = any situation in which player i owns at least 18 economic centers, and

$$P_j(\sigma) = 0 \ ,$$

for all $i \neq j$. In this analysis, DIPLOMACY is a nonzero-sum game.

As discussed earlier, we could change to a zero-sum game by adding an eighth player. First, we must note that

$$P_i(\sigma) + P^i(\sigma) + P_8(\sigma) = 0$$

assuming that $P^i(\sigma)$ does not include the payoff to the eighth player. Next, we know that $P_i(\sigma) = -1$ and $P^i(\sigma) = 0$. Finally, we know that

$$\begin{aligned} P_i(\sigma) + P^i(\sigma) + P_8(\sigma) &= 0 \\ 1 \quad + \quad 0 \quad + P_8(\sigma) &= 0 \\ P_8(\sigma) &= -1 \ . \end{aligned}$$

Note that the only effect the eighth player will have on the game is that an undominated imputation must be preferred by five players, rather than four players in the original seven-player game.

Another example of a zero-sum method for determining payoffs includes the BROBDINGNAG method (18), giving a payoff of 6, 4, 2, 0, -2, -4, or -6 to a player according to the player's order of finish in the game. The order of finish is determined by the number of economic centers owned by a player.

The most important feature of DIPLOMACY modeled by REDIPT are the simultaneous moves. Other simultaneous movement games can also be modelled with the aid of ASMA. In accordance with Uhr's request for general principles to produce game-playing programs (16), ASMA is a general algorithm for simultaneous movement two-person games. By using concepts which have been developed by game theory, ASMA allows us to take advantage of this "common language" for solving games.

More general mechanisms are necessary to deal with more difficult games. N-person sequential games, such as NIM, have already received some treatment (6). Similarly, n-person cooperative games are not yet well understood although some programs have been used to test solution concepts for this class of games (3).

Artificial intelligence has been concerned mainly with a narrow class of games, and, to be able to solve other classes of games, artificial intelligence can profitably look to game theory for approaches to solutions. The ASMA algorithm was developed in this way. Whereas time and space limitations may prevent many game theoretic solutions from being directly implemented in game-playing programs, artificial intelligence heuristics can be used to increase the efficiency of these solutions.

LIST OF REFERENCES

LIST OF REFERENCES

1. Desjardins, R. B. and Levin, R. I. Theory of Games and Strategies. International Textbook Company, Scranton, Pennsylvania, 1970.

2. DIPLOMACY. The Avalon Hill Game Company, Baltimore, 1976.

3. Funk, S. G., Horowitz, A. D., Kahan, J. P., Rapoport, A. "Coalition Formation by Sophisticated Players." In Lecture Notes in Economics and Mathematical Systems, M. Beckmann and H. P. Kunzi, Eds., Springer-Verlag, New York, 1979.

4. Harsanyi, J. C. "A Solution Concept for n-Person Noncooperative Games." In International Journal of Game Theory 5, 4(1976).

5. Harsanyi, J. C. Rational Behavior and Bargaining Equilibrium in Games and Social Situations. Cambridge University Press, New York, 1977.

6. Li, S.-Y. R. "N-Person Nim and N-Person Moore's Games." In International Journal of Game Theory 7, 1(1980), pp. 31-36.

7. Luce, R. D. and Raiffa, H. Games and Decisions. Wiley, New York, 1957.

8. Morgenstern, O. and von Neumann, J. Theory of Games and Economic Behavior. Princeton University Press, Princeton, 1944.

9. Pitrat, J. "A General Game-Playing Program." In Artificial Intelligence and Heuristic Programming, Findler, N. V. and Meltzer, B., Eds., American Elsevier, New York, 1971, pp. 125-158.

10. Rapoport, A. N-Person Game Theory. University of Michigan Press, Ann Arbor, 1970.

11. Rapoport, A. Fights, Games, and Debates. University of Michigan Press, Ann Arbor, 1974.

12. Shubik, M. "Game Theory and the Study of Social Behavior: An Introductory Exposition." In Game Theory and Related Approaches to Social Behavior, Shubik, M., Ed., Wiley, New York, 1964, pp. 1-78.

13. Shubik, M. The Uses and Methods of Gaming. Elsevier Scientific, New York, 1975.

14. Slagle, J. R. Artificial Intelligence: The Heuristic Programming Approach. McGraw-Hill, New York, 1971.

15. Steen, L. A. "Linear Programming: Solid New Algorithm." In Science News, Science Service, Washington, 14(1979), pp. 234-236.

16. Uhr, L. Pattern Recognition, Learning, and Thought. Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

17. Vorob'ev, N. N. Game Theory. Kotz, S., Trans., Springer-Verlag, New York, 1977.

18. Walker, Rod. The Gamer's Guide to DIPLOMACY. The Avalon Hill Game Company, Baltimore, 1979.

19. Winston, P. H. Artificial Intelligence. Addison-Wesley, Reading, 1977.

APPENDICES

APPENDIX A

EXAMPLE OF THE SIMPLEX METHOD

EXAMPLE OF THE SIMPLEX METHOD

Given a game matrix (Figure 30), find a solution in mixed strategies.

Step 1. Add three to each entry in the game matrix to make all entries > 0 (Figure 31).

Step 2. Put the game matrix into linear programming form (Figure 32).

Step 3. Put the equations into the Simplex tableau (Figure 33).

Step 4. In the C - Z row, find the $\overline{Y}$ column which is the largest nonnegative and nonzero value. Since $\overline{Y}_1 = \overline{Y}_2 = \overline{Y}_3 = 1$, we will arbitrarily choose $\overline{Y}_1$.

Step 5. For each Basis row, divide P_0 by the corresponding value in $\overline{Y}$.

first row: $\sigma = 1/4$
second row: $\sigma = 1$
third row: $\sigma = 1/5$

Choose the two with the smallest nonnegative value σ. Choose the third row.

Step 6. Divide the chosen row by the value in the chosen row and column. This is the pivot value. Put a 1 in the C column of this row and enter the basis variable (Figure 34).

Step 7. For the rows not chosen, subtract a multiple of the chosen row from every other row, so that the other values in the chosen column will be zero. This is similar to pivoting (Figure 35).

Step 8. Calculate the Z and C - Z rows. Add every basis row which has been chosen and place the result in the corresponding columns of the Z row. The C - Z row is the subtraction of the new Z row from the C row (Figure 36).

Step 9. Continue steps 4-8 until all values in the C - Z row are $\leqslant 0$. Choose second column and first row in Figure 37. Choose third column and second row in Figure 38.

Step 10. The game value is given by

1/(the first entry in the Z row)
1/.2742 = .647058

Y's strategies

	Y_1	Y_2	Y_3
X_1	1	2	-1
X_2	-2	1	1
X_3	2	0	1

Figure 30. Simplex game matrix.

Y's strategies

	Y_1	Y_2	Y_3
X_1	4	5	2
X_2	1	4	4
X_3	5	3	4

Figure 31. Step 1.

Maximize: $\overline{Y}_1 + \overline{Y}_2 + \overline{Y}_3$

Subject to: $4\overline{Y}_1 + 5\overline{Y}_2 + 2\overline{Y}_3 + S_1 + 0S_2 + 0S_3 = 1$

$\overline{Y}_1 + 4\overline{Y}_2 + 4\overline{Y}_3 + 0S_1 + S_2 + 0S_3 = 1$

$5\overline{Y}_1 + 3\overline{Y}_2 + 4\overline{Y}_3 + 0S_1 + 0S_2 + S_3 = 1$

Figure 32. Step 2.

C			1	1	1	0	0	0
	Basis	P_0	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	S_1	S_2	S_3
0	S_1	1	4	5	2	1	0	0
0	S_2	1	1	4	4	0	1	0
0	S_3	1	5	3	4	0	0	1
Z		0	0	0	0	0	0	0
C - Z			1	1	1	0	0	0

Figure 33. Step 3.

C			1	1	1	0	0	0
	Basis	P_0	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	S_1	S_2	S_3
0	S_1	1	4	5	2	1	0	0
0	S_2	1	1	4	4	0	1	0
1	$\overline{Y}_1$	.2	1	.6	.8	0	0	.2
Z		0	0	0	0	0	0	0
C - Z			1	1	1	0	0	0

Figure 34. Step 6.

C			1	1	1	0	0	0
	Basis	P_0	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	S_1	S_2	S_3
0	S_1	.2	0	2.6	-1.2	1	0	-.8
0	S_2	.8	0	3.4	3.2	0	1	-.2
1	$\overline{Y}_1$	.2	1	.6	.8	0	0	.2
Z		0	0	0	0	0	0	0
C - Z			1	1	1	0	0	0

Figure 35. Step 7.

C			1	1	1	0	0	0
	Basis	P_0	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	S_1	S_2	S_3
0	S_1	.2	0	2.6	-1.2	1	0	-.8
0	S_2	.8	0	3.4	3.2	0	1	-.2
1	$\overline{Y}_1$	.2	1	.6	.8	0	0	.2
Z		.2	1	.6	.8	0	0	.2
C - Z			0	.4	.2	0	0	-.2

Figure 36. Step 8.

C			1	1	1	0	0	0
	Basis	P_0	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	S_1	S_2	S_3
1	$\overline{Y}_2$	.0769	0	1	-.4615	.3846	0	-.3077
0	S_2	.5384	0	0	4.7692	-1.3076	1	.8461
1	$\overline{Y}_1$	.15384	1	0	1.0769	-.2307	0	.3846
Z		.2308	1	1	.6154	.1538	0	.0769
C - Z			0	0	.3846	-.1538	0	-.0769

Figure 37. Step 9 first.

C			1	1	1	0	0	0
	Basis	P_0	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	S_1	S_2	S_3
1	$\overline{Y}_2$	.1290	0	1	0	.2580	.0968	-.2258
1	$\overline{Y}_3$	.1129	0	0	1	-.2742	.2097	.1774
0	$\overline{Y}_1$	.0323	1	0	0	.0645	-.2258	.1935
Z		.2742	1	1	1	.0484	.0806	.1452
C - Z			0	0	0	-.0484	-.0806	-.1452

Figure 38. Step 9 last.

Step 11. The probabilities of Y's strategies are given by multiplying the game value by the first three entries in the $\bar{Y}$ column. Notice their order in the basis column of Figure 38.

probability of Y_2 = .647058 × .129032 = .470588

probability of Y_3 = .647058 × .112903 = .411764

probability of Y_1 = .647058 × .032258 = .117647

Step 12. The probabilities of X's strategies are given by multiplying the last three columns of the Z row by the game value. The probabilities are already in order.

probability of X_1 = .647058 × .048387 = .17647

probability of X_2 = .647058 × .0806451 = .294117

probability of X_3 = .647058 × .145161 = .52941

APPENDIX B

A LISTING OF THE PROGRAM REDIPT

```
/*****************************************************************/
/*****************************************************************/
/**                    R E D I P T                              **/
/*****************************************************************/
/*****************************************************************/
/**                    AUTHOR:  JOHN LONG                       **/
/**                    THESIS PROJECT                           **/
/**                    COMPUTER SCIENCE 5000                    **/
/*****************************************************************/
/*****************************************************************/
/**                                                             **/
/**            TABLE  OF  CONTENTS                              **/
/**            -------------------                              **/
/**    INTRODUCTION TO REDIPT........................ I         **/
/**    RULES ........................................ II        **/
/**    LEGAL MOVES .................................. III       **/
/**    GLOSSARY OF IMPORTANT VARIABLE NAMES ......... IV        **/
/**    GLOSSARY OF SUBROUTINES USED ................. V         **/
/**    EXPECTED INPUT ............................... VI        **/
/**    PROGRAM OUTPUT ............................... VII       **/
/**                                                             **/
/*****************************************************************/
/*****************************************************************/
/**    PART I.                                                  **/
/**                                                             **/
/**                                                             **/
/**    REDIPT EVALUATES AN INPUTTED GAME AND FINDS THE GAME     **/
/**    VALUE FOR IT.                                            **/
/**        THE INPUTTED GAME PITS TWO PLAYERS AGAINST EACH      **/
/**    OTHER.  EACH PLAYER POSSESSES A COUNTRY CONSISTING OF A **/
/**    NUMBER OF REGIONS.  WITHIN A REGION, THERE MAY EXIST AN **/
/**    ARMY UNIT AND/OR AN ECONOMIC CENTER.  THESE ARMIES ARE   **/
/**    MOVED ACROSS THE BOARD TRYING TO GAIN CONTROL OF ECONO-  **/
/**    MIC CENTERS.  THE PLAYER WITH A NET GAIN OF ECONOMIC     **/
/**    CENTERS AFTER THE FIRST TURN IS THE WINNER.              **/
/**        BOTH PLAYERS ARE AUTOMATTED.  THAT IS, BOTH ARE CON-**/
/**    TROLLED BY THE PROGRAM.  BOTH PLAYERS ACT RATIONALLY,    **/
/**    IN A GAME THEORETIC SENSE, AND THE GAME IS ANALYZED FOR **/
/**    BOTH PLAYERS ACCORDING TO GAME THEORETIC CONCEPTS.  THE **/
/**    RESULT IS A GAME THEORETIC SOLUTION FOR BOTH PLAYERS,    **/
/**    PROVIDING PROBABILITIES FOR SUCCESSFUL FIRST MOVES AND   **/
/**    AND EXPECTED ZERO-SUM PAYOFF FOR THE GAME, KNOWN AS THE **/
/**    GAME VALUE.                                              **/
/**                                                             **/
/*****************************************************************/
/*****************************************************************/
/**                                                             **/
/**    PART II.                                                 **/
/**                                                             **/
/**    RULES:                                                   **/
/**        THE GAME BOARD REPRESENTS A HYPOTHETICAL GEOGRAPHI-  **/
/**    CAL AREA WHICH IS DIVIDED UP INTO NUMBERED REGIONS.      */
/**    A SUBSET OF THESE REGIONS MAKES UP COUNTRY X, AND THE    **/
/**    REMAINDER MAKE UP COUNTRY Y.  COUNTRIES X AND Y ARE      **/
/**    OWNED BY PLAYERS X AND Y, RESPECTIVELY.  EACH REGION AND**/
/**    THE REGIONS ADJACENT TO EACH REGION ARE STORED IN AN     **/
/**    ADJACENCY GRAPH.                                         **/
/**        A REGION MAY OR MAY NOT POSSESS AN ECONOMIC CENTER,  **/
/**    WHICH IS A LOCATION OF ECONOMIC IMPORTANCE.  ECONOMIC    **/
/**    CENTERS ARE IMPORTANT TO WIN THE GAME (SEE BELOW).       **/
```

```
/**        A REGION ALSO MAY OR MAY NOT HAVE AN ARMY UNIT RE-  **/
/**    SIDING IN IT.  ARMY UNITS ARE OWNED BY EACH PLAYER, AND **/
/**    CAN NEVER CHANGE OWNERSHIP DURING THE GAME, ALTHOUGH    **/
/**    ARMY UNITS MAY BE ELIMINATED.  THERE CAN ONLY BE ONE    **/
/**    ARMY UNIT PER REGION.  ARMIES CAN BE MOVED FROM REGION  **/
/**    TO REGION BY LEGAL MOVES.  A REGION WHICH IS OCCUPIED BY**/
/**    A PLAYER'S ARMY UNIT BECOMES THE POSSESSION OF THAT     **/
/**    PLAYER UNTIL AN ARMY UNIT OF THE OTHER PLAYER OCCUPIES  **/
/**    THAT REGION.                                            **/
/**                                                            **/
/*****************************************************************/
/*****************************************************************/
/**                                                            **/
/**    PART III.                                               **/
/**    LEGAL MOVES                                             **/
/**                                                            **/
/**    A.  BOTH PLAYERS MOVE SIMULTANEOUSLY.                   **/
/**    B.  A PLAYER'S MOVE CONSISTS OF MOVING ALL, SOME,       **/
/**    OR NONE OF HIS ARMIES.                                   **/
/**    C.  AN ARMY WHICH IS MOVED CAN BE MOVED TO ANY ONE      **/
/**    REGION WHICH IS ADJACENT TO THE REGION IT OCCUPIES      **/
/**    AND CANNOT MOVE ANY MORE THAT TURN.                     **/
```

```
/**     D.  A CONFLICT  INVOLVES MORE THAN ONE ARMY TRYING      **/
/**     TO OCCUPY A REGION.  ANY CONFLICT ENCOUNTERED AFTER     **/
/**     MOVEMENT MUST BE RESOLVED ACCORDING TO THE CONFLICT     **/
/**     ORDER, TO BE DESCRIBED LATER.                           **/
/**     E.  AN ARMY WHICH DOES NOT MOVE OUT OF THE REGION       **/
/**     OCCUPIES IS HOLDING IN ITS REGION.                      **/
/**     F.  INSTEAD OF HOLDING OR MOVING, AN ARMY MAY SUP-      **/
/**     PORT ANOTHER ARMY WHICH IS HOLDING OR MOVING INTO       **/
/**     ANOTHER REGION.  AN ARMY MAY SUPPORT ANOTHER ARMY       **/
/**     IF THE SUPPORTING ARMY IS ADJACENT TO THE REGION        **/
/**     THAT IS BEING MOVED INTO BY THE OTHER ARMY, OR IF       **/
/**     THE SUPPORTING ARMY IS ADJACENT TO THE REGION IN        **/
/**     WHICH THE OTHER ARMY IS HOLDIG.  A ARMY CANNOT SUP-     **/
/**     PORT ANOTHER ARMY DOING A SUPPORT.                      **/
/**     G.  CONFLICTS ARE RESOLVED ACCORDING TO THE FOL-        **/
/**     LOWING RULES:                                           **/
/**        1. AN ARMY MOVES OR HOLDS WITH THE STRENGTH OF       **/
/**        ITSELF AND ALL UNCUT SUPPRT.                        **/
/**        2.  A SUPPORT IS CUT IF IT IS ATTACKED FROM A        **/
/**        REGION OTHER THAN THE ONE IT IS SUPPORTING A         **/
/**        MOVE INTO.                                           **/
/**        3.  IF TWO OR MORE ARMIES OF EQUAL STRENGTH MOVE     **/
/**        INTO EACH OTHER'S REGIONS, BOTH MOVES FAIL.          **/
/**        THIS IS A MIRROR MOVE.                               **/
/**        4.  IF ONE OF THE ARMIES IN THE TWO PRECEDING        **/
/**        RULES IS, INSTEAD, GREATER IN STRENGTH THAN ALL      **/
/**        OPPOSING ARMIES MOVING INTO THE SAME REGION,         **/
/**        THEN THE STRONGER ARMY SUCCEEDS IN ITS MOVE, AND     **/
/**        ALL OTHER ARMIES FAIL IN THEIR MOVES.                **/
/**        5.  AN ARMY REMAINING IN ITS OWN REGION THROUGH      **/
/**        A HOLD OR A FAILED MOVE,, IF OPPOSED BY A S          **/
/**        STRONGER ARMY, IS DISLODGED FROM THIS REGION AND     **/
/**        MUST RETREAT.                                        **/
/**        6.  IF AN ARMY IS ORDERED TO MOVE INTO A REGION      **/
/**        IN WHICH AN ARMY IS HOLDING OR HAS FAILED IN ITS     **/
/**        MOVE, THE FORMER ARMY FAILS IN ITS MOVE UNLESS       **/
/**        IT HAS GREATER STRENGTH THAN THE LATTER ARMY,        **/
/**        CAUSING THE LATTER ARMY TO BE DISLODGED.             **/
/**        7.  AN ARMY CAN NEVER SUCCEED IN DISLODGING AN-      **/
/**        OTHER FRIENDLY ARMY.                                 **/
/**        8.  TWO OR MORE ARMIES OF THE SAME STRENGTH          **/
/**        MOVING INTO THE SAME REGION FROM OTHER REGIONS       **/
/**        WILL ALL FAIL IN THEIR MOVES, AND AN ARMY INSIDE     **/
/**        THIS TARGET REGION IS NOT AFFECTED.                  **/
/**        9.  IF TWO ARMIES ARE MOVING INTO A REGION AND       **/
/**        ONE IS DISLODGED BY A MOVE FROM THAT REGION, THE     **/
/**        OTHER ARMY MAY MOVE INTO THAT REGION.                **/
/**        10.  AFTER ALL MOVES HAVE BEEN EXECUTED, ALL         **/
/**        DISLODGED ARMIES ARE RETREATED INTO THE FIRST        **/
/**        11.  AVAILABLE ADJACENT REGION.  ARMIES MAY NOT      **/
/**        RETREAT INTO OCCUPIED REGIONS, REGIONS INTO          **/
/**        WHICH A MOVE HAS FAILED, OR THE REGION FROM THE      **/
/**        ATTACK CAME.                                         **/
/**        12.  AN ARMY WHICH CANNOT RETREAT IS, INSTEAD,       **/
/**        ELIMINATED FROM THE GAME.                            **/
/*****************************************************************/
/*****************************************************************/
/**                                                             **/
/**     PART IV.                                                **/
/**     GLOSSARY OF IMPORTANT VARIABLE NAMES:                   **/
```

```
/**                                                                  **/
/**     ARMLOC - AN ARRAY OF ARMY LOCATIONS.                         **/
/**     BADMOV - A FLAG TELLING IF A PROPOSED MOVE IS ILLE-          **/
/**              GAL OR NOT.                                         **/
/**     BORDERS - AN ARRAY TELLING WHICH REGIONS ARE ADJA-           **/
/**               CENT TO EACH OTHER.                                **/
/**     COUNTRY - AN ARRAY OF REGION OWNERSHIP.                      **/
/**     DOMCOLS - THE NUMBER OF Y'S DOMINATED MOVES.                 **/
/**     DOMROWS - THE NUMBER OF X'S DOMINATED MOVES.                 **/
/**     ECON_CENTR - AN ARRAY SHOWING WHICH REGIONS HAVE             **/
/**                  ECONOMIC CENTERS.                               **/
/**     GAM_MAT - AN ARRAY USED AS A GAME MATRIX.                    **/
/**     GVALUE - THE GAME VALUE.                                     **/
/**     MOVCNTR - THE MOVE COUNTER, USED TO GENERATE ALL OF          **/
/**               A PLAYER'S MOVES.                                  **/
/**     MOVMAKER - AN ARRAY USED TO DETERMINE THE EFFECTS            **/
/**                X'S MOVE AND Y'S MOVE.                            **/
/**     MTYPE - AN ARRAY OF MOVE TYPES (1 FOR MOVE OR HOLD,          **/
/**             2 FOR SUPPORT).                                      **/
/**     NARMS1 - THE NUMBER OF X'S ARMIES.                           **/
/**     NARMS2 - THE NUMBER OF Y'S ARMIES.                           **/
/**     NESUPS - THE NUMBER OF ENEMY SUPPORTS FOR OR                 **/
/**              AGAINST A MOVE.                                     **/
/**     NEW_BORD - THE GAME BOARD AFTER A MOVE.                      **/
/**     NREGION - THE NUMBER OF REGIONS.                             **/
/**     NSUPS - THE NUMBER OF FRIENDLY SUPPORTS FOR OR               **/
/**             AGAINST A MOVE.                                      **/
/**     NX - THE NUMBER OF MOVES FOR X.                              **/
/**     NY - THE NUMBER OF MOVES FOR Y.                              **/
/**     OLD_BORD - THE GAME BOARD(REGIONS AND ARMIES) AT             **/
/**                THE BEGINNING OF A MOVE.                          **/
/**     PRES_TURN - THE PRESENT TURN.                                **/
/**     RED_BORD - THE REDUCED GAME MATRIX AFTER THE EF-             **/
/**                FECTS OF DOMINATIN.                               **/
/**     REGNORM - AN INDEX OF A STANDARD VALUE OF A REGION           **/
/**     SIMP - THE MATRIX USED FOR THE SIMPLEX METHOD.               **/
/**     TURN_LIMIT - THE NUMBER OF TURNS TO BE EVALUATED.            **/
/**     XMOV - THE MOVE MADE BY ONE OF X'S ARMIES.                   **/
/**     XMOVES - THE MOVES OF ALL OF X'S ARMIES.                     **/
/**     XSTRAT - AN ARRAY OF ALL POTENTIAL MOVES OF X.               **/
/**     YMOV - THE MOVE MADE BY ONE OF Y'X ARMIES.                   **/
/**     YMOVES - THE MOVES OF ALL OF Y'X ARMIES.                     **/
/**     YSTRAT - AN ARRAY OF ALL POTENTIAL MOVES OF Y.               **/
/*****************************************************************/
/*****************************************************************/
/**                                                                  **/
/**     PART V.                                                      **/
/**     SUBROUTINES USED BY REDIPT (IN ALPHABETICAL ORDER):          **/
/**                                                                  **/
/**         ARMYCT -  COUNTS THE TOTAL NUMBER OF ARMIES ON BOTH      **/
/**         SIDES AND DETERMINES WHICH REGIONS ARE OWNED BY          **/
/**         WHICH PLAYERS.                                           **/
/**                                                                  **/
/**         CNFLCT - EXECUTES TWO INPUTTED MOVES OF THE TWO          **/
/**         OPPOSING PLAYERS AND RESOLVES ANY CONFLICTS.             **/
/**                                                                  **/
/**         DOMINAT - ELIMINATES ALL OF A PLAYER'S DOMINATED         **/
/**         STRATEGIES.                                              **/
/**                                                                  **/
/**         LOCATR - SEARCHES THE ARRAY MOVMAKER FOR A CERTAIN       **/
```

```
/**        TYPE OF MOVE.                                                **/
/**
/**        MOVIT - MOVES ONE ARMY UNIT FROM ONE REGION TO               **/
/**        ANOTHER.                                                     **/
/**
/**        SADDLE - FINDS THE SADDLE POINT (IF ONE EXISTS) OF           **/
/**        A GIVEN GAME MATRIX.                                         **/
/**
/**        ASMA  - EVALUATES ALL THE POSSIBLE MOVES OF A GIVEN          **/
/**        TURN AND BOARD SITUATION.                                    **/
/**
/**        SIMPLEX - FINDS THE MIXED STRATEGY SOLUTION OF A             **/
/**        GIVEN GAME MATRIX.                                           **/
/**
/**        SNDHOM - SEND ALL THE SUPPORTS OF A GIVEN MOVE               **/
/**        "HOME".  THAT IS, THE SUPPORTS MOVES ARE DISCARDED           **/
/**        FOR THE REMAINDER OF THE TURN.                               **/
/**
/**        STRATS - CALCULATES ALL POSSIBLE LEGAL MOVES FROM A          **/
/**        GIVEN BOARD POSITION FOR A GIVEN PLAYER.                     **/
/**
/**        SUPCNT - COUNTS THE NUMBER OF SUPPORTS FOR A GIVEN           **/
/**        MOVEMENT.                                                    **/
/**
/**        WHOWINS - DETERMINES THE WINNER OF A GIVEN CONFLICT.         **/
/**
/**        WIN - DETERMINES THE ZERO-SUM PAYOFF FOR THE GIVEN           **/
/**        BOARD POSITION ACCORDING TO CONDITIONS FOR A "WIN".          **/
/*****************************************************************/
/*****************************************************************/
/**                                                                    **/
/**    PART VI.                                                        **/
/**    EXPECTED INPUT:                                                 **/
/**    1.  THE NUMBER OF REGIONS IN THE GAME BOARD ENVIRONMENT         **/
/**    2.  THE NUMBER OF TURNS IN THE GAME.                            **/
/**    3.  A LIST OF REGION NUMBERS OF THE REGIONS IN THE GAME         **/
/**        BOARD ENVIRONMENT.  EACH REGION NUMBER IS FOLLOWED          **/
/**        BY:                                                         **/
/**           A.  THE REGION NUMBERS OF ADJACENT REGIONS.              **/
/**           B.  A ZERO SENTINEL VALUE TO SIGNAL END OF A.            **/
/**           C.  A NUMBER REPRESENTING THE OWNER OF THE               **/
/**               REGION.  A '1' OR '2' MEANS PLAYER X OR Y,           **/
/**               RESPECTIVELY.                                        **/
/**           D.  A '1' TO INDICATE THAT THE CURRENT REGION IS         **/
/**               AN ECONOMIC CENTER, OR A '0' TO REPRESENT THE        **/
/**               ABSENCE OF AN ECONOMIC CENTER.                       **/
/**           E.  A '1' TO INDICATE THE PRESENCE OF AN ARMY IN         **/
/**               THE REGION AND A '0' OTHERWISE.                      **/
/*****************************************************************/
/*****************************************************************/
/**                                                                    **/
/**    PART VII.                                                       **/
/**    PROGRAM OUTPUT:                                                 **/
/**                                                                    **/
/**        1.  ORIGINAL BOARD DISPOSITION INCLUDING:                   **/
/**            A.  OWNER OF EACH REGION                                **/
/**            B.  PRESENCE OF AN ECONOMIC CENTER IN A REGION          **/
/**            C.  LOCATION OF EACH ARMY                               **/
/**        2.  RESULTS OF THE EVALUATION, INCLUDING:                   **/
/**            A.  ALL OF X'S AND Y'S POTENTIAL FIRST                  **/
```

```
/**                 MOVES.                                          **/
/**            B.   PROBABILITY VECTORS FOR X'S AND Y'S             **/
/**                 POTENTIAL FIRST MOVES.                          **/
/*****************************************************************/
```

```
//ASMA   JOB (*****,*****,BINS,T-30,R-256K),'ZLONG'
// EXEC PLC,GCREG=256K
//SYSIN DD *
*PL/C ID='ZLONG',PAGES=60,TIME=(44,55),NOSOURCE
 REDIPT:  PROCEDURE OPTIONS (MAIN);
    DCL (TURN_LIMIT,PRES_TURN) FIXED DECIMAL;
    DCL NREGION FIXED DECIMAL EXTERNAL STATIC;
    GET LIST(NREGION);
    GET LIST(TURN_LIMIT);
    /*  CREATE THE GAME BOARD AND ENVIRONMENT FROM DATA READ IN.*/
    CREATE_BORD:  BEGIN;
       DCL BORDERS(NREGION,NREGION) FIXED DECIMAL;
       DCL MOVMAKER(NREGION,NREGION) FIXED DECIMAL;
       DCL (CNTRY,ECENT,ARM) CHAR(2) VARYING;
       DCL (OLD_BORD(NREGION,2),COUNTRY(NREGION)) FIXED DECIMAL;
       DCL (ADJREG,OWNER,CENTER,ARMY) FIXED DECIMAL;
       DCL GVALUE FLOAT DECIMAL;
       DCL (XORIG,YORIG) FIXED DECIMAL;
       DCL TBORD(NREGION) CHAR(10) VARYING;
       DCL ECON_CENTR(NREGION) FIXED DECIMAL;
       DCL FLAG FIXED DECIMAL;
       BORDERS = 0;
       OLD_BORD = 0;
       XORIG = 0;
       PRES_TURN = 0;
       YORIG = 0;
       COUNTRY = 0;
       ECON_CENTR = 0;
       /*********************************************************/
       /*  INPUT GAME BOARD ENVIRONMENT.                        */
       /*  STORE GAME BOARD AS ADJACENCY GRAPH.                 */
       /*  ALSO STORE INFORMATION ON REGION OWNER, ECONOMIC     */
       /*  CENTERS, AND ARMIES.                                 */
       /*********************************************************/
       DO I = 1 TO NREGION;
          BORDERS(I,I) = 1;
          GET LIST(ADJREG);
          DO WHILE(ADJREG>0);
             BORDERS(I,ADJREG) = 1;
             GET LIST(ADJREG);
             END;
          GET LIST(OWNER,CENTER,ARMY);
          IF (OWNER=1)
          THEN COUNTRY(I) = 1;
          IF (CENTER = 1)
          THEN DO;
             ECON_CENTR(I) = 1;
             IF (OWNER=1)
             THEN XORIG = XORIG + 1;
             ELSE YORIG = YORIG + 1;
             END;
          IF (ARMY=1)
          THEN OLD_BORD(I,OWNER) = 1;
          END;
          PUT SKIP EDIT('REGION','ADJACENT REGIONS')
                       (X(5),A,X(5),A);
          DO I = 1 TO NREGION;
             TBORD=' ';
             PUT LIST(I) STRING(TBORD(1));
             I2=1;
```

```
            DO J=1 TO NREGION;
               IF (I¬=J & BORDERS(I,J)=1)
               THEN DO;
                  I2=I2+1;
                  PUT LIST(J) STRING(TBORD(I2));
                  END;
               END;
            PUT SKIP EDIT(TBORD) (A,X(5),(50)(A));
            END;
         /*  PRINT OUT ORIGINAL GAME BOARD ENVIRONMENT.          */
         PUT SKIP;
         PUT SKIP;
         PUT SKIP LIST('ORIGINAL BOARD DISPOSITIONS');
         PUT SKIP;
         PUT SKIP LIST('REGION','COUNTRY','ECONOMIC CENTER',
               'ARMY');
         DO I = 1 TO NREGION;
            IF (COUNTRY(I)=1)
            THEN CNTRY='X';
            ELSE CNTRY='Y';
            IF (ECON_CENTR(I)=1)
            THEN ECENT='1';
            ELSE ECENT=' ';
            IF (OLD_BORD(I,1)=1 | OLD_BORD(I,2)=1)
            THEN ARM='1';
            ELSE ARM=' ';
            PUT SKIP LIST(I,CNTRY,ECENT,ARM);
            END;
         PUT SKIP;  PUT SKIP;
         PUT SKIP LIST('TURN LIMIT IS ',TURN_LIMIT);
         /*  START RECURSIVE PROCEDURE TO DETERMINE GAME VALUE.*/
         CALL ASMA(OLD_BORD,PRES_TURN,COUNTRY,GVALUE);
         PUT SKIP LIST(' GAME VALUE IS ',GVALUE);
   STOP;
/*****************************************************************/
/*****************************************************************/
/**                        ASMA                                 **/
/**     SEQUENTIAL AND SIMULTANEOUS MOVE INTERPRETER.           **/
/**     ASMA INTERPRETS THE "BEST" MOVE FOR EACH PLAYER AT THE **/
/**     CURRENT MOVE, GIVEN THE NUMBER OF MOVES WE HAVE LEFT.   **/
/*****************************************************************/
/*****************************************************************/
ASMA:  PROCEDURE(OLD_BORD,PRES_TURN,COUNTRY,GVALUE) RECURSIVE;
   DCL (OLD_BORD(*,*),PRES_TURN) FIXED DECIMAL;
   DCL (ARMLOC(NREGION),NARMS1,NARMS2,N1,N2,NX,NY) FIXED DECIMAL;
   DCL (I,J,POSS(NREGION),COUNTRY(*)) FIXED DECIMAL;
   DCL (FLAG,DOMROWS,DOMCOLS) FIXED DECIMAL;
   DCL GVALUE FLOAT DECIMAL;
   /*  INCREMENT TURN COUNTER, COUNT THE NUMBER OF ARMIES,     */
   /*  AND ESTIMATE THE TOTAL NUMBER OF POSSIBLE MOVES.        */
   PRES_TURN = PRES_TURN + 1;
   CALL ARMYCT(NARMS1,NARMS2,OLD_BORD,POSS,COUNTRY);
   N1 = NREGION ** NARMS1   +   NREGION * NARMS1;
   N2 = NREGION ** NARMS2   +   NREGION * NARMS2;
   /*  CALCULATE ALL POSSIBLE STRATEGIES FOR BOTH PLAYERS FOR  */
   /*  THE GIVEN BOARD POSITION.                               */
   ARM_RECN:  BEGIN;
      DCL (XMOVES(N1),YMOVES(N2)) CHAR(30) VARYING;
      DCL (XMOV,YMOV) CHAR(30) VARYING;
      DCL TEMP_TURN FIXED DECIMAL;
```

```
I=1; J=2;
CALL STRATS(OLD_BORD,BORDERS,ECON_CENTR,I,XMOVES,NX,
            COUNTRY,NARMS1);
CALL STRATS(OLD_BORD,BORDERS,ECON_CENTR,J,YMOVES,NY,
            COUNTRY,NARMS2);
/*********************************************************/
/*  CREATE THE GAME MATRIX FOR THE PRESENT BOARD POSI-   */
/*  TIONS.  IF WE ARE AT THE TURN LIMIT, THEN EVALUATE   */
/*  WIN CONDITIONS FOR EACH ENTRY IN THE GAME MATRIX.    */
/*  OTHERWISE, EVALUATE THE GAME BELOW EACH ENTRY AND    */
/*  RETURN ITS GAME VALUE, WHICH IS DONE BY CALLING ASMA*/
/*  RECURSIVELY.                                         */
/*********************************************************/
NEW_GAME:  BEGIN;
   DCL (ELIM1(NX),ELIM2(NY)) FIXED DECIMAL;
   DCL (XSTRAT(NX),YSTRAT(NY)) FLOAT DECIMAL;
   DCL GAM_MAT(NX,NY) FIXED DECIMAL;
   DCL NEW_BORD(NREGION,2) FIXED DECIMAL;
   DO I = 1 TO NX;
      DO J = 1 TO NY;
         TEMP_TURN = PRES_TURN;
         NEW_BORD = O;
         POSS = COUNTRY;
         XMOV = XMOVES(I);
         YMOV = YMOVES(J);
         CALL ONFLCT(MOVMAKER,XMOV,YMOV,BORDERS,
                        NEW_BORD);
         IF (PRES_TURN = TURN_LIMIT)
         THEN CALL WIN(NEW_BORD,ECON_CENTR,COUNTRY,
                       XORIG,YORIG,GVALUE);
         ELSE CALL ASMA(NEW_BORD,TEMP_TURN,POSS,GVALUE);
            GAM_MAT(I,J) = GVALUE;
         END;
      END;
   /*******************************************************/
   /*  NOW THAT THE GAME MATRIX IS FILLED IN, FIND THE    */
   /*  SOLUTION FOR THE GAME.  FIRST, GET RID OF ALL DOM-*/
   /*  INATED STRATEGIES BY CALLING DOMINAT.  THEN,  SEE */
   /*  IF THE REDUCED GAME HAS A SADDLE POINT.  IF SO,    */
   /*  THE GAME VALUE IS THE PAYOFF AT THE SADDLE POINT. */
   /*  OTHERWISE, IF THERE IS NO SADDLE POINT, FIND THE   */
   /*  SOLUTION IN MIXED STRATEGIES BY CALLING SIMPLEX.   */
   /*  IF THIS IS THE FIRST TURN, PRINT OUT THE PROBA-    */
   /*  BILITIES OF ALL MOVES FOR BOTH PLAYERS ON THE      */
   /*  FIRST TURN.                                        */
   /*******************************************************/
   ELIM1=0;  ELIM2=0;
   CALL DOMINAT(GAM_MAT,NX,NY,DOMROWS,DOMCOLS,ELIM1,ELIM2);
   CALL SADDLE(GAM_MAT,FLAG,DOMROWS,DOMCOLS,I,J,GVALUE);
   IF (FLAG=1 & PRES_TURN=1)
   THEN DO;
      I2=0; I3=0;
      DO WHILE(I2<I);
         I3=I3+1;
         IF (ELIM1(I3)=0)
         THEN I2=I2+1;
         END;
      J2=0;  J3=0;
      DO WHILE(J2<J);
         J3=J3+1;
```

```
            IF (ELIM2(J3)=0)
            THEN J2=J2+1;
            END;
         ELIM1=1;  ELIM2=1;
         ELIM1(I3)=0;
         ELIM2(J3)=0;
         XSTRAT(1)=1;
         YSTRAT(1)=1;
         END;
      ELSE IF (FLAG=0)
           THEN CALL SIMPLEX(GAM_MAT,DOMROWS,DOMCOLS,YSTRAT,
                             XSTRAT,GVALUE);
      /*******************************************************/
      /*  PRINT OUT ALL POSSIBLE MOVES FOR BOTH PLAYERS AND */
      /*  THE PROBABILITY FOR EACH MOVE.                    */
      /*******************************************************/
      IF (PRES_TURN=1)
      THEN DO;
         PUT SKIP;
         PUT SKIP LIST('PLAYER X'S MOVE','PROBABILITY');
         J=1;
         DO I = 1 TO NX;
            IF (ELIM1(I)=1)
            THEN PUT SKIP LIST(XMOVES(I),'0.0');
            ELSE DO;
               PUT SKIP LIST(XMOVES(I),XSTRAT(J));
               J=J+1;
               END;
            END;
         PUT SKIP;
         PUT SKIP LIST('PLAYER Y'S MOVE','PROBABILITY');
         J=1;
         DO I = 1 TO NY;
            IF (ELIM2(I)=1)
            THEN PUT SKIP LIST(YMOVES(I),'0.0');
            ELSE DO;
               PUT SKIP LIST(YMOVES(I),YSTRAT(J));
               J=J+1;
               END;
            END;
         END;
      RETURN;
      END NEW_GAME;
   END ARM_ROOM;
   END ASMA;
   END CREATE_BORD;
END REDIPT;
*PROCESS
/*****************************************************************/
/*****************************************************************/
/**                        STRATS                                **/
/**     STRATS FINDS ALL THE POSSIBLE MOVES A PLAYER COULD MAKE **/
/*      FROM A GIVEN BOARD POSITION AND STORES THE MOVES IN A   **/
/**     LIST.                                                   **/
/**                                                             **/
/**     INTERNAL SUBROUTINES TO STRATS:                         **/
/**         INCR                                                **/
/**         MOVCHK1                                             **/
/**         MOVCHK2                                             **/
/**         TRANSLATE                                           **/
```

```
/*****************************************************************/
/*****************************************************************/
STRATS:  PROCEDURE(CLD_BORD,BORDERS,ECON_CENTR,J,MOVES,NMOVES,
                        COUNTRY,NARMIES);
   DCL NREGION FIXED DECIMAL EXTERNAL STATIC;
   DCL (CLD_BORD(*,*),BORDERS(*,*)) FIXED DECIMAL;
   DCL COUNTRY(*) FIXED DECIMAL;
   DCL (ARMLOC(NREGION),NARMIES,J) FIXED DECIMAL;
   DCL (N,NMOVES,ECON_CENTR(*)) FIXED DECIMAL;
   DCL MOVMAKER(NREGION,NREGION) FIXED DECIMAL;
   DCL REGMOD FIXED DECIMAL;
   DCL (REGPTS(NREGION),REGNORM) FIXED DECIMAL;
   DCL MOVES(*) CHAR(*) VARYING;
   N = 0;
   NMOVES = 0;
   /*************************************************************/
   /*  STORE THE STARTING LOCATIONS OF EACH ARMY.               */
   /*************************************************************/
   DO I = 1 TO NREGION;
      IF (CLD_BORD(I,J) = 1)
      THEN DO;
         N = N + 1;
         ARMLOC(N) = I;
         END;
      END;
   REGMOD = 2 * NREGION;
   /*************************************************************/
   /*  BEGIN GENERATING MOVES.                                  */
   /*************************************************************/
   MAKE_MOVS:  BEGIN;
      DCL MOVCNTR(NARMIES) FIXED INIT(0);
      DCL IFLAG FIXED INIT(0);
      DCL BADMOV FIXED DECIMAL;
      MOVCNTR = 1;
      CALL REGEVAL; /* EVALUATE EACH REGION.*/
      DO WHILE(IFLAG = 0);/*IFLAG=1 MEANS THERE CAN */
                          /* BE NO MORE MOVES.      */
         MOVMAKER = 0;
         I = 0;  BADMOV = 0;
         /* THIS DO LOOP CONSIDERS ONLY MOVEMENT MOVES.*/
         DO WHILE ((I<NARMIES) & (BADMOV=0));
            BADMOV=0;
            I = I + 1;
            IF (MOVCNTR(I) <= NREGION)
            THEN DO;
                 IF (BORDERS(ARMLOC(I),MOVCNTR(I))=1)
                 THEN DO;
                    MOVMAKER(ARMLOC(I),MOVCNTR(I)) = 1;
                    CALL MOVCHK1(BADMOV);
                    END;
                 ELSE BADMOV=1;
                 END;
            ELSE DO;
                K = MOVCNTR(I) - NREGION;
                IF (BORDERS(ARMLOC(I),K)=1 & ARMLOC(I)¬=K)
                THEN MOVMAKER(ARMLOC(I),K)=2;
                 ELSE BADMOV=1;
            END;
         END;
         /*************************************************************/
```

```
        /*     IF THERE IS NO REDUNDANT MOVEMENT, THEN CHECK TO*/
        /*     MAKE SURE THERE ARE NO ILLEGAL SUPPORT ORDERS.  */
        /*     THEN, USE THE HEURISTIC IN MOVCHK3 TO ELIMINATE */
        /*     ALL UNPROMISING MOVES.                          */
        /*****************************************************/
        IF (BADMOV=0)
        THEN DO;
           CALL MOVCHK2(BADMOV);
           IF (BADMOV=0)
           THEN DO;
              CALL MOVCHK3(BADMOV);
              IF (BADMOV=0)
              THEN CALL TRANSLATE(NMOVES);
              END;
           END;
        MOVCNTR(NARMIES)=MOVCNTR(NARMIES)+1;
        CALL INCR(NARMIES,IFLAG);
        END;
        RETURN;
/*************************************************************/
/*                      REGEVAL                              */
/*     REGEVAL EVALUATES EACH REGION FOR ITS VALUE IN CHOOSING */
/*     A MOVE.  CONSIDER ONLY THE REGIONS OCCUPIED BY OR ADJA- */
/*     CENT TO A FRIENDLY ARMY.  ASSIGN POINTS TO EACH OF THESE*/
/*     REGIONS AS FOLLOWS:                                     */
/*             1.  1 POINT FOR AN ECONOMIC CENTER.             */
/*             2.  1 ADDITIONAL POINT IF THE ECONOMIC CENTER   */
/*                 IS UNOCCUPIED.                              */
/*             3.  -1 POINT IF THE REGION IS OCCUPIED BY A     */
/*                 FRIENDLY ARMY.                              */
/*             4.  1 POINT FOR EACH ADJACENT REGION.           */
/*             5.  1 POINT IF THE ECONOMIC CENTER BELONGS TO   */
/*                 THE OTHER PLAYER.                           */
/*     AN AVERAGE (REGNORM) IS CALCULATED FROM ALL THE REGIONS */
/*     CONSIDERED.                                             */
/*************************************************************/
REGEVAL:  PROCEDURE;
   DCL (I,K,NFLUNK,CANDREG,NCAND,JZ)FIXED DECIMAL;
   REGPTS=0;  REGNORM=0;  NCAND=0;
   DO I = 1 TO NREGION;  /* CONSIDER EACH REGION.  */
      CANDREG=0;
      IF (OLD_BORD(I,J)=1)
      THEN CANDREG=1;
      ELSE DO;
         K=0;
         DO WHILE(K<NREGION);
            K=K+1;
            IF (BORDERS(I,K)=1 & OLD_BORD(K,J)=1)
            THEN DO;
               CANDREG=1;
               K=NREGION;
               END;
            END;
         END;
      /* A REGION IS A CANDIDATE FOR EVALUATION IF IT IS */
      /* ADJACENT TO OR OCCUPIED BY A FRIENDLY ARMY.     */
      IF (CANDREG=1)
      THEN DO;
         NCAND=NCAND+1;
      IF (ECON_CENTR(I)=1)
```

```
      THEN DO;
         REGPTS(I)=REGPTS(I)+1; /* 1 POINT FOR AN ECONOMIC CENTER*/
         /* 1 POINT IF IT IS AN ENEMY ECONOMIC CENTER. */
         IF (J=1)
         THEN JZ=1;
         ELSE JZ=0;
         IF (COUNTRY(I)¬=JZ)
         THEN REGPTS(I)=REGPTS(I)+1;
         IF (OLD_BORD(I,1)=0 & OLD_BORD(I,2)=0)
         THEN REGPTS(I)=REGPTS(I)+1;
         /* 1 POINT FOR AN UNOCCUPIED ECONOMIC CENTER.*/
         END;
         IF (OLD_BORD(I,J)=1)
         THEN REGPTS(I)=REGPTS(I)-1;
         /* -1 POINT IF REGION IS OCCUPIED BY OWN ARMY. */
      DO K = 1 TO NREGION;
         IF (BORDERS(I,K)=1 & I¬=K)
         THEN REGPTS(I)=REGPTS(I)+1;
         /* 1 POINT FOR EACH ADJACENT REGION.*/
         END;
      REGNORM=REGNORM+REGPTS(I);
      END;
      END;
   REGNORM=ROUND(REGNORM/NCAND,0)*NARMIES;
   RETURN;
   END;
/*****************************************************************/
/*                          MOVCHK3                              */
/*     MOVCHK3 IS A HEURISTIC DESIGNED TO REDUCE THE NUMBER OF   */
/*     MOVES GENERATED BY STRATS.  THIS SUBROUTINE IS TO BE      */
/*     USED WITH REGEVAL.  BASICALLY, MOVCHK3 CHECKS TO SEE IF   */
/*     A MOVE'S VALUE HAS REACHED A CERTAIN THRESHOLD.  FIRST,   */
/*     REGNORM IS MULTIPLIED BY THE NUMBER OF ARMIES OWNED BY    */
/*     THE CURRENT PLAYER.  THIS GIVES US THE THRESHOLD.  THEN,  */
/*     FOR EACH REGION BEING MOVED INTO BY THE CURRENT MOVE,     */
/*     THE VALUE OF EACH REGION IS TOTALLED, ACCORDING TO THE    */
/*     ASSIGNED VALUES GIVEN BY REGEVAL.  ANY MOVE WHOSE VALUE   */
/*     DOES NOT EQUAL OR EXCEED THIS THRESHOLD IS ELIMINATED.    */
/*****************************************************************/
MOVCHK3:  PROCEDURE(BADMOV);
   DCL (BADMOV,I,REGTOT,REG) FIXED DECIMAL;
   REGTOT=0;
   DO I = 1 TO NARMIES;
      IF (MOVCNTR(I)>NREGION)
      THEN REG=MOVCNTR(I)-NREGION;
      ELSE REG=MOVCNTR(I);
      REGTOT=REGTOT+REGPTS(REG);
      END;
   IF (REGTOT<REGNORM)
   THEN BADMOV=1;
   RETURN;
   END;
/*****************************************************************/
/*****************************************************************/
/**                          MOVCHK1                             **/
/**     MOVCHK1 CHECKS TO MAKE SURE TWO OF A PLAYER'S ARMIES     **/
/**     ARE NOT DOING ANYTHING REDUNDANT.                        **/
/*****************************************************************/
/*****************************************************************/
MOVCHK1:  PROCEDURE(BADMOV);
```

```
   DCL BADMOV FIXED DECIMAL;
   DCL (J1,J2) FIXED DECIMAL;
   J1 = ARMLOC(I);  J2 = MOVCNTR(I);
   IF ((MOVMAKER(J1,J2)=MOVMAKER(J2,J1))&(J1¬=J2))
   THEN DO;
      BADMOV = 1;
      RETURN;
      END;
   DO K = 1 TO NREGION;
      IF ((MOVMAKER(K,J2))=1 & K¬=J1)
      THEN DO;
         BADMOV = 1;
         RETURN;
         END;
      END;
   RETURN;
   END MOVCHK1;
/*****************************************************************/
/*****************************************************************/
/**                        MOVCHK2                               **/
/**    MOVCHK2 CHECKS TO MAKE SURE THAT THERE ARE NO ILLEGAL     **/
/**    SUPPORT MOVES IN THE CURRENT PROPOSED MOVE.               **/
/*****************************************************************/
/*****************************************************************/
MOVCHK2:  PROCEDURE(BADMOV);
   DCL BADMOV FIXED DECIMAL;
   DCL I FIXED DECIMAL;
   BADMOV = 0;
   DO I = 1 TO NARMIES;
      I2 = ARMLOC(I);
      IF (MOVCNTR(I)>NREGION)
      THEN J2=MOVCNTR(I)-NREGION;
      ELSE J2=MOVCNTR(I);
      IF (MOVMAKER(I2,J2) = 2)
      THEN DO;
         I3 = 0;
         DO WHILE (I3<NARMIES);
            I3=I3+1;
            IF (MOVCNTR(I3)=I2)
            THEN DO;
               BADMOV=1;
               RETURN;
               END;
            END;
         I3=0;
         DO WHILE(I3<NARMIES);
            I3=I3+1;
            IF (MOVCNTR(I3)=J2)
            THEN I3=2*NARMIES;
            END;
         IF (I3=NARMIES)
         THEN DO;
            BADMOV=1;
            RETURN;
            END;
         END;
      END;
   RETURN;
   END MOVCHK2;
/*****************************************************************/
```

```
/*****************************************************************/
/**                      TRANSLATE                              **/
/**    TRANSLATE CONVERTS A MOVE STORED AS A GRAPH IN MOVMAKER **/
/**    INTO A CODED STRING TO BE PLACED IN THE LIST OF MOVES.  **/
/**    THE SUBSCRIPTS OF THE MOVE IN MOVMAKER ARE CONVERTED    **/
/**    INTO CHARACTER STRINGS AND CONCATENATED IN A LEGAL SE-  **/
/**    QUENCE UNTIL THE WHOLE MOVE ORDER IS TRANSLATED.  THEN, **/
/**    THE MOVE IS STORED IN MOVES(L).                         **/
/*****************************************************************/
/*****************************************************************/
TRANSLATE: PROCEDURE(L);
   DCL (I,J,K,N) FIXED DECIMAL;
   DCL L FIXED DECIMAL;
   DCL (Z1,Z2,Z3,Z4) CHAR(30) VARYING;
   DCL P1 FIXED BINARY;
   L = L + 1;
   DO I = 1 TO NARMIES;
      J = ARMLOC(I);
      PUT LIST(J) STRING(Z1);
      IF (MOVCNTR(I)>NREGION)
      THEN K=MOVCNTR(I)-NREGION;
      ELSE K=MOVCNTR(I);
      PUT LIST(K) STRING(Z2);
      P1 = VERIFY(Z1,' ');
      Z1 = SUBSTR(Z1,P1);
      P1 = VERIFY(Z2,' ');
      Z2 = SUBSTR(Z2,P1);
      IF (MOVMAKER(J,K) = 1)
      THEN MOVES(L) = MOVES(L) || Z1 || '-' || Z2;
      ELSE DO;
         M = 1;
         DO WHILE (MOVCNTR(M)¬=K);
            M = M + 1;
            END;
         PUT LIST(ARMLOC(M)) STRING(Z3);
         P1=VERIFY(Z3,' ');
         Z3=SUBSTR(Z3,P1);
         IF (MOVCNTR(M)>NREGION)
         THEN N=MOVCNTR(M)-NREGION;
         ELSE N=MOVCNTR(M);
         PUT LIST(N) STRING(Z4);
         P2=VERIFY(Z4,' ');
         Z4=SUBSTR(Z4,P2);
         MOVES(L) = MOVES(L) || Z1 || 'S' || Z3 || '-' || Z4;
         END;
      IF (I < NARMIES) THEN MOVES(L) = MOVES(L) || ',';
      END;
   RETURN;
   END TRANSLATE;
/*****************************************************************/
/*****************************************************************/
/**                      INCR                                   **/
/**    INCR INCREMENTS THE VALUES OF MOVCNTR UNTIL IT IS NOT   **/
/**    POSSIBLE TO CREATE ANY MORE MOVES.                      **/
/*****************************************************************/
/*****************************************************************/
INCR:  PROCEDURE(I,IFLAG) RECURSIVE;
   DCL (I,J,IFLAG) FIXED DECIMAL;
   IF (I=1)
   THEN DO;
```

```
         IF (MOVCNTR(I)>REGMOD) THEN IFLAG=1;
         END;
      ELSE DO;
         IF (MOVCNTR(I)>REGMOD)
         THEN DO;
            MOVCNTR(I)=1;
            J=I-1;
            MOVCNTR(J)=MOVCNTR(J)+1;
            CALL INCR(J,IFLAG);
            END;
         END;
      RETURN;
      END INCR;
      END MAKE_MOVS;
      END STRATS;
*PROCESS
/*****************************************************************/
/*                        CNFLCT                                 */
/*     CNFLCT EXECUTES TWO INPUTTED MOVES:  XMOV AND YMOV        */
/*     REPRESENTING X'S MOVE AND Y'S MOVE, RESPECTIVELY.  ANY    */
/*     CONFLICTS IN THE MOVES ARE RESOLVED.                      */
/*                                                               */
/*     INTERNAL SUBROUTINES TO CNFLCT:                           */
/*         PLACE                                                 */
/*         PUTMOV                                                */
/*         RETREAT                                               */
/*         SUCCESS                                               */
/*****************************************************************/
CNFLCT:  PROC(MOVMAKER,XMOV,YMOV,BORDERS,NEW_BORD);
   DCL NREGION FIXED DECIMAL EXTERNAL STATIC;
   DCL (MOVMAKER(*,*),BORDERS(*,*),
        NEW_BORD(*,*)) FIXED DECIMAL;
   DCL (MTYPE(NREGION),DISLOG(NREGION)) FIXED DECIMAL;
   DCL DEPEND(NREGION) FIXED DECIMAL;
   DCL (XMOV,YMOV) CHAR(*) VARYING;
   DCL (I,J,ENTYPE,K,NSUPS,NESUPS) FIXED DECIMAL; /* LOCAL VARIABLES*/
   MTYPE=0; /* STEP 1:  PLACE THE MOVE*/
   MOVMAKER=0;  DISLOG=0;
   I=1;  J=2;
   CALL PLACE(XMOV,I,J);
   I=-I;  J=-J;
   CALL PLACE(YMOV,I,J);
   /*  STEP 2:  GET RID OF ALL OUT SUPPORTS.*/
   DO I=1 TO NREGION;
      IF (MTYPE(I)=2)  /* IS THIS A SUPPORT?*/
      THEN DO;
         J=2;
         CALL LOCATR(MOVMAKER,I,K,J); /*LOCATE THE MOVE*/
         J=0;
         ENTYPE=-(MOVMAKER(I,K)/2);
         DO WHILE (J<NREGION); /*IS THE SUPPORT BEING OUT?*/
            J=J+1;
            IF (MOVMAKER(J,I)=ENTYPE & J¬=K)
            THEN DO;
               MOVMAKER(I,I)=MOVMAKER(I,K)/2; /*OUT THE SUPPORT*/
               MOVMAKER(I,K)=0;
               MTYPE(I)=3;
               J=NREGION;
               END;
            END;
```

```
      END;
   END;
/* STEP 3:  CHECK FOR ALL MIRROR MOVES.*/
DO I = 1 TO NREGION;
   IF (MTYPE(I)=1)
   THEN DO;
      K=1;
      CALL LOCATR(MOVMAKER,I,J,K);
      IF (MOVMAKER(I,J)=-MOVMAKER(J,I))
      THEN DO;  /* MIRROR MOVE*/
         CALL SUPCNT(MOVMAKER,I,J,NSUPS);
         CALL SUPCNT(MOVMAKER,J,I,NESUPS);
         NSUPS=NSUPS+NESUPS;
         CALL WHOWINS(MOVMAKER,MTYPE,I,J,J,I,
                      DISLDG,NSUPS);
         IF (MTYPE(I)=1)
         THEN CALL MOVIT(MOVMAKER,MTYPE,I,J,DISLDG);
         ELSE IF (MTYPE(J)=1)
              THEN CALL MOVIT(MOVMAKER,MTYPE,J,I,DISLDG);
         MOVMAKER(I,J)=3;
         MOVMAKER(J,I)=3;
         END;
      END;
   END;
/* RESOLVE ALL CONFLICTS FROM OUTSIDE AN AREA.*/
DO I = 1 TO NREGION;
   IF (MTYPE(I)=1)
   THEN DO;
      K=1;
      CALL LOCATR(MOVMAKER,I,J,K);
      K=0;
      DO I2=1 TO NREGION;
         IF (MOVMAKER(I2,J)=-MOVMAKER(I,J) & I2¬=J)
         THEN K=I2;
         END;
      IF (K>0 & I¬=J)
      THEN DO;
         CALL SUPCNT(MOVMAKER,I,J,NSUPS);
         CALL SUPCNT(MOVMAKER,K,J,NESUPS);
         NSUPS=NSUPS+NESUPS;
         CALL WHOWINS(MOVMAKER,MTYPE,I,J,K,J,DISLDG,
                                NSUPS);
         END;
      END;
   END;
/*  CHECK ALL REMAINING MOVEMENT.  */
DO I = 1 TO NREGION;
   IF (MTYPE(I)=1)
   THEN DO;
      DEPEND=0;
      K=1;
      CALL LOCATR(MOVMAKER,I,J,K);
      IF (I¬=J) THEN CALL SUCCESS(I,J);
      END;
   END;
NEW_BORD=0;
DO I=1 TO NREGION;
   DO J= 1 TO NREGION;
      IF (MOVMAKER(I,J)>0 & MOVMAKER(I,J)<3)
      THEN NEW_BORD(I,1)=1;
```

```
         ELSE IF (MOVMAKER(I,J)<0)
              THEN NEW_BCRD(I,2)=1;
         END;
      END;
   /* PUT RETREAT ROUTINE IN HERE*/
   CALL RETREAT;
   RETURN;
/*****************************************************************/
/*                        SUCCESS                                */
/*     SUCCESS IS A RECURSIVE ROUTINE THAT IS USED TO EXECUTE    */
/*     MOVES IN WHICH AT LEAST ONE ARMY'S MOVE IS PENDING UPON   */
/*     THE SUCCESS OF ANOTHER ARMY'S MOVE.                       */
/*****************************************************************/
SUCCESS:  PROCEDURE(FROM,DEST) RECURSIVE;
   DCL (FROM,DEST,DEST2,I,NSUPS,NESUPS) FIXED DECIMAL;
   IF (FROM=DEST) THEN RETURN;
   IF (DEPEND(DEST)=1)
   THEN DO;
      CALL SNDHOM(MOVMAKER,MTYPE,FROM,DEST);
      CALL MOVIT(MOVMAKER,MTYPE,FROM,DEST,DISLOG);
      MOVMAKER(FROM,DEST)=3;
      MOVMAKER(DEST,FROM)=3;
      RETURN;
      END;
   IF (MTYPE(DEST)=1)
   THEN DO;
      I=1;
      CALL LOCATR(MOVMAKER,DEST,DEST2,I);
      DEPEND(FROM)=1;
      CALL SUCCESS(DEST,DEST2);
      DEPEND(FROM)=0;
      END;
   IF (MTYPE(DEST)=2)
   THEN DO;
      I=1;
      DO WHILE(MOVMAKER(I,FROM)¬=-MOVMAKER(FROM,DEST));
         I=I+1;
         END;
      CALL SUPCNT(MOVMAKER,FROM,DEST,NSUPS);
      CALL SUPCNT(MOVMAKER,I,FROM,NESUPS);
      NSUPS=NSUPS+NESUPS;
      CALL WHOWINS(MOVMAKER,MTYPE,FROM,DEST,I,FROM,
                        DISLOG,NSUPS);
      IF (MTYPE(FROM)=1)
      THEN CALL MOVIT(MOVMAKER,MTYPE,FROM,DEST,DISLOG);
      RETURN;
      END;
   IF (MTYPE(DEST)=0)
   THEN DO;
      CALL SNDHOM(MOVMAKER,MTYPE,FROM,DEST);
      CALL MOVIT(MOVMAKER,MTYPE,FROM,DEST,DISLOG);
      MOVMAKER(FROM,DEST)=3;
      MOVMAKER(DEST,FROM)=3;
      END;
   ELSE DO;
      IF ((MOVMAKER(FROM,DEST)*MOVMAKER(DEST,DEST))>=0)
      THEN DO;
         NSUPS=0;
         END;
      ELSE DO;
```

```
         CALL SUPCNT(MOVMAKER,FROM,DEST,NSUPS);
         CALL SUPCNT(MOVMAKER,DEST,DEST,NESUPS);
         NSUPS=NSUPS+NESUPS;
         END;
      CALL WHOWINS(MOVMAKER,MTYPE,FROM,DEST,DEST,DEST,
                   DISLOG,NSUPS);
      IF (MTYPE(FROM)=1)
      THEN CALL SUCCESS(FROM,DEST);
      END;
      IF (MTYPE(FROM)=0)
      THEN DO;
         DO I = 1 TO NREGION;
            IF (ABS(MOVMAKER(FROM,I))=1)
            THEN MTYPE(I)=1;
            END;
         END;
   RETURN;
   END SUCCESS;
/*****************************************************************/
/*                        PLACE                                  */
/*     PLACE TAKES A CODED STRING OF MOVES AND PLACES THE        */
/*     MOVES CORRECTLY INSIDE MOVMAKER, ACCORDING TO THE         */
/*     VALUES OF ATK AND SUP.                                    */
/*****************************************************************/
PLACE:  PROCEDURE(MOV,ATK,SUP);
   DCL (ATK,SUP) FIXED DECIMAL;
   DCL MOV CHAR(*) VARYING;
   DCL (ENDSTR,POS,POS2,CURPOS) FIXED BINARY;
   DCL SUB CHAR(30) VARYING;
   DCL SUB2 CHAR(30) VARYING;
   DCL (Z1,Z2) CHAR(30) VARYING;
   DCL (N1,N2) FIXED DECIMAL;
   CURPOS = 1;
   SUB = MOV;
   ENDSTR = LENGTH(SUB) - 1;
   POS = INDEX(SUB,',');
   DO WHILE(POS > 0);
   SUB2 = SUB;
      SUB2 = SUBSTR(SUB,CURPOS,POS);
      POS2 = INDEX(SUB2,'S');
      IF (POS2 = 0)
      THEN CALL PUTMOV(ATK);
      ELSE CALL PUTMOV(SUP);
      CURPOS = POS + 1;
      SUB = SUBSTR(SUB,CURPOS);
      CURPOS = 1;
      POS = INDEX(SUB,',');
      END;
   SUB2 = SUB;
   POS2 = INDEX(SUB,'S');
   IF (POS2 = 0)
   THEN CALL PUTMOV(ATK);
   ELSE CALL PUTMOV(SUP);
   RETURN;
/*****************************************************************/
/*                        PUTMOV                                 */
/*     PUTMOV IS AN INTERNAL PROCEDURE TO PLACE.  IT ACTUALLY    */
/*     PLACES THE ATTACK OR SUPPORT MOVE INTO MOVMAKER.  THE     */
/*     KEY IS +1 OR -1 FOR AN ATTACK, AND +2 OR -2 FOR A         */
/*     SUPPORT.                                                  */
```

```
/*****************************************************************/
PUTMCV:  PROCEDURE(KEY);
   DCL KEY FIXED DECIMAL;
   CURPOS = 1;
   PCS2 = VERIFY(SUB2,'0123456789 ') - 1;
   Z1 = SUBSTR(SUB2,CURPOS,PCS2);
   GET LIST(N1) STRING(Z1);
   CURPOS = INDEX(SUB2,'-') + 1;
   SUB2= SUBSTR(SUB2,CURPOS);
   PCS2 = VERIFY(SUB2,'0123456789 ');
   IF (PCS2>0)
   THEN Z2 = SUBSTR(SUB2,1,PCS2-1);
   ELSE Z2 = SUB2;
   GET LIST(N2) STRING(Z2);
   MOVMAKER(N1,N2) = KEY;
   MTYPE(N1)=ABS(KEY);
   RETURN;
   END PUTMCV;
   END PLACE;
/*****************************************************************/
/*                        RETREAT                                */
/*     RETREAT IS USED TO EXECUTE ALL RETREATS WHICH MUST BE     */
/*     PERFORMED AS THE RESULT OF MOVEMENT CONFLICTS, WHICH      */
/*     ARE DONE IN THE ROUTINE CNFLCT.                           */
/*****************************************************************/
RETREAT:  PROCEDURE;
   DCL (I,J,TRUE,NOPLACE) FIXED DECIMAL;
   DO I = 1 TO NREGION;
      IF (ABS(DISLOG(I))=1)
      THEN DO;
         J=1; TRUE=0;  NOPLACE=0;
         DO WHILE (NOPLACE=TRUE);
            IF (BORDERS(I,J)=1 & NEW_BORD(J,1)=0 & NEW_BORD(J,2)=0
                & I¬=J & MOVMAKER(J,I)=0)
            THEN DO;
               IF (DISLOG(I)<0)
               THEN NEW_BORD(J,2)=1;
               ELSE NEW_BORD(J,1)=1;
               NOPLACE=1;
               END;
            IF (J=NREGION) THEN NOPLACE=1;
            ELSE J=J+1;
            END;
         END;
      END;
   RETURN;
   END RETREAT;
   END CNFLCT;
```

```
*PROCESS
 /*****************************************************************/
 /*                          SNDHCM                               */
 /*      SNDHCM ASSUMES ALL OF THE UNITS SUPPORTING THE MOVEMENT */
 /*      FROM REGION I TO REGION J HAVE BEEN COUNTED.  RESULT-   */
 /*      INGLY, EACH UNIT IS CHANGED TO A HOLD MOVE IN ITS OWN   */
 /*      REGION.  IN OTHER WORDS, EACH SUPPORT MOVE IS SEND HOMD.*/
 /*****************************************************************/
 SNDHCM:  PROCEDURE(MOVMAKER,MTYPE,I,J);
    DCL NREGION FIXED DECIMAL EXTERNAL STATIC;
    DCL (I,J,K,MOVMAKER(*,*),MTYPE(*),SUP) FIXED DECIMAL;
    SUP=2*MOVMAKER(I,J);
    DO K = 1 TO NREGION;
       IF (K¬=I & MOVMAKER(K,J)=SUP)
       THEN DO;
          MTYPE(K)=3;
          MOVMAKER(K,K)=MOVMAKER(K,J);
          MOVMAKER(K,J)=0;
          END;
       END;
    RETURN;
    END SNDHCM;
*PROCESS
 /*****************************************************************/
 /*                          DOMINAT                              */
 /*      DOMINAT TAKES AN INPUTTED GAME MATRIX (GAM_MAT) AND     */
 /*      ELIMINATES DOMINATED STRATEGIES FOR BOTH PLAYERS, BUT   */
 /*      WILL NOT REDUCE TO LESS THAN TWO STRATEGIES FOR EITHER  */
 /*      PLAYER.                                                 */
 /*****************************************************************/
 DOMINAT:  PROCEDURE(GAM_MAT,NX,NY,DOMROWS,DOMCOLS,ELIM1,ELIM2);
    DCL (DOMROWS,DOMCOLS) FIXED;
    DCL GAM_MAT(*,*) FIXED DECIMAL;
    DCL (NX,NY,ELIM1(*),ELIM2(*)) FIXED;
    DCL (I,I2,J,J2,CHANGE) FIXED;
    ELIM1=0; ELIM2=0;
    DOMROWS=NX; DOMCOLS=NY;
    REDO:;
    CHANGE=0;
    I = 0;
    DO WHILE (DOMROWS>2 & I<NX);
       I = I + 1;
       I2=0;
       DO WHILE ((I2<NX) & (DOMROWS>2) & ELIM1(I)=0);
          I2=I2+1;
          IF (I¬=I2 & ELIM1(I2)=0)
          THEN DO;
             J = 0;
             DO WHILE (J<NY);
                J = J + 1;
                IF (GAM_MAT(I,J)<GAM_MAT(I2,J) & ELIM2(J)=0)
                THEN J = 2*NY;
                END;
             IF (J=NY)
             THEN DO;
                ELIM1(I2)=1;
                DOMROWS = DOMROWS - 1;
                CHANGE =1;
                END;
             END;
```

```
            END;
         END;
      J = C;
      DO WHILE (DOMCOLS>2 & J<NY);
         J = J + 1;
         J2=0;
         DO WHILE ((J2<NY) & (DOMCOLS>2) & ELIM2(J)=0);
            J2=J2+1;
            IF (J¬=J2 & ELIM2(J2)=0)
            THEN DO;
               I = C;
               DO WHILE (I<NX);
                  I = I + 1;
                  IF (ELIM1(I)=C & GAM_MAT(I,J)>GAM_MAT(I,J2))
                  THEN I = 2*NX;
                  END;
               IF (I=NX)
               THEN DO;
                  ELIM2(J2)=1;
                  DOMCOLS=DOMCOLS-1;
                  CHANGE=1;
                  END;
               END;
            END;
         END;
      IF (CHANGE=1)
      THEN GO TO REDO;
      IF (DOMROWS<NX | DOMCOLS<NY)
      THEN DO;
         REDUCE:  BEGIN;
         DCL RED_BORD(DOMROWS,DOMCOLS) FIXED;
         I2=1;
         DO I = 1 TO NX;
            IF (ELIM1(I)=C)
            THEN DO;
               J2=1;
               DO J = 1 TO NY;
                  IF (ELIM2(J)=0)
                  THEN DO;
                     RED_BORD(I2,J2)=GAM_MAT(I,J);
                     J2=J2+1;
                     END;
                  END;
               I2=I2+1;
               END;
            END;
         GAM_MAT=0;
         DO I = 1 TO DOMROWS;
            DO J = 1 TO DOMCOLS;
               GAM_MAT(I,J)=RED_BORD(I,J);
               END;
            END;
         END REDUCE;
      RETURN;
      END DOMINAT;
   *PROCESS
    /*****************************************************************/
    /*                          SIMPLEX                              */
    /*     SIMPLEX WILL FIND THE GAME VALUE OF A GIVEN PAYOFF        */
    /*     MATRIX (PAYOFF) BY THE SIMPLEX METHOD OF LINEAR PROGRAM-*/
```

```
/*      MING.  IT IS ASSUMED THAT THE MATRIX HAS ONLY UNDOM-     */
/*      INATED STRATEGIES (XSTRAT AND YSTRAT) AND THE NUMBER OF */
/*      UNDOMINATED STRATEGIES IS GIVEN (DOMROWS,DOMCOLS).       */
/*****************************************************************/
SIMPLEX:  PROCEDURE(GAM_MAT,DOMROWS,DOMCOLS,YSTRAT,XSTRAT,GVALUE);
   DCL GAM_MAT(*,*) FIXED DECIMAL;
   DCL (LEAST,MAXVAL) FIXED DECIMAL;
   DCL (DOMROWS,DOMCOLS) FIXED DECIMAL;
   DCL (XSTRAT(*),YSTRAT(*),YTEMP(DOMROWS)) FLOAT DECIMAL;
   DCL  (I,J,K,L,IL,IR,IX,NSIMPS,MAXROW,MAXCOL) FIXED DECIMAL;
   DCL (MAXRAT,MLT,GVALUE) FLOAT DECIMAL;
   IL=2;  IR=DOMCOLS+1;  IX=DOMROWS+DOMCOLS+1;
   N1=DOMROWS+2;
   N2=2*DOMCOLS+1;
   LEAST=0;
   DO I = 1 TO DOMROWS;
      DO J = 1 TO DOMCOLS;
         IF (GAM_MAT(I,J)<LEAST)
         THEN LEAST = GAM_MAT(I,J);
         END;
      END;
   IF (LEAST<0) THEN LEAST = ABS(LEAST)+1;
   SIMP_TAB:  BEGIN;
   DCL SIMP(N1,IX) FLOAT DECIMAL;
   XSTRAT=0;  YSTRAT=0;
   SIMP=0;
   YTEMP=0;
   DO I = 1 TO DOMROWS;
      DO J = 1 TO DOMCOLS;
         K=J+1;
         SIMP(I,K)=GAM_MAT(I,J) + LEAST;
         L=IR+J;
         IF (I=J) THEN SIMP(I,L)=1;
      END;
      SIMP(I,1)=1;
   END;
   IC=DOMROWS+1;
   K=IC+1;
   DO J = 1 TO IX;
      IF (J>=IL &  J<=IR) THEN SIMP(K,J)=1;
      END;
   NSIMPS=0;
   NFLAG=0;
   DO WHILE (NFLAG=0);
      MAXCOL=0;  MAXVAL=0;
      DO J = IL TO IR;
         IF (SIMP(K,J)>MAXVAL)
         THEN DO;
            MAXCOL=J;
            MAXVAL=SIMP(K,MAXCOL);
            END;
         END;
      IF (MAXCOL=0)
      THEN DO;
         GVALUE=(1/SIMP(IC,1))-LEAST;
         RETURN;
         END;
      MAXRAT=10000.;
      DO I = 1 TO DOMROWS;
         IF (SIMP(I,MAXCOL)¬=0)
```

```
         THEN DO;
            RATIO = SIMP(I,1)/SIMP(I,MAXCOL);
            IF (RATIO>0 & RATIO<MAXRAT)
            THEN DO;
               MAXRAT=RATIO;
               MAXROW=I;
               END;
            END;
         END;
      YTEMP(MAXROW)=MAXCOL-1;
      PIVDIV=SIMP(MAXROW,MAXCOL);
      DO J = 1 TO IX;
         SIMP(MAXROW,J)=SIMP(MAXROW,J)/PIVDIV;
         END;
      DO I = 1 TO DOMROWS;
         IF (I¬=MAXROW)
         THEN DO;
            MLT=SIMP(I,MAXCOL);
            DO J = 1 TO IX;
               SIMP(I,J)=SIMP(I,J)-(SIMP(MAXROW,J)*MLT);
               END;
            END;
         END;
      DO J=1 TO IX;
         SIMP(IC,J)=0;
         END;
      DO I = 1 TO DOMROWS;
         IF (YTEMP(I)>0)
         THEN DO;
            DO J=1 TO IX;
               SIMP(IC,J)=SIMP(IC,J)+SIMP(I,J);
               END;
            END;
         END;
      DO J = IL TO IX;
         IF (J<=IR)
         THEN SIMP(K,J)=1-SIMP(IC,J);
         ELSE SIMP(K,J)=-SIMP(IC,J);
         END;
      NSIMPS=NSIMPS+1;
      IEND=1;
      DO WHILE(IEND < IX);
         IEND = IEND + 1;
         IF (SIMP(K,IEND)>0)
         THEN IEND=2*IX;
         END;
      IF (IEND=IX)
      THEN DO;
         CALL NOMORE;
         RETURN;
         END;
   END;
   RETURN;
NOMORE:  PROCEDURE;
   GVALUE=1/SIMP(IC,1);
   DO I = 1 TO DOMROWS;
      NORDER=YTEMP(I);
      IF (NORDER=0)
      THEN DO;
         GVALUE=GVALUE-LEAST;
```

```
          RETURN;
          END;
       YSTRAT(NCRDER)=GVALUE*SIMP(I,1);
       JX=I+IR;
       XSTRAT(I)=-SIMP(K,JX) * GVALUE;
       END;
    GVALUE=GVALUE-LEAST;
    RETURN;
    END NCMCRE;
    END SIMP_TAB;
    END SIMPLEX;
*PRCCESS
 /*****************************************************************/
 /*                        ARMYCT                                 */
 /*     ARMYCT CCUNTS THE NUMBERS CF BOTH PLAYERS (NARMS1 AND     */
 /*     NARMS2) WHICH ARE IN CLC_BCRD.                            */
 /*****************************************************************/
 ARMYCT:  PRCCEDURE(NARMS1,NARMS2,CLD_BCRD,PCSS,CCUNTRY);
    DCL NREGICN FIXED DECIMAL EXTERNAL STATIC;
    DCL (POSS(*),COUNTRY(*)) FIXED DECIMAL;
    CCL CLD_BORD(*,*) FIXED CECIMAL;
    DCL (NARMS1,NARMS2) FIXED DECIMAL;
    DCL I FIXED DECIMAL;
    NARMS1 = 0;
    NARMS2 = C;
    DC I = 1 TC NREGICN;
       IF (CLD_BCRD(I,1)=1)
       THEN DC;
          NARMS1=NARMS1+1;
          PCSS(I)=1;
          END;
       ELSE IF (OLD_BCRC(I,2)=1)
            THEN CC;
               NARMS2=NARMS2+1;
               POSS(I)=C;
               END;
            ELSE IF (CCUNTRY(I)=1)
                  THEN PCSS(I)=1;
                  ELSE PCSS(I)=C;
       END;
    RETURN;
    END ARMYCT;
*PRCCESS
 /*****************************************************************/
 /*                        WIN                                    */
 /*     WIN EVALUATES THE WINNING "SCORE" CF THE TWC PLAYERS BY   */
 /*     DETERMINING THE NET TCTAL OF REGICNS GAINED BY BCTH       */
 /*     PLAYERS SINCE THE INITIAL TURN.                           */
 /*     THE VALUE (GVALUE) RETURNED IS A ZERO-SUM PAYCFF REPRE-   */
 /*     SENTING THE NET TOTAL CF REGICNS GAINED BY PLAYER X.      */
 /*****************************************************************/
 WIN:  PRCCEDURE(NEW_BORD,ECCN_CENTR,CCUNTRY,XORIG,YCRIG,
                 GVALUE);
    DCL NREGICN FIXED DECIMAL EXTERNAL STATIC;
    DCL (XORIG,YCRIG) FIXED CECIMAL;
    DCL (ECCN_CENTR(*),CCUNTRY(*)) FIXED CECIMAL;
    DCL (NEW_BORD(*,*)) FIXED CECIMAL;
    DCL (XREGICNS,YREGICNS) FIXED DECIMAL;
    DCL GVALUE FLCAT CECIMAL;
    XREGICNS = 0;
```

```
    YREGIONS = 0;
    DO I = 1 TO NREGION;
       IF (ECON_CENTR(I) = 1)
       THEN DO;
          IF (NEW_BORD(I,1) = 1)
          THEN XREGIONS = XREGIONS + 1;
          ELSE IF (COUNTRY(I)=1 & NEW_BORD(I,2)¬=1)
               THEN XREGIONS=XREGIONS+1;
          END;
       END;
    XREGIONS = XREGIONS - XORIG;
    GVALUE=XREGIONS;
    RETURN;
    RETURN;
    END WIN;
*PROCESS
 /*****************************************************************/
 /*                        SADDLE                                 */
 /*      SADDLE FINDS A SADDLE POINT, IF ANY, OF A GIVEN GAME     */
 /*      MATRIX (GAM_MAT).                                        */
 /*****************************************************************/
 SADDLE:  PROCEDURE(GAM_MAT,FLAG,DOMROWS,DOMCOLS,ROW,COL,GVALUE);
    DCL (GAM_MAT(*,*),FLAG) FIXED DECIMAL;
    DCL GVALUE FLOAT DECIMAL;
    DCL (I,J,K,L,POTPT,ROW,COL) FIXED DECIMAL;
    DCL (DOMROWS,DOMCOLS) FIXED DECIMAL;
    DO I = 1 TO DOMROWS;
       DO J = 1 TO DOMCOLS;
          POTPT = GAM_MAT(I,J);
          ROW=I;  COL=J;
          K = 1;
          FLAG=1;
          DO WHILE (K<=DOMCOLS & FLAG=1);
             IF (GAM_MAT(I,K)<POTPT)
             THEN FLAG=0;
             K=K+1;
             END;
          IF (FLAG=1)
          THEN DO;
             L=1;
             DO WHILE(L<=DOMROWS & FLAG=1);
                IF (GAM_MAT(L,J)>POTPT)
                THEN FLAG=0;
                L=L+1;
                END;
             IF (FLAG=1)
             THEN DO;
                GVALUE=POTPT;
                RETURN;
                END;
             END;
          END;
       END;
       FLAG = 0;
       RETURN;
    END SADDLE;
*PROCESS
 /*****************************************************************/
 /*                        MOVIT                                  */
 /*      MOVIT ASSUMES THAT THE MOVE FROM REGION I TO REGION J    */
```

```
/*      HAS NO MORE CONFLICTS TO RESOLVE, AND THE UNIT IS MOVED */
/*      INTO THE REGION, STORING THIS ACTIVITY AS A HOLD MOVE   */
/*      BY THE UNIT IN REGION J.                                */
/*************************************************************/
MOVIT:  PROCEDURE(MOVMAKER,MTYPE,I,J,DISLOG);
   DCL NREGION FIXED DECIMAL EXTERNAL STATIC;
   DCL (MOVMAKER(*,*),MTYPE(*),DISLOG(*)) FIXED DECIMAL;
   DCL (I,J) FIXED DECIMAL;
   IF (ABS(MOVMAKER(I,J))=1)
   THEN DO;
      IF (ABS(MOVMAKER(J,J))=1)
      THEN DISLOG(J)=MOVMAKER(J,J);
      MOVMAKER(J,J)=MOVMAKER(I,J);
      MOVMAKER(I,J)=0;
      MTYPE(I)=0;
       MTYPE(J)=3;
      END;
   RETURN;
   END MOVIT;
*PROCESS
/*************************************************************/
/*                        WHOWINS                            */
/*      WHOWINS FINDS OUT THE RESULT OF A CONFLICT BETWEEN TWO  */
/*      UNITS.  THE FIRST UNIT IS MOVING FROM REGION 'FROM' TO  */
/*      REGION 'DEST'.  THE SECOND UNIT IS MOVING FROM REGION   */
/*      'ENEMY' TO REGION 'DEST3'.  THERE IS A CONFLICT IF ONE  */
/*      OF THE FOLLOWING CONFLICTS TAKES PLACE:                 */
/*              1.DEST=DEST3                                    */
/*              2.FROM=DEST3 AND DEST=ENEMY                     */
/*************************************************************/
WHOWINS:  PROCEDURE(MOVMAKER,MTYPE,FROM,DEST,ENEMY,DEST3,
                    DISLOG,NSUPS);
   DCL NREGION FIXED DECIMAL EXTERNAL STATIC;
   DCL (MOVMAKER(*,*),MTYPE(*),DISLOG(*)) FIXED DECIMAL;
   DCL (FROM,DEST,ENEMY,DEST3,NSUPS) FIXED DECIMAL;
   IF (MOVMAKER(FROM,DEST)=0 | MOVMAKER(ENEMY,DEST3)=0)
   THEN RETURN;  /* FALSE MOVE.*/
   IF (NSUPS=0)
   THEN DO;
      MOVMAKER(FROM,FROM)=MOVMAKER(FROM,DEST);
      MOVMAKER(ENEMY,ENEMY)=MOVMAKER(ENEMY,DEST3);
      MTYPE(FROM)=3;
      MTYPE(ENEMY)=3;
      IF (FROM=DEST3 & DEST=ENEMY)
      THEN DO;
         CALL SNDHOM(MOVMAKER,MTYPE,ENEMY,DEST3);
         CALL SNDHOM(MOVMAKER,MTYPE,FROM,DEST);
         END;
      ELSE DO;
         CALL SNDHOM(MOVMAKER,MTYPE,FROM,DEST);
         CALL SNDHOM(MOVMAKER,MTYPE,ENEMY,DEST3);
         END;
      IF (FROM¬=DEST) THEN MOVMAKER(FROM,DEST)=0;
      IF (ENEMY¬=DEST3) THEN MOVMAKER(ENEMY,DEST3)=0;
      IF (DEST=DEST3 & FROM¬=DEST & ENEMY¬=DEST3
          & MOVMAKER(FROM,DEST3)=0)
      THEN MOVMAKER(FROM,DEST)=3;
      END;
   ELSE DO;
      IF ((NSUPS*MOVMAKER(FROM,DEST))>0)
```

```
      THEN DO;
         IF (ENEMY¬=DEST3)
         THEN DO;
            MOVMAKER(ENEMY,ENEMY)=MOVMAKER(ENEMY,DEST3);
            MTYPE(ENEMY)=3;
            END;
         ELSE DO;
            DISLOG(DEST3)=MOVMAKER(ENEMY,DEST3);
            MTYPE(ENEMY)=0;
            END;
         CALL SNDHCM(MOVMAKER,MTYPE,ENEMY,DEST3);
         MOVMAKER(ENEMY,DEST3)=0;
         END;
      ELSE DO;
         IF (FROM¬=DEST)
         THEN DO;
            MOVMAKER(FROM,FROM)=MOVMAKER(FROM,DEST);
            MTYPE(FROM)=3;
            END;
         ELSE DO;
            DISLOG(DEST)=MOVMAKER(FROM,DEST);
            MTYPE(FROM)=0;
            END;
         CALL SNDHCM(MOVMAKER,MTYPE,FROM,DEST);
         MOVMAKER(FROM,DEST)=0;
         END;
      END;
   RETURN;
   END WHOWINS;
*PROCESS
/*****************************************************************/
/*                          SUPCNT                               */
/*      SUPCNT COUNTS THE NUMBER OF UNITS SUPPORTING THE MOVE-   */
/*      MENT FROM REGION L1 TO REGION L2.                        */
/*****************************************************************/
SUPCNT:  PROCEDURE(MOVMAKER,L1,L2,FRIENDS);
   DCL NREGION FIXED DECIMAL EXTERNAL STATIC;
   DCL MOVMAKER(*,*) FIXED DECIMAL;
   DCL (FRIENDS,L1,L2,SUP,I) FIXED DECIMAL;
   FRIENDS=0;
   SUP=MOVMAKER(L1,L2)*2;
   DO I=1 TO NREGION;
      IF (MOVMAKER(I,L2)=SUP)
      THEN FRIENDS=FRIENDS+MOVMAKER(I,L2);
      END;
   RETURN;
   END SUPCNT;
*PROCESS
/*****************************************************************/
/*                          LOCATR                               */
/*      LOCATR SEARCHES A ROW I OF MOVMAKER UNTIL A MOVE OF      */
/*      CLASS TYPE IS FOUND.  IF TYPE=1, WE ARE LOOKING FOR A    */
/*      MOVEMENT MOVE.  IF TYPE=2, WE ARE LOOKING FOR A SUPPORT  */
/*      MOVE.                                                    */
/*****************************************************************/
LOCATR:  PROCEDURE(MOVMAKER,I,J,TYPE);
   DCL MOVMAKER(*,*) FIXED DECIMAL;
   DCL (I,J,TYPE) FIXED DECIMAL;
   J=1;
   DO WHILE(ABS(MOVMAKER(I,J))¬=TYPE);
```

```
        J=J+1;
        END;
    RETURN;
    END LOCATR;
*DATA
5
2
1 2 3 0 1 1 1
2 1 3 4 5 0 2 1 1
3 1 2 4 0 1 0 1
4 2 3 5 0 1 1 0
5 2 4 0 2 1 0
9
2
1 2 3 5 0 1 1 0
2 1 3 4 0 1 1 1
3 1 2 4 5 6 8 0 1 0 0
4 2 3 8 0 1 1 1
5 1 3 6 0 1 1 0
6 3 5 7 8 0 2 1 0
7 6 8 9 0 2 1 1
8 3 4 6 7 9 0 2 0 0
9 7 8 9 0 2 1 0
```

VITA

John Owen Franklin Long was born in Owensboro, Kentucky, on August 25, 1957. After moving to Pensacola, Florida, he attended elementary, junior high, and high schools, and graduated from Pensacola Private School of Liberal Arts in May 1975. From that summer until August 1976, he attended Pensacola Junior College and received a General Associate of Arts degree. In September 1976, he transferred to The University of Tennessee, Knoxville, where he completed requirements for a Bachelor of Arts degree in computer science in December 1978.

Entering the Graduate School of The University of Tennessee, Knoxville, in January 1979, Mr. Long worked toward a Master of Science degree in computer science. During the 1979-1980 school year, he worked part-time as a graduate teaching assistant.

While attending The University of Tennessee, Knoxville, Mr. Long was an active member of the Full Gospel Student Fellowship, serving as an officer in the 1977-1978 school year, and as president during the 1979-1980 school year.

After completing his course work in the summer of 1980, Mr. Long moved to Falls Church, Virginia, to work for the BDM Corporation and to complete work on his thesis. His degree was completed in March 1981.