ELSEVIER

Artificial Intelligence 134 (2002) 241–275

Artificial Intelligence

www.elsevier.com/locate/artint

World-championship-caliber Scrabble [☆]

Brian Sheppard

Sheppard Company, Inc., 296 Old Malboro Road, Concord, MA 01742, USA

Abstract

Computer Scrabble programs have achieved a level of performance that exceeds that of the strongest human players. MAVEN was the first program to demonstrate this against human opposition. Scrabble is a game of imperfect information with a large branching factor. The techniques successfully applied in two-player games such as chess do not work here. MAVEN combines a selective move generator, simulations of likely game scenarios, and the B* algorithm to produce a world-championship-caliber Scrabble-playing program. © 2001 Published by Elsevier Science B.V.

Keywords: Scrabble; Heuristic search; B*; Simulations; Probability-weighted search; Dictionary representations

1. Introduction

Scrabble is a board game in which players build words with small tiles containing letters of varying point values. It can be played in family environments with two to four players. In this article we only consider its two-player variant. Scrabble is a game of imperfect information; its rules are described in Section 2.

The game attracts a wide variety of human players, since it challenges the ability to form words from a given set of letters. Playing under strict tournament conditions requires inventiveness and creativity. Around 1980 the question arose as to whether computer programs were able to play competitive Scrabble. In 1982 the first attempts were published [13,17].

My first program was written in PL/1 on an IBM mainframe in the summer of 1983. About 25,000 words out of the Official Scrabble Player's Dictionary (OSPD) [12] were typed in. This was sufficient to observe that the program had significantly surpassed what was described in the literature. In addition, two more things could be concluded from the

☆ SCRABBLE® is a registered trademark. All intellectual property rights in and to the game are owned in the USA by Hasbro Inc., in Canada by Hasbro Canada Corporation, and throughout the rest of the world by J.W. Spear & Sons Limited of Maidenhead, Berkshire, England, a subsidiary of Mattel Inc.

E-mail address: SheppardCo@aol.com (B. Sheppard).

program's performances. First, the program's vocabulary is the dominant determinant of skill. Second, the tiles left on the rack are the greatest contributors to the scores of future turns, i.e., rack management is important.

My second program ran on a VAX 11/780 and was written in C in 1986. It contained the full OSPD dictionary and a rack evaluation function that was tuned by self-play. At that time the program was, in my opinion, already the world's best player. It was named MAVEN, a word of Yiddish origin that is defined in the OSPD as "expert". Later on, someone who actually knows Yiddish told me that the connotation is more "know-it-all" than "expert". So much the better!

In the fall of 1986 MAVEN had a complete dictionary, a fast move generation, and a good rack evaluator. The program entered in its first Scrabble tournament in December 1986. Since MAVEN's rudimentary endgame capability was not actually working, it was disabled. MAVEN scored 8–2 over a grandmaster field of human players, and finished second after tiebreak. It was the beginning of an advance that ended a decade later by surpassing the world's top players by quite a margin. Empirical evidence of this claim will be given in Section 10. Here we outline the increase of MAVEN's playing strength by its determining factors.

At first MAVEN's advantage was attributable to accurate move generation (in principle, it considers all possible moves) and fine-tuned positional evaluation. But humans learned from MAVEN's play and then the best humans could challenge MAVEN on almost equal terms. Then MAVEN's endgame player presented humans with a new challenge. Again humans adjusted their play by allocating extra time to endgame analysis during tournament games, and by specifically practicing endgames. However, no human actually caught up with MAVEN in endgame skill.

MAVEN was one of the earliest programs to employ "simulated games" for the purpose of positional analysis, though as far as this author knows the concept was introduced first in backgammon programs [1,16]. The term "rollout" seems to date from the late 1980s, whereas MAVEN first employed the technique in 1987.

MAVEN was also one of the first programs to employ simulated games for the purpose of making moves in competitive games. This is now also done in bridge [8] and poker [5]. Surprisingly enough, only since 2000 have backgammon programs been able to make moves at over-the-board speeds using simulated games. Of course, the ability to use simulated games depends on domain-specific speed and accuracy tradeoffs. The idea has been floating around since 1990.

Again, humans learned the lessons of simulations from MAVEN. In the years between 1990 (when simulation became available as an analytical tool) and 1996 (when simulations were first used in competitive play) humans improved their positional skills by studying simulation results. However, it may be doubted that the improvement ever compensated for MAVEN's advantages in move generation and endgame play.

Human improvement notwithstanding, the author believes that MAVEN has maintained at least a slight superiority over human experts since its debut in 1986. This may be the earliest time at which a computer program achieved world-class status over human masters in a non-trivial game of skill. With the advent of competitive play using simulated games, MAVEN is now out of reach of human experts. No human will ever challenge MAVEN on equal terms. In brief: a program that plays almost perfectly is technically feasible at this point, and MAVEN is close.

The course of the paper is as follows. Section 2 describes the rules of the game and the basic ideas about scoring points. Section 3 describes the basic components and how they interact. Section 4 discusses move generation. Simulations are described in Section 5. The advantage of perfect knowledge in the endgame is discussed in Section 6. Section 7 deals with the pre-endgame phase. Section 8 presents the overall design decisions. Since there are all kinds of myths and rumors about how to play Scrabble well, we list three of them together with their refutations in Section 9. Experimental evidence on playing strength is given in Section 10. Section 11 contains suggestions to make MAVEN even stronger. Finally, Section 12 contains conclusions.

2. Scrabble

Scrabble is popular worldwide, with numerous national and international Scrabble associations and a biennial world championship. It is a language game for two, three or four players. As stated above we only consider the game for two players.

2.1. Rules and terminology

The goal in Scrabble is to create words, using the same constraints as in crossword puzzles. An example board is given in Fig. 1. In this article we refrain from providing a full description of the range of letter values, rules of how to form a word, and the exact marking system on the board. We assume that any reader is familiar with Scrabble and that a superficial knowledge of Scrabble rules is sufficient to understand the essence of this article which focuses on the development of artificial intelligence techniques. For every word created by your tiles, a score is given dependent on the letters used and the marks on the squares where the tiles are placed. Your score for a game is the sum of your scores for all turns. You win the game if your score at the end is greater than the opponent's. Ties are possible although they rarely occur.

In the beginning all tiles (including two "blank" tiles which can be used as any letter) are placed in a bag in which the letters are invisible. Each player starts off with 7 tiles. The players move in turn. A move places some tiles on the board forming one or more words with the tiles already on the board. For every letter placed on the board a replacement is drawn from the bag. The game ends when one player is out of tiles and no tiles are left to draw, or both players pass.

The game is played on a 15×15 Scrabble board. The squares on a Scrabble board are referred to by using a coordinate system. The rows of the board are numbered 1 through 15, and the columns are numbered A through O. A number and a letter specify a square on the board. For example, 1A is the upper left-hand corner. To specify a move you need to specify a word, a starting square and a direction. The convention is that giving the row of the square first specifies a horizontal play (e.g., WORD (1A) is a horizontal move) and giving the column of the square first is a vertical play (e.g., WORD (A1) is vertical). See Fig. 1, where MAVEN played MOUTHPART as its last move at 1A (note the use of the

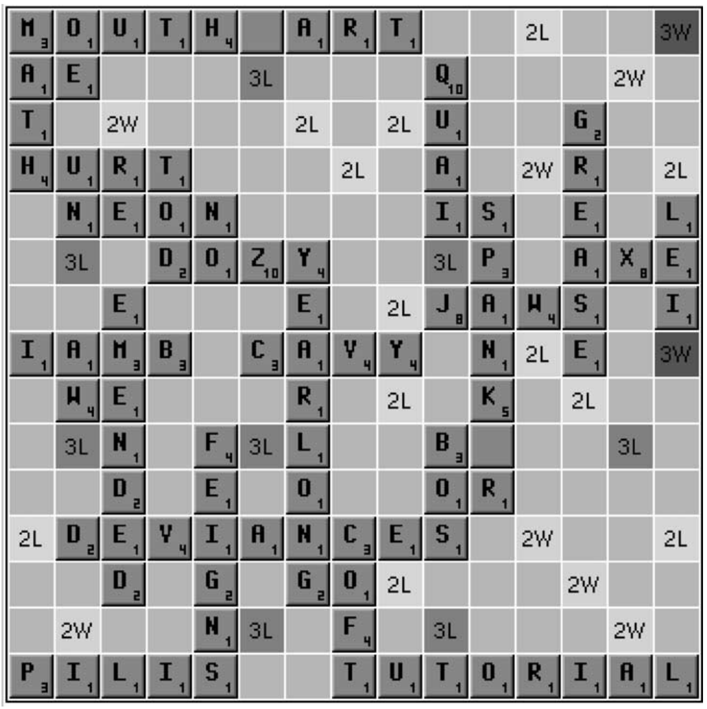

Fig. 1. MAVEN versus Adam Logan (exhibition match, AAAI-98). MAVEN's last move is MOUTHPART at 1A.

blank for the P). In a Scrabble analysis, a move is usually annotated with a triplet that contains the move's location, score, and the resulting rack leave (see below); sometimes the third coordinate is suppressed.

There are several Scrabble terms used in this paper. The set of tiles that a player holds is called the *rack*. The set of tiles left after a player has moved is called the *rack leave*. A *bingo* is a move where a player uses all seven of the rack tiles in a single move. For achieving this feat a player earns a 50-point bonus. For example, MAVEN's move in Fig. 1 (i.e., playing "UTH?ART", where the '?' represents the blank) is a bingo (and also happens to end the game, see Appendix A). A *hot spot* is a place on the board where high-scoring plays are likely.

In general, on a given turn a player wants to score as many points as possible. This means that exploiting squares with high bonus points is preferable, as is using letters with a high value. Of course, one should have adequate knowledge of the playable words and a good estimate of the potential use of the letters left.

The human cognitive problem of finding high-scoring plays is very different from the programming problem of generating them. For a human the process involves anagramming the contents of the rack and looking for hot spots. There are a variety of methods that help experts carry out this process.

2.2. Human Scrabble strategy

Human Scrabble strategy can roughly be defined according to the following four phases:
(1) search for a bingo,
(2) search for hot spots,
(3) try to improve upon the result found, and
(4) consider the rack leave.
Below this strategy is illustrated.

First, you should always try to find a bingo. You are looking for either a 7-letter word using the entire rack or an 8-letter word that uses one tile from the board. In passing we note that a 7-letter word should be legally connected to the letters on the board. Experts group tiles into "prefixes" and "suffixes" and then mentally shuffle the remaining tiles. For example, assume the rack is DFGIINN. An expert will see the ING group, and physically rearrange the rack into DFIN-ING. It then takes but a second to anagram FINDING. The top 200 prefixes and suffixes cover 80 percent of all 7-letter and 8-letter words. So this method is pretty thorough if you have sufficient expert knowledge to carry it out.

Second, you should look for hot spots. The most important aspect of a hot spot is that it involves premium squares. There are different types of premium square, e.g., doubling and tripling the value of a letter, as well as doubling or tripling the value of a word. You can scan the board looking for ways to hit specific premium squares.

Third, once you found a good move, you can start to improve upon that move by asking more specific questions. For instance, given a 30-point move, you can ask how you can possibly score more than that. Some hot spots are not sufficiently productive to meet the standard (i.e., of 30 points), so you can save time by skipping them.

Finally, you should consider the impact of your rack leave on future turns. The highest-scoring move is often not best because it leaves bad tiles in your rack. Much better is to give up a few points in order to play away bad tiles. It pays to play away the worst tiles (e.g., duplicated tiles or the Q) even at the expense of points.

3. Overview of MAVEN

In this section, a brief description of each of the major components of MAVEN is given. A move once played changes three factors: the score, the rack, and the board. Hence, the move generator should find all legal plays and then assign the appropriate number of points they yield, the rack evaluation should estimate the future scoring, and the board evaluation should give an evaluation of the new board position. In subsequent Sections 4, 5, 6, and 7, the move generator, the simulator, the endgame, and the pre-endgame are discussed extensively.

3.1. Vocabulary

Knowing all the words is vitally important. The Official Scrabble Player's Dictionary (OSPD) is the definitive source for North American words. Accurate word lists (consisting of some 100,000 words) are nowadays available on the Internet.[1] In tournaments with participants from all over the world other vocabularies have been used too. For instance, in the third Computer Olympiad in Maastricht 1991, two dictionaries were used to decide whether a word was acceptable: the British “Official Scrabble Words” and the OSPD. In that time the dictionaries mentioned contained words of at most eight characters and therefore in most Scrabble tournaments longer words were accepted if they appeared in Webster’s Ninth Collegiate dictionary. One program had interpreted the rule of a word appearing in one of the two books literally and challenged ICONICITY and NONCAUSAL [18]. In Maastricht, it was decided that both words were acceptable and that the challenge was not penalized (an incorrect challenge normally loses a turn).

The following statistics underline the importance of the vocabulary. With a deficient vocabulary MAVEN scored around 20 points per turn. With every word of length 5 or less and all the J, Q, X, Z words (around 25,000 words) the program had an average score of 23 to 24 points. With the entire OSPD the program achieved an average of 30 points per move.

Watching Scrabble Grandmaster Joe Edley find the word METHADONE through separated tiles inspired us to add the 9-letter and longer words to MAVEN. Subsequently the Merriam-Webster Collegiate Dictionary was scanned for such words. The result was cross-checked against the work of another person for the 9-letter words, but the 10-letter and longer words remained rife with errors until 1996, when the NSA (National Scrabble Association) developed an official, computerized list.

We discussed the topic of the vocabulary quite extensively because massive knowledge bases are a very important component of AI systems. There is a tendency to dismiss problems of scale as mere matters of technique, but this author disagrees with that viewpoint. Intelligent behavior inevitably confronts issues of scale, so it is best to have an arsenal of methods for dealing with large knowledge bases.

3.2. Move generation

In 1983 a straightforward move generator was developed using constraints to find solutions. First the set of all squares that satisfied certain constraints was pre-computed. For speed, the constraints were represented as bit vectors. Then, to compute the set of all squares that satisfy a series of constraints the bit vectors’ intersection was computed. The algorithm was called the “bit-parallel” algorithm.

In 1988 Appel and Jacobson’s algorithm [3] came to my attention. It represents the dictionary as a dawg (directed acyclic word graph). This dictionary representation enables clever algorithms. For example, you can search the state machine constrained by the tiles on the rack, tiles on the board, and constraints from overlapping words. It is easy to make a move generator on this basis. Moreover, Appel and Jacobson showed how to use the state machine to generate moves with minimal effort. A comparison of their algorithm against my own showed that both were equally fast, but Appel and Jacobson’s was simpler thanks

[1] In the early 1980s, electronic versions of the dictionary were not available. The obstacle was resolved by manually making the entire OSPD available to the program. This meant creating a database of 95,000 words by typing them in one by one. It took roughly 200 hours of typing, 200 hours of verification, and years of part-time work to keep it up to date.

to the state machine representation. Therefore my algorithm was ditched in favor of theirs. The Appel and Jacobson's algorithm was used in all subsequent versions of the program.

MAVEN's move generator generates and scores each move. Then each move is passed off to a caller-supplied handler function for further processing. This "handler function" architecture facilitates reuse of the move-generator module in a variety of search engines. All search engines in MAVEN use the same move generator with different handlers. MAVEN has a variety of heuristics for assessing moves. Each heuristic routine identifies up to 10 candidate moves. Merging the candidate move lists results in 20–30 moves to be considered by the search. The rest are ignored. A description of the data structure used is given in Section 4.

3.3. The rack evaluation

It is clear that there is a need to evaluate rack leaves. A program having no bias against bad tiles will often hold racks like IIIUVVW, because it plays away its good tiles until only bad tiles remain. After all, there are *more* moves that use good tiles, so without any effort specifically directed at playing bad tiles it is assured that the program will play more good tiles.

Holding bad tiles must have a penalty sufficient to cause MAVEN to trade off the current score against any future scoring. The balance between present and future scoring is at an optimal level when total scoring is maximized over the whole game. Therefore we want to keep good tiles, but we do not want to give up too many points as a consequence.

After a move the rack contains from zero to seven tiles. Since the value of the rack might be a highly non-linear function of the tiles, the evaluation problem was resolved by an architecture that could handle the rack evaluation adequately. The rack evaluator has a list of *combinations* of tiles, each combination having an evaluation parameter. The evaluator is linear, but combinations allow for the modeling of complicated factors (cf. [6]). The combinations may include any group that seems to have an effect on the game, and the list may be arbitrarily large.

Our initial rack evaluator included a value for every tile (Blank, A, B, C, ...) and every duplicated tile (Blank-Blank, AA, BB, ...), and every triplicated tile (AAA, DDD, EEE, ...). For quadruples and higher the value used for triples was taken. There was also a value for QU, since these are individually bad tiles, but not bad when held together.

3.3.1. Learning parameters

What remained to be settled was an adequate estimation of all parameters. Various alternatives were considered, like simulated annealing, which was all the rage in those days (i.e., 1986). Finally, we arrived at the opportunity to "learn" parameters through a feedback loop. The idea was to play games, then value each combination by the impact that it had on future scores.

For instance, assume that the rack leave of a particular move is EQU. This rack contains the following combinations: E, Q, QU, and U. So this turn is an observation for four parameters. The future score is the difference in score between the side to move and the opponent over the rest of the game. After collecting many such observations you have an over-constrained system of linear equations. You can solve that system in a number of ways. The solution we used at the time exploited domain-specific properties. Today, we would choose least-squares, following the recommendation by Michael Buro [6].

Our approach was not an on-line algorithm (i.e., MAVEN did not learn as it played). It started from no initial knowledge (i.e., all parameters were 0) and gathered statistics about the parameters. A day's worth of self-play games was used as training data for parameter learning using the observed parameter values. After a third iteration it was found that parameter values converged.

Since that time temporal differences (TD) learning has emerged as a powerful learning algorithm. Sutton's pioneering research predated our work [14]. Having no access to the Internet in those days, we did not find out about TD until 1996. Meanwhile we worked out a technique that is equivalent to TD(0) around 1990, and tried to learn Scrabble parameters using it. TD(0) did learn something, but it was not the right answer. It learned values that were compressed towards 0 compared to the parameters we assumed to be correct. Just to make sure, MAVEN's original parameters were pitted against the parameters learned using TD(0), and the original parameters won. Why did TD(0) fail? The only explanation we have is our exploration bias: an on-line learner learns to avoid terrible racks like III before their values have converged to their true (and truly awful) values.

A comparison between backgammon and Scrabble is in order, since TD(0) works like a charm in backgammon [15], whereas it did not work in Scrabble (which has a *much* simpler parameter space). Sutton's theoretical arguments show that TD should work if given occasional opportunities to "explore" paths that seem sub-optimal [14]. In Scrabble you are almost never *forced* to keep III, so learning can stop even the start of an awful convergence. In backgammon, by contrast, the dice often force you to accept all sorts of positional defects, even if you play ideally. The lesson is that in applying TD you must ensure adequate exploration.

As an important aside we mention that we are convinced that MAVEN's parameters are very close to optimal. In 1994 Steven Gordon published a hill-climbing search of the parameter space and confirmed that MAVEN's parameters are optimal according to his method [10]. Steven's method was to modify parameters by steps of 0.5 using a competitive co-evolution strategy. His parameters agree (on the whole) with MAVEN's to within 0.5. Other Scrabble developers (e.g., Jim Homan on CrossWise [11] and James Cherry on ACBot [7]) have reported similar results. For a full understanding of the method and the reported results we refer to the references given above.

3.3.2. Extensions

Some value should be associated with drawing a tile from the bag. For instance, the rack leave AET should be thought of as "AET plus 4 tiles from the bag". This is important because AET may be compared to "AERT plus 3 tiles from the bag". MAVEN's tile turnover value is the average of the values of the unseen tiles. Thus, if the tiles in the bag are better than the tiles in the rack then MAVEN has a reason to play its tiles, but if the tiles in the bag are worse then MAVEN prefers to keep its tiles. In addition to this heuristic, MAVEN uses three other heuristics described below, that are important for good rack management.

The first heuristic is called "Vowel/Consonant Balance". For example, SNRT is not as good as the sum of its tiles because there are too many consonants. MAVEN includes bonuses for all possible counts of vowels and consonants. We tested a dynamic bonus that depends on the ratio of vowels to consonants among the unseen tiles, but it did not improve performance. Then we settled on a Vowel/Consonant term after trying to include a term in the evaluation function for every pair of tiles (i.e., 351 different pairs). We found that there was a positive value associated with almost every Vowel + Consonant combination, and a negative value associated with almost every Vowel + Vowel or Consonant + Consonant combination, whereupon the much simpler Vowel/Consonant balance term seemed like a better choice.

The second heuristic, "U-With-Q-Unseen", is sometimes important. The idea is that holding a U when there is a Q unseen is better than it seems because a U-less Q costs 12 points, whereas a QU together is neutral. A linear function implements this concept.

The third heuristic is called "First-Turn Openness". This heuristic expresses the theory that it pays to play the first turn "tightly" by using a few tiles only. The reason is that the reply to the first move can be very awkward if the opponent has no access to double word squares. We have computed a table of bonuses associated with starting games by playing $2, 3, \ldots$ up to 7 tiles. MAVEN has implemented this feature, but we are uneasy about it because it is in MAVEN's general interest to open the board against humans. In this article we do not analyze MAVEN's performances against other programs.

3.4. Search and evaluation

There are three phases of the game, each requiring a different search algorithm: normal game, pre-endgame and endgame. A typical Scrabble position averages 700 possible moves (i.e., legal words), and there are situations (e.g., when the mover holds two blanks) when 8,000 moves are possible. Such a space precludes deep exhaustive searching, necessitating the program to be selective about which moves it will consider.

MAVEN uses different search engines, each specializing in one phase of the game. The basic engine is a one-ply search with a static evaluation function (see below). The static evaluation function is very fast and accurate enough for world-class play in the beginning of the game and in the middle game (i.e., for all but the last 3 or 4 moves).

The pre-endgame commences when there are 16 unseen tiles (9 in the bag plus 7 on the opponent's rack) and continues until the bag is empty. The pre-endgame search engine is a one-ply search with a static evaluation function that is computed by an oracle (for an explanation, see below). The pre-endgame evaluator is better tuned to the peculiarities of the pre-endgame, but it is comparatively slow. The endgame engine takes over when the bag is empty. Scrabble endgames are perfect information games, because you can deduce the opponent's rack by subtracting off the tiles you can see from the initial distribution. The endgame engine uses the B* search algorithm.

3.4.1. Board evaluation

It is obvious that each move changes the board. At first, it was not clear what to make of this, so we decided to assign the board a value of 0. In other words, MAVEN assumed that changes on the board were neutral. Later, when the assumption was revisited it turned out that 0 is a good approximation!

One widespread expert theory is that "opening the board" (i.e., creating spots where bingos could be played) is bad, on the grounds that the opponent obtains a first crack at playing a bingo, and if the opponent has no bingo then he could “shut down the board”. This seems like a promising “grand strategy”, but it does not work in practice. We tried several different functions that included significant understanding of bingo chances. All were worse than including no openness function at all. There is no general penalty for board openness. To understand board openness requires simulation.

Even if openness were slightly negative, it would be bad tactics to include such a factor in MAVEN, since the program never misses a bingo, whereas human masters miss a bingo in 10 to 20 percent of the cases. Obviously, this makes up for any “first mover” advantage.

Moreover, many experts have explained to us the general principle of avoiding placing vowels adjacent to bonus squares. The idea is that a high-scoring tile can be “double-crossed” on the bonus square, leading to a good score for the opponent. For instance, we were told that it is appropriate to sacrifice 6 points to avoid placing a vowel next to the bonus squares on the initial turn. This turns out to be laughably large. When we used computer analysis to estimate this factor, it transpired that the largest such penalty is only 0.7 points! Most vowel-bonus square combinations had penalties of 0.01 points, meaning that you should avoid it only if there was no other consideration.

These experiences showed two things. The first was to treat expert guidance with skepticism. Experts often do not know *which* move is best, let alone *why* it is best. We resolved to test every theory. The second was an understanding as to why board-related factors should generally be neutral. There are three reasons for this. First, you have to subtract off the average value of a play from the extra value of a hot spot. For example, if the opponent scores 40 points in a spot, that does not mean that a spot cost 40 points. If the opponent’s average score in a position is 35 points, then the net loss is just 5 points, which is not a big deal considering that you are not sacrificing points to block the spot. Second, creating a spot does not matter unless it is the *biggest* spot. If you create the second-biggest spot and your opponent takes the biggest, then *you* benefit. Third, the board is shared between you and the opponent. Maybe the opponent benefits, but maybe no one does, and maybe you do. Granted, the opponent moves first, so the opponent’s chances are better, but how much better? Assume that there are 11 moves left in the game (which there are, on average) so the opponent takes the spot 6 times, and you take it 5 times. What is the net deficit, on average? One-eleventh of the extra points gained? It is insignificant. And so the initial theory that the board is neutral was substantiated by qualitative and quantitative arguments.

There is one exception: “Triple Word Squares”. The Triple Word Squares are the highest premium squares on the board. It is very easy to score 30+ points by covering one such square. In fact, it is so easy to obtain a big score that this is the only exception to the general rule that board factors are irrelevant. MAVEN has a table of parameters that determine how damaging it is to open a triple word square as a function of the “shape” of the opening. The penalties range from 0 (e.g., for putting a Q on A14) to 12 points (e.g., for putting an E on 1E when both 1A and 1H are open).

The simple evaluation strategy discussed above improved scoring from 30 points per move to 35. This concludes the middle-endgame search and the middle-game evaluation. The endgame search and the pre-endgame search are discussed in the Sections 6 and 7 respectively.

3.5. Simulation

Simulation is a means of leveraging a fast but possibly inaccurate evaluation function into a slow but accurate evaluation function. The idea is to sample random racks and play out the game for a few turns using the basic engine to select moves. Moves that have a high average point differential in the simulated games are best.

One of the great things about simulation is that it is able to compare moves that were evaluated by different evaluation functions. For example, assume that the basic search engine considers move A to be best, but the pre-endgame engine prefers move B. Which move is better? The answer is probably move B, but it may be move A. Simulation provides an answer. MAVEN can simulate games after moves A and B, and then compare the two moves on the basis of point differential (or winning percentage). A detailed description and some history is given in Section 5.

4. Move generation

At a conceptual level, the dictionary is simply a list of words. But that list must be indexed so as to allow rapid retrieval of words based upon letter content. The dictionary representation of choice is called a Directed Acyclic Word Graph (or "dawg") (cf. [2]). This representation is a reduction of a letter trie, so that data structures will be described first.

Given a list of words you construct a letter trie by subdividing the list according to the initial letter. In Scrabble you start with 95,000 words, then break that down into 26 lists. You can index the lists by creating a search tree whose root node has 26 pointers, one for each list. If you continue this procedure recursively until every list has zero entries then you have a letter trie. If you look at the nodes of the trie you will find that there are many duplicates. For example, the following pattern is repeated thousands of times: "the prefix that you have traversed thus far is a word, and you can also add an 'S', but there is no other way to extend this prefix to make a longer word".

A dawg is a trie after all identical subtrees have been reduced, for example for postfixes such as "-s", "-ing" and "-ed". Also, among the 26 pointers, in each node many null pointers exist, and it is wasteful to represent them. Instead, a node is represented as a variable-length packet of edges, where only the non-null edges are represented. A dawg is isomorphic to the minimal finite-state recognizer for the regular language represented by the word list.

A dawg is a compact representation of the word list. For example, the 95,000 words in the OSPD take about 750 K bytes when represented as a word list. The dawg represents the same list using about 275 K bytes. The efficiency comes from "sharing" common prefixes and suffixes over many different words.

4.1. Brief history of move generation

Our first move generator straightforwardly searched for spots to place words on the board such that the rules of the game were satisfied. Let us take a specific word (QUA) and see how this might operate. The question is to identify the set of all squares on which the word QUA may start. Without loss of generality, assume that QUA is horizontal. To play QUA, several constraints must be satisfied simultaneously:

(1) An empty square (or the edge) to the left of the Q (since Q is the first letter).
(2) An empty square that is adjacent to an occupied square between 0 and 2 squares of the Q (since QUA has to connect to the board).
(3) Either a Q in our rack or a Q on the board.
(4) Either a U in our rack or a U one square to the right of the Q.
(5) Either an A in our rack or an A two squares to the right of the Q.
(6) An empty square (or the edge) to the right of the A (since A is the last letter in QUA).
(7) The Q, U, and A must either not form crosswords or they must make valid crosswords with letters already on the board.

To make it easier to find solutions to such constraints, we pre-computed the set of all squares that satisfied any such constraints (e.g., the set of all squares that had a U one square to the right). The constraints were represented as bit vectors, using one bit per square. Intersecting a number of these bit vectors identified all squares satisfying all of the corresponding constraints.

There are other algorithms. For instance, James Cherry devised a move generator based upon permuting the letters in the rack [7]. Cherry's algorithm is not as fast as Appel and Jacobson's [3] but it can be implemented on small-memory computers (e.g., we implemented it once on an 8-bit microprocessor using only 96 bytes of RAM). Our "bit-parallel" method described above works well on machines with large words (e.g., 64-bits). In the end we followed the suggestions from Appel and Jacobson. Steve Gordon, among others, supported this choice by remarking that the algorithm created more sophisticated state-machine representations [9,10]. He implemented the fastest known move generator by following a recommendation from Appel and Jacobson. Actually, with RAM being in abundant supply nowadays, Gordon's algorithm is probably a better choice than Appel and Jacobson's, as it gains a factor of two in speed, and costs a factor of 5 in space for storing the dictionary (from 0.5 MB for a dawg to 2.5 MB for Gordon's data structure), which is a negligible expense.

As stated before, MAVEN has stuck with Appel and Jacobson's algorithm. It balances speed, memory and simplicity better than any other choice.

5. Simulations

The basic evaluation function in MAVEN incorporates only "first-order" factors. In other words, we do not try to understand any interactions between factors. For example, a W is usually a bad tile, but assume that it happens to play very well on a specific board. MAVEN then does not identify that situation and can make an error. That specific combination of events will not happen very often, and when it does it is not a big deal (see Section 3.4.1).

It is possible, in theory, to incorporate sophisticated factors into a positional evaluation. One could, for example, analyze whether there are any productive hot spots for a W, and if there are then increase the evaluation weight of the W. However, that would address only that single, relatively miniscule, problem. What about the numerous other potential dependencies? Even the number of *categories* of possible dependencies is huge. Extending the evaluation function is a bottomless pit.

The problem is that an evaluation function takes a "deconstructive" approach to positional analysis. Breaking a position down into independent factors is helpful in that we can trade off factors, but it is self-limiting because a "synthesis" of the position never occurs.

Another view of the problem is that the evaluation function applies "average" values of individual factors to the current position. Of what relevance are average values to this position? Usually the relevance is good enough, but sometimes not.

Let us recall the discussion on how MAVEN's rack evaluation parameters were derived. The value of a tile was set to the average amount by which it increased scoring over the rest of the game. Would better positional play result if we had an oracle that could tell us the true values of all evaluation parameters *starting from the current position*? Simulation is the oracle we seek. Simulation provides a whole-position evaluation of moves that have been evaluated within the context of the current position. The mechanism is to average the scores resulting from playing out the game for a few moves. As long as the basic evaluator is reasonably accurate, the simulation will uncover whatever "tactical" factors happen to exist in the position.

5.1. History of simulation

The start was a fruitful discussion with Ron Tiekert at a tournament in December 1987. Ron was a legendary player, a National Champion, and the player who achieved the highest NSA (National Scrabble Association) rating (2167, a record that still stands). Ron was well known for his skill at evaluation, and the hope was to learn from him. Evaluation is, after all, the only mistake that MAVEN ever makes. The reasoning behind this statement is that for every game, an AI program only makes evaluation errors, provided that the program considers all legal moves (as MAVEN does).

One particular problem was the opening rack AAADERW. There are two prominent moves: AWARD (8H, 22, AE) and WARED (8D, 26, AA). Expert opinion favored AWARD, and MAVEN agreed. But Ron chose the unusual play AWA (8G, 12, ADER), claiming it was far and away the best play. Ron said that he was initially attracted to AWA because it had a nice balanced rack leave (ADER) that is more likely to make a bingo on the next turn than the two-vowel leave of AE after AWARD. Also, AWA does not open access to the double-word squares, whereas AWARD does. AWARD can be "hooked" with an S for big plays down the M-column.

But Ron was not certain that AWA was better until he played out both AWA and AWARD using parallel racks *fifty times each*. Ron said that AWA finished with much better results, so he was confident that AWA was better. Ron also mentioned, in passing, that with MAVEN being as fast as it is, it should be able to crunch this type of problem pretty quickly. For me, this experience was like a ray of light from heaven. Here is Ron's procedure. First he drew racks for the opponent. He ensured that the racks had a balance of tiles representative of the current distribution of unseen tiles. Then he played a move for the opponent after each candidate play. Then he filled the rack of the side to move, using parallel draws to the greatest extent possible. Then he played a move for that side. He totaled the scores and folded in an estimate of the value of the two racks left at the end.

We will call this process *simulation*. Simulation provides the deepest insight into Scrabble positions. It is a tremendous tool. You can control the moves to be compared and the number of moves you look ahead; moreover you can choose whether you evaluate endpoints by score or by winning percentage. Any position can be evaluated using this technique. The only limitation is the question of whether MAVEN is playing the variations well. In fact, even that consideration is muted by the fact that simulation results might be valid even if MAVEN plays the variations badly, provided that MAVEN plays "equally badly" after each candidate. Such "cancellation of errors" is a very important factor in simulations. The only remaining drawback is that simulations are long. The standard deviation of a two-move sequence in Scrabble is about 30 points. This means that even after 1,800 iterations you have a 1-point standard deviation in a comparison of two moves. If you want to achieve really fine judgments it can take a simulation that runs overnight. Still, it is at least possible.

5.2. Search depth

The key issue is to search sufficiently deep to allow hidden factors to come to light. While it is possible for factors to take several moves to work out, it usually requires just two ply in Scrabble. Scrabble racks turn over pretty often. An average turn involves 4.5 tiles, so after a couple of moves there is usually nothing left from the original rack. Hence, two ply is the general case, but there are exceptions. Consider the opening rack CACIQUE. The word CACIQUE can be played from 8B for 96 points. Or it can be played from 8H for 110 points. A 14-point difference would usually be a no-brainer, but in this case the 8H placement suffers from a huge drawback: if the opponent holds an S then he can pluralize CACIQUE while hitting the triple-word square at O8. For example, the riposte FINDS (O4) would score 102 points. After CACIQUE (8B) there is no such volatile placement. A two-ply simulation shows that the 8H placement is better, but simulations to four or more ply show that the 8B placement is better.

One must be wary about using deeper simulations! They take more iterations to reach statistical accuracy and they are slower as well. For example, a four-ply lookahead has double the variance of a two-ply simulation, and each iteration takes twice as long. The total cost is about three times as much CPU time when using a four-ply lookahead. We maintain that if it requires an extreme situation like CACIQUE to see the value of a four-ply simulation then they are not worth doing.

6. The endgame

The endgame is a game of perfect information. That is because each player can subtract the known tiles (i.e., those on the player's rack and those on the board) from the fixed initial distribution to arrive at the set of tiles that the opponent must have. It was surprising to find out that human experts do this routinely. Some players even allocate their time so that they have 10 minutes (out of 25) for thinking through the endgame and "pre-endgame" moves.

What characterizes good endgame play? Well, for one thing scoring since you have to score points; but not at the expense of going out. Going out first is very important since if your opponent goes out then you lose points in three ways: your opponent gets an extra turn, you do not receive his tile penalty, and he obtains your tile penalty.

There are also defensive aspects to endgame play. We provide three examples. First, if your opponent has one big hot spot then you want to block it. Second, if your opponent has just one place to play a specific tile, then you want to take that opportunity away. Third, if your opponent cannot play out because (s)he is stuck with a tile then you want to maximize the value of your tiles by playing out slowly.

What sort of algorithm leads to good endgame moves? Let us start by investigating the obvious. Full-width alpha-beta search is the conventional technique for perfect-information two-player games. However, it works badly in Scrabble for many reasons. First, the branching factor is rather large: about 200 for a 7-tile versus 7-tile endgame. Second, search depth can be an issue. When one player is stuck with the Q it may take 14 ply to justify the correct sequence, which is often to play off one tile at a time. Third, alpha-beta's performance depends critically on good move ordering. Unfortunately, Scrabble move generators produce moves in prescribed orders. Of course, you can generate the moves and then sort them, but a good general-purpose heuristic for scoring moves is not obvious. For example, in stuck-with-Q endgames it will often be a low-scoring move that works best. Finally, move generation in Scrabble is computationally expensive. I brag about how fast MAVEN is, but compared to the speed required for a real-time 14-ply full-width search, MAVEN is slow. Therefore we rejected full-width alpha-beta search. However some argue that due to the increasing computing power available alpha-beta may have a comeback in the future. We do not believe this since branching factors of 200 and depths of 13 ply are not so easily dismissed. To understand the endgame search problems we provide an example where one player is stuck with the Z.

Fig. 2 gives a position that illustrates many endgame issues. It occurred in a game between Joe Edley and Paul Avrin, with Joe to move. Joe's rack was IKLMTZ and Paul's: ?AEINRU. Joe's problem is to play away his Z, which is unplayable in the position now.

Joe's move, TSK (4L, 18), is strongest. It threatens to follow with ZITI (L2, 13). The most straightforward plan for the defender is given below:

Joe:	TSK	4L	18
Paul:	DRIVEN	8J	−36
Joe:	ZITI	L2	13
Paul:	AUK	N2	−14
Joe:	(remaining LM)		−8 = −27 net.

To surpass this result requires a truly deep assessment of the position. MAVEN quotes the following variation as best. It might not actually be best, but it is certainly impressive, and I have not found anything better.

Joe:	TSK	4L	18 (Best for offense. What should the defense do?)
Paul:	URN	2J	−3 (Blocks ZITI (L2), but that allows...)

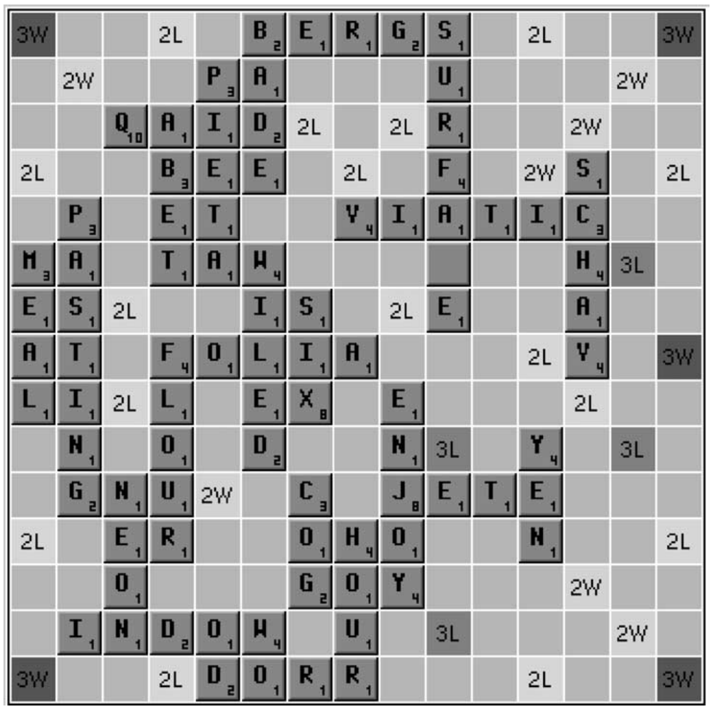
Fig. 2. Joe Edley versus Paul Avrin. An example of complex endgame play.

Joe:	VIM	8M	24 (The defender to get the triple-word square.)
Paul:	MU	O8	−4 (Defender plays out one tile at a time.)
Joe:	pass		0
Paul:	ENJOYS	I9	−16 (Can he get back enough points?)
Joe:	pass		0
Paul:	IN	12K	−4 (What is he getting at?)
Joe:	PAL	2E	7
Paul:	VIATICA	5H	−18 (Oh!)
Joe:	pass		0 (No play for the Z, of course.)
Paul:	HE	6M	−16 (The point!)
Joe:	(remaining Z)		−20 = −32 net

This variation, 12 moves long, does not have to be seen in its entirety in order to choose TSK. TSK may be selected after noting that it is high-scoring and sets up a strong threat to counter the opponent's play DRIVEN.

By contrast, to choose the defender's first move requires seeing the entire 11-move sequence that follows, since it is only the stunning setup VIATICA followed by HE that finally justifies the initial sacrifices of points. Note that the opponent passes to delay VIATICA, because it can be followed by VIATICAL. The defender finally forces his opponent to play off the L with PAL, and then VIATICA and HE follow.

6.1. Selective search

The obvious next step is to consider selective search, and that implies the need for an accurate evaluation function. There are two factors that make such an evaluation function possible: the game is rapidly converging, and the position is largely static. A converging game is one in which the number of moves reduces to zero along every variation. Scrabble has this property because tiles never return to the bag, and when one player is out then the game is over. Scrabble endgames are largely static because the great majority of the tiles (at least 86 out of 100) are already on the board, and they will not move. As the game progresses each addition to the board further freezes the position.

How can these properties be exploited to build an evaluation function for Scrabble endgames? By using dynamic programming. Because endgame positions are largely static, the evaluation depends largely on the two racks that the players hold. In other words, if the position reduces to one where we hold ER and our opponent holds AW, then the evaluation of that position is largely determined. It might vary as a function of changes made to the board position, but we can deal with that in the search engine.

It is impractical to build a table of all possible racks that we hold versus all possible racks that our opponent holds, as there are up to 128 possible racks for each side. But fortunately we do not need the full set. Simply knowing what one side can achieve within N turns (for $N = 0, \dots, 7$) is sufficient.

We can statically compute the values of the racks if $N = 0$ (i.e., if the opponent goes out before we move). That is simply -2 times the sum of the tiles. For $N > 0$ we use dynamic programming. For every move that can be played out of each rack we compute the score of that move plus the evaluation of the remaining tiles assuming that $N - 1$ turns remain. The value of a rack in N turns is the largest value of that function over all possible moves.

The beauty of this algorithm is that we can statically determine how many turns it will take a given rack to play out. If we wish to know whether DERTTT will play out in N moves, we simply check to see whether there are 0 tiles left if DERTTT has N moves to play. Now assume that MAVEN moves first, holding DERTTT and the opponent has EJVY. What is the outcome? MAVEN starts by computing the result assuming that DERTTT will play eight moves (and EJVY will play seven moves). That gives us one sequence which we will minimax to determine the right answer. Another sequence is if EJVY has the option of playing out in seven turns, which gives DERTTT the right to move seven times. If that is better for EVJY then EVJY will prefer that alternative. Then DERTTT has the opportunity to improve if it can play out in seven turns. And so on until neither player can improve.

This evaluation function is static and quite fast. It takes two full-board move generations to construct the tables. We can then do an evaluation of any position in the search space. Our first really successful endgame player did a two-ply search of moves ordered by this evaluation function. This program was really good, because it could execute the key skills outlined above. It scored points, because it included all the high-scoring plays. It could block, provided that any blocking play was included among the top N moves (which is usually the case; hot spots for the opponent are usually at least somewhat hot for us). It could play out in two moves because the evaluation of positions after two ply includes the observation that the tiles we keep will be out in one. It could play out one tile at a time if the opponent is stuck, because it would see that it scores more points in seven turns than in two turns.

We suspected that this algorithm resulted in play that was better than human experts, and indeed, it was a huge leap forward for computer programs. Unfortunately, the approach had its share of weaknesses. For instance, blocking occurred largely as a matter of chance. There were at least three questions that turned out to be obstacles. What about *set-ups*, in which one plays tiles knowing that the remaining tiles will combine well with the tiles just played? What about *negative set-ups* where your move places tiles in an unfortunate spot that sets up the opponent? What about obvious moves? As it stood, the algorithm spent the same amount of time on every play.

6.2. The opponent's possibilities

Part of the problem was that the plausible-move generator did not take into account the opponent's possibilities. That was easy to fix, since we were already generating the opponent's plays while building the evaluation tables. We simply saved a set of promising opposing moves. Then, after generating moves for the side-to-move we could determine which of these opposing moves were not blocked, and score the resulting variations after a two-move sequence was played.

This tweak was a big improvement, but I noticed some disturbing problems. For example, assume the opponent holds ROADWAY, and he can play ROAD for a small score, and WAY for a big score. Actually, WAY is the opponent's only big score, and we must block it. But when we evaluate a move that blocks we find that the evaluation may be off. For example, assume that among the opponent's replies we find ROAD, which is fine so far. But if we are not out-in-one after our initial move then the evaluation of the future is "way off", because the evaluation tables believe that holding WAY is valuable. The change in position is not reflected in the evaluation tables until the opponent generates plausible moves.

The problem vexed me for a while. We considered modifying the evaluation function to account for the fact that WAY was blocked, but that would just introduce other problems. For example, assume that WAY could be played in two different locations, one scoring 25 points and the other 28 points. The gain from blocking the high-scoring spot is only 3 points. How would the evaluation function know what the gain was?

6.3. The B algorithm*

The last question set me on the path to the right solution. The real issue is the difference between the highest and second-highest score using a set of tiles. Accordingly, the evaluation function was generalized to include an analysis of the risk associated with an N-turn sequence. In effect, there was an optimistic evaluation and a pessimistic evaluation for every variation. The idea was that the true evaluation would be found within those bounds. Computing such bounds is tricky, but with sufficient test cases you can eventually make a system that almost always works.

The problem is that minimax search does not work with intervals. Thus, we needed a new search engine that worked with an evaluation function that returned intervals. The answer is Hans Berliner's B* algorithm [4]. We used his original formulation, and it worked like a charm. We know of only one other application where B* is superior to all other known techniques [19]. In part this is because the evaluation function and plausible-move generator are tremendously accurate. In part this is because the domain is converging and not very deep.

7. The pre-endgame

There is a big difference between a middle-game simulation and a pre-endgame simulation. In the middle-game it is not possible for the game to end within the two-ply horizon. So we can ignore the future, since it is equal across all moves (i.e., the opponent moves first in every case, and the game lasts about the same number of moves, etc.). But in the pre-endgame this is no longer the case. Sometimes two-ply continuations finish the game outright, and in other times the game still has a long way to go. Since the future is highly variable between these two endpoints it cannot be ignored. Therefore, we decided that lookahead in the oracle used in pre-endgames (see Section 3.4) should happen to the end of the game rather than to a fixed number of plies.

When there is one tile left in the bag then the true endgame will commence with the next move (barring a pass). Such a situation is called a "PEG-1" (Pre-EndGame-1). There are up to eight possible endgames to consider. In theory, we could enumerate all possible one-tile draws from the bag and solve the resulting endgames, which would result in perfect play (provided that the best move is considered). In a PEG-2 there are 36 endgames to consider, and there is also a new possibility: the side-to-move can play exactly one tile, which would bring up a PEG-1. We can no longer always reduce the problem to repeated use of the endgame analyzer, but any move that empties the bag could be exactly analyzed that way. In a PEG-8 it is no longer possible to empty the bag, so no endgames can arise immediately. Every play leads to another PEG. Moreover, there are a plethora of possible continuations, so the impact of the draw is immense.

It is clear from the sequence given above that the value of precise analysis decreases as the number of tiles in the bag increases. In other words, MAVEN's statistical evaluation model becomes progressively more accurate as the game lengthens. It is only very close to the endgame that we really have to worry about specifics. We made a design decision that the pre-endgame begins with PEG-9.

7.1. Pre-endgame issues

To play well in pre-endgames you have to master several tactical issues. Of course, scoring points is still important, as is keeping good tiles. These things are always important, but in the pre-endgame these factors have to be balanced against other issues. Below we discuss three issues. First, you want to obtain a good endgame. Mostly this means that you want to move first against an opponent that holds an almost full rack. Avoiding the Q can also be an issue. These are "timing" issues with respect to the end of the game. You want to score well and to time things so that the opponent will empty the bag. Second, there are the blocking issues. You want to block openings that your opponent can use to score well. Usually this means blocking bingo lines, but it can be other things as well. Finally, we have "fishing" issues. You want to create chances for your own big plays, especially bingos.

7.2. Probability-weighted search

In this subsection we provide solutions for two of the three issues mentioned in Section 7.1. Good endgame timing is resolved by a simple table-lookup indexed by the number of tiles left in the bag after the play. There are several cases, depending on whether blanks and the Q are still in play, but that is just a detail. It is easy to code and creates better endgame timing. This feature has the effect of emphasizing endgame play, which is to MAVEN's advantage.

Blocking is much harder, and thereafter comes fishing for bingos which is even harder. Blocking requires, in theory, a two-ply search, and fishing for bingos requires a three-ply search (or simulation). It seems clear that the two-ply problem must come before the three-ply problem (i.e., before fishing), so we focus below on the problem of blocking the opponent.

The evaluation function of a two-ply search takes the form of

$$\mathit{OurScore} - \mathit{OpponentScore} + \mathit{Future}$$

where *Future* is determined by the rack evaluator and endgame timing factors, and the scores are determined by the moves. *OurScore* is known from the move we are generating, but the *OpponentScore* must be estimated because the opponent's rack is not known.

At first, we considered using simulations (see Section 5), but at the time we faced the problem (circa 1992) we did not have computers that were fast enough to simulate plays in real time. Besides, the problem was to *generate* good moves and simulations depend on the quality of the moves fed into them. So, it was decided to compute an approximate distribution function of the opponent's move. Here is how it worked: we threw all of the unseen tiles into one large "rack" and generated all of the moves that could be played. For each move we computed the probability of holding those tiles, and created a list of the high-scoring spots and calculated the chances of playing there. Then we noticed that scores in a spot could be very different when a move used a Q, in particular, compared to the same spot not using the Q. Therefore, the definition of "spot" was expanded to include a specification of the highly-valued tiles used by the move. (To our chagrin, obvious facts like this had escaped our attention during design.)

Once a distribution is created we can approximate a two-ply search. For every move of ours we scan the distribution to determine which replies are not blocked. Then we weight each reply according to its likelihood.

The pre-endgame engine required extensive debugging. There are all sorts of constraints on the behavior of the opponent that affect the distribution of moves. For example, our first program could have projected that the opponent would play the Q 100 percent of the time, even though there was only a 7/8 chance that the opponent even held the Q. The scores needed all sorts of tweaks to account for the quality of the tiles played, and so on.

There are many parameters to tune too, and it turned out that when there are two blanks unseen then the pre-endgame engine would take too long for PEG-9s. Moreover, there was much tweaking before we were satisfied with the engine (i.e., the results were reasonably accurate).

7.3. Unclear benefits

The engine unquestionably reduced the number of bingos by the opponent and also other adverse big plays. But it made plenty of weird moves too. For instance, it could defend against threats that were just not significant or sometimes block spots that were favorable on balance. So, the engine was sensitive to defensive considerations, but not sensitive to other considerations. Benchmarking the new generator showed uneven results. Yes, the insight was sharp, the algorithm was clever and the implementation was good, but all these did not change which side won the game (at least not in the MAVEN self-play trials performed). The explanation must be that the new generator helps when MAVEN is ahead, since it unquestionably reduces the number of bingos played by the opponent. But it must hurt when MAVEN is behind, since the overall results are even. What should we do?

The obvious answer is to use the new generator when ahead and the original when behind, but we never did that. We intended to extend the new generator to cover the come-from-behind case. Unfortunately, we never succeeded in doing so. The reason is that it is not easy to extend the framework. The core concept of this technique is to evaluate moves on the basis of *Our Score minus the Opponent's Score*, whereas coming from behind requires including our next turn as well. The framework covers our future turns by using rack evaluation, but this is insufficient for guiding a comeback strategy.

In summary, we regarded the pre-endgame generator as a small positive asset to the program. It eliminated one possible weakness in MAVEN (that an opponent might win by fishing for bingos at the end), while it maintained MAVEN's overall level of play. Since humans will be in the trailing situation in most pre-endgames against MAVEN, the defensive orientation is clearly beneficial. The pre-endgame engine really shines as a generator of moves for simulations, since it finds defensive moves that would otherwise be missed.

8. Odds and oddities

This section deals with eight strategic decisions taken on the basis of observations or as an answer to questions. The first observation is that a PEG-1 is a key special case. It is possible to analyze exhaustively all possible endgames that can arise, since the bag will be emptied by this play (barring a Pass). Hence, the simulation is not a simulation, actually. It is really a call to the endgame engine to determine how the game will play out. This capability is valuable, and therefore implemented in MAVEN.

The next question is how "wide" must the search be in a simulation? The "width" of a simulation refers to the number of moves considered, and obviously the more the better. But there are diminishing returns. To structure the considerations on the width, the following information should be taken into account. MAVEN's basic engine orders the very best move first in 50 percent of all turns. Each successive move in MAVEN's list takes another hefty fraction, so that by the time you get to 10 moves there is very little left (about 5 percent or so is our guess). This guess influenced our decision on the width.

Third, we have a question on the cooperation of the basic engine and the pre-endgame engine. The basic engine does not produce key moves in the pre-endgame. Therefore we decided to augment the move list with moves from the pre-endgame engine, and also add in several other moves. For example, we added "ultra-paranoid moves" which sought to block bingos regardless of score and rack leave. Moreover, we added "score at all cost" moves, which disregard all future factors (including rack leave), since sometimes your opponent is guaranteed to bingo out and only by playing off tiles and scoring points can you manage to stay ahead. You also have to consider "one-tile" plays. All in all, there were 5 move generators for the pre-endgame, each capable of generating up to 10 moves. In practice there was a great deal of overlap, and we rarely saw more than 25 unique moves in a pre-endgame simulation.

The fourth strategic decision had to do with fishing. MAVEN's generators as described above are almost perfect (in the sense that they generate the best moves), but they do sometimes miss a key move. Most of the errors have to do with fishing. A fishing generator is not hard to write, but we have never started to write it.

The fifth strategic decision contains an intriguing question: how many samples are needed? The goal is to achieve statistical separation between the highest move and the others. In Scrabble you have about a 30-point standard deviation per iteration, so it takes about 900 iterations to reduce the sampling error to 1 point. If that is sufficient to separate the moves then you need not search any more. A different perspective is that if you search for 900 iterations and then make an error due to a sampling error of 1 point then it is no big deal. Losing a point is not significant. Accordingly, our simulation controller sets an arbitrary limit of 1000 iterations.

The sixth strategic decision is on statistical control of simulations. It is a crucial one, since we cannot afford to simulate 10 plays to 1000 iterations of two-ply lookahead for up to 12 moves in a game. We have to cut down the computation by about a factor of 8. The key insight is that many moves can be rejected after only a few iterations. Since the standard deviation of a two-move sequence is about 30 points, it follows that after only 16 iterations we can already reject (with over 90 percent confidence) moves that are 15 or more points below the highest-rated move. As the number of iterations increases, the confidence interval narrows, and thus more moves can be pruned. Our controller simulated every move for at least 17 iterations, and then pruned on the basis of point differential (or winning percentage in the pre-endgame). Any move that was two or more standard deviations below the top move was pruned. This rule caused rapid convergence of the simulation when one move was obvious. It also rapidly pruned moves that were truly terrible. The remaining moves could go the distance, but it was very rare that even 3 moves simulated to the full 1000 iterations.

A seventh strategic decision is on pruning. In simulations the problem of similar moves may happen. It often happens that two words play in the same spot and use the same tiles. There can be zero difference between such plays, and such simulations would always go to the full 1000 iterations. Therefore, MAVEN prunes the lower of two such moves at iteration 31.

Finally, we have a special rule on bingos. At move 100 MAVEN prunes all bingos except the highest-rated ones. This is a rather similar issue as the strategic decision above. Two bingos often score the same, and we do not want to simulate to 1000 iterations only to find that one of them is 0.1 points better.

These pruning rules save the equivalent of a factor of 16 compared with simulating 10 moves to iteration 1000. Moreover, these rules have good accuracy. A review of 14 games showed only one instance where a pruned move proved to be best, and that only cost 3 points. There were many more occasions where the best move was number 8 or 9 or 10 on the list—a move that we could not afford to consider without pruning.

Obviously, simulation is a huge win for MAVEN. Our first test of simulation games with MAVEN (endowed with the results of the strategic decisions) against the basic MAVEN engine was so amazing that we spent two days searching for a bug in how we collected the resulting data. We had not seen such a huge advance in MAVEN's skill since we implemented the first rack evaluator. The simulator defeated the basic engine by winning 57 percent of the games played, with a 15-point advantage per game. For all this, one should keep in mind that the basic engine is a championship-class player. Just as impressive were the types of moves that the program found. They were the kind of moves that human experts would tell us, namely the kind that humans occasionally find in a blinding flash of inspiration. Only now it seemed to happen to MAVEN in every game.

9. Myths and rumors

Human experts were the source of several evaluation-function ideas that turned out not to be as the human players believed. These concepts, below called fictions, were routinely used by the experts of the day, and they unquestionably hurt their play. This section repeats three differences, earlier given as illustration in the text, by listing the human and computer perspective on three popular Scrabble concepts. The computer perspectives are mentioned as facts.

Fiction 1. Each tile played is worth between 2 and 3 points. That is, if you play an extra tile then you could afford to sacrifice 2 to 3 points off the score of a move. The idea is that by turning over tiles you were more likely to draw the blanks, each of which could turn into a bingo. In addition, the value of drawing a blank is magnified by the fact that not only do you draw the blank, but also the opponent does not draw it.

Facts. This line of reasoning does not work. Trying to turn over extra tiles not only decreases the score, but also decreases turnover! The key observation is that it is easier to play good tiles than bad ones, so a turnover bonus encourages the program to keep bad tiles. MAVEN's rack values are computed as a globally optimal balance between present and future scoring, which subsumes the concept of turnover.

Fiction 2. “Opening the board” (i.e., creating spots where bingos could be played) is bad. This seems reasonable because the opponent obtains the first crack at playing a bingo, and if the opponent has no bingo then he can just “shut down the board”.

Facts. Although this seems like a promising strategy, it does not work in practice. Several different functions were implemented that contained significant understanding of bingo chances. All were worse than including no openness function at all. MAVEN does not include a general penalty for board openness. To evaluate board openness accurately requires simulation. Even if openness had a slight negative score, it would be bad tactics to include such a factor in MAVEN. MAVEN, after all, never misses a bingo, whereas human masters miss a bingo 10 to 20 percent of the cases. Such a frequency makes up for any first-mover advantage.

Fiction 3. Experts use the general principle of avoiding placing vowels adjacent to bonus squares. The idea is that a heavy tile can be “double-crossed” on the bonus square, leading to a good score for the opponent. The theory goes that it is appropriate to sacrifice up to 6 points to avoid placing a vowel next to the bonus squares on the initial turn.

Facts. The human estimate is grossly in error. Computer analysis showed that the largest such penalty is only 0.7 points! Most vowel-bonus square combinations had penalties of 0.01 points, meaning that you should avoid it only if there was no other consideration.

These experiences illustrate two things. First, it is important to treat expert advice with skepticism. Second, human experts often do not know *which* move is best, let alone *why* it is best.

10. Experiments

There are several ways of demonstrating that MAVEN is capable of super-human play. These include self-play results, experience against strong human players, and anecdotal evidence. Below we provide relevant snapshots of all three possibilities.

10.1. Skill metrics

The obvious measure of skill is the average number of points scored per game. Obviously, if Player A averages 450 points per game and Player B averages 350 then A is a better player than B. But there is a huge problem with this metric, and if you play Scrabble for any length of time then you will find this out. The quality of the opponent matters a great deal; one can score more points against weaker opponents than against top masters. Players A and B, in our example above, could be roughly equal players, but A has played weaker opponents than B.

The reason why this happens is that the number of turns in a game increases when the players are weak. Strong players use 4 to 4.5 tiles per turn, so the games are finished in 11 to 12 turns per player. Weak players may only dispose of 3 tiles per turn, in which case the game drags on for 16 turns per player. In a match-up of strong versus weak players, the strong player has the chance to play more moves than he normally would when playing against his peers.

A slight modification to this metric, measuring the average number of points scored per turn, is a much more reliable indicator of a player's skill level. It works because the players alternate turns, so a game ends with each player having the same number of moves (to within one). Indeed, this measure is remarkably consistent.

An opponent's style can influence the number of points-per-turn scored. Some players create very constricted positions in which it is difficult to score. Other players prefer wide-open positions. There are limits to the impact of this factor, since carrying stylistic preferences to an extreme will reduce your winning chances by more than it hurts the opponent. Experience suggests that if defensive measures reduce your opponent's score by more than a couple of points per turn then you are hurting yourself more than your opponent.

Points-per-turn is not used to measure human skill. Humans rate players using an Elo-rating system akin to the system used in chess. Here, both your competitive results (wins, losses and draws) and the strength of your opponent are used to determine whether your rating goes up or down. On that scale novices rate around 700 and the highest player is usually around 2100. A 200-point differential in rating corresponds to a 77-percent winning percentage. Any player rated over 2000 has a legitimate chance to win the North American Championship.

Rating is a very good measure for assessing tournament play, since it permits indirect comparisons of players by "factoring out" the quality of the opponent. However, rating is a more variable function than points-per-turn. MAVEN's self-play games contain 21 moves on average (almost 11 per player). So estimates based upon points per turn converge $\sqrt{21/2}$ times faster than estimates based upon rating.

We do not really need one single measure of strength; the important thing is to have measures that are appropriate for what we are trying to accomplish. In comparing consecutive versions of MAVEN, points-per-turn is used since it is easier to reach conclusions instantaneously. In comparing MAVEN versus humans, rating proves to be the most useful metric.

10.2. Tournaments and matches

10.2.1. MAVEN *vs.* MAVEN

MAVEN averages 35.0 points per move. MAVEN-versus-MAVEN games are over in 10.5 moves per side on average. Each MAVEN plays 1.9 bingos per game. These statistics are considerably better than human experts. The best humans average around 33 points per move, 11.5 moves per game and about 1.5 bingos per game.

10.2.2. Man–machine experience

MAVEN has played in a number of human tournaments and arranged matches. The following summarizes the program's career.

- December 1986. First tournament against top experts. MAVEN finished 8–2, including some sparkling, high-scoring romps over top championship contenders.

- October 1987. Pairs tournament where two players jointly choose which moves to make. Ostensibly, MAVEN and the author were entered as a pair, but the latter automatically seconded MAVEN's moves. MAVEN won 5 games against no losses. The opposition was expert but not championship caliber.
- December 1987. Tournament against experts, finishing 7–3. This was the first event in which MAVEN had a working endgame player.
- July 1997. Exhibition match against North American Champion Adam Logan at AAAI-97 in Providence. MAVEN lost 0–2. Guided by the wish to demonstrate MAVEN's simulation player for the first time, little time was left for preparation. MAVEN did not play so badly, though; it simply did not demonstrate what was hoped for.
- March 1998. MAVEN defeated World Champion Joel Sherman and runner-up Matt Graham 6–3 in a match sponsored by the New York Times. This match was played against the commercial Scrabble CD-ROM, and therefore did not feature simulations.
- July 1998. MAVEN defeated Adam Logan by 9–5 at AAAI-98 in Madison. MAVEN employed simulation to tremendous benefit. The twelfth game of the match is published with annotations in Appendix A.

MAVEN's total match and tournament record is 35 wins and 15 losses against an average rating of 1975. The average opponent rating is low, since we credited human champions earlier with a 2050 rating rather than their inflated post-championship ratings of 2100 or higher. A 70 percent score against such opposition is a major result. On the basis of these results MAVEN's rating has been computed as 2124. Human players occasionally have had a higher rating than that, but no human has maintained a rating much over 2050 for any substantial time period.

Finally, what qualities of MAVEN make it such a tough player? The answer is that MAVEN is well prepared in all facets of the game. MAVEN knows all the words, it evaluates the moves well, it plays fast (which puts opponents into time trouble), it never loses a challenge, and it plays endgames perfectly. It never gets tired or inattentive, which is a significant factor in match and tournament play. These are formidable skills.

10.2.3. MAVEN–*Logan*

In this subsection, we briefly discuss the 1998 AAAI match against Adam Logan, which MAVEN won by 9 to 5. In Appendix A the moves of MAVEN's twelfth game in that match are analyzed. Adam was (and still is) one of the top six players in the world (he finished fourth in the 2000 World Scrabble championship). There were 14 games played in which 168 move decisions per side were taken. Adam played well, but it is abundantly clear that MAVEN played better. MAVEN's errors came from three sources:

(1) A bug that prevented it from choosing exchanging moves. There were three moves in 14 games where it should have exchanged but did not, costing 9 points.
(2) A bug where the endgame player would choose inferior moves. An "optimization" inserted to speed pre-endgame search was incorrectly used in the endgame. Fortunately, it was impossible for this bug to affect the outcome of a game.
(3) Search control issues. There was one instance in which MAVEN cut off a move that proved to be best. This cost 3 points. Then there were cases in which MAVEN did not consider the best move. It is hard to estimate how many errors were attributable

to missing the best play, since if MAVEN does not generate the move then it is very unlikely that its developer finds the move. Of course, once found such a move can be analyzed with the help of MAVEN. Consequently, there is an interesting way to measure the impact of the search-control problem. MAVEN can be used to annotate Adam's moves, and see how often Adam's move proved to be better than any of the ones MAVEN generated. This analysis showed that Adam's move was better only 5 times in the 14 games, yielding a total of 15 points. Of course, Adam will not necessarily find the best move in every case, so this is a lower bound on MAVEN's error rate. A likely upper bound is 10 errors in 14 games yielding a total of 30 points (2.1 points per game), on the grounds that Adam will come up with the best move in at least 50 percent of his plays.

The post-mortem analysis indicates that MAVEN made errors that averaged a total of 6 points per game, with easily-fixed bugs accounting for over half of the total. The current version of MAVEN likely has an error rate of 3 points per game, a playing level that is far beyond anything a human player could hope to achieve. An advantage of 10 points per game corresponds to a winning percentage of about 54 percent. Thus, a perfect human player could only win 51 percent of the games against MAVEN.

10.3. Summary

A few years ago, a demo version of MAVEN (without simulations) was made available to the Scrabble community. Numerous top players used this program for training. No player reported being able to win more than 48 percent of his games. Given that MAVEN has made significant advances in the interim, the gap between MAVEN and the top human players has grown substantially.

An important believer in MAVEN's analysis is Joe Edley, the editor of the US National Scrabble Association's newspaper. MAVEN's analysis is so compelling that Joe Edley decreed that the NSA would only publish annotations of games that have been checked using MAVEN simulations.

11. Making MAVEN stronger

The biggest challenge in the world of Scrabble AI is met, because no one seriously disputes that MAVEN is better than any human. Strictly speaking, this has been demonstrated only within North America. A different dictionary is used in the United Kingdom, and yet another for the rest of the English-speaking world. And, of course, there are other languages. However, adapting MAVEN to a new dictionary is not a challenge. This has already been done for UK English, International English, French, Dutch, and German. There is no doubt in my mind that MAVEN is superhuman in every language. No human can compete with this level of consistency. Anyone who does not agree should contact me directly to arrange a challenge match.

The real challenge is whether MAVEN can be improved upon. There are several avenues to explore, but the total improvement will be only a few points per game.

The first avenue is to make the simulation controller scalable, so that it operates on slower machines, yet takes advantage of faster machines. The actual cutoffs (e.g., 10 moves, 17 iterations before any pruning, 1000 iterations, etc.) were tuned by experience on a Pentium II/300. I now believe that an ideal simulation controller would be able to run with *some* benefit even on a 486/66.

The insight that allows simulations to scale down to slow machines is that the chance that the Nth move is best is a geometric function of N. It follows that simulating 2 moves provides greater value per CPU speed than simulating 3 moves, for example. Simply sliding down the number of moves simulated would allow simulations to scale down to slow machines. There are also performance advantages to be gained in the other cutoffs, as well.

The second avenue, also to scalability, is the ability to reduce errors on faster machines. Increasing the cutoff values could extend our algorithms, but we believe that a neural network controller would have even better results. A neural network could learn the error function as a function of the data from the simulation. A controller that directly sought to balance the CPU usage of the current move against the CPU usage of future moves could possibly prune much more effectively, and provide scalability both up and down.

A third important avenue is to emphasize competitive skill by making moves quickly. A human opponent would like to use the program's clock time. When MAVEN plays quickly then its opponents must hurry to finish their moves before their clocks fall. The 10-point per minute penalty for overtime is not really a large penalty, but *fear* of overtime is. The errors induced by hurrying are often far larger than 10 points. Simulations that last for 2 minutes are of questionable value if that time allows the opponent to improve on his decisions. So another goal of a scalable controller could be to use faster CPU speed for moving faster.

The fourth avenue is to improve on a modeling fallacy. MAVEN assumes the opponent's rack is drawn from a uniform tile distribution. But we can infer a more accurate distribution of tiles from the opponent's last play. For instance, assume the opponent's opening play is BARD (8G, 14). Did he keep an O? No, certainly not, because then he would have played BOARD (8D, 22). Did he keep an E? (No! BARED.) Did he keep an I? (No! BRAID.) Did he keep a U? (Probably not, since DRUB would be better than BARD, but he would play BARD if he held another A also.) We can go on like this through all the tiles, and combinations of tiles, to come up with a list of racks that the opponent could have held. This process is called "inferring" the opponent's rack.

Inferences change the distribution of tiles that the opponent could have kept, which biases the future distribution. Now we have indicated how such inferences might be computed, it is easy to see how they might be used. We do not know how large the effect would be. Manual experiments suggest that the distribution of opponent's tiles is unimportant, but it would be worthwhile to test this hypothesis.

A fifth avenue deals with a simple inference, viz. the distribution of tiles in pre-endgame situations. We have collected probability distributions showing the chance that, for example, a V is in the bag given that it is one of the 8 tiles unseen in a pre-endgame. This information gives MAVEN a way to distinguish between two pre-endgame moves that win an equal number of endgames; we can weight the endgames by the chances of the tiles being in the bag.

A sixth avenue is reducing the expense of move generation. It is possible to make the simulation faster by exploiting the fact that the opponent's rack is duplicated on the first iteration. Since the boards being simulated have only one move changed between them, and the opponent's rack is always the same, it is possible to "share" move generation expense across candidate moves. This has the potential to cut simulation time by about 30 percent.

A seventh avenue reconsiders the speed of the simulation. They can be much faster if we use approximate rules for generating racks that contain blanks. Because of the very high value of a blank, there are only four things you can do with one: play a bingo, play the Q using the blank as a U, use a bunch of heavy tiles to hit a triple-word square, or keep the blank for next turn. All four possibilities can be handled using special move generators at a total cost not much larger than a normal move generator for a non-blank rack. This would almost double simulation throughput.

Since fishing may be MAVEN's biggest weakness, research in this direction constitutes the eighth avenue. A "fish" is a try at drawing the one or two tiles needed to convert the rack leave into a bingo. For example, racks like AEINST can produce a 7-letter bingo on 90 percent of the draws from the bag, so it can be very lucrative to fish in the right circumstance. All of the moves that MAVEN overlooked in its MAVEN against Adam Logan match were fishes. In the absence of simulation we cannot add a generator for fishing plays, since there is no way to compare such plays against regular moves. But simulation rationalizes the comparison of moves that were generated by incomparable generators.

Recent experience showed the ninth avenue. We found some cool improvements in the pre-endgame, thanks to a technical advance. The technical advance is an endgame evaluator that does not play out the position to the end. In the match versus Adam Logan, MAVEN "evaluated" endgames by playing them out using the general evaluator. The resulting estimate had a 13-point standard error. Since it was sufficient for simulation purposes, it illustrates very well the value of playing "equally badly for all variations". Our new estimator produces an estimate of the score using only 3 move generations. It is slightly slower than playing out the game using the normal move generator, but the estimate has just a 5-point standard error.

The new evaluator improves the accuracy of pre-endgame simulations, and also enhances exhaustive pre-endgame analysis of bag-emptying moves. It can evaluate any move that empties the bag using the equivalent of just 3 move generations. This makes it feasible to analyze exhaustively the 330 endgames that could arise in a PEG-4, for instance.

The new evaluator also improves the winning chance estimation used in pre-endgames. Prior to this function the simulator used 1.0 for a win, 0.5 for a tie and 0.0 for a loss. In view of the 13-point standard error the reality is hardly as sharp as the estimates. The new function knows its error function, which allows MAVEN to estimate the winning percentage as a function of the projected point differential. For instance, if MAVEN estimates that it will win the endgame by 8 points, then that game will be won with probability 0.87. By making the winning percentage continuous we make it much more robust.

Finally, the tenth avenue, it may be possible to bias middle-game simulations to encourage winning. For instance, once MAVEN had a choice between a conservative move that protected its lead and an aggressive move. MAVEN preferred the aggressive move by a tiny amount, and Adam Logan responded by playing a monster bingo off of it. Adam was certain that there was little to recommend the aggressive play. It may be possible to use a winning percentage model by estimating the winning percentage of simulation endpoints. In fact, we have developed a neural network using temporal differences that estimates winning percentage. This is worth pursuing, but there are caveats with respect to "hopeless" positions—MAVEN still has to play well (i.e., optimizing the point differential) because point differential is used to break ties in tournament standing.

12. Conclusions

MAVEN is a good example of the "fundamental engineering" approach to constructing intelligent behavior. Design, testing, and quality control contributed more to the quality of MAVEN than any grand concepts of artificial intelligence. We will not hesitate to adopt the same approach to new projects though, of course, one cannot say in advance to what extent it will be successful.

Most of the effort on MAVEN was invested in the education of the program's author, not in the education of the program. The real problems were always conceptual; once the author understood what had to be done the solution was straightforward. But sometimes years would elapse without any significant advances while the author awaited enlightenment. Advice was always in plentiful supply. There were numerous examples to show what MAVEN was doing wrong, and plenty of theories as to what could be done improve the situation. But most proposed solutions were cosmetic in nature, hiding the deeper, conceptual understanding needed to address the problem properly.

The experience has taught me the value of following a sound engineering plan that accounts for known factors, and then following through on that plan. Not a bad lesson.

Appendix A

The following game is in the author's opinion the best Scrabble board game ever played in a tournament or match. The game is the 12th game in the AAAI-98 exhibition match between MAVEN and Adam Logan. After losing three of the first four games, MAVEN had come back strongly to take a 7 to 4 lead. In total, there were 14 games scheduled. The game has been published previously in the *Scrabble Players News*.[2]

Player	Rack	Move	Points	Total
MAVEN:	ACNTVYZ	CAVY 8F	24	24

The alternative is ZANY for 32, but the CVT rack leave is poor. Much better is 24 points with an NTZ leave. As to placement, a better choice than MAVEN's is probably CAVY 8G. This version of MAVEN was not ideal at first-turn placement, for inexcusable internal

[2] The editors thank the *Scrabble Players News* editor Joe Edley for providing permission to reprint the analyses. Some editing changes have been made.

reasons. Fortunately this is not a significant skill factor compared to scoring and keeping good tiles. MAVEN is almost ideal at those skill factors.

Logan:	EGLNORY	YEARLONG G6	66	66

Adam finds the only bingo. Humans specifically practice finding such plays, since they score a 50-point bonus.

MAVEN:	ADNNOTZ	DOZY 6D	37	61

It is DOZY (6D, 37, ANNT) vs. AZLON (10E, 34, NTD) or ZOON (11E, 26, ADNT). DOZY's extra points and retention of a vowel win despite duplicate Ns.

Logan:	ADEFOTV	OFT H13	21	87

Adam's choice is best. He also has VOTED (5A, 27, AF), OVA (H13, 21, DEFT), FOVEAL (10B, 22, DT), and ADVENT (12C, 22, FO). Adam did not think long, and since the choices are so close it does not pay to think long!

MAVEN:	AENNNOT	NEON 5B	15	76

NEON (5B, 15, ANT) edges ANON (5B, 15, ENT). I am not sure why, but clearly ENT and ANT are both good rack leaves, and there must be some benefit to avoiding a FANON hook in the A column. It may also be that ANON's vowel-consonant-vowel-consonant pattern is easier to overlap than NEON.

Logan:	ACDEEIV	DEVIANCE 12B	96	183

Adam finds the only bingo.

MAVEN:	AHINRTU	HURT 4A	34	110

HUNT would usually surpass HURT, because R is better than N, but here there are three Ns already on the board versus one R. It is important to note that MAVEN did not choose HUNT for this reason; MAVEN chose HUNT because in 1000 iterations of simulation it found that HUNT scored more points than HURT. Our reason (that three Ns have been played versus one R) is an interpretation of that same body of data.

Logan:	DDEEMMN	EMENDED C7	26	209

EMENDED is a good play, following sound principles: score points, undouble letters. Simulations give a 2-point edge to MEM (13A, 25, EDDN), however. Possibly the 8A-8D spot weighs against EMENDED, plus keeping an E is a valuable benefit for MEM. These advantages outweigh the extra point and duplicated Ds.

MAVEN:	ABEINNP	IAMB 8A	33	143

IAMB is really the only play, doubled Ns notwithstanding.

Logan:	AILMTTU	MATH A1	27	236

MATH (A1, 27, ILTU) is best, with UTA (3A, 20, ILMT) second. The advantage of MATH over UTA is its 7 extra points, but the disadvantage is keeping a U. These almost wash, with an edge to MATH.

MAVEN:	EFGNNPS	FEIGN E10	18	161

FEIGN is the only good move. FENS (J9, 24, GNP) is higher scoring, but FEIGN keeps better tiles; NPS easily makes up the scoring deficit plus a lot more on top.

Logan:	AILORTU	TUTORIAL 15H	77	313

Adam finds the only bingo. (Actually, TUTORIAL also plays at 15F, but scores only 59 there.)

MAVEN:	?ABNOPS	BOS J10	26	187

See Fig. A.1. MAVEN made a great draw from the bag, and then made one of the most difficult plays of the game. MAVEN has no bingos, and has to choose how to make one. Playing off the B and P is indicated, so plays like BAP or BOP (7I, 20) come to mind. But MAVEN finds 2 stronger, and surprising, alternatives: BOS (J10, 26, ?ANP) and BOPS (J9, 25, ?AN). These plays score a few extra points as compensation for playing the S, and they open the K-column for bingo-making. We would have thought that BOPS would win out, but BOS is better. BOS does show a higher point differential, but that is not why it is better. It is better because the chance of getting a big bingo is higher due to the creation of a spot where a bingo can hit two double word squares. The great majority of human masters would reject BOS. BOP is the first play that comes to mind in that situation and

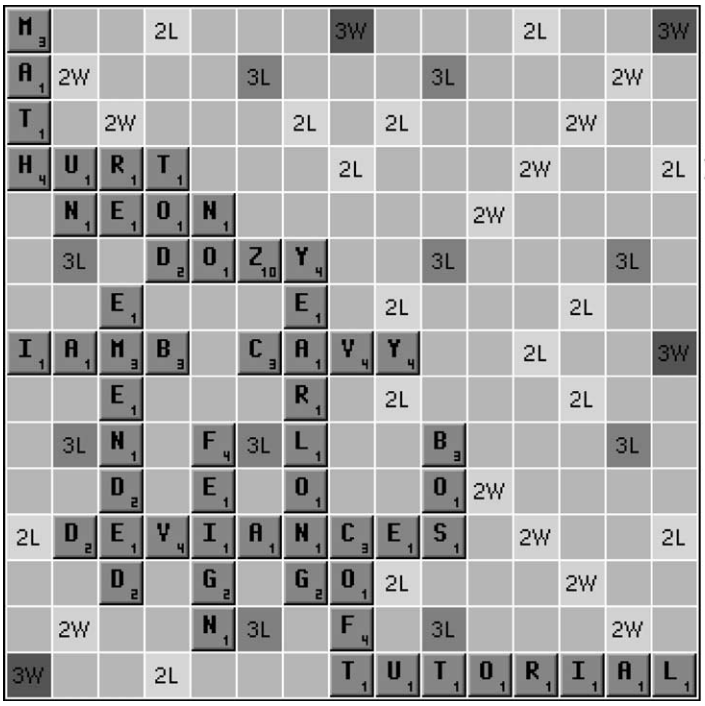

Fig. A.1. MAVEN versus Adam Logan (exhibition match, AAAI-98). MAVEN plays BOS (J10, 26).

nearly every master would play it. BOS is a fantastic play, and yet, there are 2 plays still to come in this game that are even more difficult.

Logan: IILPRSU PILIS 15A 34 347

PILIS, PULIS, PILUS, and PURIS are all good. Adam's choice is best because there are only 2 Us left, and Adam does not want to risk getting a bad Q. When you lead the game you have to guard against extreme outcomes.

MAVEN: ?AKNPRS SPANKER K5 105 292

This is the only bingo, and a big boost to MAVEN's chances. In my role as MAVEN's operator I saw SPANKER but I was not sure it was legal, so I was sitting on the edge of my seat. Being down 160 points is depressing. Worse than depressing; it is nearly impossible to come back from that far behind. The National Championship Tournament gives a prize to the greatest comeback, and in this 31-round, 400-player event there is often only one game that features such a comeback.

Logan: EEEORUS OE B1 12 359

Adam plays the best move again. This play scores well, as his highest-scoring play is just 13 points (ERE L6). OE dumps vowels while keeping all his consonants (an edge over ERE). It also keeps the U as "Q-insurance", an edge over MOUE (1A,7,EERS). And it blocks a bingo line. Overall good value, and a good example of how to make something positive happen on every rack.

MAVEN: ?HJTTWW JAW 7J 13 305

MAVEN's draw is bad overall, but at least there is hope if MAVEN can get rid of the bad letters. Any play that dumps two of the big tiles is worth considering, with JAW, WORTH (11I, 16, ?JW), and WAW (B7, 19, ?JHTT) as leading contenders. JAW wins because the WH and TH are bearable combinations, and the TT is not too bad either.

Many players would exchange this rack, but MAVEN did not consider doing so. I do not know how exchanging (keeping ?HT, presumably) would fare, but I suspect it would not do well; there are few good tiles remaining, and drawing a Q is a real risk.

Logan: AEEGRSU GREASE M3 31 390

Simulations show AGER (L9, 24, ESU) as 3 points superior to GREASE, but I suspect that GREASE does at least as good a job of winning the game, since it takes away -S bingos off of JAW. It also pays to score extra points, which provide a cushion if MAVEN bingos. And it pays to turn over tiles, which gives MAVEN fewer turns to come back.

MAVEN: ?HRTTWX AX 6M 25 330

MAVEN's move is brilliant. Who would pick AX over GOX (13G, 36)? Would you sacrifice 11 points, while at the same time creating a huge hook on the O-column for an AX-E play? And do so when there are 2 Es unseen among only 13 tiles and you do not have an E and you are only turning over one tile to draw one? It seems crazy, but here is the point: among the unseen tiles (AAEEIIIILOQUU) are only 2 consonants, and one of them is the Q, which severely restricts the moves that can be made on the O-column. If Adam has

EQUAL then MAVEN is dead, of course, but otherwise it is hard to get a decent score on the O-column. In effect, MAVEN is getting a free shot at a big O-column play.

AX is at least 10 points better than any other move, and gives MAVEN about a 20 percent chance of winning the game. The best alternative is HAW (B7, 19). GOX is well back.

Logan: EIIILQU LEI O5 13 403

Adam sensibly blocks, and this is the best play. The unseen tiles from Adam's perspective are ?AAEHIORTTUW, so Adam's vowelitis stands a good chance of being cured by the draw.

MAVEN: ?AHRTTW WE 9B 10 340

Again a problem move, and again MAVEN finds the best play. In fact, it is the only play that offers real winning chances. MAVEN calculates that it will win if it draws a U, with the unseen tiles AEIIIOQUU. There may also be occasional wins when Adam is stuck with the Q. This move requires fantastic depth of calculation. What will MAVEN do if it draws a U?

Logan: AIIIOQU QUAI J2 35 438

Adam's natural play wins unless there is an E in the bag. AQUA (N12, 26), QUAIL (O11, 15), QUAI (M12, 26), and QUA (N13, 24) also win unless there is an E in the bag, but with much, much lower point differential than QUAI because these plays do not block bingos through the G in GREASE.

There is no better play. If an E is in the bag then Adam is lost.

MAVEN: ?AHRTTU MOUTHPART 1A $92 + 8$ 440

See Fig. 1. MAVEN was fishing for this bingo when it played WE last turn. With this play MAVEN steals the game on the last move. Adam, of course, was stunned, as it seemed that there were no places for bingos left on this board. If I had not felt so bad for Adam, who played magnificently, I would have jumped and cheered.

This game put MAVEN up by 8 games to 4, so winning the match was no longer in doubt.

References

[1] B. Abramson, Expected-outcome: A general model of static evaluation, IEEE Transactions on Pattern Analysis and Machine Intelligence 12 (1990) 182–193.

[2] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.

[3] A.W. Appel, G.J. Jacobson, The world's fastest Scrabble program, Comm. ACM 31 (5) (1988) 572–578, 585.

[4] H.J. Berliner, The B^* tree search algorithm: A best-first proof procedure, Artificial Intelligence 12 (1) (1979) 23–40.

[5] D. Billings, A. Davidson, J. Schaeffer, D. Szafron, The challenge of poker, Artificial Intelligence 134 (2002) 201–240 (this issue).

[6] M. Buro, From simple features to sophisticated evaluation functions, in: H.J. van den Herik, H. Iida (Eds.), Proc. First International Conference on Computers and Games (CG-98), Lecture Notes in Computer Science, Vol. 1558, Springer, Heidelberg, 1998, pp. 126–145.

[7] J. Cherry, Web site, www.doe.carleton.ca/~jac/scrab.html.
[8] M. Ginsberg, GIB: Steps toward an expert-level bridge-playing program, in: Proc. IJCAI-99, Stockholm, Sweden, 1999, pp. 584–589.
[9] S.A. Gordon, A comparison between probabilistic search and weighted heuristics in a game with incomplete information, in: AAAI Fall 1993 Symposium on Games: Playing and Learning, AAAI Press Technical Report FS9302, Menlo Park, CA, 1993.
[10] S.A. Gordon, A faster Scrabble move generation algorithm, Software Practice and Experience 24 (2) (1994) 219–232.
[11] J. Homan, Web site, www.cygcyb.com/catalog/software.html#CrossWise.
[12] Official Scrabble Players Dictionary, Merriam-Webster, 1995.
[13] S.C. Shapiro, Scrabble crossword game-playing programs, SIGART Newsletter 80 (1982). Reprinted in: M.A. Bramer (Ed.), Computer Game-Playing: Theory and Practice, Ellis Horwood Ltd, Chichester, UK, 1983, pp. 221–228.
[14] R. Sutton, Learning to predict by the method of temporal differences, Machine Learning 3 (1988) 9–44.
[15] G. Tesauro, Temporal difference learning and TD-GAMMON, Comm. ACM 38 (3) (1995) 58–68.
[16] G. Tesauro, G.R. Galperin, On-line policy improvement using Monte Carlo search, in: Neural Information Processing Systems Conference, Denver, CO, 1996, Unpublished Report.
[17] P.J. Turcan, A competitive Scrabble program, SIGART Newsletter 80 (1982). Reprinted in: M.A. Bramer (Ed.), Computer Game-Playing: Theory and Practice, Ellis Horwood Ltd, Chichester, UK, 1983, pp. 209–220.
[18] I. Uljee, Letters beyond numbers, in: H.J. van den Herik, L.V. Allis (Eds.), Heuristic Programming in Artificial Intelligence 3, The Third Computer Olympiad, Ellis Horwood, Chichester, UK, 1992, pp. 63–66.
[19] L. Yedwab, On playing well in a sum of games, Technical Report MIT/LCS/TR-348, Massachusetts Institute of Technology, Cambridge, MA, 1985.