## Mastering Scrabble

*Brian Sheppard, Hasbro*

If I had to identify one factor that enabled recent advances in game AI, it would be that programmers have a large arsenal of methods to adapt to their needs. An example is in order, so I'll describe Maven, my Scrabble AI.

The most important skill in Scrabble is the ability to find high-scoring plays. So Maven includes an exhaustive move generator, which produces each legal move, along with a score and a list of the tiles remaining on the rack. Thus, Maven achieves this critical skill using full-width search.

A note about word lists is in order. Nowadays, I can get a computerized word list from the National Scrabble Association (NSA). But this is a recent advance. The development of Maven included several man-months of data entry. I will describe the process, because every game AI project involves similar drudge work.

I bought a copy of the *Official Scrabble Players Dictionary* (OSPD) from my local bookstore and started typing the words. I couldn't bear to type every single word, so I invented a "little language" of the form "v assert -or -ors," which I postprocessed into "ASSERT ASSERTED ASSERTING ASSERTS ASSERTOR ASSERTORS." This trick cut the typing in half.

Then came a verification stage, where I statistically profiled the word list to determine its error rate. My initial data entry omitted 2% of the words and misspelled 1%. There ensued a proofreading chore to correct these errors. I then validated key lists of words such as the two-letter words, JQXZ words, and so forth, against printed lists from the NSA. This step ensured that any remaining errors were unlikely to matter, because they would occur among low frequency words. Eight errors remained (out of 95,000 words), which I found several years later when I compared my list against the list of another person who had undertaken the same task.

Then I decided to add the "long words" to my list, because the OSPD only contains main entries up to eight letters long. This process involved scanning *Webster's 10th Collegiate Dictionary*, proofreading, profiling, cross-checking, and so on. It was a huge task, but very typical of game AI development.

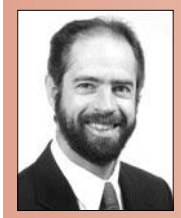
### Evaluation functions

To evaluate a position properly you have to model the factors that are important to the domain. Maven's development is interesting because there wasn't a well-developed positional model of Scrabble at the time. At least, there was none that I could find. Of course, experts used certain precepts in choosing plays, but I didn't know any expert players, and I didn't have access to any of their writings. I had to model the domain "from first principles." I might have been lucky in this regard, because almost every precept held by experts prior to the advent of Maven has been proven false. Since modeling is a messy task that nearly every game AI developer has to do sometime, I will walk you through the steps I followed.

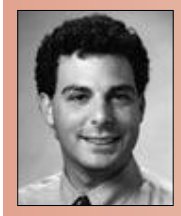
I reasoned that a move changes three things: the score, the tiles held by the player, and the position. So, in gross terms, I have the equation Evaluation = Score + Rack + Position. This is a good start because my move generator already computes the score and the tiles left on the rack (the *rack leave*). I was confident that I could build an evaluator for rack leaves, because I had a trick up my sleeve. But what should I do about this annoying *Position* term? Did I have to develop a complicated (and slow) pattern-matching algorithm for evaluating the myriad possible changes in position that could occur as a result of a move?

Upon reflection, I decided that the Position term was usually very close to 0, so I could ignore it (with one exception). The reason is that the board is a resource that affects both players, so any openings for high scores tend to cancel out. The opponent's advantage is that he moves first, so a hot spot is more likely to benefit him. Maybe you should penalize hot spots by a small amount, but maybe not. You have to consider that the opponent is a weaker player than Maven, so hot spots disproportionately benefit Maven. The only


**Michael Buro** is a scientist at the NEC Research Institute. He wrote Logistello, the world champion-class Othello program. He earned a diploma in computer science from the Technical University of Aachen and a PhD in machine learning in games from the University of Paderborn, Germany. He is a member of the AAAI and the ICCA. Contact him at NEC Research Inst., 4 Independence Way, Princeton, NJ 08540; mic@research.nj.nec.com; www.neci.nj.nec.com/homepages/mic/mic.html.



**Richard E. Korf** is a professor of computer science at the University of California, Los Angeles. He received his BS from MIT, and his MS and PhD from Carnegie-Mellon University, all in computer science. His research is in the areas of problem solving, planning, and heuristic search in artificial intelligence. He received an NSF Presidential Young Investigator Award and is a fellow of the AAAI. Contact him at the Computer Science Dept., UCLA, Los Angeles, CA 90095; korf@cs.ucla.edu; www.cs.ucla.edu/~korf.



**Michael Littman** is an assistant professor of computer science at Duke University. His main interests are in machine learning, examining algorithms for decision-making under uncertainty, and statistical natural-language processing. He received his PhD from Brown University, and his master's and bachelor's degrees from Yale University. His crossword work was chosen for the best paper award at AAAI 99. Contact him at Box 90129, Duke Univ., Durham, NC 27708-0129; mlittman@cs.duke.edu; www.cs.duke.edu/mlittman/.

**Brian Sheppard** is director of technology at Hasbro Interactive. His research interests include heuristic search and multiplayer network games. He received a BA in mathematics from Harvard College. He is the author of the Scrabble program Maven, one of the first programs to achieve championship caliber in any game. Contact him at Hasbro Interactive, 50 Dunham Rd., Beverly, MA 01915; bsheppard@hasbro.com.

**Jonathan Schaeffer** is a professor in the Department of Computing Science at the University of Alberta. His research interests include heuristic search and parallel-computing environments. He received a BSc from the University of Toronto and an M.Math and a PhD from the University of Waterloo. He is a member of the IEEE, ACM, AAAI, and ICCA. Contact him at the Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta, Canada T6G 2H1; jonathan@cs.ualberta.ca; www.cs.ualberta.ca/~jonathan.

## Coming Next Issue

**Special Millenium Issue: The Experts Expound on AI's Greatest Trends and Controversies**

exception is that direct access to triple-word squares is a factor that should be evaluated, because such a spot is high scoring, easy to use, and unlikely to be left around for the next turn. Still, calculation showed that direct access to triple-word square is only worth a few points (usually under three).

As for Rack evaluation, I whipped out my trick: I would use self-play to generate games and "feed back" the impact of holding specific tiles into the evaluation function. This method worked well in Scrabble. In fact, the evaluation function improved from zero initial knowledge to beyond the level of the human champions of the day, while using only a single day of training.

Self-play combined with feedback is a fundamental method employed in most competitive programs. It works in other nondeterministic games, and in deterministic games, too, if combined with tricks that ensure exploration.

But self-play can only take you so far. Self-play brings a program into greater internal consistency, but if a fundamental computational process is missing, you won't discover it through self-play. Actual comparison against human experts is required to diagnose such deficiencies.

The most direct form of comparison is competition. Competition measures skill using the same standards that humans use. You can also participate in post-mortem discussions that provide guidance about where to invest additional effort.

Maven's competitive games showed that Maven was championship caliber. They showed that I could stop worrying about things that I always believed were unimportant, but experts told me were huge. For example, was it important that Maven didn't vary its play as a function of the score? Was it important to consider the skill of the opponent? Was it important to *block* or *open* the board? Well, maybe it was important, but it was insignificant compared to Maven's skill in scoring points and keeping good tiles.

Other forms of comparison are indirect. For example, I compared Maven's moves against moves made by experts and against annotations written by experts. I published annotations "written" by Maven, to elicit feedback from experts. All of these things helped somehow, if only to provide reassurance to the author.

### The endgame

I also learned about the importance of the endgame, which is the phase of the game where there are no tiles in the bag, and so the game becomes deterministic. Maven made serious endgame errors by failing to block a good spot for the opponent or failing to leave itself a way to play off all of its tiles.

Achieving good endgame play required that I scrap Maven's whole approach, because it is impossible to build a static evaluator that evaluates an endgame position using only one ply of lookahead. Clearly, the searching techniques of perfect-information games needed to be brought to bear, but the leading candidate (full-width alpha-beta) had serious shortcomings for this application.

For one thing, alpha-beta requires almost best-first move ordering for good search efficiency, whereas my move generator produces moves in order of rows of the board. The prospect of ordering moves after generating them was unattractive, because there are an average of 200 moves at the start of an endgame, and there could be many, many more if the side-to-move held two blanks. Also, move generation was comparatively slow (about 1 second on the hardware of the day), which limited us to about 120 nodes per search. Obviously, you can't search a tree whose branching factor is 200 at the root if you have only 120 nodes to work with. As if that weren't enough, there are vitally important endgames where the one side is "stuck with the Q" and cannot play out. In such cases, the best strategy may be to play out "one tile at a time." Such endgames can last 14 ply, with several hundred legal moves per ply, and the highest-scoring moves are almost always bad!

What was needed was a search algorithm that was naturally full-width, variable-depth, and appropriately selective (that is, able to distribute 120 nodes of search so as to explore a potentially huge space). Fortunately, I have read nearly every paper about search algorithms ever written, so Berliner's B* algorithm was familiar to me. The technique of applying B* is very interesting and novel, but alas, this is not the right forum for describing it, as it is highly domain-specific. To continue our topic, every game programmer needs to be familiar with the literature, because there are many general-purpose methods available. The programmer must also accurately judge the applicability of each method to his domain. Finally, general methods usually require domain-specific adaptation to reach their full potential.

### Statistical lookahead

Finally, is developing one novel technique too much to ask of a game programmer? Actually, most successful game programmers have contributed a novel method. It seems that one cannot conquer a new game simply by applying previously known techniques. So I tentatively put forward that Maven was the first to use the technique of statistical lookahead for playing games.

Statistical lookahead is now recognized as a general method, having been applied (and independently discovered) by pioneers in games such as backgammon, bridge, and poker. The technique might be new to readers, so I will take a moment to describe it.

The idea is to evaluate moves by "playing them out" at high speed. The move with the highest average outcome is selected. During the process, you can gain speed by pruning moves that have proven to be inferior. This technique has many domain-specific details, such as the question of which alternatives are considered, how the game is played out, how to prune moves, how to model opponent's behavior, and so forth. Virtually any domain with randomness (such as backgammon) or hidden information (bridge) or both (Scrabble and poker) can benefit from statistical lookahead. I think this technique will produce a treasure-trove of research results (and practical results) because of its adaptability.

So, successful game AI results from combining many general methods. Maven would not be what it is without full-width search, evaluation functions, self-play, feedback, competition, indirect comparisons, knowledge engineering, perfect-information search techniques, and statistical lookahead. Plus a lot of luck.