Kurz vorgestellt: Leela

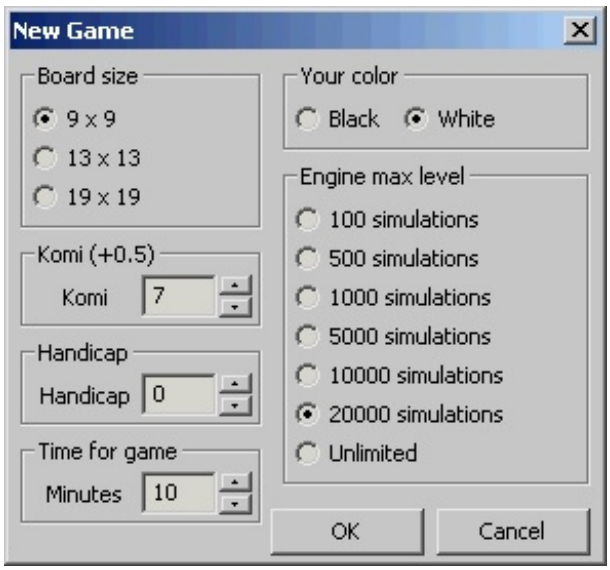
Leela heißt das Go-Programm von Gian-Carlo Pascutto, und das beste daran: es ist einfach. Nicht einfach zu schlagen natürlich, im Gegenteil, es spielt brutal stark, aber einfach zu bedienen. Und es eignet sich auch noch als Analysetool. Schluss also mit aufwendigen Konfigurations-Orgien, um MoGo unter irgendeiner Windows-GUI zum Laufen zu bringen, mit Leela gehts leichter!

Die Go-Szene bewegt sich. Nach jahrelanger Stagnation brachten die UCT-Programme frischen Wind und machen momentan nicht nur die Computer-Go-Meisterschaften unter sich aus, sondern stoßen erstmals auch in den Bereich menschlicher Dan-Spieler vor. Doch am Computer-Go-Fan geht diese Entwicklung weitgehend vorbei, denn außer dem Olympiasieger MoGo gibt es kein einziges Programm, das man irgendwo herunterladen oder wenigstens kaufen könnte. MoGo selbst ist zwar gratis, spielt aber nur in der Linux-Version mit voller Kraft, nicht aber in der Windows-Version, und es ist eine reine Engine, die sich zudem nur sehr kompliziert in vorhandene GUIs einbinden lässt und zudem nicht unter allen funktioniert. Den Olympia-Zweiten Crazy Stones gibt es gar nicht mehr öffentlich, kurz gesagt: von der neuen Herrlichkeit profitierten bisher nur wenige Go-Freunde, und auch die nur unter Schmerzen.

Doch jetzt gibt es Leela! Gian-Carlo Pascutto, Computerschach-Freunden als Autor der Engine DeepSjeng bekannt, hat sich erbarmt und nicht nur eine extrem starke UCT-Engine geschrieben, sondern auch noch eine kinderleicht zu installierende und bedienende, absolut einsteigertaugliche Windows-Oberfläche dazu, die neben reinem Spielspaß auch noch etliche Analyse-Funktionen bietet.

Leela

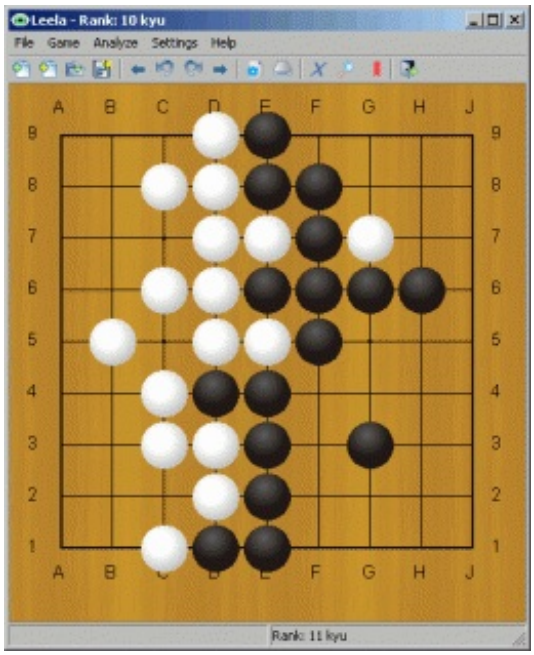
Beim Programmstart präsentiert Leela ein frei in der Größe veränderbares Spielfenster, in dem auf einem 9x9-Brett der Spieler fünf Vorgabesteine hat. Das ist auch für ein Programm, das 9x9 auf Hoch-Dan-Niveau spielt, ein bisschen zu optimistisch, denn natürlich bekommt auch der weltbeste Go-Spieler da nichts mehr ans Leben. Es handelt sich um den Meisterschafts-Modus; per "rated game"-Button kann man sein Rating (am Anfang 30. Kyu) verbessern und bekommt dabei immer weniger Vorgabesteine. Natürlich ist man nicht auf gewertete Spiele beschränkt, sondern kann auch mit frei konfigurierbaren Parametern ein Spielchen wagen, also klickerdiklack, neue Partie. Der Benutzer bekommt ein kleines Einstellungs-Fenster:



Neben leichtverständlichen Einstellungen wie Brettgröße, Farbe, Komi, Handicap und Zeit pro Partie taucht auch eine Möglichkeit auf, die Anzahl der Simulationen pro Zug zu wählen. Das ist eine Eigenheit der Technik, auf der die Engine basiert: sie spielt Zufallspartien gegen sich selbst und wählt den Zug, der am besten abschneidet (detaillierter wird das im Abschnitt "So denken UCT-Programme" erklärt). Über die Anzahl der Simulationen steuert man die Genauigkeit der Zugermittlung, letztlich also die Spielstärke. Je mehr Simulationen, desto stärker spielt Leela.

Und sie kann verdammt stark spielen; auf dem kleinen Brett so gut wie ein Profi-Dan, auf 19x19 immerhin noch wie ein an der Schwelle zum Amateur-Dan stehender Kyu. Übersetzt in Schachausdrücke hätten wir also einen 9x9-Großmeister und einen 19x19-FIDE-Meister.

Die GUI sieht sehr übersichtlich und aufgeräumt aus; die wichtigsten Funktionen wie Laden, Speichern, Neues Spiel, Zug vor/zurück, Passen, Ziehen, Aufgeben, Analysemodus und Spiel auszählen warten zusätzlich zum Menü noch in einer Buttonleiste darauf, angeklickt zu werden. Als Format beim Speichern und Laden benutzt Leela den Internet-Standard SGF und kann daher Partien aller gängigen Programme verarbeiten. Per Menübefehl kann man Leela veranlassen, die bisher (wahrscheinlich) eroberten Gebiete und eventuelle Moyos, also besonders große Gebietsanlagen, anzuzeigen und im weiteren Spiel markiert zu lassen.



Leela spielt nicht nur 9x9, sondern auch auf 13x13 und 19x19 Felder großen Brettern

Ein wichtiger Unterschied zu anderen Go-Programmen besteht im Analysemodus. Bei Schachprogrammen seit Urzeiten selbstverständlich, war bislang anscheinend niemand daran besonders interessiert, eines Go-Programmes Meinung zu einer Stellung einzuholen, weshalb praktisch keine Go-Software eine stinknormale Daueranalyse sinnvoll unterstützt. Außer Leela, das sich nach dem Willen ihres Programmierers Gian-Carlo Pascutto immer weiter in Richtung Analysetool entwickeln soll: "Ich habe großes Augenmerk auf eine einfach zu benutzende Oberfläche gelegt, die ideal für ein schnelles Spielchen ist. Es wird in der Zukunft noch viele weitere Funktionen geben, die Analysen unterstützen. Das letzte Update 0.3.8 war ja schon ein Schritt in diese Richtung."

Die Vollversion von Leela kostet 50 Euro; wer das einäugige Mädel erstmal ausprobieren möchte, kann gratis eine

eingeschränkte Version von Gian-Carlos Webseite herunterladen.

Zufällig gut: So denken UCT-Go-Programme

Backgammon- und Schachprogramme spielen stärker als die menschlichen Weltmeister, Go-Programme dagegen bewegten sich lange, lange auf dem Niveau eines talentierten Anfängers, der vielleicht hundert Partien gespielt hat. Die Go-Programme des Jahres 2005 spielten nur unwesentlich stärker als zehn Jahre alte DOS-Veteranen, obwohl in ihnen viele Mannjahre versickerten. Die allgemein vorherrschende Ansicht, auf einem 19x19 Felder großen Go-Brett sei die Anzahl der möglichen Züge einfach zu groß, um mit Alpha-Beta nebst Erweiterungen erfolgreich zu spielen, findet allerdings nicht die Zustimmung von Schachprogrammierern, die sich dem Go zugewandt haben.

Reines Alpha-Beta reduziert den Verzweigungsfaktor von 36 im Schach ihn bereits auf 6, und all die Verfeinerungen drücken ihn auf ca. 2, wobei Erweiterungen der Suche ihn zwar wieder auf 3 bis 4 bringen, aber der Spielstärke-Steigerung dienen. Ein Schachprogramm untersucht also von den durchschnittlich 36 pro Stellung möglichen Zügen nicht mehr als drei bis vier, was für eine Rechentiefe von etwa 18 Halbzüge reicht; einige Varianten nur sieben oder acht, andere dafür über 40 Halbzüge tief. Alpha-Beta würde beim Go den Verzweigungsfaktor schon auf weniger als 19 reduzieren; mit den aus dem Schach bekannten Tricks sogar auf unter zehn. Damit könnte ein Go-Programm ca. zehn Halbzüge tief suchen, was im Schach schon fast für Großmeister-Stärke ausreicht. Das kleine Go-Brett von 9x9 Feldern wäre sogar vergleichbar komplex (oder simpel) wie Schach.

Das Problem liegt woanders: Schach ist ein ungeheuer taktisches Spiel. Großmeister wissen zwar eine Menge mehr über Strategie als Programme, werden jedoch regelmäßig taktisch zu Boden geschlagen. Beim Go gibt es zwar auch Taktik, aber wesentlich langzügiger. Eine Schachpartie ist zuende, wenn der König mattgesetzt wurde, und wenn er angegriffen wird, dann muss er verteidigt werden. Bei Go dagegen kann es auf dem großen Brett an vielen Stellen gleichzeitig brennen, die einzelnen taktischen Kämpfe finden gleichzeitig statt und haben Auswirkungen auf die strategische Gesamtsituation. Wenn eine große Gruppe abhanden kommt, kann es sein, dass an anderen Stellen des Brettes genug Vorteile zusammenkamen, um dies auszugleichen, und in jedem Fall dauert es Dutzende Züge, bis eine Gruppe zusammenhängender Steine wirklich vom Gegner gefangen wird, wobei sie auch noch selten wirklich geschlagen wird, sondern meist bis zum Spielende auf dem Brett verbleibt und durch ihre schiere Existenz für weitere Gefahr sorgt. Die Go-Taktik ist also viel tiefer als Schachtaktik, was zusammen mit einer geringeren Suchtiefe dazu führte, Alpha-Beta als ungeeignet abzutun. Traditionelle Programme zerlegen die Stellung in Sub-Spiele, einzelne Brandherde, die sie auch separat zu löschen versuchen, wobei sie allerdings die Gesamtstrategie völlig aus dem Auge verlieren.

Ein weiteres Problem besteht in der Bewertung: beim Schach ist ein Läufer eben ein Läufer. Er kann unter ungünstigen Umständen auch mal im Wege stehen, dann muss man ihn woanders hinziehen, aber einen Nachteil stellt er praktisch niemals dar. Im Go geht es nicht um einzelne Steine, sondern um die freien Punkte, die sie umschließen. Der Wert eines einzelnen Steins hängt von der Stellung ab, und er kann, wenn er im eigenen freien Gebiet steht, sogar schaden. Was sicheres Gebiet ist und was nicht, kann man auch während des Spielaufbaus nur schwer abschätzen. Dazu kommt der aus der Frühzeit der Schachprogrammierung bekannte Horizont-Effekt: das Programm sieht einen Turmverlust auf sich zukommen und versucht, dieses unerfreuliche Ereignis aus dem begrenzten Bereich seiner Suchtiefe hinauszuschieben. Zum Beispiel, indem es Bauern opfert, denn ein Bauer ist weniger wert als ein Turm. Irgendwann sind die Bauern aber alle und der Turm muss dennoch über die Klinge springen. Durch Sucherweiterungen haben Schachprogramme dieses Problem heute nicht mehr. Beim Go aber gibt es jede Menge Möglichkeiten, ein Unglück über den Suchhorizont zu schieben, zum Beispiel, indem man einen Stein so ins feindliche Territorium setzt, dass der Gegner darauf reagieren muss. Letztlich wird der Gegner dadurch nur stärker, aber das wirkliche Problem hat das Programm sich für den Moment erfolgreich gesundgerechnet.

Den Go-Programmierern kam der Zufall zur Hilfe. Man stelle sich eine Stellung vor, in der eine Seite im Vorteil ist. Zwischen zwei perfekten Spielern würde dieser Vorteil immer zum Gewinn reichen. Zwischen zwei starken, aber nicht perfekten Spielern wird es auch meistens reichen, weil zwar Fehler geschehen, aber etwa gleichverteilt auf beiden Seiten. Aber was, wenn man Zufallszüge spielte? Zufall ist per Definition gleichverteilt, sodass auch hier die Seite mit Vorteil eine größere Gewinn-Wahrscheinlichkeit hätte. Um eine Go-Stellung perfekt zu bewerten, reichte es also, unendliche viele Zufallspartien spielen zu lassen und zu gucken, welche Seite öfter gewinnt. Weil nur selten unendlich viel Zeit zur Verfügung steht, bescheiden sich die Programme mit einer endlichen Anzahl von Zufallspartien und einer nicht perfekten Bewertung.

Die Idee heißt "Monte Carlo" und ist aus der numerischen Optimierung bekannt. Die einfachste Art der Zugermittlung wäre, jeden in einer Stellung möglichen Zug zu erzeugen und mit einer bestimmten Anzahl Zufallspartien auszuspielen. Der Zug mit dem besten Score würde vom Programm gewählt. Das funktioniert, aber es ist nicht effizient. Besser funktioniert es, nicht für jeden Zug dieselbe Anzahl von Zufallspartien spielen zu lassen; ein einfaches Verfahren solcher Skalierung hat bereits Richard Epstein in „Theorie of Gambling“ vorgeschlagen: man probiert einen Zug, so lange er gewinnt, danach den nächsten usw., bis man reihum ist. Das geschieht so lange, bis die zur Verfügung stehende Bedenkzeit aufgebraucht ist., und wählt dann den Zug mit dem besten Score. Das funktioniert viel besser, als alle Züge mit gleicher Wahrscheinlichkeit zu probieren. Es gibt aber noch ein besseres Verfahren, das die aktuell stärksten Go-Programme verwenden: UCT (Upper confidence bounds applied to trees)

Wie bei Epstein erzeugt ein UCT-Programm alle Züge, berechnet die Qualität aber nach einer Formel wie

$$Q = MC + K * \sqrt{\ln(TP) / TM}$$

MC ist die Gewinn-Wahrscheinlichkeit, die sich aus den Monte-Carlo-Zufallspartien ergibt, TP die Gesamtzahl der

Zufallspartien in der Stellung, TM die Anzahl der für diesen Zug gespielten Zufallspartien, und K ein Faktor, der das Verhältnis zwischen Monte-Carlo-Wahrscheinlichkeit und Wichtigkeit des Zuges bestimmt.

UCT wichtet die Gewinn-Rate der Monte-Carlo-Versuche also, indem es nicht allein der Gewinnwahrscheinlichkeit aus den Zufallspartien die Palme zuerkennt, sondern auch die Anzahl der Zufallspartien in der aktuellen Stellung und für den untersuchten Zug in die Kalkulation einbezieht. Untersucht wird immer der Zug mit der besten Qualität. Am Anfang haben alle Züge die gleiche Qualität, und das Programm spielt für einen Zug so lange Zufallspartien, bis seine Qualität unter die der anderen Züge fällt. Das passiert mit den anderen Zügen ebenfalls, einmal reihum, wonach schon eine nach vermutlicher Qualität sortierte Zug-Liste existiert, die wiederum in absteigender Folge durchprobiert wird.

Dadurch fallen offensichtlich schlechte Züge sehr schnell, nach nur wenigen Zufallspartien, aus dem Raster, und das Programm konzentriert seine Bemühungen und seine Rechenkraft auf die guten Züge. Diese Idee verwenden UCT-Programme nicht nur an der Wurzel des UCT-Suchbaums, sondern auch während der Zufallspartien selbst. UCT skaliert ungeheuer gut und pendelt sich rasch auf den stärksten Zug ein; bei mehreren etwa gleichguten Zügen verteilt das Verfahren die Rechenzeit etwa gleich auf alle. Die schlußendliche Qualität der Zugauswahl hängt wesentlich von der Anzahl der gespielten Zufalls-Partien ab; Experimente ergaben, dass bis zu zwei Millionen Monte-Carlo-Partien pro Stellung die Spielstärke wächst. Darüber hinaus gibt es keine veröffentlichten Untersuchungen, sodass der Breakeven der Methode noch unbekannt ist.

Die erfolgreichsten Vertreter der UCT-Gattung setzen auch Go-Muster und Bayes-Filter zum Lernen ein, um die Qualität der Zufallspartien zu steigern und damit die nötige Anzahl zu verringern. Dieses Wissen dient nicht dazu, wie im Schach Stellungen zu bewerten, es handelt sich also keineswegs um eine Art Bewertungsfunktion, die eine Position in eine Dezimalzahl quetscht, sondern es steuert die UCT-Suche. Der einzige Sinn der enthaltenen Go-Muster besteht darin, erfolgversprechende Varianten so früh wie möglich zu erkennen und von ihnen ausgehend besonders intensive Simulationen des weiteren Spielverlaufs zu starten. Es ist ja auch logisch: Bei perfekten Spielern reicht *eine einzige Partie*, um die Wahrheit herauszufinden, bei Zufallsspielern braucht man (theoretisch) *unendlich viele*. Je mehr Wissen in der UCT-Suche steckt, desto geringer ist also die Anzahl der nötigen Simulationen, oder umgekehrt, desto aussagekräftiger ist das Ergebnis bei gleicher Anzahl von Simulationen. Wie groß der Aufwand dafür ist, beschreibt Gian-Carlo Pascutto: "Wenn Leela kein Go-Wissen benutzt, berechnet sie ungefähr 40.000 Playouts pro Sekunde auf meinem 1,7-GHz-Rechner. Mit Wissen sind es gerade noch 6.000 Playouts pro Sekunde! Wissen und Suchbaum-Beschneidung kosten also viel Zeit, aber insgesamt ist es das natürlich wert."

Bemerkenswert an UCT ist unter anderem, dass die Bewertung der Wurzelstellungen eben nicht, wie bei Alpha-Beta, der Bewertung genau eines einzigen Blattknotens entspricht, sondern einem Querschnitt des Suchraums. Auch bekommen durch UCT auch erstmal als schwächer bewertete Züge immer wieder eine Chance, ihren Score zu verbessern, sodass auch versteckte taktische Möglichkeiten in Reichweite des Verfahrens liegen. Im Vergleich zu den traditionellen Go-Programmen mit Subspiel-Suche erkennen UCT-Programme die Zusammenhänge zwischen den einzelnen Gruppen, bemerken also nicht nur, dass von zwei angegriffenen Gruppen nur eine überleben kann, sondern sehen auch, welche davon wichtiger ist.

Das Ende einer Go-Partie zu erkennen, bereitet ungeübten Spielern oft Schwierigkeiten, und für traditionelle Programme ist es nicht leichter. UCT-Programme haben dieses Problem nicht, weil sie in ihren Simulationen so lange weitermachen, bis alle freien Punkte zugesetzt sind. Das führt zu einer interessanten Spielweise, denn die UCT-Programme, auch Leela, machen absolut keinen Unterschied zwischen einem Sieg mit einem Punkt Vorsprung und einem mit 100 Punkten. Dasselbe gilt für Niederlagen – kein Unterschied, ob sie völlig vom Brett geputzt werden oder nur ganz knapp unterliegen. Praktisch spielen sie daher sehr sicher und machen, falls sie sich vorn sehen, gern Züge im eigenen Gebiet. Das wird von spielstarken Menschen oft als unästhetisch empfunden ("So ein sinnloser Sicherungszug, es drohte doch gar nichts!"), aber das UCT-Programm weiß bereits, dass es gewinnt, und schert sich nicht um den einen verlorenen Punkt.

So geht es immer: Leela gewinnt, aber nur ganz knapp

So entwickeln sich ziemlich ausgeglichene Partien, der Mensch, egal welcher Spielstärke, verliert fast immer nur ganz knapp gegen das Programm und kann sich in dem (trügerischen) Gefühl wiegen, er hätte es ja fast geschafft. Dass dieser letzte Schritt, nämlich die nötigen mickerigen zwei Punkte mehr zu erobern, unvergleichlich viel schwerer gewesen wäre, fällt durch den logischen Spielverlauf nicht auf. Sieht sich ein UCT-Programm im Nachteil, schlägt es wild um sich. UCT hat ja keineswegs den Anspruch, den objektiv stärksten Zug zu ermitteln, sondern es errechnet, welcher Zug die größten Chancen bietet. Darum sind UCT-Programme beinharte Verteidiger; wenn man nicht ganz genau weiß, wie man eine Stellung im feindlichen Gebiet zum Leben bringt, nutzt das Programm die allerkleinste Ungenauigkeit brutal aus. Und es spielt immer die unangenehmsten Züge, die den schmalsten Pfad zum möglichen Überleben und Sieg offenlassen! Wenn das Programm einen scheinbar überflüssigen Sicherungszug ausführt, kann man ziemlich sicher sein, dass schon alles gelaufen ist – zu Gunsten von Leela.

Obwohl UCT-Go-Programmen ein kleiner Quantensprung im Vergleich mit heuristischen Programmen gelungen ist, spielen auch sie noch auf (allerdings höherem) Amateur-Niveau, zumindest auf dem 19x19-Brett. Nach der Computer-Olympiade 2007 trat die chinesische Profi-Spielerin Guo Juan gegen den Sieger MoGo an, wobei sie nur zehn Minuten Bedenkzeit hatte und dem Programm neun Steine vorgab, vergleichbar mit einer Damen-Vorgabe im Schach. Trotzdem schlachtete sie das UCT-Programm ganz mühelos. Von den drei danach ausgetragenen Partien auf dem 9x9-Brett verlor sie allerdings eine. Auf dem kleinen Brett erreichen die UCT-Programme bereits Profi-Dan-Level!