AD-A043 453 CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 9/2
BKG. A PROGRAM THAT PLAYS BACKGAMMON.(U)
JUL 77 H J BERLINER

UNCLASSIFIED F44620-73-C-0074
AFOSR-TR-77-1143 NL

| OF |
AD
A043453

END
DATE
FILMED
9-77
DDC

BKG --
A Program that plays Backgammon

Hans J. Berliner

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)
NOTICE OF TRANSMITTAL TO DDC
This technical report has been reviewed and is approved for public release IAW AFR 190-12 (7b).
Distribution is unlimited.
A. D. BLOSE
Technical Information Officer

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) **REPORT DOCUMENTATION PAGE**

READ INSTRUCTIONS BEFORE COMPLETING FORM

(18) 1. REPORT NUMBER: AFOSR-TR- 77-1143

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

(6) 4. TITLE (and Subtitle): BKG, A PROGRAM THAT PLAYS BACKGAMMON.

(9) 5. TYPE OF REPORT & PERIOD COVERED: Interim rept.

6. PERFORMING ORG. REPORT NUMBER: 7

(10) 7. AUTHOR(s): Hans J. Berliner

(15) 8. CONTRACT OR GRANT NUMBER(s): F44620-73-C-0074

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Carnegie-Mellon University
Computer Science Dept.
Pittsburgh, PA 15213

(16) 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F 2304/A2 (17) A2

(11) 11. CONTROLLING OFFICE NAME AND ADDRESS
Defense Advanced Research Projects
1400 Wilson Blvd
Arlington, VA 22209

12. REPORT DATE: July 1977

13. NUMBER OF PAGES: 29

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Air Force Office of Scientific Research
Bolling AFB, DC 20332

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

(12) 28p.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper is both a description of an existing backgammon program and a theoretical discussion of some important issues in evaluation. The program plays a generally competent game at an intermediate level of skill. It correctly solves a high percentage of intermediate level problems in books. Although it only doubles and accepts doubles during the running game, it does several non-trivial things perfectly. In discussing the structure of evaluation, we consider partitioning all game states into mutually exclusive state-classes. These state-classes are very useful in a knowledge-based system, as they allow relatively easy assimilation of new knowledge. They also permit the building of opponent models based upon what evidence shows the opponent knows in each state-class.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

403 081

BKG --
A Program that plays Backgammon

Hans J. Berliner

Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pa. 15213

July, 1977

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (contract F44620-73-C-0074) and is monitored by the Air Force Office of Scientific Research.

ABSTRACT

This paper is both a description of an existing backgammon program and a theoretical discussion of some important issues in evaluation. The program plays a generally competent game at an intermediate level of skill. It correctly solves a high percentage of intermediate level problems in books. Although it only doubles and accepts doubles during the running game, it does several non-trivial things perfectly.

In discussing the structure of evaluation, we consider partitioning all game states into mutually exclusive state-classes. These state-classes are very useful in a knowledge-based system, as they allow relatively easy assimilation of new knowledge. They also permit the building of opponent models based upon what evidence shows the opponent knows in each state-class.

I. Why Yet Another Game?

Backgammon is a game of skill and chance. It is played on a vector of 24 "points". From a predefined starting position, both sides move in opposite directions, with the object of first removing all 15 of one's own men off the end of the vector. There are rules for moving, capturing, re-entering captured men, blocking points so an opponent cannot move there, and finally removing men from the board near the end of the game. The reader not familiar with backgammon would do well to scan the rules of a standard work such as Jacoby & Crawford [Ja73] before proceeding into this paper in great detail.

The thing that makes backgammon an interesting object of study for AI is that in any given position (of with there are 10^{20} [Le76]), there are 21 possible combinations that the throw of two dice can produce. Each of these, can be played legally in the average board position about 40 different ways. Thus if one were to investigate a backgammon position by tree searching, it would be necessary to deal with a branching factor of more than 800 (!!) at every node. Clearly this is completely impractical. Therefore backgammon must be approached with evaluation and knowledge in mind. Position P1 will have to preferred over position P2 because it has features that more endear it to the player who can produce it than the features that obtain in P2.

In a game such as chess, it has been customary to search very large trees of 5000 to 2 million terminal nodes. In such a paradigm, the execution of a terminal evaluation function requires a certain amount of time, which must then be multiplied by the expected number of terminal nodes in the search. Thus designers of chess programs are very circumspect in creating evaluation functions which require lengthy execution times. For this reason certain knowledge that is not trivial to compute is usually left out so that the program may operate faster and search more. Since there can be little or no searching in a practical backgammon program, these contingencies will not apply. On the contrary, it is desirable to apply all possible knowledge to successor positions of the root node, in an attempt to find the best next move. Further, the fact that modern backgammon involves doubling and accepting doubles places an even greater emphasis on the use of knowledge for knowing when to double and when to accept doubles. It is the encoding of knowledge and the subsequent selection effectiveness of the evaluation function that is of interest. This subject is treated in depth in sections IV and V. However, before going to that part of our research, we describe certain peripheral artifacts that are important for understanding the whole project.

II. The Structure of BKG

BKG is an interactive backgammon program. It is the result of about one man-year of effort since mid-1974. It is written in BLISS [Wu71], a system implementation language, and runs under the TOPS-10 monitor on the PDP-10's at Carnegie-Mellon University. It encompasses more than 80 pages of code, occupies 19K of 36-bit words in core and a further 11K of tables on secondary storage. BKG runs interactively on four different kinds of video terminals (including a graphics terminal which moves the men with a graphic hand, and will upset the board

occasionally when BKG loses), and a standard model 33 teletype. These routines were written by Phil Karlton and Steve Rubin at Carnegie-Mellon University.

BKG operates, similarly to many game playing programs, by executing a minor cycle which makes moves within a major one which plays games. When BKG is running interactively, it displays appropriate outputs and prompts at each step of the cycles. The minor cycle (see Figure 1) operates as follows: First, BKG checks to see if the side whose turn it now is wishes to double (checking first that it would be legal for that side to do so). If a double is made, BKG asks if the double is accepted. If so, BKG adjusts the position and denomination of the doubling cube and proceeds. If not, the game is over and BKG exits to the major cycle. Next BKG checks to see if it is to throw dice, or receive a roll from the console. BKG uses a random number generator to throw dice. It then generates a list of all possible legal moves for the given roll. If it is the program's turn to play, it serves these potential moves up, one at a time, to the evaluation procedure. It then selects the best. If it is not the machine's turn to play it waits to receive a move from its environment. It then checks the legal move list to see if this move is on the list. If not, it requests a legal move. If the received move is legal, BKG puts it into canonical form. It then executes the legal move by updating the board configuration. If the conditions for one side winning have been met, it exits to the major cycle.

The major cycle is activated whenever a new game is about to begin. If a game is just over, BKG adjusts the overall record of the current competion in favor of the side that just won. It then asks if another game is desired (unless it has been preset for a certain number of games and this number has not been reached as yet). If the answer is negative, the program terminates. Otherwise, it sets the board up, throws dice to see who starts, and yieids control to the minor cycle.

BKG can operate in several different modes. It's usual mode is to play a human opponent interactively at a video terminal. However, it can also monitor a game between two opponents while rolling dice for both, and doing the bookkeeping for doubling and accepting decisions. It can also play itself, either while displaying all actions at the terminal, or by only reporting the results of a series of games. For this simulation mode, it is neccessary to type in a starting position and the number of itterations desired. For certain types of positions that BKG can play well, this mode can be used to determine within reasonable limits of accuracy what the chances of the respective sides are.

When one or more human opponents are involved, there is the option of letting the humans throw dice for themselves and entering the roll when the program prompts for it. For ordinary purposes this is much too slow a way of playing the program. There is, however, a practical reason for this mode. In tournaments it will be necessary to actually throw dice at the table for both sides, and a method must exist for entering the rolls. It is rather interesting to note as an aside, that while we have seen quite a few illegal moves made in tournaments, this should never happen to BKG, as if a move is not on its legal move list, it would not accept the opponent playing it.

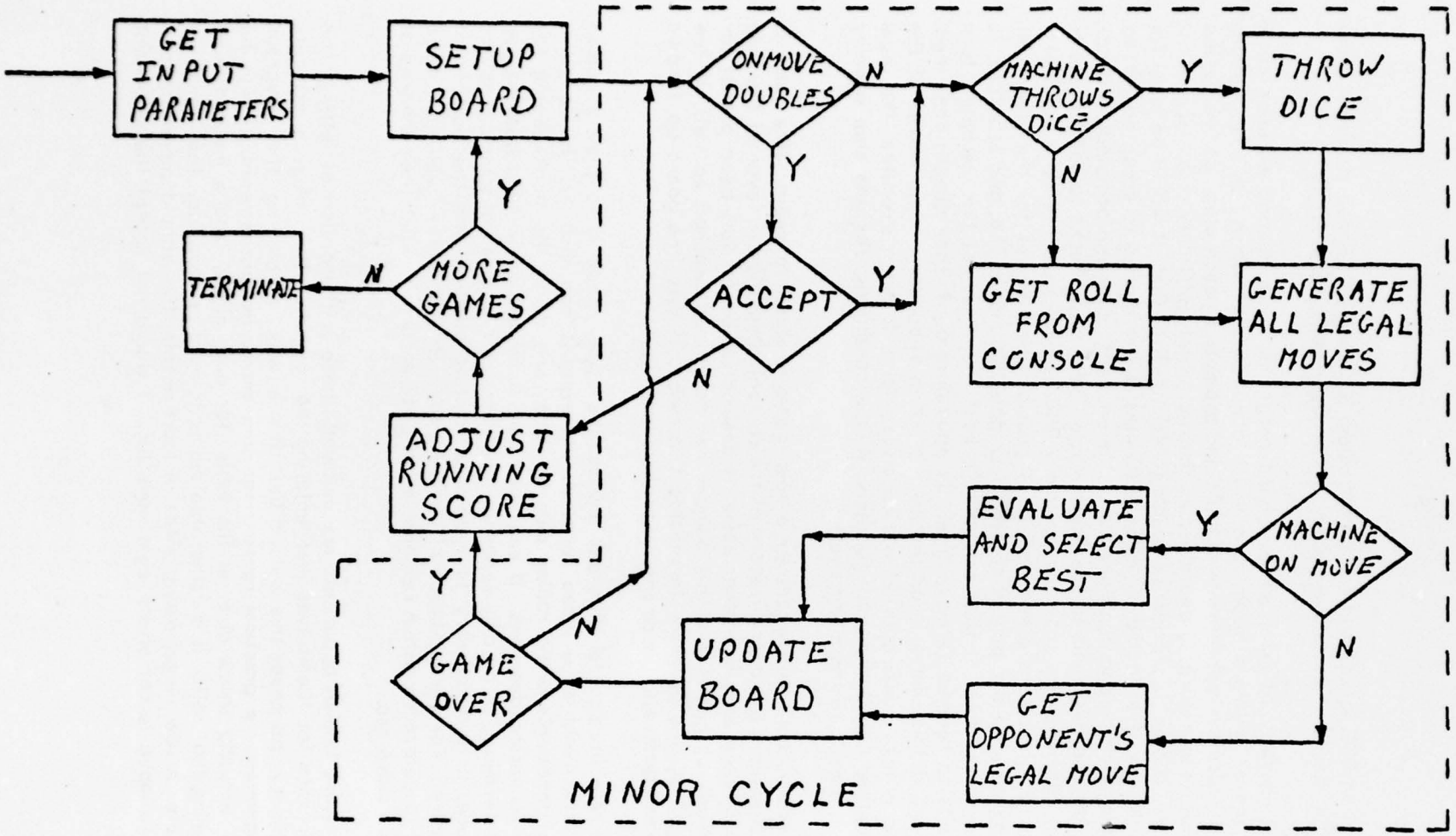
GET INPUT PARAMETERS
SETUP BOARD
ONMOVE DOUBLES
N
MACHINE THROWS DICE
Y
THROW DICE
Y
N
TERMINATE
N
MORE GAMES
Y
ACCEPT
Y
GET ROLL FROM CONSOLE
GENERATE ALL LEGAL MOVES
N
ADJUST RUNNING SCORE
EVALUATE AND SELECT BEST
Y
MACHINE ON MOVE
Y
N
GAME OVER
N
UPDATE BOARD
GET OPPONENT'S LEGAL MOVE
MINOR CYCLE

Figure 1

III. Basic Procedures

A. The Move Generator

In discussing move generation, we will refer to the movement of a single man in accordance with the value of a single die face as a move, and the total set of moves associated with a roll of the dice as a play. At first blush it would seem that the design of a move generator for a backgammon program should be a trivial exercise. However, this turns out to be not quite so easy. The reason is that, while it is no problem to attempt to move each man in turn the number of spaces corresponding to the pips on a die face, there are several special conditions that control the way a roll may be played. The basic situation is this: each man may be moved the number of pips on one of the die faces, it the destination space is not off the board and not blockaded by the opponent (he has two or more men on it). ...g made one move of one man corresponding to one die face, the same ...edure is applied with respect to the other die face. In the case of doubles, the ...omination of one die face is applied four times. By simple recursion it is possible to apply each die face to each man in turn, and thus enumerate the whole set of legal moves.

However, there are problems. Firstly, the above procedure will generate many duplicates which we would hope to avoid or eliminate after generation. Secondly, and more importantly, the procedure outlined will not work for situations in which the moving side has men on the bar (and the only legal moves consist of entering men from the bar), and for those situations where all the men are in the home board (when it is legal to move men to the next point beyond the end of the board and in certain cases even beyond that). Thus on careful examination we determine that there are three distinct states that a board situation can be in:

1) Men on bar,

2) Able to bear off,

3) All other situations.

Further, it is possible for the state to change during a single play, and the move generator must be able to come to grips with such a situation to generate all legal moves, and only legal moves. A final complication introduced by the rules of backgammon requires that if a full play cannot be made according to the roll, the player must play the largest possible part of the roll. This means that if he can play the full roll, he must do so; and if he can play either die face, he must play the larger.

The move generator understand exactly what moves are legal in each of the three states. As moves are generated which could form a legal play, these are put into a tree form. When move generation has been completed, the tree is scanned and those plays that are legal by our final criterion (use maximum part of roll) are marked as legal plays.

In view of these considerations, we implemented the move generator in the following

way. We have a recursive procedure which can call itself a maximum of three times (for doubles) and once for non-doubles. The procedure starts at the location of the man that is furthest away from home for the moving side. It then scans toward home, pausing each time a point is found containing one or more men of the moving side. The die faces are labelled arbitrarily FACE1 and FACE2. The move generator first determines which of the three states the board position is in. It then attempts to apply the current die face to a man on the current point. If it fails in this, it continues the scan until there are no more men or points, whereupon it backtracks. If it succeeds, it updates the board, and if there are still more die faces to apply, it calls itself. In this case, a parameter of this call is the point at which the next scan should commence. For non-doubles this is the same point where the first recursion began. However, for doubles a large saving can be realized by using the current point as argument (since all opportunities to apply the denomination of the die face earlier in the recursion must already have been tried!!). This algorithm will generate all legal moves and only legal moves. For instance, in the case of doubles, if there are more than one man on a point, when the recursive call occurs, the algorithm will attempt to apply the remaining die faces to be played to the remaining men on the point before continuing the scan. Thus, for doubles this algorithm will not generate duplicate plays, while for non-doubles it will. For this reason we use a small modification of the procedure for non-doubles, but this does not concern us here.

B. Special program functions

There are several special functions which the program must be able to perform in order to be able to play an interactive game. These include receiving moves, updating the internal represention and that of any display device being used, sending appropriate prompts and messages to the user, being able to double, accept doubles, resign, and accept resignations. We consider all but doubling and resignation to be quite straight forward, so we will only describe the peculiarities of those four functions here.

1. Doubling

According to the rules of modern backgammon, the game is played with a doubling cube which has both a denomination and position. The cube is initialized at value "1", and located in the center (between the two players). The rules specify that either player may before rolling, if the cube is not on his opponent's side, offer to double the stakes by saying "I double", changing the value of the cube to be twice its current value, and offering it to his opponent by placing it on his side. If the opponent accepts the double, he gains possession of the cube, so that he is the only player who is next entitled to double, and the game continues for twice the previous stake. If he refuses the game is over, and the appropriate game ending actions must be taken.

It is generally considered by backgammon experts that doubling is what separates the men from the boys (meaning that it is relatively easy to make the right move most of the time but hard to know when a position is good enough to double and bad enough to refuse an opponents double). To implement even a mediocre

doubling and accepting algorithm is an extremely difficult task. Because of this, BKG at present only permits doubling (in games in which it is playing) in situations where the two sides have disengaged so that captures are no longer possible. This limitation is being remedied in the version we are currently working on. The problem of constructing such decision algorithms are treated in later sections; here we discuss only the requirement for the program to handle doubling in game.

Since it would be rather boring to ask each human player before his roll (if he were legally entitled to double) "Do you want to double?", we have instead created a doubling flag for each human opponent. If a human feels he may want to double on the next roll, he should enable the doubling flag before making his current move. This will result in BKG prompting him with an asterisk before the dice are rolled for him next time (and until the flag is turned off). When the prompt appears, the player may double or just continue. When the doubling flag is not set, BKG will just roll the dice without asking about doubling. This speeds up the game considerably. Of course, for itself BKG does not need such a flag as it can consider its doubling actions in a few microseconds.

When BKG has been doubled, it decides whether or not to accept using the same procedures it uses in deciding whether to double. Whenever a double has been accepted or rejected, the bookkeeping decisions that follow are rather trivial.

2. Resigning

It is, of course, possible to play every game out until someone has actually won. However, it is not infrequent that a situation is reached in which is is no longer possible for one side to win, no matter how fortuitously the fates may treat him. In such a situation it seems appropriate to resign in the interest of time saving and start the next game. BKG will only attempt to resign or allow resignation after the two sides have disengaged. It then calculates after each move the maximum and minimum number of rolls that it could take for each side to get all its men off. Clearly, if the minimum number of rolls of one side is greater than the maximum number of the other side, the first side should resign. This criterion is currently used both for resigning and accepting resignations. However, we do allow the program to try to resign in situations where it may get gammoned (although only when this is somewhat remote) in the hope that the opponent may accept such a resignation. It is also possible to resign a gammon or backgammon, using as criterion the number of rolls needed to get one man off and to get all men out of the opponent's home board. The question of attempting to resign and accept resignations before disengagement will also be taken up in the version we are currently bringing up. This too will involve judgements as to when a position is too good to accept a resignation (the gammoning chances are too high) and when it is so bad that one should be happy to resign (because one could quite easily lose a gammon).

C. Evaluation

The real knowledge and intelligence of BKG are in the procedures that evaluate moves and positions. We describe these in detail in the next two sections.

IV. The Evaluation Procedures

In this section, we describe issues in measuring certain important facets of a board position. It the next section, we describe how the outputs of these measuring functions are used in the over-all evaluation. Finally, in section VI, we describe limitations of various evaluation approaches, and give what we currently feel is the best approach to this problem. In our discussions, we will refer to the two sides as *Onmove* and *Notonmove* (before a move is made) and *Justmoved* and *Nexttomove* after a move is made.

A. Blot danger calculation

A blot is a man that is by itself on its point. Such a man is in potential danger of being hit (either right away or at some later time), and being sent back to await its opportunity to re-enter and come around the board again. All other things being equal, it is undesirable to leave blots. However, all other things are seldom equal, and for a variety of reasons (including that it cannot be avoided) blots are left at the end of a play. This procedure calculates the danger to the set of all blots of the moving side, and delivers several values for use by the evaluation procedure.

There are many intricacies to appraising the danger that a set of blots is in. When we first brought the program up, BKG merely noted the existence of blots. It considered all blots equally likely to be hit, and merely delivered a value that represented the total pipcount that would be lost if all blots were to be hit. This was a term that the evaluation procedure attempted to minimize.

However, this measure proved very inadequate; it failed most importantly to consider whether any particular blot could be hit by an opposing man. Our next (very small) improvement was to determine that for a blot to be in danger at all, there would have to be an opposing man somewhere in front of it. This produced a small improvement in performance. However, it still failed to get at the degree of endangerment of any blot.

Next we noted that for a blot to be hit it must be a distance of 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 20, or 24 in front of the hitter, and that for each such distance there is a hit probability corresponding to the number of possible combinations. This new fact produced a very large increment in performance, but it still left untouched several important situations. One of these is that even when a hitter is the right distance away from a hittee, it is at times necessary to have available a set of intermediate points where the hitter is to land. Second, is what is called in backgammon jargon duplication: it is impossible to apply a single die face to more than one move. Thus there is a certain safety for a pair of blots if they can be hit by different men (say) 4 pips away. The point is that if a 4 is rolled only one of the blots can be hit, not both. Thirdly, there was the question of more than one blot being hit with a single roll (something that it is usually wise to avoid), and the question of a single blot being hit simultaneously by two men. This is known as pointing on a blot, and is also something that it is wise to avoid.

As a result of these considerations, we finally implemented a very detailed hit

probability procedure which we now describe. We consider any man which is not paired on its point to be a blot, unless it is on the 1 or 2 point in the opponent's home board and the opponent has not yet made more than two points in his home board. We consider any man to be a hitter unless it is part of a pair (and only a pair) in its home board. The reason for the latter is that moving one of such a pair would expose the other to the hittee coming in on the next play, and thus would not really constitute a threat.

We note that there are 40 different ways in which an arbitrary hitter can hit a hittee. To illustrate this consider that there are 3 different ways in which a hitter 3 pips away from a hittee can hit it: by a 3, a 2,1, or using 3 parts of a 1,1. This information is encoded in a table of masks which specify which intermediate points must be free, or that only one of a set of two points need be free.

Our procedure scans the board, starting with the most advanced blot of the moving side, and determines if it is in range of any hitters. If so, then for every hitter it determines all combinations that could be used to hit this blot. It then checks whether the intermediate landing point conditions are satisfied. If so, it enters the location of the hitter in the word corresponding to this combination in the 40-word vector of bit-vectors *Ahit*. At the end of this first pass, the vector *Ahit* contains all the locations of potential hitters, the combination used for the hit, and the location (implicitly) of the hittee.

Next the procedure determines whether the side next-to-move has 0, 1, or more than 1 men on the bar. There is separate section of code for each of these situations. Basically, the procedure examines each word of *Ahit* to see if this combination can be used to hit a man. It starts with the combinations that use only one die face and then goes to the more involved combinations. Each time it finds such a combination it checks to see if the die faces needed for this combination have not been used yet. If so, it marks the faces as used, updates the information on hitting (always assuming that the most advanced blot will be hit if there is a choice), and continues.

If it finds that a combination using only one die face can be used to hit more than one blot, it knows that there exists a combination (the double with that die face) that can be used to hit two blots. If it finds that blots can be hit with more than one single face combination, this means either that a blot can be pointed on or that two blots can be hit with a certain throw. Whenever a blot can be hit by a combination involving two die faces, it checks whether a blot exists on either intermediate point. If so, this combination would hit two blots. In all cases, the values are multiplied by the unused number of ways that this combination can be rolled. At the end of this computation the following values are available:

Piploss = The total number of pips that may be lost due to blots being hit multiplied by the number of rolls that can be used for each hit.

P1 = The number of rolls that hit one or more men.

P2 = The number of ways that more than one blot may be hit, plus the number of ways that a blot may be pointed on.

These data are very adequate to the task at hand. For instance, in situations near the end of the game where one side must break up its safe position and run across "no-mans-land" exposing some blots, BKG does a remarkable job of distributing blots so as to minimize the chance of any being hit. This can be quite difficult even for a master player at times, but this is one of about 3 or 4 area of backgammon play where BKG plays perfectly.

However, there are still some inadequacies to the current approach. The principal one is that the hit probability calculation assumes that it would be desirable to hit any exposed blot. This is usually valid but not always. For instance, the man that would be the hitter might be part of some important blockading or defensive structure and would therefore not want to give that up in order to hit a blot. Also there is the possiblity that if the blot were hit it would in turn leave a blot for the opponent. Under certain circumstances (of which we already indicated one earlier) the disadvantages of leaving a blot in the process of hitting a man outweigh the advantages of making such a hit. We have in mind to improve the calculation to report which men are the hitters and name points on which opponent's blots would be left after hitting a blot. However, the program does not seem to be as limited by the lack of this information as by some other things, so this improvement will be postponed until such a time as it seems necessary. We are also aware of the ad hoc nature of defining what is a blot and what is a hitter, and will at some future time make these definitions more sensitive to the overall board situation. However, as of this writing the hit probability computation is by far the most sophisticated thing in BKG, encompassing some seven pages of code.

B. Blockading Factor

A blockade consists of a set of points "made" by one side, which prevent an opposing man from having access to those points. Clearly, such points can have a great effect on the opponent's movements, and their location is of great importance. The blockading calculation also has an evolutionary history. It became apparent very early in the development of BKG that it is necessary to distinguish between blockades that have one or more men trapped in front of them, and those that do not. The latter consist only of a potential trap for any man that may be hit.

Initially, we counted the maximum number of contiguous blockading points and squared this number to give greater weight to longer blockades. However, this method overlooks the fact that a blockade of six-in-a-row cannot be spanned, the fact that seven-in-a-row is not better than six-in-a-row, the fact that blockading points do not always have to be contiguous to be effective, and the fact that blockading strength is affected by the distance that a potential blockade runner is away from the blockade. To overcome these objections we developed a table of potential blockades.

We note that since there are only 15 men on a side, it is impossible to have more than 7 blockading points. We then computed all combinations of zero to seven blockading points at a distance of 1 to 12 spaces in front of a man. For each configuration we computed the number of rolls that could legally be played by the blockade runner. This is the best measure of the strength of a blockade that we

have found. Since a byte of 12 bits uniquely identifies any blocking configuration, and since the number of rolls that can result in getting past the blockade must be between 0 and 36, we constructed a table with bytes of size 6 bits accessed by a 12 bit code describing the blockading configuration. This method results in quick lookup of the essential data. We keep track of:

Escapes = number of rolls that can be legally played from this point.

Contsq = The point between the location of the furthest back man and own 9 point where the value of *Escapes* is lowest.

Acontain= 36- *Escapes*(*Contsq*).

Contain= 36- the lowest value of *Escapes* between our 24 point and our 9 point.

It has been found that this information satisfies our needs at the moment.

C. The Running Game

BKG will disengage the forces whenever it has the opportunity to do so, if it is even just slightly ahead in the running game (the race to get all men off). Plays are then evaluated in the following way: Any move that brings a man not already in the home board into the homeboard gets credit for 3 heuristic points (HP). From this is subtracted the number of men already on this point in order to give some encouragement to spreading men out. If the man is brought to the 6 point it gets 10 more HP's, and 10 HP's are subtracted for every space to its destination beyond the 5 point. If all men are in the home board at the end of a play, 200 HP's are added.

At the end of any potential play, BKG evaluates the placement of all men not yet in the home board. For each such man, BKG computes the number of board-crossings the man still has to make to get into the home board, and where the man is placed in its current board. In backgammon terminology a board is a sequence of 6 points (of which there are four, the two home boards and the two outer boards). In general, for the same number of total pips remaining, it is most desirable to have the fewest men left, and have them placed as far back as possible in the board they currently occupy. This allows the most efficient utilization of the most throws. Thus BKG will multiply (3 - number of board crossings) by the depth of man in his board. This function seems to produce the desired effect, because it gives the greatest weight to the location of men closest to entering the home board. If the program's side is in danger of gammon, BKG will try to maximize potential rolls of doubles. Thus it will place men so that they can be brought home and one borne off most easily in the event of double 6, 5, 4, 3, or 2.

BKG makes and accepts doubles only during the running game. Until the position is advanced enough so that bearoff tables that give the expected number of rolls to get all men off can be referenced, it uses an algorithm developed by Emmett Keeler [Ke75] of the RAND Corp. This algorithm uses an adjusted pipcount which adds 4/3 pips for every man on the 1 point, 2/3 pips for every man on the 2 point, and 1/3 pip for every man on the 3 point. If one side has borne off fewer men then the

other, the (*Acount*) of that side is increased by 2 pips for every extra man. Finally, it checks the number of gaps (points unoccupied by own men) in the home board of each side and subtracts the difference from the side having the fewest gaps. This is the adjusted pipcount (*Acount*) used in the following computation.

BKG will double if the cube is in the center if its opponent's *Acount* is 107 minus 2 pips greater than its own. It will double when it owns the cube if its opponent's *Acount* is 107 minus 1 pip greater than its own. It will accept doubles when it is not more than 107 + 2 pips behind. This algorithm performs in a basically satisfactory way.

D. Bearing Off

To support decision making during the bearing off phase BKG has extensive tables which give the probability for a given position of one side, of bearing off all men in 1,2,-- 8 rolls and the expected number of rolls (ENR) to bear all men off. These tables were computed by James J. Gillogly of the RAND Corp. for our use. They are used both in selecting a move and in making doubling and accepting decisions.

The tables cover all situations for up to and including 8 men in the home board, and up to and including 25 pips worth of men in the home board. The latter value assures that the tables can handle situations where, for instance, one side has all his men on the 1 and 2 points, but at least 5 on the 1 point.

The use of the tables in move selection is simple. BKG moves to the position with the lowest ENR. There is one exception to this case; that is when it is far behind or far enough ahead to have a chance of winning a gammon. In the former case, it moves to the position which has the greatest probability of bearing all men off in the number of rolls that are expected for the opponent to get off. Here the values of probabilities of getting off in N rolls are very useful. When it is far ahead it moves to the position which gives it the greatest chance of bearing all men off in the number of rolls it expects to have before the opponent gets his first man off. It does this by giving additional weight in the computation to the probability of getting off in 1 -- n rolls, where n is the number of rolls it expects to have before the opponent meets his objective.

For doubling and accepting doubles the situation is more intricate. Whenever, BKG can legally double during this phase, or when it has been doubled, it executes a win probability calculation. If the bearoff position of either side is not in the bearoff probability table, then it uses the Keeler method described above. In that case, the ENR is computed to be the larger of the *Pipcount*/7 and the (number of men left+1)/2. This will usually overestimate the real ENR somewhat. If the ENR's for both sides are in the table then BKG can calculate the exact probability of the side on move winning by itteratively calculating over N the following equations:

W←W+(1.0-W-V)*AWPRB(N)

V←V+(1.0-W-V)*AVPRB(N)

Until W+V=1.0

Here, W is probability of *Onmove* winning, and V is probability of *Notonmove* winning. The AWPRB(N) are computed from the bearoff probability tables by the following equation:

AWPRB(N)=WPRB(N)/(1.0-PRBZ)

Where: PRBZ= SUM i=1 to N-1 of WPRB(i).

If the moving side has a 65% chance of winning, BKG will always double. If the moving side has more than a 79% chance of winning, BKG will always resign if it is the non-moving side. In between there are many situations in which it makes a great deal of difference how many rolls are left in the game, who has the cube (one should be more cautious in doubling when giving up the cube), and where in between 51% and 65% the win percentage is. We use a set of ad hoc tables here, and periodically adjust them when the program appears to be doubling too early or too late.

It should be noted that BKG does a vastly better job of doubling and accepting when using this algorithm than when relying on the Keeler approximation. This can be noticed, for instance, when the position is close but appears to favor one side. When the position changes so that both positions can now be looked up, BKG will frequently double, even though very little appears to have changed from the situation one roll ago for both sides. Upon inspecting the value of the win probability for the moving side, it is not unusual to find it between 60% and 65%; values which were not sufficient to activate Keeler's approximation. For this reason, we are looking at more precise ways of dealing with running game doubling, with the view of being able to obtain an estimate of the win probability of the side on move, rather than the double/don't double decision which Keeler's algorithm estimates. To this end, we are looking at the work of Thorpe [Th75], and have some ideas of our own for simple approximations which can be tested using our simulation facility.

An example of the kind of thing we are talking about is the position in Figure 2.

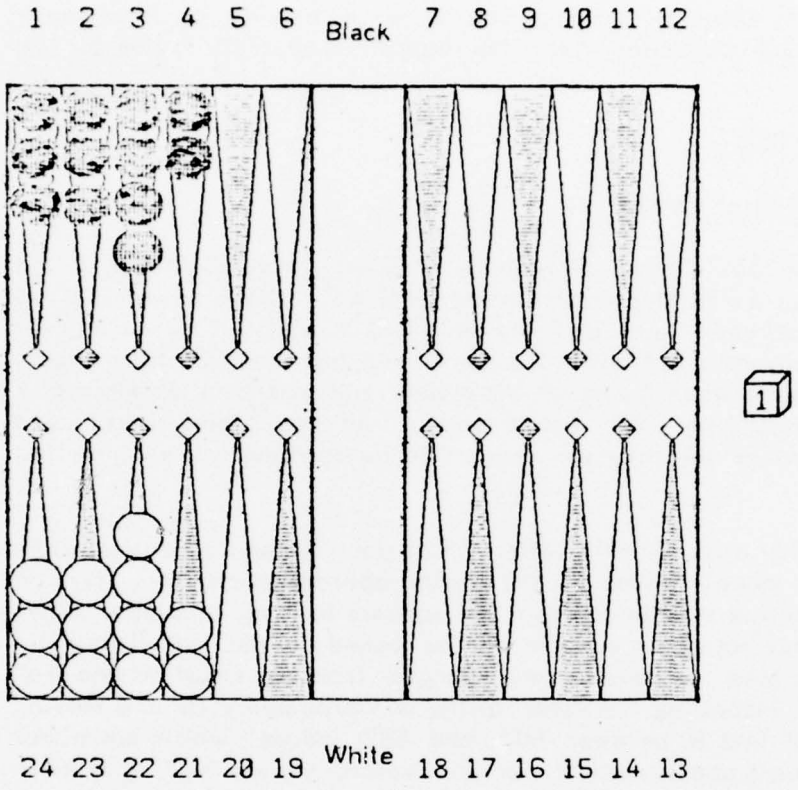

Figure 2

Here the *Acount* for both sides is 36. 36*1.1 - 2 = 38, so that *Onmove* cannot double. Yet *Onmove* has approximately a 65% chance of winning here, as evidenced by simulations.

The power of the bear-off tables is very impressive. Here is another area where the program plays perfectly. To illustrate the type of thing BKG does to amaze its author, we show two examples.

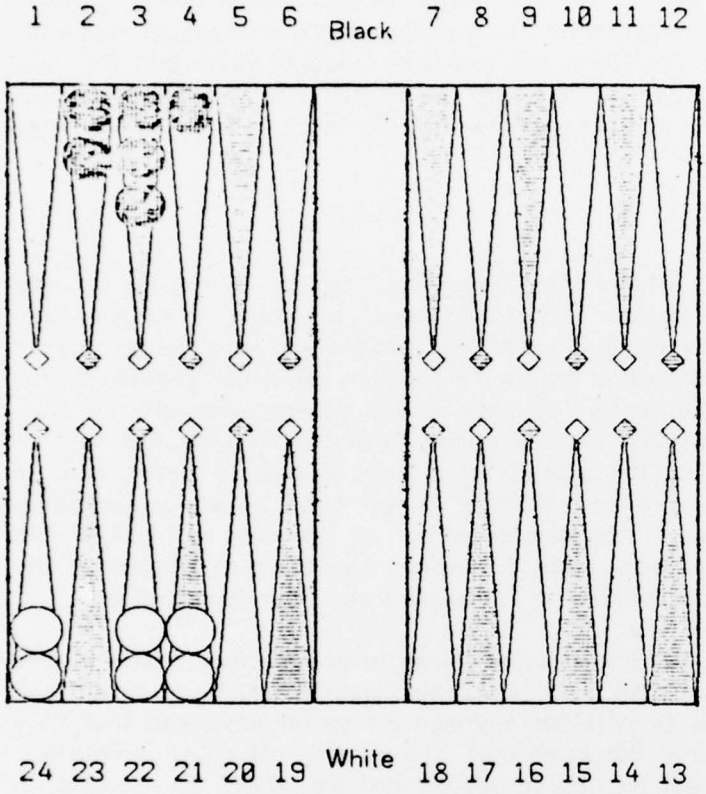

Figure 3

In the bottom part of Figure 3, White is to play a 6,2. The 6 must obviously be played from the 21 point. But what is the correct way to play the 2? Almost every human player would say 21-23. However, this is not correct; 22-24 is better. The bear-off tables report the respective ENR's to be 2.748 and 2.739. Upon examination, it turns out that all sequences of future rolls produce the same results in the two positions except when one of the next two rolls is 1,1. If this occurred on the first roll, it would in both positions allow taking 3 men off. In the preferred postion this would leave men on the 21 and 23 points, which allows 6 additional combinations of getting them both off on the next roll over the other position where the two men would both be on the 22 point. The situation is similar if the 1,1 occurs on the second roll.

The second example in the top part of Figure 3 has similar features. Here Black is to play a 6,1. The 6 must be played from the 4 point; the question is how to play the 1. Again human players would automatically play 2-1, but this is incorrect. 3-2 is better in all future sequences. Again only those sequences involving a 2,2 make a difference. This would allow getting off in 2 rolls unless the next roll is 2,1 in the preferred case, but would do no good in the other. The respective ENR's are 2.794 and 2.777.

It may seem that the advantage gained by making the correct play in these cases is trivial; however, it is certainly worthwhile to improve one's chances in the play when it can be done at no risk. For these two examples it would seem that the rule: "When there are a small odd number of men on the board, play to maximize use of doubles" would seem to be the correct way for humans to capture the knowledge that is contained in the tables.

E. Spacing of men around the board

It is important in backgammon, all other things being equal, not to let groups of men become disconnected from one another. Thus one does not want to advance any men too far, or have any fall too far behind. At the moment we are taking care of this problem by using a 2nd moment of inertia calculation which in general works quite well. We first calculate the center of mass of the moving side after a play. We then calculate *Moment2* -- the 2nd moment of inertia of the men located back of the center of mass. In the evaluation process this term will be a debit, and in general BKG will try to minimize this term all other things being equal. It should be noted that this computation also has the desired effect on the men in front of the center of mass (not too far advanced), since advancing men too far, advances the center of mass, thus increasing the number of stragglers and their contribution.

However, as well as this does, it still leaves some things to be desired. Basically, the real issue is whether any man or men is so far behind that it will be very difficult for it to ever join the main force. Or whether any men are so far advanced that they will not play any meaningful role in the game until the bearing off stage is reached. These two issues are probably best treated separately and we intend to do this in our next revision. It is desirable to maintain stepping stones of safe points for lagging men to join the main force, and it should be relatively simple to devise a measure of the number and nearness to each other of such stepping points as an index of the ease which which lagging men can be brought to safety. Likewise, far advanced men can be debited according to how far advanced they are beyond the last safe man of the opponent. Even with such measures available, we would still plan to retain the 2nd moment term as a useful measure of dispersion of the total force.

F. Other variables

BKG also computes the values of the following variables which are used in the evaluation process.

1. *Closehoard* -- Number of points closed in own home board --This is a simple counting operation, except at present we subtract 1 for every enemy point in the other's homeboard, and 1 for every two own blots in a homeboard. These approximations are heuristic and will probably be replaced by a more exact calculation at some future time.

2. *Homblots* -- The number of blots in own home board.

3. *Builders* -- BKG at the end of a play decides which point it would next like to

make (based on those already made and a table of values). It then counts the number of men that are extra (single or more than 2 on a point) that are within 6 points in front of such a point. These men are builders that are available to make this point on a future roll.

4. *Slotted* -- When BKG has identified the point it next wants to make, if one of its men is already on this point (either as a result of the present play or a previous one) BKG will assign a value to *Slotted* commensurate will the value of the point (from the point table). Of course, if such a slotted man is in danger of being hit by the opponent on the next roll, this value will be traded off against the risk measured by the hit probability calculation.

5. *Histack* -- Accumulates values according to how overloaded this side's points are. It adds 1 point for the third man, 1 for the fourth, 2 more for the fifth, 3 more for the sixth, 4 more for the seventh, etc. This is a quantity to be minimized in the evaluation procedure.

6. In addition to the above variables which require minimal computation, BKG also assign to variables the following quantities:

Mostback -- The point on which the furthest back man is located (could be the bar).

Onbar -- The number of men on the bar.

Expallin,Exploff,Expturns,Maxturns,Minturns,Maxloff,Minloff -- For these variables the number of half-turns required to meet the stated goal are computed after a situation has been reached in which disengagement is near. Minimum and maximum quantities are self-explanatory. The EXP quantities are based on the sum of the ceilings of the distance that must be covered by each man involved divided by 3 1/2.

Mobility = Sum of *Escapes* over all men of a side.

Getbackin(Closeboard,Onbar) -- Heuristic table of values for each situation.

Winfactor = (*Pipcount(Justmoved)* + 4 + *Acontain(Justmoved)**2) / (*Pipcount(Nexttomove)*+*Acontain(Nexttomove)**2).

Caploss = Number of pips lost by *Nexttomove* due to captures.

Edge = *Pipcount(Nexttomove)*-*Pipcount(Justmoved)*-4.

Contfactor = (*Contain(Onmove)*^{3})/300.

7. BKG also can compute the following functions on demand (necessary for understanding next section):

Ownmen(side,point) = Number of men of "side" on "point".

Manhome(point) = Returns binary value; true if "point" is in *Onmove's* homeboard.

Distance(A,B) = Number of pips required to move from "A" to "B".

V. The Evaluation Process

The evaluation process has a long history of experimentation which is still going on. It was apparent, even in the early days of BKG, that a different evaluation procedure had to be used for the running game (forces disengaged) from that used in non-running game situations. This is apparent when one considers that *Acontain*, *Builders*, etc., have no bearing on running game situations. Similarly, there are certain factors which measure the aggressive worth of a position, the defensive potential of position, and the ease with which men could be brought into the home board, given that the forces are still engaged. Clearly, all these factors are not applicable all the time.

To evaluate any position, we must decide which evaluation computations are applicable. The first step in this process is to decide whether it is a running game position or the forces are still engaged. We explained in Section IV how running game evaluation is done. When the forces are still engaged the evaluation proceeds as follows:

The unit of evaluation is the heuristic point (HP). For all positions where the sides are not yet disengaged the following *GENERAL* evaluation is performed:

HP←HP+*Caploss*
+ *Ownmen*(*Offboard*) * (if *P1*=0 then 22 else 3)
+ *Mobility*/5
+ *Getbackin*(*Closeboard*(*Justmoved*),*Onbar*(*Nexttomove*))
- *Piploss*/ *Winfactor*
- (if not *Manhome*(*Mostback*(*Justmoved*)) then *Moment2*)
- 2 * (*Ownmen*(*Justmoved*,1) + *Ownmen*(*Justmoved*,2))
- *Homblots*(*Justmoved*)^2.

The evaluation now becomes more sensitive to specific situations. First we determine whether *Acontain*(*Justmoved*) ≤ 6 in which case *Justmoved* is considered near disengagement and the *NEARDIS* evaluation is performed:

If *Justmoved* appears to be winning; i.e. *Edge*≥0 or *Acontain*(*Nexttomove*) ≥ 16 then:

If *P1*≤3 then the following SAFE evaluation is done:

HP←HP+ (if *Edge*>0 then *Edge*^{2} else *Acontain*(*Nexttomove*)^3/300)
- *Histack*
+ (if *Ownmen*(*Mostback*(*Justmoved*))≤2 then 10)
(This makes it easier to break this point and move the men up)
+ (if *Manhome*(*Mostback*(*Justmoved*)) then
begin if *Ownmen*(*Mostback*(*Justmoved*)) MOD 2 =0 then 10;
if (*Ownmen*(*Mostback*(*Justmoved*))
+ *Ownmen*(*Mostback*(*Justmoved*)+1))≥ 5 then 10;

end).

If this is the same state-class we were in previously then
if *Caploss*=0 or *P1*>(26-*Edge*)/5 then
the following GAP evaluation is done:

HP←HP-(GAP(*Justmoved*)+*P1*)
*(*Contfactor*+*Getbackin*(*Closeboard*(*Nexttomove*),1)/4)*6
/*Distance*(Offboard,*Mostback*(*Justmoved*).

Else if this is a new state class then:

HP←HP+*Acontain*(*Nexttomove*)*2-(IF *Edge*>0 then *Edge**3).

This factor is a measure of how desirable it is to be getting near disengagement, and thus encourages or discourages getting into this state.

Else if *Justmoved* is losing then:

HP←HP+*Acontain*(*Nexttomove*)^3/300.
This factor encourages maintaining the best possible actual containment because disengagement is not in the interest of *Justmoved*.

If *Distance*(*Mostback*(*Justmoved*),*Mostback*(*Nexttomove*))≤2
or GAP(*Justmoved*)≤1 then:
HP←HP+(*Expallin*(*Onmove*)-*Expallin*(*Justmoved*)/10.

This factor encourages *Justmoved* to bring his men efficiently near or into the homeboard, while being applicable only when *Justmoved*'s position has no holes in it. For this section the important ideas are: If we are ahead and were near disengaged then try to bring men up with minimum danger.

If *Justmoved* is not near disengagement then the following *NOTNEARDIS* computation is performed:

HP←HP+*Slotted*(*Justmoved*)
+ *Builders*(*Justmoved*)
- *Histack*(*Justmoved*)
+ *Acontain*(*Nexttomove*)^3/300
+ *Contain*(*Nexttomove*)^2/50
+ (If *Onbar*(*Justmoved*)=0 then
Escapes(*Justmoved*,*Mostback*(*Justmoved*))*2)
- *Escapes*(*Nexttomove*,*Mostback*(*Nexttomove*))*2.

If *Justmoved* is 10% behind in the *Pipcount* then the *DEFENSE* evaluation function is computed. First we check to see if there is any danger of gammon or backgammon. In that case:

if *Contain*(*Nexttomove*)≥15 or *Closeboard*(*Justmoved*)≥4 then

we consider if we wish to stay around in the hope of hitting a man. This is because we have some defensive potential in case we do hit the man. In that case the following defensive potential is added in:

HP←HP + N^3*$Distance(Mostback(Justmoved),Mostback(Nexttomove))$*2-PRB(1)*100,

where N is the number of men that must cross, and PRB(1) is the probability of their crossing in 1 roll. The latter quantity is found by using the bearoff tables with the input shifted to make it appear that the edge of the board coincides with the location of $Mostback(Justmoved)$.

If $Acontain(Nexttomove)$≥16 then:

HP←HP+150+$Acontain(Nexttomove)$*3.

This encourages keeping the opponent blocked in if at all possible.

Unless $Acontain(Nexttomove)$=36 ($Justmoved$ has men trapped in front of a prime), the following *BLOT DANGER* calculation is done:

HP←HP-($P1$*($Getbackin(Closeboard(Nexttomove),1)+Contfactor$))
/$Winfactor$

-($P2$*($Getbackin(Closeboard(Nexttomove),2)+Contfactor$))
/$Winfactor$.

If the current play involves changing the location of Mostback then if the new location is further up than $Contsq(Onmove)$ then:

HP←HP+($Acontain(Onmove)^2-Acontain(Justmoved)^2$)/50;

It is noteworthy that these evaluation functions produce almost all the recommended ways of playing the opening rolls, thus obviating the need for "opening book knowledge", and also confirming the validity of the evaluation process.

VI. State-Classes and their Utilization

The above evaluation functions informally partition the state-space into a considerable number of classes. This partitioning is defined purely by the recognition predicates which invoke some evaluation functions and ignore others. The reader will have noted that there are some terms in these functions which are invoked only if this is or is not a new state. This type of recognition produces in effect a different state-class for two identical positions, given that one is reached from a member of the same state-class and the other not. This action is but one unfortunate side-effect of our current method of doing business. Another is that edge-effects exist. An edge-effect is caused by the fact that there are sharp boundaries between state-classes. Thus the program may stay in one state-class because transistion to the next normal state-class in the progression toward winning cannot be done favorably -- at least not as defined by the evaluation function for the new

state-class. This may result in loitering in the current state-class until there is no choice but to make the transition, possibly under much worse circumstances than would have been possible earlier. It is conceivable that if no state-classes existed this type of transistion could be done more smoothly.

However, it is almost self-evident that one evaluation function cannot serve to order all positions. To do this, incredible complexities would have to be introduced. Consider the *Slotted* term. Before the opponent has build a strong defensive position, it is worthwhile to expose a man in a slotted context in order to improve one's own position. As the opponent's position gets better, this becomes less and less worthwhile. However, in certain desperate situations, it may again be desirable to take such risks. Finally, in the running game the term has no meaning at all. Clearly, the coefficient of this term must be sensitive to a great deal of context, which in effect makes the coefficient non-linear and the resulting evaluation function non-linear. Rather, than evaluate one gigantic non-linear evaluation function, it seems wiser to evaluate each position in its proper context.

Thus the whole issue of state-classes and their associated evaluation functions is born. The issues associated with state-classes are these: We assume that it is possible to partition all game positions into mutually exclusive state-classes. This is not difficult and can be accomplished by merely having recognizers for a set of state-classes, invoking these recognizers in a canonical order, and putting all not-recognized positions into a grab-bag class. We further assume that within a state-class, a linear polynomial function exists which can order the members of this class according to goodness. It is apparent that this is true in the limit, when there is a state-class for each position; however, the degree to which this is possible when there are a large number of members in a given class is not clear. In practice it is possible to get very good (if not perfect) orderings, and to split a state-class when the ordering procedure becomes too difficult.

In general, state-classes can be classified into the following categories: 1) Essentially won, 2) Favorable, 3) About even, 4) Unfavorable, 5) Essentially lost. In all cases it is possible to further subdivide the classes into stable and unstable, where in general stability can be thought of as the variability of the end result. Thus the following ordering of these categories represents their general desirability: 1) Won stable, 2) Won unstable, 3) Favorable stable, 4) Favorable unstable, 5) Even stable, and even unstable, 6) Unfavorable unstable, 7) Unfavorable stable, 8) Lost unstable, 9) Lost stable. Frequently, in order to reach a more favorable class it will be necessary to go first to an unstable class; i.e. take a chance. In general, the side that is closest to winning will want more stable positions and the side that is closest to losing more unstable ones.

However, this is not always the case. The side that has a slightly better position may want to introduce some instability in the hope it will result in an even more favorable position (or even won), while risking losing the advantage or possibly getting slightly the worse of it. This kind of decision is very difficult to program, if all positions are evaluated only on their expectation or game-theoretic value.

Actually, notions such as progress and risk are crutches that are not needed when a

universal measure of goodness such as expectation exists, as it does in backgammon. Thus we should always move to the state with the greatest expectation, and state-classes are not needed at all. However, as we pointed out earlier this is only in a system with perfect knowledge. When there is imperfect knowledge, such crutches allow for much smoother performance.

To give an example of move selection across class boundaries, consider the position of Figure 4.

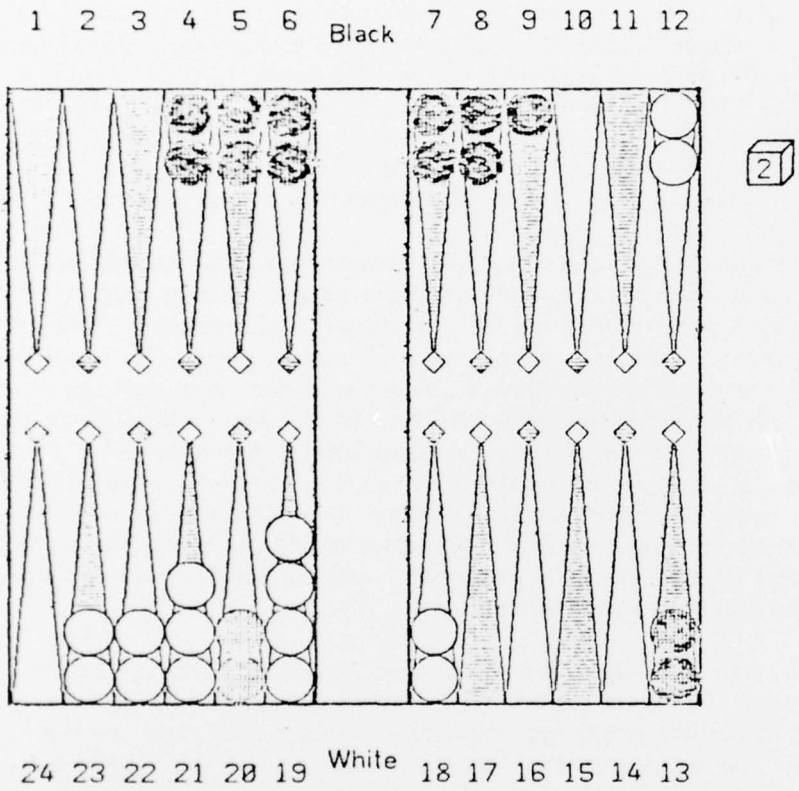

Figure 4

In this case *Onmove* is White. The current state-class (1) is characterized by *Onmove* being ahead in the pipcount by 15 pips or more, having to cross exactly two more enemy points with his Mostback, and having $P1 \leq 3$. We can now imagine an evaluation function for this class which would consider the position of the doubling cube, the exact difference in the pipcount, the distance between Mostback and each of the two points that must be crossed, *Contain*(White), the number of stepping stone points of White between the two points of *Nexttomove* that must be crossed, the value of *Closeboard*(White), and the number of pips that are available to be played as slack before any of these values are decreased. Such a function could conceivably deliver an output that would indicate the expectation of White in the particular member of the state-class. Such a function could be derived either by

analytic means or by actual simulation of positions in the class to find out how each of the above variables affected the expectation of White. Further, such a function could be tuned as experience is accumulated. Let us assume that such a function exists and predicted that in the situation of Figure 4 White should win 66% of the time, win a gammon 2% of the time, lose 30% of the time and lose a gammon 2% of the time, for a net expectation for White of +.36.

Now from this position it is possible to move to four state-classes: the present one (class I), an unstable state-class where White has two points to cross, but has a blot in danger of being hit, i.e. $P1>3$ (class II), an unstable class where White has only one more point left to cross (class IV), and a stable class where White has only one point left to cross (class III). Further, if a blot is hit in state-classes II or IV, we have another state-class (V) in which White has a man on the bar which must enter in front of Black's blocking position. Each of these state-classes will have their own evaluation functions. Thus when deciding how to play a roll, the play yielding the best expectation will be chosen.

White's win probability, W, in a state where he is to move is the SUM i=1 to n $(T_i * W_i)$ where T_i is the probability of transiting to state i on the play by playing it optimally, and W_i is the probability of winning once state i is reached. If it is Black to play, White's win probability can be computed in a like manner.

This method can be used to decide between plays that result in differing state-classes even though one class may be unstable and the other not. Let us illustrate by an example. Assume that W= .92 for positions in state-class III. W for positions in state-class II is PH * W5 + (1-PH) * W_2, and W for state class IV is PH * W_5 + (1-PH) * W_4. Here PH = $P1$/36, W_2 = .85, and W_4 = .92. To get W_5 we must compute the probability of White escaping over the blockade on his next roll, as otherwise he will be doubled and will have to resign. If he does escape, he has about an even chance in the resulting position. These constants should make clear the computation below. It should be noted that when W for a side that is on roll and can double is ≥ .75 (i.e. his expectation ≥ .50) he can double and force his opponent's resignation. Thus such terms should be ignored as their value drops to 0. This is true for instance of the term dealing with the situation where White is hit and contained.

We now use this method to decide how to play a difficult roll, 6-1, in Figure 4. There are basically two plays: run one man from the 12 point resulting in a position of class II, or play both men from the 18 point remaining in class I. For the first play:

W= 20/36 * .85 + 16/36 * 2/36 *.5 = .48.

For the second play, there are 5 rolls which result in transition to state-class III (2-2, 3-3, 4-4, 5-5, 6-6), 2 rolls that result in a class II position (6-2), 8 rolls that result in class IV positions (6-1, 6-3, 6-4, 6-5), and 21 rolls that result in remaining in state-class I. The appropriate computation is:

```
W=  5 ☆ .92                                  (state-class III)
  + 8 (16/36 ☆  2/36 ☆.5 + 28/36 ☆ .85)      (state-class II)
  + 2 (23/36 ☆ 2/36 ☆.5 + 13/36 ☆.75)        (s-c IV, 2 blots)
  + 21 ☆ .55                                 (s-c I, new W = .55)
```

/36 = 20.61 / 36 = .57.

Therefore, it can be seen that it is better to make play two. It should be noted that as the probability of containing a hit man varies with Black's defensive formation, this calculation will also vary accordingly.

The method we have described above can be used for deciding the very important problem of when to move to a state that is in a state-class different from the one we are currently in. However, the whole method assumes that an accurate evaluation function exists for each state-class; i.e. it both orders properly and produces the correct expectation for each member of its class. This is obviously never the case.

Therefore, it is necessary to show how such a system can operate adequately and can be improved in the face of error. For each state-class there are new state-classes that can be reached in one optimal play for each side without a capture being made. We call these classes forward with respect to the original class. Likewise, there are new classes that can be reached from the current class in one optimal play by each side, when there has been at least one capture of a man. We call these classes backward with respect to the original class.

It is possible to start with a class for which we have excellent expectation data, i.e. the class of bearing off positions that can be looked up in our tables (class B). Next, we consider all classes for which class B is forward, and improve the evaluation function for those classes, tuning the coefficents of existing terms and adding new ones as required. This will improve these evaluation functions. We also note all classes that are backward to this class, and put them on a list together with the name of the current class. We can continue this process indefinitely, but painfully until every class has been encountered. Whenever the evaluation function of a class that is on the backward list is improved, we go back and modify all the evaluation functions of the affected classes. We can then continue our process or go back to one of the classes whose function has just been modified and start anew from there. It is clear that this is a converging procedure. It would probably be necessary to eventually automate this proceedure, if for no other reason than that eventually the evaluation functions would become so good that they would do a better job of ordering members of a class than the experimenter would. Such automation except for the introduction of new terms has been previously done by Samuel [Sa59] for checkers. It would appear likely that for a game such as backgammon, it would be possible to get a selection of terms such that no new ones will ever be required. Then it will be merely a matter of tuning old evalution functions, pulling in a new (but known) term every once in a while to see if it can improve prediction.

As data are collected and the evaluation functions improve, two things become possible. It is possible to keep track of how the prediction works out for the program's own play, which can be used as an indicator of which functions need to be tuned next. It is also possible to keep track of individual opponent's results and come to the conclusion that they don't appraise certain state-classes correctly, and use this information in future games.

VII. Testing of BKG

When testing BKG on typical beginners books, it gets the right answer in excess of 70% of the time. A much better appraisal of the program can be obtained by analyzing its successes and failures on more difficult tasks. For this we chose the problems in a very fine intermediate level book [Ho74]. There are 74 doable problems in this book (BKG could not do those which involve doubling decisions before disengagement). We have classified the problems according to the major knowledge required to get the right answer. This is a rather arbitrary way of looking at things, but it is helpful in trying to understand the strengths and lacks in the program. We divided the problems into seven categories:

1) General positional,
2) Running game: bearoff,
3) Engaged: bearoff,
4) Back game (this a special defensive posture),
5) Timing (this involves advantages that presently exist going away because one side or the other must destroy his position),
6) Defensive plays,
7) Advanced defensive plays (including the return play).

We followed the practice in scoring the results of giving BKG part credit for answers that were not perfectly correct but showed it understood the main point of the problem, although the execution was not perfect. We also deducted part credit when it got the correct answer without understanding what the main problem was. Table I below shows the results of the tests.

TABLE I- Tests of BKG on "Better Backgammon"

Position Class	Number	Right	Wrong	Percent Correct
Positional	28	18 3/4	9 1/4	67
Running Bearoff	5	5	0	100
Engaged Bearoff	11	2	9	22
Back Game	8	3 1/4	4 3/4	41
Timing	13	4	9	31
Advanced Defense	6	1/2	5 1/2	6
Defense	3	1	2	33

In evaluating these results, several things should be noted. The subject matter is relatively advanced, and would for the most part come up in only one of 20 or more games. There are usually on the order of three plausible answers to a problem. BKG is good enough in almost every case to know what these are; thus attaining a score of 33% or less could be regarded more or less as the result of chance. We can see that BKG is extremely good in running game play. Also it has a good understanding of the relative positional advantages. However, its performancce in other intermediate level aspects of the game is at best mediocre. It has heuristics to help it do bearing off while still engaged, but these are for run-of-the-

mill situations, not for the more sophisticated ones in the test set. It has no specific understanding of the back game. Since the objectives in the back game are rather different than anything else in backgammon, it will be necessary to implement a specific set of state-classes which recognize back-game potential and how to maintain and destroy it. The problem of timing is one that will be resolved soon. Essentially, this requires having a measure of how many men are presently bound to essential roles in the current evaluation, and how many pips are available to be played by the remaining men before the important men will have to be moved. BKG's only knowledge of defense is that described earlier. It does not understand the concept of coverage, i.e. controlling points on which an opponent's blot may land in the next roll or two. It does not understand that at times it may be beneficial to expose a blot in dire circumstances or to make the "return" play. Thus this series of tests has pinpointed some specific knowledge that BKG lacks and that is not subsumed in its present knowledge base.

VIII. Use of Simulation Facility

It is possible to give BKG a position and ask it to play both sides repeatedly any number of times. In doing these simulations it will play as it ordinarily does; double when allowed and appropriate. There are certain types of positions that BKG plays almost perfectly so that doing such simulations generates useful information for evaluating the position. Included in this class are all running game positions and those where one side is bearing off with one or fewer points to cross. Thus it is, for instance, possible to determine the expectation of the bearing off side when there are two opponent's men on the bar, and his homeboard is closed. Such information is very useful for understanding where certain break even points are. We have, in fact, considered publishing tables of such data for general consumption.

IX. A game

BKG does not play a brilliant game, as one could for instance say of a chess program when it makes a sacrifice. It plays well and consistently, using its knowledge of probability and positional facets of the game. It makes errors when these facets are misinterpreted due to its evaluation functions or when the appropriate knowledge is not present. The game below is representative of the good games it plays.

White: Berliner		Black:	BKG
Roll	Play	Roll	Play
5,4	1-5,12-17	5,6	24-18-13
2,5	5-7-12	3,6	24-18-15
1,3	12-15X,1-2	3,4	25-21,13-10
4,3	12-15,19-23	1,5	6-5,10-5
4,6	15-21X,17-21	6,3	25-22-16
3,1	12-15-16X	2,5	25-23X-18
6,1	0-1,12-18	1,4	25-24-20
(Doubtful play as this blot is too exposed, but Black is lucky).			
4,3	17-20X,16-20	3,4	25-22-18X
2,4	0-4,2-4		
(Now would be a good time for Black to double and White should refuse. However, BKG does not double until disengaged).			
		2,6	13-11,18-12X
(BKG does not know about back-game possibilities and thus is happy to hit all the blots it safely can. Here the back-game is unsound anyway).			
5,1	0-1,17-22	1,4	8-7,11-7
3,3	19-22,17-20-23,15-18	3,4	12-9,13-9
3,4	19-23,18-21	6,5	13-7,13-8
2,6	4-10-12	4,5	8-4X,9-4
1,4	0-1,12-16	5,4	13-8,9-5
(Making the 3 point improves Black's gammon chances, but BKG does not want to leave any shots).			
1,2	1-3,16-17	3,6	5-2,8-2
1,3	17-20,21-22	6,1	8-2,8-7
6,2	3-9,22-24	6,3	7-4
2,4	20-24,9-11	6,2	7-5
6,5	11-17-22	5,3	7-2,7-4
1,2	22-24,23-24	3,1	4-3,6-3
5,3	19-24,19-22	1,4	4-0,6-5
6,1	1-7,22-23	3,6	6-0,6-3
6,3	7-13-16	4,6	5-0,5-1X
2,5	No legal move (no luck)	2,2	5-3,5-3-1,2-0 (safe now!)
2,5	0-5-7	4,2	2-0,4-0
5,6	7-12-18	6,4	4-0,3-0
4,5	16-21,18-22	Double	(no chance of gammon remains)
Resign			

REFERENCES

[Ho74], Holland, Tim, "Better Backgammon", *Reiss Games, Inc.*, New York, 1974.

[Ja73], Jacoby, Oswald & Crawford,John R., "The Backgammon Book", *Bantam Books*, 1973.

[Ke75], Keeler, Emmett & Gillogly, James, "Playing the Runnning Game in Backgammon", *Popular Bridge*, June 1975, Vol. 9 Number 3, pp. 34-38.

[Le76], Levner, David, "Is Brute Force Backgammon Possible?", *SIGART Newsletter 58*, June 1976, p. 20.

[Sa59], Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers", *IBM Journal of Research and Development*, Vol. 3, No. 3, 1959, pp. 210-229.

[Th75], Thorpe, Edward O., "Backgammon: Part I, The Optimal Strategy for the Pure Running Game", Presented at the *Second Annual Conference on Gambling*, Nevada, June 1975.

[Wu71], Wulf, W. A., Russell, D. B., and Habermann, A. N., "Bliss: A language for Systems Programming", *Communications of the Association for Computing Machinery*, Vol. 14, No. 12, December 1971.