Al Invades Go Territory

See Also

- Computers Still Dominate Chess
- Unorthodox Chess From an Odd Mind
- Awarding the Brains Behind AI
- Chess: Man vs. Machine Plays Out
- Chess Conspiracy Unconvincing
- Rage Against the (Chess) Machine

By Brendan Borrell Also by this reporter

02:00 AM Sep. 19, 2006

Chess was once the pinnacle of geekdom, but then the artificial intelligence geeks got too smart for chess and turned to Go. Why Go?
The game is more than a thousand years older than chess, and the number of possible moves in a game of Go exceeds the number of atoms in the universe. But most importantly, computer programs haven't yet beaten the human masters of Go.

Around the world, dedicated coders trade secrets on the computer go mailing list and compete monthly during the KGS Computer Go Tournaments.

In the past year, a new strategy implemented by computer scientist Rémi Coulom at the Université de Lille in France, has revolutionized the way these programmers have approached the problem. Coulom's program Crazy Stone won a gold medal at the 2006 Computer Olympiad in Torino, Italy. Recently, Coulom spoke to Wired News to explain some of the challenges of Go and what makes Crazy Stone work so well.

Wired News: What makes programming go so much tougher than chess?

Rémi Coulom: In Go, you don't capture pieces, and so it's very difficult to say that black is ahead or white is ahead just by looking at the board. In order to survive, a group of stones needs to surround two "eyes" -- empty areas that can't be invaded by the opponent.

On a 19-by-19(-line) board, you'll have plenty of stones whose life or death status is undecided, and this is extremely difficult to analyze statically. This is different from the situation with chess or (checkers), where you can look at the board and say, "I have one more pawn than you."

WN: What are "Monte Carlo" methods and how do they apply to Go?

Coulom: Monte Carlo methods are named after a quarter of Monaco that's famous for its casinos. In the case of Go, the basic idea goes like this: To evaluate a potential move, you simulate thousands of random games. And if black tends to win more often than white, then you know that move is favorable to black.

WN: With 250 moves in a typical game, that must take a lot of computational power.

Coulom: The version of Crazy Stone in the Torino Olympiad ran on a four-CPU machine -- two dual-core AMD Opterons at 2.2 GHz -- and did about 50,000 random games per second. Unlike traditional algorithms, the Monte Carlo approach is extremely easy to parallelize, so it can take advantage of the multi-core architecture of the new generation of processors.

WN: Crazy Stone was not the first program to use Monte Carlo methods, but it was successful enough that it started a trend among Go programmers. What was your innovation?

Coulom: Because you can't sample every possible random game, the Monte Carlo algorithm can easily fail to find the best moves. For instance, if most of the random games resulting from a certain move are losses, but one is a guaranteed win, the basic algorithm would take the average of those games and still evaluate it as a bad position.

Crazy Stone is clever enough to avoid this problem. When it notices that one sequence of moves looks better than the others, it tends to play it more often in the random games.

WN: Why have people like Nick Wedd, the moderator of the monthly KGS tournaments, complained that
watching games played by Monte Carlo programs can be boring?

**Coulom:** Monte Carlo programs maximize the probability of winning, not the margin that they win by. When they're very far ahead of the opponent, then they'll always play a safe move, which might look boring compared to more aggressive alternatives. It may be boring to watch, but it's more efficient in winning games.

**WN:** I've heard that a lot of the top Go programs are written by top Go players. What's your experience with the game?

**Coulom:** Before I started to write my first Go program, I decided I was going to play well enough to beat the other programs out there. But I don't think being a strong player is important to write a strong program. When I was still programming chess, this was obvious: my program was immensely stronger than me.

Some of the programs out there do use these set sequences of play, called *joseki*, but I avoid hard-coding this knowledge. I see some programs blunder because they blindly apply a hard-coded pattern.