




- [Main page](#)
- [Contents](#)
- [Featured content](#)
- [Current events](#)
- [Random article](#)
- [Donate to Wikipedia](#)

- Interaction
  - [Help](#)
  - [About Wikipedia](#)
  - [Community portal](#)
  - [Recent changes](#)
  - [Contact Wikipedia](#)

- Toolbox
- Print/export

- Languages
  - [Français](#)
  - [한국어](#)
  - [日本語](#)

Article [Discussion](#) [Read](#) [Edit](#) [View history](#)

Candidates are currently being accepted for the Wikimedia Board of Trustees Election. [Become a candidate.](#)

[\[Hide\]](#)  
[\[Help with translations!\]](#)

# Computer Go

From Wikipedia, the free encyclopedia

**Computer Go** is the field of [artificial intelligence](#) (AI) dedicated to creating a [computer program](#) that plays [Go](#), a traditional [board game](#).

## Contents [\[hide\]](#)

- 1 Performance
  - 1.1 Recent results
- 2 Obstacles to high level performance
  - 2.1 Size of board
  - 2.2 Most moves are possible
  - 2.3 Additive nature of the game
  - 2.4 Techniques in chess that cannot be applied to Go
  - 2.5 Evaluation function
  - 2.6 Combinatorial problems
  - 2.7 Endgame
  - 2.8 Speculations on why humans are better at Go
  - 2.9 Order of play
- 3 Tactical search

Part of a series of articles on  
**[Go \(board game\)](#)**



## Game specifics

- [Go rules](#)
- [Go handicaps](#)
- [Go proverbs](#)

#### 4 State representation

#### 5 System design

##### 5.1 New approaches to problems

##### 5.2 Design philosophies

###### 5.2.1 Minimax tree search

###### 5.2.2 Knowledge-based systems

###### 5.2.3 Monte-Carlo methods

###### 5.2.4 Machine learning

#### 6 Competitions among computer Go programs

##### 6.1 History

##### 6.2 Rule Formalization Problems in computer-computer games

##### 6.3 Testing

#### 7 See also

#### 8 Notes and references

#### 9 Further reading

#### 10 External links

##### 10.1 Related websites

##### 10.2 Computer programs

## Performance

[[edit](#)]

Go has long been considered a difficult challenge in the field of [AI](#) and is considerably more difficult to solve than [chess](#). The first Go program was written by [Albert Zobrist](#) in 1968 as part of his thesis on [pattern recognition](#). It introduced an [influence function](#) to estimate territory and [Zobrist hashing](#) to detect [ko](#).

Recent developments in [Monte Carlo Tree Search](#) and [machine learning](#) have brought the best programs to high [dan](#) level on the small 9x9 board. In 2009, the first such programs appeared which could reach and hold low [dan-level ranks](#) on the [KGS Go Server](#) also on the 19x19 board.

- [Go provers](#)
- [Go terms](#)
- [Go strategy and tactics](#)

### History and culture

- [History of Go](#)
- [Go equipment](#)
- [Hikaru no Go](#)
- [Go variants](#)
- [Four go houses](#)

### Players and organizations

- [Go players](#)
- [Go ranks and ratings](#)
- [Go professional](#)
- [Go organizations](#)
- [Go competitions](#)

### Computers and mathematics




- [Go and mathematics](#)
- **[Computer Go](#)**
- [Go software](#)
- [Internet Go server](#)

This box: [view](#) • [talk](#) • [edit](#)

Currently, the best Go programs running on stock hardware are ranked as 2 dan - 3 kyu.<sup>[1]</sup> Only a decade ago, very strong players were able to beat computer programs at handicaps of 25–30 stones, an enormous handicap that few human players would ever take. There was a case in the 1994 World Computer Go Championship where the winning program, Go Intellect, lost all 3 games against the youth players on a 15-stone handicap.<sup>[2]</sup> In general, players who understood and exploited a program's weaknesses could win with much larger handicaps than typical players.<sup>[3]</sup>

## Recent results

[\[edit\]](#)

In 2008, thanks to an efficient message-passing parallelization, [MoGo](#)  won [one game](#)  (out of three) against [Catalin Taranu](#), 5th Dan Pro, in 9x9 with standard time settings (30 minutes per side). MoGo was running on a cluster provided by "Bull" (32 nodes with 8 cores per node, 3 GHz); the machine was down during one of the lost games. The results of this event were approved by the [French Federation of Go](#) . MoGo also played a 19x19 Game against Catalin Taranu and lost in spite of 9 stones handicap. However, MoGo was in good position during most of the game, and lost due to a bad choice in a ko situation at the end. The machine used for this event (the IAGO challenge, organized by the company "Recitsproque") is a good one, but far from the top level in industry<sup>[\[citation needed\]](#)</sup>.

On August 7, 2008, the computer program MoGo running on 25 nodes (800 cores, 4 cpus per node with each core running at 4.7 GHz to produce 15 Teraflops)<sup>[\[citation needed\]](#)</sup> of the Huygens cluster in Amsterdam beat [professional Go player](#) Myungwan Kim (8p) in a nine stone handicap game on the 19x19 board on the [KGS Go Server](#). MoGo won by 1.5 points. Mr. Kim used around 13 minutes of time while MoGo took around 55; however, he felt that using more time would not have helped him win. In after-game commentary, Kim estimated the playing strength of this machine as being in the range of 2–3 amateur dan.<sup>[4]</sup> MyungWan and MoGo played a total of 4 games of varying handicaps and time limits, each side winning two games. The game records are accessible on [KGS](#) where MoGo played as MogoTitan. In a rematch on September 20, Kim won two games giving MoGo nine stones.<sup>[5]</sup> On August 26, 2008, Mogo beat an Amateur 6d with five stones of handicap, this time running on 200 cores of the Huygens cluster.<sup>[6]</sup>

On September 4, 2008, the program [Crazy Stone](#) running on an 8-core personal computer won against 30 year-old professional, Aoba Kaori (4p), receiving a handicap of eight stones. The time control was 30 seconds per move. White resigned after 185 moves. The game was played during the FIT2008

conference in Japan.<sup>[7]</sup>

In February 2009, MoGo won two 19x19 games against professional Go players in the Taiwan Open 2009. With a 7-stones handicap the program defeated [Zhou Junxun](#) (9p), and with a 6-stones handicap it defeated Li-Chen Chien (1p).<sup>[8]</sup>

On February 14, 2009, Many Faces of Go running on a 32-core Xeon cluster provided by Microsoft won against James Kerwin (1p) with a handicap of seven stones. The game was played during the 2009 AAAS general meeting in Chicago.<sup>[9]</sup>

On August 7, 2009, Many Faces of Go (version 12) resigned against Myungwan Kim (8p) in a 7-stone handicap game.<sup>[10]</sup> Many Faces was playing on a 32 node system provided by Microsoft. The "Man vs. Machine" event was part of the 2009 US Go Congress, which was held in Washington DC from August 1 to August 9.<sup>[11]</sup>

On August 21 and 22, 2009, [Zhou Junxun](#) (9p) beat Many Faces of Go, MoGo, and Zen in full-board 7-stone games, beat MoGo in an even 9x9 game, and won one and lost one even 9x9 game against Fuego.<sup>[12]</sup>

On July 20, 2010, MoGoTW won an even 9x9 game as white against [Zhou Junxun](#) (9p).<sup>[13]</sup>

On July 20, 2010, at the 2010 IEEE World Congress on Computational Intelligence in Barcelona Spain computer program Zen played professional 4 dan Ping-Chiang Chou of Taiwan 19x19 Go. Yamato of Japan wrote Zen. Zen had 6 stone handicap. Each side had 45 minutes. Zen won the game.<sup>[14]</sup>

On July 28, 2010, at the 2010 European Go Congress in Finland computer program MogoTW played European professional 5 dan Catalin Taranu 19x19 Go. MogoTW had a 7 stone handicap. The computer won. MogoTW is a joint project between the MoGo team and a Taiwanese team.<sup>[15]</sup>

In December 2010, computer program Zen reached the rank 4 dan on the server KGS. Japanese programmer Ojima Yoji writes Zen.<sup>[16]</sup>

## Obstacles to high level performance

[\[edit\]](#)



This section **needs additional citations for verification**.

Please help [improve this article](#) by adding [reliable references](#). Unsourced material

For a long time it was a widely held opinion that computer Go posed a problem fundamentally different to computer chess insofar as it was believed that methods relying on fast global search compared to human experts combined to relatively little domain knowledge would not be effective for Go. Therefore, a large part of the computer Go development effort was during these times focused on ways of representing human-like expert knowledge and combining this with local search to answer questions of a tactical nature. The result of this were programs that handled many situations well but which had very pronounced weaknesses compared to their overall handling of the game. Also, these classical programs gained almost nothing from increases in available computing power per se and progress in the field was generally slow.

A few researchers grasped the potential of probabilistic methods and predicted that they would come to dominate computer game-playing,<sup>[17]</sup> but many others considered a strong Go-playing program something that could be achieved only in the far future, as a result of fundamental advances in general artificial intelligence technology. Even writing a program capable of automatically determining the winner of a finished game was seen as no trivial matter.

The advent of programs based on [Monte Carlo](#) search starting in 2006 changed this situation in many ways, although the gap between strong human players and the strongest Go programs remains considerable.

## Size of board

[\[edit\]](#)

The large board (19x19, 361 intersections) is often noted as one of the primary reasons why a strong program is hard to create. The large board size is a problem to the extent that it prevents an [alpha-beta searcher](#) without significant search extensions or pruning heuristics from achieving deep look-ahead.

So far, the largest game of Go being completely solved has been played on a 5×5 board. It was achieved in 2002, with black winning by 25 points (the entire board), by a computer program called MIGOS (Mini GO Solver).<sup>[18]</sup>

## Most moves are possible

[\[edit\]](#)

Continuing the comparison to chess, Go moves are not as limited by the rules of the game. For the first move in chess, the player has twenty choices. Go players begin with a choice of 55 distinct legal moves,

accounting for symmetry. This number rises quickly as symmetry is broken and soon almost all of the 361 points of the board must be evaluated. Some are much more popular than others, some are almost never played, but all are possible.

## Additive nature of the game

[\[edit\]](#)

As a chess game progresses (as well as most other games such as checkers, draughts, and backgammon), pieces disappear from the board, simplifying the game. Each new Go move, on the contrary, adds new complexities and possibilities to the situation, at least until an area becomes developed to the point of being 'settled'.

## Techniques in chess that cannot be applied to Go

[\[edit\]](#)

The fact that computer Go programs are significantly weaker than [computer chess](#) programs has served to generate research into many new programming techniques. The techniques which proved to be the most effective in computer chess have generally shown to be mediocre at Go.

While a simple material counting evaluation is not sufficient for decent play in chess, it is often the backbone of a chess evaluation function, when combined with more subtle considerations like isolated/doubled pawns, rooks on open files (columns), pawns in the center of the board and so on. These rules can be formalized easily, providing a reasonably good evaluation function that can run quickly.

These types of positional evaluation rules cannot efficiently be applied to Go. The value of a Go position depends on a complex analysis to determine whether or not the group is alive, which stones can be connected to one another, and heuristics around the extent to which a strong position has influence, or the extent to which a weak position can be attacked.

## Evaluation function

[\[edit\]](#)

Another problem comes from the difficulty of creating a good [evaluation function](#) for Go. More than one move can be regarded as the best depending on how you use that stone and what your strategy is. In order to choose a move, the computer must evaluate different possible outcomes and decide which is best. This is difficult due to the delicate trade-offs present in Go. For example, it may be possible to capture some enemy stones at the cost of strengthening the opponent's stones elsewhere. Whether this is a good trade



or not can be a difficult decision, even for human players. The computational complexity also shows here as a move might not be immediately important, but after many moves could become highly important as other areas of the board take shape.

## Combinatorial problems

[\[edit\]](#)

Sometimes it is mentioned in this context that various difficult combinatorial problems (in fact, any [NP-complete](#) problem can be converted to Go-like problems on a sufficiently large board)<sup>[\[citation needed\]](#)</sup>; however, the same is true for other abstract board games, including [chess](#) and [minesweeper](#), when suitably generalized to a board of arbitrary size. [NP-complete](#) problems do not tend in their general case to be easier for unaided humans than for suitably programmed computers: it is doubtful that unaided humans would be able to compete successfully against computers in solving, for example, instances of the [subset sum problem](#). Hence, the idea that we can convert some NP-complete problems into Go problems does not help in explaining the present human superiority in Go.

## Endgame

[\[edit\]](#)

Given that the endgame contains fewer possible moves than the opening or middle game, one could suppose that it was easier to play, and thus that computers should be easily able to tackle it. In chess, computer programs perform worse in endgames because the ideas are long-term, unless the number of pieces is reduced to an extent that allows taking advantage of solved endgame [tablebases](#).

The application of [surreal numbers](#) to the endgame in Go, a general game analysis pioneered by [John H. Conway](#), has been further developed by [Elwyn R. Berlekamp](#) and [David Wolfe](#) and outlined in their book, *Mathematical Go* ([ISBN 978-1-56881-032-4](#)). While not of general utility in most playing circumstances, it greatly aids the analysis of certain classes of positions.

Nonetheless, although elaborate study has been conducted, Go endgames have been proven to be [PSPACE-hard](#). There are many reasons why they are so hard:

- Even if a computer can play each local endgame area flawlessly, we cannot conclude that its plays would be flawless in regards to the entire board. Additional areas of consideration in endgames include [Sente](#) and [Gote](#) relationships, prioritization of different local endgames, territory counting & estimation, and so on.

- The endgame may involve many other aspects of Go, including 'life and death', which are also known to be [NP-hard](#).<sup>[19][20]</sup>
- Each of the local endgame areas may affect one another. In other words, they are dynamic in nature although visually isolated. This makes it much more difficult for computers to deal with. This nature leads to some very complex situations like [Triple Ko](#), [Quadruple Ko](#), [Molasses Ko](#) and [Moonshine Life](#).

Thus, it is very unlikely that it will be possible to program a reasonably fast algorithm for playing the Go endgame flawlessly, let alone the whole Go game.<sup>[21]</sup>

## Speculations on why humans are better at Go

[\[edit\]](#)

Go has features that might be easier for humans than computers.<sup>[22]</sup> The pieces never move about (as they do in [Chess](#)), nor change state (as they do in [Reversi](#)). Some speculated that these features make it easy for humans to "read" (predict possible variations) long sequences of moves, while being irrelevant to a computer program, while no rigorous cognitive neuroscience evidence indicating so.

In those rare Go positions known as "[ishi-no-shita](#)", in which stones are repeatedly captured and re-played on the same points, humans have reading problems<sup>[citation needed]</sup>, while they are easy for computers.

## Order of play

[\[edit\]](#)

Current, Monte-Carlo-based, Go engines can have difficulties in solving problems when the order of moves is important.<sup>[23]</sup>

## Tactical search

[\[edit\]](#)

One of the main concerns for a Go player is which groups of stones can be kept alive and which can be captured. This general class of problems is known as [life and death](#). The most direct strategy for calculating life and death is to perform a [tree search](#) on the moves which potentially affect the stones in question, and then to record the status of the stones at the end of the main line of play.



However, within time and memory constraints, it is not generally possible to determine with complete accuracy which moves could affect the 'life' of a group of stones. This implies that some [heuristic](#) must be applied to select which moves to consider. The net effect is that for any given program, there is a trade-off between playing speed and life and death reading abilities.

## State representation

[\[edit\]](#)

An issue that all Go programs must tackle is how to represent the current state of the game. For programs that use extensive searching techniques, this representation needs to be copied and/or modified for each new hypothetical move considered. This need places the additional constraint that the representation should either be small enough to be copied quickly or flexible enough that a move can be made and undone easily.

The most direct way of representing a board is as a 1 or 2-dimensional array, where elements in the array represent points on the board, and can take on a value corresponding to a white stone, a black stone, or an empty intersection. Additional data is needed to store how many stones have been captured, whose turn it is, and which intersections are illegal due to the [Ko rule](#).

Most programs, however, use more than just the raw board information to evaluate positions. Data such as which stones are connected in strings, which strings are associated with each other, which groups of stones are in risk of capture and which groups of stones are effectively dead is necessary to make an accurate evaluation of the position. While this information can be extracted from just the stone positions, much of it can be computed more quickly if it is updated in an incremental, per-move basis. This incremental updating requires more information to be stored as the state of the board, which in turn can make copying the board take longer. This kind of trade-off is indicative of the problems involved in making fast computer Go programs.

An alternative method is to have a single board and make and takeback moves so as to minimize the demands on computer memory and have the results of the evaluation of the board stored. This avoids having to copy the information over and over again.

## System design

[\[edit\]](#)

## New approaches to problems

[\[edit\]](#)

Historically, [GOF AI](#) (Good Old Fashioned AI) techniques have been used to approach the problem of Go AI. More recently, [neural networks](#) are being looked at as an alternative approach. One example of a program which uses neural networks is WinHonte.<sup>[24]</sup>

These approaches attempt to mitigate the problems of the game of Go having a high [branching factor](#) and numerous other difficulties.

Computer Go research results are being applied to other similar fields such as [cognitive science](#), [pattern recognition](#) and [machine learning](#).<sup>[25]</sup> [Combinatorial Game Theory](#), a branch of [applied mathematics](#), is a topic relevant to computer Go.<sup>[25]</sup>

## Design philosophies

[\[edit\]](#)

The only choice a program needs to make is where to place its next stone. However, this decision is made difficult by the wide range of impacts a single stone can have across the entire board, and the complex interactions various stones' groups can have with each other. Various architectures have arisen for handling this problem. The most popular use:

- some form of [tree search](#),
- the application of [Monte-Carlo methods](#),
- the application of [pattern matching](#),
- the creation of [knowledge-based systems](#), and
- the use of [machine learning](#).

Few programs use only one of these techniques exclusively; most combine portions of each into one synthetic system.

## Minimax tree search

[\[edit\]](#)

One [traditional AI](#) technique for creating game playing software is to use a [minimax tree search](#). This involves playing out all hypothetical moves on the board up to a certain point, then using an [evaluation function](#) to estimate the value of that position for the current player. The move which leads to the best hypothetical board is selected, and the process is repeated each turn. While tree searches have been very

effective in [computer chess](#), they have seen less success in Computer Go programs. This is partly because it has traditionally been difficult to create an effective evaluation function for a Go board, and partly because the large number of possible moves each side can make each leads to a high [branching factor](#). This makes this technique very computationally expensive. Because of this, many programs which use search trees extensively can only play on the smaller 9×9 board, rather than full 19×19 ones.

There are several techniques, which can greatly improve the performance of search trees in terms of both speed and memory. Pruning techniques such as [Alpha-beta pruning](#), [Principal Variation Search](#), and [MTD-f](#) can reduce the effective branching factor without loss of strength. In tactical areas such as life and death, Go is particularly amenable to caching techniques such as [transposition tables](#). These can reduce the amount of repeated effort, especially when combined with an [iterative deepening](#) approach. In order to quickly store a full sized Go board in a transposition table, a [hashing](#) technique for mathematically summarizing is generally necessary. [Zobrist hashing](#) is very popular in Go programs because it has low collision rates, and can be iteratively updated at each move with just two [XORs](#), rather than being calculated from scratch. Even using these performance-enhancing techniques, full tree searches on a full sized board are still prohibitively slow. Searches can be sped up by using large amounts of domain specific pruning techniques, such as not considering moves where your opponent is already strong, and selective extensions like always considering moves next to groups of stones which are [about to be captured](#). However, both of these options introduce a significant risk of not considering a vital move which would have changed the course of the game.

Results of computer competitions show that pattern matching techniques for choosing a handful of appropriate moves combined with fast localized tactical searches (explained above) are sufficient to produce a competitive program. For example, [GNU Go](#) is competitive.

## Knowledge-based systems

[\[edit\]](#)

Novices often learn a lot from the game records of old games played by master players. There is a strong hypothesis that suggests that acquiring Go knowledge is a key to make a strong computer Go. For example, Tim Kinger and [David Mechner](#) argue that "it is our belief that with better tools for representing and maintaining Go knowledge, it will be possible to develop stronger Go programs." They propose two ways: recognizing common configurations of stones and their positions and concentrating on local battles. "... Go

programs are still lacking in both quality and quantity of knowledge."<sup>[26]</sup>

After implementation, the use of expert knowledge has been proved very effective in programming Go software. Hundreds of guidelines and rules of thumb for strong play have been formulated by both high level amateurs and professionals. The programmer's task is to take these [heuristics](#), formalize them into computer code, and utilize [pattern matching](#) and [pattern recognition](#) algorithms to recognize when these rules apply. It is also important to have a system for determining what to do in the event that two conflicting guidelines are applicable.

Most of the relatively successful results come from programmers' individual skills at Go and their personal conjectures about Go, but not from formal mathematical assertions; they are trying to make the computer mimic the way they play Go. "Most competitive programs have required 5–15 person-years of effort, and contain 50–100 modules dealing with different aspects of the game."<sup>[27]</sup>

This method has until recently been the most successful technique in generating competitive Go programs on a full sized board. Some example of programs which have relied heavily on expert knowledge are Handtalk (later known as Goemate), The Many Faces of Go, Go Intellect, and Go++, each of which has at some point been considered the world's best Go program.

Nevertheless, adding knowledge of Go sometimes weakens the program because some superficial knowledge might bring mistakes: "the best programs usually play good, master level moves. However, as every games player knows, just one bad move can ruin a good game. Program performance over a full game can be much lower than master level."<sup>[27]</sup>

## Monte-Carlo methods

[\[edit\]](#)

One major alternative to using hand-coded knowledge and searches is the use of [Monte-Carlo methods](#). This is done by generating a list of potential moves, and for each move playing out thousands of games at random on the resulting board. The move which leads to the best set of random games for the current player is chosen as the best move. The advantage of this technique is that it requires very little domain knowledge or expert input, the trade-off being increased memory and processor requirements. However, because the moves used for evaluation are generated at random it is possible that a move which would be excellent except for one specific opponent response would be mistakenly evaluated as a good move. The result of

this are programs which are strong in an overall strategic sense, but are weak tactically. This problem can be mitigated by adding some domain knowledge in the move generation and a greater level of search depth on top of the random evolution. Some programs which use Monte-Carlo techniques are [Fuego](#), [The Many Faces of Go v12](#), [Leela](#), [MoGo](#), [Crazy Stone](#), [MyGoFriend](#), Olga and Gobble.

In 2006, a new search technique, *upper confidence bounds applied to trees* ([UCT](#)), was developed and applied to many 9x9 Monte-Carlo Go programs with excellent results. UCT uses the results of the *play outs* collected so far to guide the search along the more successful lines of play, while still allowing alternative lines to be explored. The UCT technique along with many other optimizations for playing on the larger 19x19 board has led MoGo to become one of the strongest research programs. Successful early applications of UCT methods to 19x19 Go include MoGo, Crazy Stone, and [Mango](#). MoGo won the 2007 [Computer Olympiad](#) and won one (out of three) blitz game against Guo Juan, 5th Dan Pro, in 9x9 Go. [The Many Faces of Go](#) won the 2008 [Computer Olympiad](#) after adding UCT search to its traditional knowledge-based engine.

## Machine learning

[\[edit\]](#)

While knowledge-based systems have been very effective at Go, their skill level is closely linked to the knowledge of their programmers and associated domain experts. One way to break this limitation is to use [machine learning](#) techniques in order to allow the software to automatically generate rules, patterns, and/or rule conflict resolution strategies.

This is generally done by allowing a [neural network](#) or [genetic algorithm](#) to either review a large database of professional games, or play many games against itself or other people or programs. These algorithms are then able to utilize this data as a means of improving their performance. Notable programs using neural nets are NeuroGo and WinHonte.

Machine learning techniques can also be used in a less ambitious context to tune specific parameters of programs which rely mainly on other techniques. For example, [Crazy Stone](#) learns move generation patterns from several hundred sample games, using a generalization of the [Elo rating system](#).<sup>[28]</sup>

## Competitions among computer Go programs

[\[edit\]](#)

Several annual competitions take place between Go computer programs, the most prominent being the Go events at the [Computer Olympiad](#). Regular, less formal, competitions between programs occur on the [KGS Go Server](#) (monthly) and the [Computer Go Server](#) (continuous).

Prominent go-playing programs include [Fuego](#), North Korean Silver Star/[KCC Igo](#), ZhiXing Chen's [Handtalk](#), Michael Reiss's [Go++](#) and David Fotland's [Many Faces of Go](#). [GNU Go](#) is a free computer Go implementation which has also won computer competitions.

## History

[\[edit\]](#)

The first computer Go competitions were sponsored by [USENIX](#). They ran from 1984-1988. These competitions introduced Nemesis, the first competitive Go program from [Bruce Wilcox](#), and G2.5 by [David Fotland](#), which would later evolve into Cosmos and The Many Faces of Go.

One of the early drivers of computer Go research was the Ing Prize, a relatively large money award sponsored by Taiwanese banker [Ing Chang-ki](#), offered annually between 1985 and 2000 at the World Computer Go Congress (or Ing Cup). The winner of this tournament was allowed to challenge young professionals at a handicap in a short match. If the computer won the match, the prize was awarded and a new prize announced: a larger prize for beating the professionals at a lesser handicap. The series of Ing prizes was set to expire either 1) in the year 2000 or 2) when a program could beat a 1-dan professional at no handicap for 40,000,000 [NT dollars](#). The last winner was Handtalk in 1997, claiming 250,000 NT dollars for winning an 11-stone handicap match against three 8-9 year old professionals. At the time the prize expired in 2000, the unclaimed prize was 400,000 NT dollars for winning a 9-stone handicap match.<sup>[\[29\]](#)</sup>

Many other large regional Go tournaments ("congresses") had an attached computer Go event. The European Go Congress has sponsored a computer tournament since 1987, and the USENIX event evolved into the US/North American Computer Go Championship, held annually from 1988-2000 at the US Go Congress.

Japan has recently started sponsoring its own computer Go competitions. The FOST Cup was held annually from 1995-1999 in Tokyo. That tournament was supplanted by the Gifu Challenge, which was held annually from 2003-2006 in Ogaki, Gifu. The [UEC](#) Cup has been held annually since 2007.



## Rule Formalization Problems in computer-computer games

[\[edit\]](#)

When two computers play a game of Go against each other, the ideal is to treat the game in a manner identical to two humans playing while avoiding any intervention from actual humans. However, this can be difficult during end game scoring. The main problem is that Go playing software, which usually communicates using the standardized Go Text Protocol (GTP), will not always agree with respect to the alive or dead status of stones.

While there is no general way for two different programs to “talk it out” and resolve the conflict, this problem is avoided for the most part by using customized rulesets such as variations on Chinese or Tromp-Taylor rules in which continued play (without penalty) is required until there is no more disagreement on the status of any stones on the board. In practice, such as on the KGS Go Server, the server can mediate a dispute by sending a special GTP command to the two client programs indicating they should continue placing stones until there is no question about the status of any particular group (all dead stones have been captured). The CGOS Go Server usually sees programs resign before a game has even reached the scoring phase, but nevertheless supports a modified version of Tromp-Taylor rules requiring a full play out.

It should be noted that these rulesets create a small risk that a program which was in a winning position at the traditional end of the game (when both players have passed), could be penalized for poor play that is made *after* the game was technically over, but this is not a common occurrence.

The main drawback to the above system is that some [rule sets](#) (such as the traditional Japanese rules) penalize the players for making these extra moves, precluding the use of additional playout for two computers. Nevertheless, most modern Go Programs support Japanese rules against humans and are competent in both play and scoring (Fuego, Many Faces of Go, SmartGo, etc.).

Historically, another method for resolving this problem was to have an expert human judge the final board. However, this introduces subjectivity into the results and the risk that the expert would miss something the program saw.

## Testing

[\[edit\]](#)

Many programs are available that allow computer Go engines to play against each other and they almost always communicate via the Go Text Protocol (GTP).

GoGUI and its addon gogui-twogtp can be used to play two engines against each other on a single computer system.<sup>[30]</sup> SmartGo and Many Faces of Go also provide this feature.

To play as wide a variety of opponents as possible, the KGS Go Server allows Go engine vs. Go engine play as well as Go engine vs. human in both ranked and unranked matches. CGOS is a dedicated computer vs. computer Go server.












## See also




















[\[edit\]](#)

- [Go \(game\)](#)
- [Go Text Protocol](#)
- [Computer chess](#)
- [Computer Othello](#)
- [Computer shogi](#)
- [Arimaa](#)

## Notes and references




[\[edit\]](#)






- <sup>^</sup> AyaMC, [Crazy Stone](#), Leela, Many Faces of Go, MoGo, and Zen all have/had accounts in this range on the [KGS Go Server](#) before or in August 2009
- <sup>^</sup> [Program versus Human Performance](#) 
- <sup>^</sup> See for instance [http://www.intgofed.org/history/computer\\_go\\_dec2005.pdf](http://www.intgofed.org/history/computer_go_dec2005.pdf)  [Archived](#)  May 28, 2008 at the [Wayback Machine](#).
- <sup>^</sup> Computer Beats Pro at U.S. Go Congress [http://www.usgo.org/index.php?%23\\_id=4602](http://www.usgo.org/index.php?%23_id=4602) 
- <sup>^</sup> [September 21, 2008; Volume 9, #49 SPECIAL EDITION!](#) 
- <sup>^</sup> [Sensei's Library: MoGo](#) 
- <sup>^</sup> [Crazy Stone defeated 4-dan professional player with a handicap of 8 stones.](#) 
- <sup>^</sup> ["French software and Dutch national Supercomputer Huygens establish a new world record in Go"](#) . The Netherlands Organization for Scientific Research (NWO). 25 February 2009. Retrieved 2009-03-06.
- <sup>^</sup> [Many Faces of Go defeated 1-dan professional player with a handicap of 7 stones.](#) 
- <sup>^</sup> [AGA News](#) 
- <sup>^</sup> [2009 US Go Congress](#) 

12. ^ [2009 IEEE International Conference on Fuzzy Systems](#) 
13. ^ [Computers vs Humans in Barcelona \(WCCI 2010\)](#) 
14. ^ [Computer program Zen with 6 stone handicap beat professional 4 dan Ping-Chiang Chou of Taiwan](#) 
15. ^ [Computer program MogoTW with 7 stone handicap beat European professional 5 dan Catalin Taranu](#) 
16. ^ [Sensei's Library KGS Bot Ratings](#) 
17. ^ [Game Tree Searching with Dynamic Stochastic Control](#)  pp.194-195
18. ^ [5×5 Go is solved by Mlni GO Solver](#) 
19. ^ On page 11: "Crasmaru shows that it is NP-complete to determine the status of certain restricted forms of life-and-death problems in Go." (See the following reference.) Erik D. Demaine, Robert A. Hearn (2008-04-22). *Playing Games with Algorithms: Algorithmic Combinatorial Game Theory*. [arXiv:cs/0106019](#) 
20. ^ Marcel Crasmaru (1999). "On the complexity of Tsume-Go.". *Lecture Notes in Computer Science* (London, UK: Springer-Verlag) **1558**: 222–231. doi:10.1007/3-540-48957-6\_15 
21. ^ See [Computer Go Programming](#)  pages at [Sensei's Library](#)
22. ^ Raiko, Tapani: "The Go-Playing Program Called Go81"  section 1.2
23. ^ [example of weak play of a computer program](#) 
24. ^ [WinHonte 2.01](#) 
25. ^ <sup>a</sup> <sup>b</sup> Müller, Martin. [Computer Go](#) , Artificial Intelligence 134 (2002): p150
26. ^ Müller, Martin. [Computer Go](#) , Artificial Intelligence 134 (2002): p151
27. ^ <sup>a</sup> <sup>b</sup> Müller, Martin. [Computer Go](#) , Artificial Intelligence 134 (2002): p148
28. ^ [Computing Elo Ratings of Move Patterns in the Game of Go](#) 
29. ^ [World Computer Go Championships](#) 
30. ^ [Using GoGUI to play go computers against each other](#) 

## Further reading

[\[edit\]](#)

- [AI-oriented survey of Go](#) 
- [Co-Evolving a Go-Playing Neural Network](#) , written by Alex Lubberts & Risto Miikkulainen, 2001
- *Computer Game Playing: Theory and Practice*, edited by M.A. Brauner (The Ellis Horwood Series in Artificial Intelligence), Halstead Press, 1983. A collection of computer Go articles. The American Go Journal, vol. 18, No 4. page 6. [ISSN 0148-0243]
- [A Machine-Learning Approach to Computer Go](#) , Jeffrey Bagdis, 2007.

- [Minimalism in Ubiquitous Interface Design](#)  Wren, C. and Reynolds, C. (2004) Personal and Ubiquitous Computing, 8(5), pages 370 - 374. [Video of computer Go vision system in operation](#)  shows interaction and users exploring [Joseki](#) and [Fuseki](#).
- [Monte-Carlo Go](#) , presented by Markus Enzenberger, Computer Go Seminar, University of Alberta, April 2004
- [Monte-Carlo Go](#) , written by B. Bouzy and B. Helmstetter from Scientific Literature Digital Library
- [Static analysis of life and death in the game of Go](#) , written by Ken Chen & Zhixing Chen, 20 February 1999

## External links





[\[edit\]](#)

## Related websites

[\[edit\]](#)



Wikibooks has a book on the topic of  
***Computer Go***

- [Mick's Computer Go Page](#) 
- [Extensive list of computer Go events](#) 
- [All systems Go](#)  by David A. Mechner (1998), discusses the game where professional Go player [Janice Kim](#) won a game against program [Handtalk](#) after giving a 25-stone handicap.
- Kinger, Tim and Mechner, David. [An Architecture for Computer Go](#)  (1996)
- [Computer Go](#)  and [Computer Go Programming](#)  pages at [Sensei's Library](#) 
- [Computer Go bibliography](#) 
- [Another Computer Go Bibliography](#) 
- [Computer Go mailing list](#) 
- Published articles about computer Go on [Ideosphere](#)  gives current estimate of whether a Go program will be best player in the world
- [Information on the Go Text Protocol](#)  commonly used for interfacing Go playing engines with graphical clients and internet servers
- The Computer Go Room on the [K Go Server](#)  (KGS) for online discussion and running "bots"
- [Two Representative Computer Go Games](#) , an article about two computer Go games played in 1999, one with two computers players, and the other a 29-stone handicap human-computer game
- [What A Way to Go](#)  describes work at Microsoft Research on building a computer Go player.

- [Cracking Go](#), by Feng-hsiung Hsu, [IEEE Spectrum magazine](#), October 2007 [↗](#) argues why it should be possible to build a Go machine stronger than any human player

## Computer programs

[\[edit\]](#)

See also: [Go software](#)

- [AYA](#) [↗](#) by Hiroshi Yamashita
- [Crazy Stone](#) by Rémi Coulom
- [Fuego](#) [↗](#), an [open source](#) Monte Carlo program
- [GNU Go](#), an open source classical Go program
- [Go++](#) [↗](#) by Michael Reiss (sold as *Strongest Go* or Tuyoi Igo in Japan)
- Go Intellect by Ken Chen
- Handtalk/Goemate, developed in China by Zhixing Chen (sold as Shudan Taikyoku in Japan)
- Haruka by Ryuichi Kawa (sold as Saikouhou in Japan)
- Indigo by Bruno Bouzy
- Katsunari by Shin-ichi Sei
- KCC Igo, from North Korea (sold as Silver Star or Ginsei Igo in Japan)
- [Leela](#) [↗](#), the first Monte Carlo program for sale to the public
- [The Many Faces of Go](#) [↗](#) by David Fotland (sold as AI Igo in Japan)
- [MyGoFriend](#) [↗](#) by Frank Karger
- [MoGo](#) [↗](#) by Sylvain Gelly; parallel version <http://www.lri.fr/~teytaud/mogo.html> [↗](#) by many people.
- [Smart Go](#) [↗](#) by Anders Kierulf, inventor of the [Smart Game Format](#)
- [Zen](#) [↗](#) by "Yamato".

Categories: [Computer Go](#)

This page was last modified on 29 April 2011 at 17:15.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. See [Terms of Use](#) for

details.

Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Contact us](#)

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#)

