SapioGo
An Artificially Intelligent Go Game using Neural Networks and Genetic Algorithms

---

Ryan Flannery
squid@fuse.net

Kevin Upchurch
upchurks@email.uc.edu

Advisor: John Schlipf
john.schlipf@uc.edu

June 8, 2004

Abstract

SapioGo is a program that plays the game of Go and attempts to "learn" how to play better over time by using a genetic algorithm known as Symbiotic Adaptive Neuro-Evolution (SANE) to evolve an artificial neural network. SapioGo attempts to raise the potential of the neural network by increasing the overall size of the neural network. Co-evolution is used to compensate for the slow learning curve.

Contents

Forward

This project was more than the application of our cumulative knowledge in computer science; it was, in fact, another learning experience for us. Neither of us had any prior experience with neural networks nor genetic algorithms but we were fascinated with the theory behind them and were determined to learn more. SapioGo is the result of six months of research and development into neural networks and genetic algorithms.

More than that, this project was fun. We really had a great time creating SapioGo and although the quarter is over we hope to continue this project further.

We would also like to thank Professor Schlipf for advising us on this project and taking the time out of his busy schedule to meet with us for one hour every week.

We would like to thank Meghan Murphy for helping name the project, as well as for the milk shakes she provided for us during one of our many late-night *hack-a-thons*.

And finally we would like to thank Carrie Lucken for proofing our final document, saving us from our hopeless grammer skills.

Ryan Flannery & Kevin Upchurch

Chapter 1

The Project

1.1 The Game of Go

1.1.1 History and Evolution of the Game of Go

The game of Go is believed to have originated between 4,000 and 5,000 years ago in the area that is modern-day China and is considered to be the oldest game still played today. Mystery surrounds the creation of Go, but some believe that Go may have been a precursor to the abacus. One legend tells of an emperor who created the game in an attempt to make his dull-witted son more intelligent.

By approximately 600 B.C., Go had become one of the "Four Accomplishments," which included brush painting, poetry and music, that all Chinese gentleman were to master, thus playing a major role in Chinese culture.

Given China's close proximity to countries such as Japan and Korea, it's not surprising that Go became equally prevalent in those countries. In the early 1600's, four schools of Go were created in Japan, where students learned to master the game in a formalized setting. These schools still exist today and are a major part of Japanese culture. Japanese Go Masters are considered celebrities in their country.

Today Go is most popular in Korea where it is believed that between five and ten percent of the population plays regularly. Some of the worlds best Go players reside in Korea.

Go was brought to the United States in the mid-1800's when multitudes of Chinese workers were immigrating to the U.S.

With so many different cultures playing the game of Go, it's no wonder many different methods of playing evolved. Each culture has its own rule system, varying in the way points are tallied at the end of the game. Some even have their own timing systems which force players to make moves in a required amount of time.

1.1.2 The Rules of Go

Go is a game played on a large, flat wooden board where two opposing players place white and black stones called "ishi" on the board. The board is square with 19 horizontal lines intersecting 19 vertical lines. Although Go games may be played on any size board, 9x9, 13x13, and 19x19 boards are the most common. The two players alternate placing stones on the intersection of two lines, rather than in the squares formed by them.

When the stones are placed on the board, players have the option of trying to *capture* the stones of their opponent. Capturing the stones of one's opponent is done by entirely surrounding the stone of one's opponent. Once this is done, the opponents' stone is removed from the board. If numerous stones of the opponent are adjacent to each other, then all of the stones must be surrounded and captured at once.

The goal of a game of Go is to encircle territory on the board. This is done by surrounding large areas of the board with stones and preventing the opposing player from doing the same. Also, players want to minimize the number of stones of their color that are captured while trying to capture as many stones of the opponent as possible.

The Black player always moves first in a game of Go. This provides an unfair advantage to the Black player, and to help compensate, the White player always receives an additional $5\frac{1}{2}$ points at the end of the game. This rule is called *Komi*. The amount of Komi given to the White player is the same regardless of the board size. Also, the additional $\frac{1}{2}$ point that is awarded to White makes it impossible to have two players tie in a game of Go.

Each player has the option of passing when it is his/her turn, opting not to place any stones on the board. When both players pass consecutively, the game is considered over. This usually happens when there is no territory left on the board for one of the players to capture, or when it is obvious to one of the players that their any continuing effort would be futile.

Once the game is over, the score for each player is tallied and the player with the higher score wins. Methods for determining the score vary with the rule system being used, but they all roughly award points as follows: every intersection on the board that is completely encircled by stones of one color are awarded to that player. Every stone on the board that is responsible for encircling some territory is also added to that players score. *Note that stones on the board that do not take part in encircling some area are called Dead Stones and do not affect the score.* Stones that have been captured by a player are added to that players score.

1.1.3 Terminology

Below is a brief introduction to some common terminology used in the game of Go, as well as throughout the documentation for this project.

Atari An state when your opponent can capture one or more *ishi* in the next move.

Dame A point on the board that is useless for both players.

Fuseki The opening moves in a game before the real strategy begins.

Gote A move causing the current player to lose *initiative*. This is the opposite of *sente*.

Initiative A state where the current player can force their opponent to move in certain locations.

Ishi Used to refer to the stones in a game of Go.

Ko A situation where a single board position may be captured and recaptured infinitely by both players. *The rule of Ko* eliminates such endless cycles by not allowing a previous board state to be repeated.

Komi A number of points (usually $5\frac{1}{2}$) that is added to White's final score to compensate for Black having moved first. Some rule-sets may choose to use a different amount of points or none at all. Note that the $\frac{1}{2}$ point prevents a Go game from ever ending in a draw.

Kyu A standard ranking used to evaluate weaker players. The ranking ranges from 1 to 25, with 1 being the strongest and 25 being the weakest.

Sente A state that maintains *initiative* for the current player by forcing their opponent to move in a certain location since any alternative move would result in the loss of critical *territory*.

Territory Area on the Go board where both players battle for control.

1.1.4 Computer Go

The game of Go has a long history in the field of computer science. Its unique nature has made it very difficult to model with any algorithm and no existing approach has created a Go client capable of consistently beating even a novice.

The field of study focused on creating an intelligent Go game is known as *Computer Go,* and includes work in areas such as algorithm design & analysis, artificial intelligence, and data structures.

Computer Go is so well established that standards for file formats and communication protocols (for playing Go over a network) have already been established. There are even numerous open-source software packages available that implement these standards, allowing the development of intelligent Go systems to focus more on the design and implementation of the intelligence.

1.2 Approach

The SapioGo project is composed of three completely seperate sub-projects. The first is a generalized Go playing API that allows programmers to very quickly

and very easily write Go playing games. The second is a generalized API that can be used for learning through neural networks and genetic algorithms.

The third sub-project is SapioGo itself, and simply combines the first two API's to play Go and learn.

1.2.1 Playing Go

Computer Go is a well established computer game, with a great deal of existing work to build on. Standard protocols for "speaking" Go over a network already exist and there are even open-source libraries available that implement these protocols. There is even a standard file format, the Simple Go Format (SGF), for storing an entire Go game, or just a snapshot of one, to a file.

The most widely used protocol is known as the Go Modem Protocol (GMP), and is supported by virtually all Go clients. GMP is, however, quit old and is no longer maintained. A newer, more robust protocol called the Go Text Protocol (GTP) has already been established, and the GNU Go game provides an easy interface for using GTP.

Since GMP is widely accepted by many Go clients, SapioGo uses GMP when playing against most opponents including human opponents. However, when SapioGo plays GNU Go it utilizes GTP.

Go Modem Protocol (GMP)

The Go modem protocol was developed by Bruce Wilcox, with input from David Fotland, Anders Kierulf, and other computer Go Programmers. The hope was that all Go programs would implement GMP so that all Go program users would be able to play go no matter which program they own, and so that computer/computer competitions can be played without needing an operator to type moves back and forth. The problem we found with GMP was that it was not as functional as we would have liked it. Testing for legal moves and other Go situation like Seki was left up to the programmers to implement. Other situations like scoring were also not provided in the protocol. What we wanted was something that could work right out of the box. However we couldn't not use GMP, because of its wide acceptance in the Computer Go community.

Go Text Protocol (GTP)

The Go Text Protocol, GTP, is a text based protocol for communication with computer go programs. It is a modern alternative to the Go Modem Protocol, GMP, and may potentially replace this for use in Go tournaments in the future. It is also intended, through the use of auxiliary programs, to make it easier for go programmers to connect to go servers on the internet and do automatic regression testing. GTP is able to test for legal moves, get the final score, calculate territory.

1.2.2 Artificial Intelligence Approaches to Go

When it comes to playing games with computers, there are a number of standard approaches taken to make the computer play intelligently. These range from min-max based and heuristic algorithms to neural-networks and genetic algorithms.

Classical Approaches

When modelling other games, such as tac-tac-toe or chess, a *min-max* algorithm is generally employed, where the computer checks every possible sequence of moves that could take place and determines its move by following the sequence of moves that *minimizes* its losses and *maximizes* its gains. By doing this, the computer always makes an optimal move and, as long as the game is fair (which both tic-tac-toe and chess are) it is then impossible to beat the computer. The best one could hope to do is tie with it. Such an algorithm, however, employs no real intelligence. Rather, it exploits the computer's ability to peform billions of operations per second and always plays with the same level of difficulty (although optimal), never really improving.

Another approach commonly used to model intelligent game play by computers is to use a *heuristic* algorithm. Such an algorithm determines its move based on a set of rules that are hard-coded into the computer. Such an approach is typically not practical, as it requires heavy pattern matching (to determine what it's current "state" is) and a great deal of effort on behalf of the programmer to hard-code an optimal response for every possible state. This is, of course, assuming that an optimal or adequate response can be found for every possible state.

Neither of these approaches alone peforms well when playing Go for the following reasons:

1. A game of computer-Go could potentially last forever. In a normal game of Go, each player has only 250 pieces which means that a game of Go can last no more than 500 moves. In computer Go, however, each player is assumed to have an infinite number of pieces. Although the board size is finite, since territory can be played on, captured, and then re-played over, there is the potential for a game to never end. As such, min-max algorithms fail in enumerating every possible board state.

2. A game of computer Go has *many* board states, too many to enumerate. Even on a 9x9 board, there are 3^{9^2} potential board states (remember, each intersection has 3 potential states: empty, black, or white). That's

   $$443{,}426{,}488{,}243{,}037{,}769{,}948{,}249{,}630{,}619{,}149{,}892{,}803$$

   board states! We could certainly make use of symmetry in the board to reduce this number, but it would still not be enough to make such an approach feasible.

3. In Go, an optimal move is not always clearly defined. Master Go players often describe their moves based on the current board state, how far into a game of Go they are, and the tactics employed by their opponent. Incorporating their opponents tactics into their own strategy makes any attempt at duplicating this approach far more difficult and impractical.

For playing Go, alternative approaches must be used. Two popular methods that have had limited success in the area are neural networks and genetic algorithms.

Artificial Neural Networks

Artificial neural networks (ANNs) are designed to mimic the structure of a human brain and provide a highly generalized method of learning. ANNs key advantage is their ability to handle errors in training data.

In the human brain, there are approximately 10^11 neurons, each attached to roughly 10,000 other neurons. It is through this vast connected graph that electrical impulses travel. Of a neurons connection, one is called the 'axon' and is essentially the *output* of the neuron (where the electrical impulse exits the neuron). The rest of the connection are called 'dendrites' and are the *inputs* of the neuron (where the electrical impulses enter the neuron). Between the dendrites and the axon is the cell body, where the output of the neuron is determined based on the cumulative input from the dendrites along with a weight associated with each dendrite, as shown in equation 1.1.

$$\sum_{k=1}^{n} input_k weight_k = CellInput \tag{1.1}$$

The output of the neuron is then determined by performing some function on this input, as shown in equation 1.2. In the human brain, this function is quite complex and requires a great deal of space just to write it.

$$f(CellInput) = CellOutput \tag{1.2}$$

In ANNs, this function is typically kept very simple and is often no more complex than a simple linear function (where the output is some scalar multiple of the input).

ANNs are structured such that their input is a vector of real-values, the weights associated with each connection are real-valued, and the output of the ANN is a real-valued vector. The application using an ANN needs only to determine how to represent its data as a real-valued vector for input into the ANN and how to decipher the output vector.

A game of tic-tac-toe, for example, might encode the board state as a vector of 9 values, corresponding to the 9 playable locations on the tic-tac-toe board. For each location, a 0 is input if the location is empty, a 1 if it is occupied by the current player, and a -1 if it is occupied by the opponent. The output of the neural network could then be another vector of 9 real values, and the

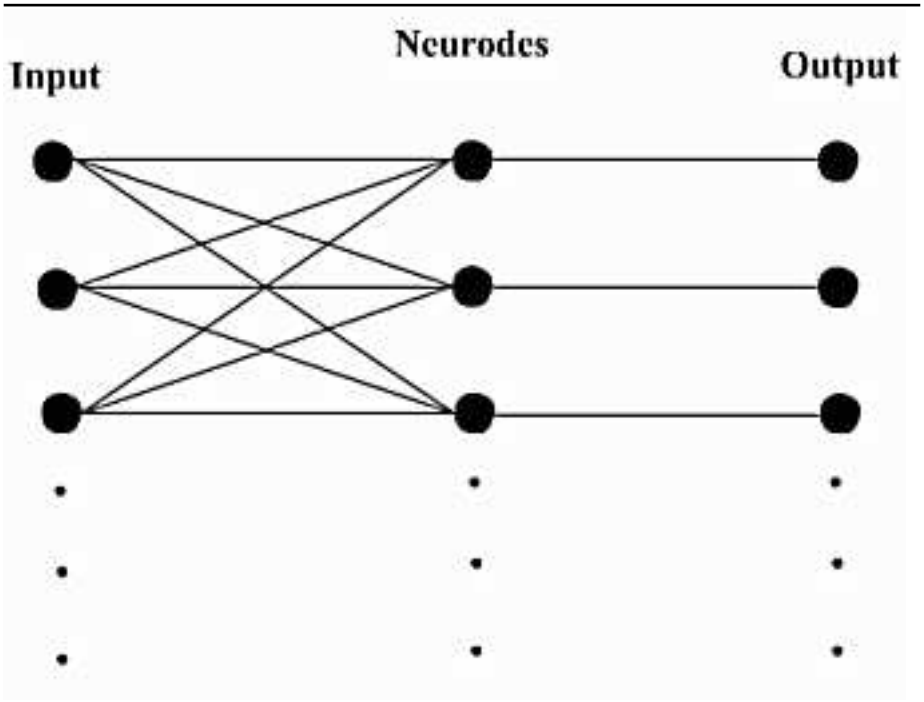


Figure 1.1: Artificial Neural Network used to play Tic-Tac-Toe.

application may choose its move, for example, by selecting the greatest of these values and then move in the corresponding board location. The ANN would then look something like figure 1.1.

This sort of structure is typically called a *feed-forward* design.

When using an ANN, the goal is to find an optimal set of weights for the connections such that a given input vector will produce desired output. Determining an optimal set of weights is where other algorithms are employed and are discussed later. The performance of an ANN is based on how *granular* the network is. Typically, the larger the input vector and the larger the population of neurons, the better the network will perform with an optimal set of weights. The performance of an ANN with optimal weights is typically called the network's *potential*.

There may also be multiple layers to an ANN, where the outputs of one layer of neurodes are the inputs to another layer of neurodes, as shown in figure 1.2.

The advantage to having multiple layers is to increase the granularity of the ANN, and thus increase the network's potential. Adding mulitple layers, however, also increases the complexity and size of the network dramatically. As such, determining an optimal set of weights becomes increasingly difficult.

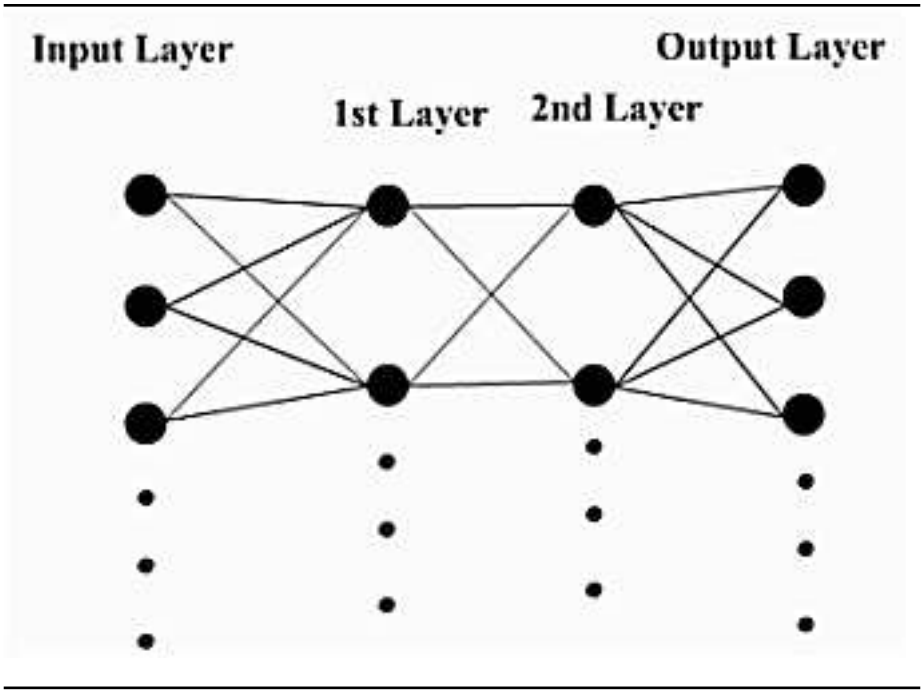


Figure 1.2: A Two-Layer, Feed-Forward Artificial Nueral Network.

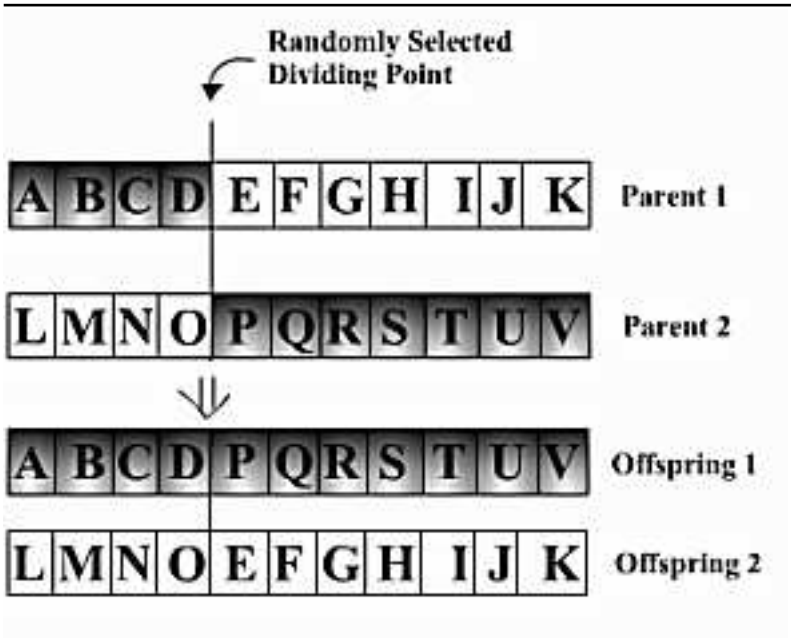


Figure 1.3: One-Point Crossover Operation.

Genetic Algorithms

Genetic algorithms try to simulate biological evolution to achieve learning, and like ANNs, they provide a highly generalized approach to learning that is robust against errors in training data.

In a genetic algorithm, the problem is broken up into many *hypotheses* which are tested against the target problem and rated according to how well they performed. Hypotheses that performed well are breed with each other to produce other hypotheses (offspring) that typically replace those hypotheses that performed poorly. To simulate real evolution, the concept of mutation is introduced during breeding, where each offspring has some chance of being "mutated."

Hypotheses are typically represented as large bit-strings, analagous to DNA. Breeding two hypotheses is accomplished via a crossover operation, where the two hypostheses are lined up, side by side, and points along the hypotheses are chosen at random. These points determing how the hypotheses will divide.

In a one-point crossover operation, as shown in figure 1.3, only one point is randomly selected along the two hypotheses. All bits before the point in one hypotheses are joined with all of the points after the point in the other hypotheses to form one offspring, while the remaining bits are used for the second offspring.

With genetic algorithms, the term "species" is used to refer to the set of all

1.	Reset all cumulative ratings for each neurode to 0
2.	Randomly select a subset (blueprint) of the neurode population
3.	Play a game of Go using only the neurodes in the selected blueprint
4.	Based on the performance of the blueprint, rate each of the neurodes in the blueprint
5.	Continue steps 2-4 until all neurodes have participated in an adaquate number of blueprints
6.	Sort the neurode population by their cumulative rating
7.	Delete the lowest performing 25% of the neurodes
8.	Breed the top performing 25% of the neurodes with each other and use their offspring to replace the lower 25% that were just deleted

Figure 1.4: Pseudo-code for one generation of SANE.

hypotheses of the same size. For example, two hypotheses that are bit-strings of the same length are said to be of the same species. For some applications, many different species of solutions are evolved against the same problem.

The SANE Algorithm

One genetic algorithm, called Symbiotic-Adaptive Neuro-Evolution (SANE for short), is a genetic algorithm that employs reinforcement learning for optimizing the weights and bias in a neural network. SANE was first conceived by David E. Moriarty and Risto Mikkulainen at the University of Texas at Austin in 1996, and has since been employed in applications ranging from games (even Go) to the classic pendulum problem.

SANE works through *reinforcement learning* to develop *cooperation* and *specialization* in the neurode population of a neural network. The theory behind SANE is that individual neurodes in the neurode population represent only a partial solution to a problem while the entire population, working *cooperatively*, represents a complete solution. Subsets of the population are then *specialized* for certain tasks.

SANE achieves specialization through reinforcement learning, where subsets of the neurode population (called "blueprints") are selected randomly and then used in a sample network for a sample task. The neurodes selected are then rated according to how well they performed the task. Blueprints are chosen, tested, and rated until all neurodes in the population have been used a minimum number of times (which is variable). Once all the neurodes have been used this minimum number of times, they are sorted according to their cumulative rating and the top 25% of the neurodes are breed with themselves (typically using a one-point crossover of the two neurode's weight vectors as illustrated in figure 1.3). Their offspring then replace the bottom 25% of the population. This is called one "generation" of the SANE algorithm. Figure 1.4 shows complete pseudo-code for the SANE algorithm.

1.2.3 The SapioGo Approach to Go

For SapioGo, the SANE algorithm was used in conjunction with a two-layer, feed-forward ANN. Since the ANN has two layers, each neurode in the hidden layer has two associated weight vectors, attached to the inputs and the output neurodes, respectively. These weight vectors are vectors of floating point values that act as the hypotheses in SANE.

Network Dimensions

One of the key differences between SapioGo and previous attempts at modelling Go using SANE is SapioGo's massive input vector. The input to SapioGo's ANN includes nearly all input that any human player has available when playing Go. In all, the input vector for the ANN consisted of the following data points:

1. *The state of each intersection.* Each intersection can have three possible states: empty, occupied by a stone of SapioGo, or occupied with a stone of the opponent. Since there are three possible states, we use two inputs to encode this with the following scheme: (1,1) if the intersection is empty, (0,1) if the intersection is occupied by SapioGo's stone, and (1,0) if the intersection is occupied by an opponent's stone.

2. *The number of liberties for the stone at each intersection.* For each intersection, we add an additional value to the input vector containing the number of liberties for the stone at that intersection. If the intersection is empty, the input is set to the board size.

3. *The intersection coordinates.* In SapioGo, an arbitrary $x - y$ coordinate system is applied to the game board, with (0,0) being treated as the upper-right intersection and the coordinates increasing in value both down and accross the board. So, for each intersection, an additional two inputs are present in the input vector.

4. *Miscellaneous Game Information.* Six additional inputs included are SapioGo's current score, the opponents's current score, the percentage of the board covered with SapioGo's stones, the percentage of the board covered by the opponent's stones, the percentage of the board covered by either, and the number of moves made so far.

The output vector of the ANN contains one real-value for each intersection on the board. SapioGo chooses its move in the following manner. If no output is above a given threshold, SapioGo passes. Otherwise, SapioGo randomly selects from the top 5 moves. This random selection of moves was chosen to make SapioGo less deterministic, and more closely simulate how humans play Go.

The hidden layer of the ANN contains between 2,000 and 4,000 neurodes, depending on the size of the Go board. The size of the blueprints also ranged from 100 to 500. These values were determined experimentally by Norman Richards in his paper "Evolving Neural Networks to Play Go". Table 1.5 summarizes the neural network dimensions used by SapioGo.

Neural Network Evaluation Function

The mapping function used in each neurode was kept as simple as possible. For each neurode, the output is equivalent to the input of the neurode.

$$f(x) = x \tag{1.3}$$

SANE Evalutation Function

After a generation is complete, we rate the blueprint using the score of the game. SapioGo uses a modified version of the Chinese ruleset, where the score for each player is calculated as follows:

	Number of Captured Enemy Stones
+	Number of Intersection in Player's Territory
+	Number of Player's Stones on the Board
-	Number of Player's Dead Stones
+	Number of Opponent's Dead Stones
+	Komi (if Player is White)
=	Total Score

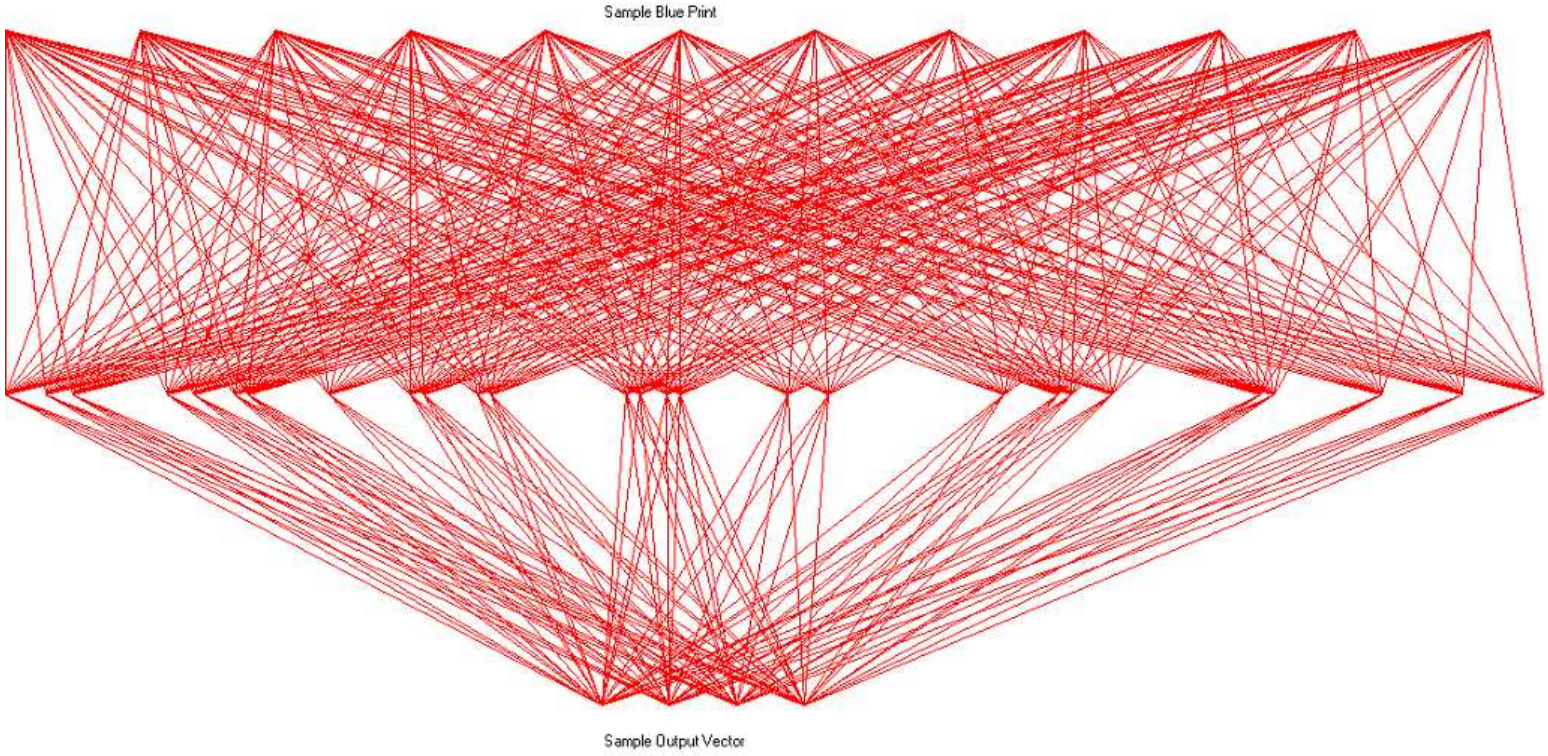
Implementation

For SapioGo, two seperate species were evolved: one on a 5x5 board and another on a 9x9 board. The goal of the 5x5 species was to get at least one win against GNU Gobefore the end of the quarter. We knew that a 5x5 board would have a smaller input and output network and thus would be able to optimize faster than the 9x9 board. Breaking our approach into 2 different board strains we were able to optimize our collective computing power by not having to waste time sending 5+ meg. files back and forth so that we would have the highest species of board state.

The goal of the 9x9 species was the same as the 5x5: to beat GNU Go. Although this has not been accomplished we feel that with enough processing time and enough generations we will be able to beat GNU Go.

Board Size	Input Size	Hidden Layer Size	Output Size	Blueprint Size
5x5	131	2,000	25	100
9x9	411	4,000	81	500
nxn	$5n^2 + 6$	n/a	n^2	n/a

Table 1.1: SapioGo Neural Network Dimensions


Sample Blue Print
Sample Output Vector

1.3 Project Requirements

1.3.1 Game Details

SapioGo will support playing a basic game of Go with traditional Chinese settings. SapioGo will allow either itself or its opponent to make the opening move, and will also support handicaps for either its opponent or itself.

Board Sizes

SapioGo will support the standard Go board sizes of 9x9, 13x13, and 19x19, and will even allow the user to specify an arbitrary board size. During development, two species of SapioGo will be evolved: the first will use a 5x5 board and the second will use a 9x9 board. Although the 9x9 board is only slightly larger than the 5x5, the neural network required for the 9x9 species will be over 6 times larger than the one required for the 5x5 board. As such, the evolution of the 5x5 species will be much faster and hopefully provide insight into the evolution of the 9x9 board.

Rule Set, Time System, and Handicaps

SapioGo will only be able to play using the standard Chinese rules of Go and will not use any timing system. It will support a standard Komi of $5\frac{1}{2}$. Also, SapioGo will not support handicaps for either player in a game.

Opponents

SapioGo will be able to play against human opponents or even other automated Go clients. It could even be set to play against itself. SapioGo will also support playing opponents over a network, allowing users to play SapioGo from remote locations.

Benchmarking

During testing, SapioGo will be benchmarked against the well established GNU Go, an open-source Go game that provides a consistent level of challenge. The goal is for SapioGo to beat GNU Go at least some small percentage of the time. Although other open-source Go games are available, GNU Go provides a consistent level of difficulty, making it easier to monitor SapioGo's progress.

For more information about GNU Go, or to download a free copy of the source code, visit the following website:

`http://www.gnu.org/software/gnugo/gnugo.html`

1.3.2 Software Requirements

SapioGo will make use of open-source software for everything from its development to its end user interface. This should allow SapioGo to be easily ported to platforms other than those directly supported.

Development Environment

SapioGo will be implemented in C++ using the GNU C++ compiler and will be targeted for a UNIX platform, such as FreeBSD 5.1, as well as for Microsoft Windows XP/2000. The source code is written in strict ANSI C++ and could be easily compiled on any other UNIX or LINUX platform.

Network Protocol

SapioGo will make use of the Go Modem Protocol (GMP) library, created by David Fotland. This library implements the standard JCGA Go Communication Protocol in C and handles all communication and interaction with other Go game clients.

The GMP library, along with sample Go applications using the library, are available for download from the cgoban website, listed below.

User Interface

SapioGo will make use of an existing Go interface program known as cgoban, version 1.9, which will provide a method of initiating new games, playing them, saving them, and restoring saved games. The more recent versions of cgoban do not provide a method for interacting with other automated Go clients, and thus SapioGo will not work with them.

Version 1.9 of cgoban can be downloaded freely at the following website:

`http://www.igoweb.org/~wms/comp/cgoban/`

1.3.3 Hardware Requirements

SapioGo will be able to run smoothly on modest hardware by today's standards. Since SapioGo performs a massive amount of floating-point computations when evaluating the neural network, a computer with a faster processor should be used if available.

Processor:	1.2 GHz Intel Pentium® III or comparable
RAM:	256 MB (preferably 512 MB)
Hard Drive Space:	500 MB
Peripherals:	Mouse, Keyboard
Display:	800x600 16-bit color display

1.4 Design

1.4.1 Go Text Protocol (GTP) Library

The GTP library requires at least version 3.4 of GNU Go in order to use this class properly. Calling this constructor it will spawn an instance of GNU Go and uses a pipe to send and receive GTP commands and responses. When the GTP object is destroyed, or the `kill()` command is called, the GNU Go instance is terminated.

Class `Gtp`

The GTP library also includes a way to talk Go Modem Protocol allowing itself to be used with programs that only talk in GMP, such as cgoban. Calling the `Connect()` command the use is able to use GMP as well as take advantage of Gtp. This is because the Connect command also flags the bool `gmp`, so that when you 'pass' or make a move with GTP you will also send a corresponding GMP request.

Some other features of the GTP library is the `tell_gnugo` command and the `ask_gnugo` command, both of which are private members. These two functions are the basis in which the GTP library is founded. The `tell_gnugo` command actually sends a GTP command to GNU Go. The `ask_gnugo` reads from the return pipe and puts the return pipe in to a private character array called `gnugo_line[]`. The meat of the GTP class is in the `MakeGtpMove(int x, int y, int iswhite)` which takes care of moving pieces on the board, as well as incrementing `numbermoves`. `Islegal(int x,int y,int iswhite)` is similar except that it returns whether the X and Y coordinate of the board is a legal move or not. The `newscore(int isWhite)` is used to calculate the score at the end of the game. If the board is mostly empty then this function may take longer than normal because it is harder to calculate what territory belongs to whom.

1.4.2 Go Modem Protocol (GMP) Library

Our implementation of the `GoGMP` class was modified from code originally done in C by William Shubert, but was not written in an Object Oriented Model. GMP is able to set up a new game, and send moves or passes. You can also query player information with the calls to functions such as `gmp_komi()`, `gmp_size()`, `gmp_handicap()`, `gmp_chineseRules()`, and `gmp_iAmWhite()`. `Gmp_startGame(int size, int handicap, float komi, int chineseRules, int iAmWhite)` sets up the board for a soon to beginning game. Once `gmp_startGame` is called you would then call the `waitForNewGame()` which waits for the other client to initialize their game. `gmp_sendMove(int x, int y)` is used to send moves while `gmp_sendPass()` is used to send passes.

1.4.3 Neural Network Library

The Neural Network library consists of two ADT's, `Neurode` and `Network`. A `Network` object contains a set of `Neurode` objects (its *neuron population*). As such, the `Neurode` class is never meant to be used by the programmer using the Neural Network library. All interaction with neurodes is encapsulated in the `Network` class.

Class `Neurode`

The `Neurode` ADT encapsulates all of the properties of an artificial neuron. It has two public data members, `InputWeights` and `OutputWeights`, which are pointers to arrays of floating point values. These two arrays correspond to the input and output weight vectors, respectively, associated with each neuron.

There are two other public data members, `InputWeightSize` and `OutputWeightSize`, which correspond to the size of the two input weight vectors.

The default constructor, `Neurode()`, creates an empty neurode with no weight vectors, and is never intended to be used. Instead, there is a constructor `Neurode(int InputSize, int OutputSize)` which creates a neurode with the given dimensions. The input and output vectors are sized according to the `InputSize` and `OutputSize` parameters, respectively. All memory is dynamically allocated.

Class `Network`

The `Network` ADT is the class that a programmer would interact with when using the Neural Network Library. Class `Network` encapsulates all of the properties and methods associated with a two-layer, feed-forward, artificial neural network.

Types `InputVector` *and* `OutputVector`. Class `Network` includes the definition of two other ADT's, `InputVector` and `OutputVector`. Both are simply `typedef`'s of the Standard Template Library's `vector` class, of type float (`vector<float>`).

Creating a `Network`. To create a `Network`, the dimensions of the neural network must be specified. These include three parameters: the input size, the hidden size, and the output size. The corresponding constructor is:

```
Network(unsigned int InputLayerSize, unsigned int
  HiddenLayerSize, unsigned int OutputLayerSize)
```

Randomizing and Clearing a `Network`. Once a `Network` has been created, all of the weights associated with each `Neurode` can be set to random floating-point values between 0 and a provided upper-bound. This is done with the `Network`'s public method, `void randomize(float UpperBound)`.

A `Network`'s structure can also be completely cleared, using the public method `void clear(void)`. All `Neurodes` are removed and the dimensions of the `Network` are set to 0.

Iterating a `Network`*.* Feeding an input vector into the network and producing an output vector is called *iterating* the network. Iterating is accomplished using the following public method:

```
void iterate(InputVector &input, OutputVector &output)
```

The `input` vector, although passed by reference for memory considerations, remains unchanged. The `output` vector is cleared and then re-filled with the output of the neural network.

Storing/Loading a `Network` *to/from a File.* Class `Network` also has the ability to store its structure to a file using the `bool save(char *filename)` public method. The network is saved in a simple ASCII text format with one weight (a `float` value) stored per-line. The `save()` method returns `true` if it finished successfully and `false` otherwise.

A `Network` can then be loaded from file using the `bool load(char *filename)` public method. Any existing network structure is cleared before the network is loaded. Similar to the `save()` method, `load()` returns `true` if it finished successfully and `false` otherwise.

There is even a constructor, `Network(char *filename)`, that will create a `Network` loaded from the provided file.

1.4.4 Sane-Network Library

The Sane-Network library consists of a single ADT, class `SaneNetwork`, which adds all the functionality of the SANE algorithm to the `Network` ADT, including selecting a random subset (blueprint) of the neurode population, iterating with a given blueprint, rating the performance of a blueprint, and breeding an entire network.

Class `SaneNetwork`

`SaneNetwork` is a child-class of the `Network` ADT described above. The descriptions below outline the design of the `SaneNetwork` class and how to use it.

Creating a `SaneNetwork`*.* To create a `SaneNetwork`, the dimensions of the underlying neural network must be specified just as for the `Network` class, along with additional parameters required for the SANE algorithm. These include three parameters: the mutation rate, the size of the blueprints selected, and the minimum number of times every neurode must be used in a blueprint before the network may be breed. The corresponding constructor is:

```
SaneNetwork(unsigned int InputLayerSize, unsigned int
HiddenLayerSize, unsigned int OutputLayerSize, unsigned int
MuatationRate, unsigned int BlueprintSize, unsigned int
MinimumUsage)
```

Selecting a Random Blueprint. To select a random blueprint from the `SaneNetwork`, there is a public method that returns a randomized blueprint. Its definition is:

```
blueprint randomizeBlueprint(void);
```

Iterating with a Blueprint. Once a blueprint has been selected, the `SaneNetwork` may be iterated just as a `Network` would, using the following method:

```
void iterate(InputVector &input, OutputVector &output, blueprint
                        *bprint = NULL);
```

Note that the third parameter is a pointer to a blueprint. If no blueprint is provided, the entire network is iterated. Otherwise, only those neurodes in the passed blueprint are used in the iteration.

Rating the Performance of a Blueprint. After a blueprint has been used for a given application, its performance may be rated using a floating-point value and the following public method of `SaneNetwork`:

```
void rate(float rating, blueprint *bprint = NULL);
```

Breeding a `SaneNetwork`. Once all of the neurodes in a `SaneNetwork` have been used in enough applications, the neurodes may be breed using the following public method of `SaneNetwork`: `void breed(void);`

1.4.5 SapioGo

The design of SapioGo is broken into two client programs: `sapiogo` and `sapiogo-coevo`. The first program is designed to play against GNU Go and human opponents, while the second program is designed for coevolution, where SapioGo plays against itself.

Program `sapiogo`

The `sapiogo` client program is relatively simple, since the `GTP` and `SaneNetwork` libraries encapsulate all aspects of playing Go and learning, respectively. The `sapiogo` program works by either creating a new, random network or loading in an existing network from a file. For each generation, random blueprints are selected until all neurodes in the network have been used the minimum number of times (a command-line parameter). For each blueprint that is selected, a game of Go played using only the neurodes in that blueprint. Afterwards, the

neurodes of the blueprint are rated with the score of the game. After all of the neurodes have been used the minimum number of times, the neurodes are breed and a new generation begins. Figure 1.5 complete is pseudo-code for the `sapiogo` program.

```
1  Create a random network or load-in a network from a file
2  For X generations...
3     Create a new SaneNetwork
4     Until all neurodes in the network have been used Y number of times...
5        Select a random blueprint from the SaneNetwork
6        Play a game of Go using only the blueprint selected
7        Rate the neurodes of the blueprint with the score of the game
8     Breed the neurodes of the SaneNetwork
```

Figure 1.5: `sapiogo` Pseudo Code.

Program `sapiogo-coevo`

The `sapiogo-coevo` client program is, like the `sapiogo` program, also relatively simple. The `sapiogo-coevo` program starts by creating either a new, random network or loading in an existing network from a file. For each generation, two random blueprints are selected at a time, one called 'white' and the other called 'black'. A game of Go is played pitting the two blueprints against each other. After the game is finished, the neurodes of each blueprint are rated with the score that blueprint received. This continues until all neurodes have been used the minimum number of times. Afterwards, the neurodes are breed and a new generation begins. Figure 1.6 is complete pseudo-code for the `sapiogo-coevo` program.

```
1  Create a random network or load-in a network from a file
2  For X generations...
3     Create a new SaneNetwork
4     Until all neurodes in the network have been used Y number of times...
5        Select a random blueprint from the SaneNetwork, call it 'white'
6        Select a random blueprint from the SaneNetwork, call it 'black'
7        Play a game of Go between the white and black blueprints
8        Rate the white blueprint with its score
9        Rate the black blueprint with its score
10    Breed the neurodes of the SaneNetwork
```

Figure 1.6: `sapiogo-coevo` Pseudo Code.

1.5 Analysis

1.5.1 Performance: GNU Go

As SapioGoplayed GNU Go, its average score increased as each generation was completed and breed to the next generation, although the increase was usually fractions of points. Both our 5x5 strain and our 9x9 strain showed the same increases to the average score. As SapioGobegins to develope better statagies would not expect clusters of wins right away, we would expect a small amount of wins to occur every couple generations before SapioGobegins to play consistently.

The 5x5 game was able to win against GNU Go. SapioGocurrently is winning around 1 out of 20,000 games on a 5x5 board, or 1 out of 75 generations, thus is not has begun to win consistently. We do expect the number of wins to increase as well as the wins per game to increase. As long as the average score continues to increase as a psudeo-linear rate, (as it is now) we are garenteed more wins.

Both species of boards showed a slow emergence of Statagies. The first statagy shown was SapioGo's beginning set of generationed favored a random opening move, as smarter generations were breed it became more and more intellegent as it possitioned its opening move toward the center of the board. The secound stratagy that emerged was SapioGooriginally favored random placement of stones on the board, the statagy that emerged was the gnugo began to favor placing its stone in groups near stones of its own color, thus making a stonger group together.

1.6 Future Work

Although the school project is complete, SapioGo is far from finished. Below is a list of possible improvements/additions to the project.

1. *Provide an initial set of weights.* Looking at the input vector used by SapioGo, many of the inputs are clearly more important than others. If the weights associated with the less important inputs were started with lower values, and the weights associated with more important inputs were started with higher values, what would the effect be? Would it learn faster? This is analogous to a human player who is advised by other, more experienced Go players.

2. *Test variations of the SANE settings.* There are numerous settings associated with the SANE algorithm, and SapioGo already has the ability to vary these settings via command-line switches. How would SapioGo perform if...

    - the mutation were raised/lowered? The default mutation rate used throughout our testing was 1%. What if a different mutation rate were used, or perhaps, a variable mutation rate?

- the minimum usage required for each neurode in the network were raised/lowered? The default minimum usage used by SapioGo is 5. Would increasing this value raise the learning curve?

3. *Create a SapioGo client capable of playing on the Korean Go Servers.* The ultimate goal of SapioGo is to produce a Go client capable of beating good human opponents. Would playing human opponents during the learning phase help SapioGo to beat its target opponent faster? The Korean Go Servers (KGS, available at `http://kgs.kiseido.com`) have hundreds, sometimes thousands, of eager Go players from all over the world logged-in and playing Go. If an interface were written that allowed SapioGo to play multiple games in parallel across the players logged-in to KGS, a single generation could be played in the time it takes to play a single game.

4. *Provide control over amount of resources used by SapioGo.* While running, SapioGo uses almost all available CPU time. This makes using the computer that it's running on virtually useless for anything else. Adding control over how much CPU time SapioGo used (via the `setpriority()` function in `C`, for example), would make running SapioGo more practical. This feature would also facilitate the next addition...

5. *Create a screen-saver version of SapioGo.* Many people leave their computers on all the time. If a screen-saver version of SapioGo existed (similar to the SETI@home project), the idle time of many computers could be harnessed to further SapioGo's learning.

6. *Create an SGF function for SapioGo.* Currently the sgf functionality being used is provided by SapioGo through GTP has only the ability to write the current board state out to file. Whereas if we would be able to write our own SFG file we would be able to monitor games and hopefully see what kind of statigies are emerging.

7. *Adding more input to the input vector.* After we began breeding our game we realized that there was even more boardstate that we could code into the input vector. Such as is a move legal, as well as other inputs such as intersection state, meaning alive, dead, or captured territory. Ideally adding more information to the board state would give our neural network more working knowledge of the board, however this would drasticly effect how quickly our network would become trained.

Chapter 2

Project Considerations

2.1 Social Impacts of the Project

Because Artificial Intelligence potentially could replace Jobs such as Airline pilots or Air Traffic Controllers, Artificial Intelligence itself could have significant impacts on our society.

However, in our project, we simply wanted to try our hand at a classical computer problem: playing the game of Go. While good Go programs still fail to beat Go masters. An ideal approach to programming go has yet to be mastered. Even assuming that our program was a complete success it would only be a minor step forward in Artificial Intelligence, although it would be a huge break through to the go community.

In conclusion we predict little to no significant social impacts of our project on Society.

2.2 ABET Considerations

Below is a list of goals that the Computer Science curriculum at UC is designed to meet for the Accreditation Board for Engineering Technologies (ABET).

- *A knowledge, and ability to apply, topics in mathematics appropriate to our discipline.* The theory behind neural networks and genetic algorithms is rich in mathematics. For example, evaluating neural networks is equivalent to evaluating a large set of linear equations.

- *An understanding of the basic theory of our discipline.* SapioGo is a project rich in theory that most would consider beyond basic. Neural networks and genetic algorithms represent some of the most popular approaches in artificial intelligence. In addition to that, creating SapioGo absolutely required following good software design principles that have been stressed heavily throughout our computer science curriculum.

- *An ability to design and conduct experiments, as well as analyze and interpret their results, and an understanding of the scientific method.* Simply put, SapioGo *is* a large experiment and this documentation represents our analysis of the results. Further, before we could begin any real design work we created numerous, smaller *proof of concept* experiments to test certain approaches for both the artificial intelligence and the playing-Go portions of the project.
- *An ability to design a system, component, or process to meet desired needs.* We started this project by first defining a strict set of requirements that we agreed our project would have to meet before it could be considered completed. After three months of design and another three of implementation , testing, and analysis, we have met those requirements.
- *An ability to identify, formulate, and solve problems that arise in our discipline and its applications.* During the course of this project, it was discovered that the GMP API was incomplete. Even worse, no complete GMP API was available. Solving this problem required designing and implementing an entirely new API of our own more than $frac23$ of the way into the project.
- *An ability to communicate effectively orally.* Much of this project required brainstorming with Professor Schlipf as well as a great deal of interaction between Kevin and Ryan.
- *An ability to communicate effectively in writing.* It was required for our project that we provide weekly progress reports in addition as well as detailed documentation along the way, including a requirements document, a detailed project time-line, a design document, a user manual, and finally an analysis of our work.
- *An ability to use the techniques, skills, and modern tools of the discipline.* SapioGo like any large project, required a great deal of coordination and effort. Good design approaches, such as breaking our project up into independent sub-projects, made its implementation much easier. The research phase of this project required us to read heavily into contemporary approaches in artificial intelligence, primarily emerging genetic algorithms (such as SANE) and how they perform.

2.3 Team Biographies

Below are brief biographies of Ryan Flannery and Kevin Upchurch, as well as a brief biography of the project advisor, Professor John Schlipf.

2.3.1 Ryan Flannery

Ryan Flannery is a student at the University of Cincinnati majoring in Computer Science in the College of Engineering as well as Mathematics in the College

of Arts & Sciences. Ryan is expected to graduate from UC with his Bachelor's of Science in Computer Science in June of 2004, and to finish his major in mathematics by Spring of 2004.

Ryan is a staunch advocate of open source software and operating systems and is a member of both the UC Free Operating Systems advocacy group (FreeOS) and the Laboratory for Recreational Computing (LaRC). After graduation, Ryan plans to continue in academia and obtain his Ph.D. in computer science.

2.3.2 Kevin Upchurch

Kevin Upchurch is a Computer Science major at the The University of Cincinnati and is planning on graduating June of 2004.

Kevin worked two years at Sanger and Eby Design in Cincinnati working as a web developer, where he created custom web applications.

Kevin followed that co-op experience with a final co-op at Check Mark Inc., where he futhered his intrest in web applications developement.

Kevin recently accepted a full time position at Check Mark Inc. where he will be starting in July.

2.3.3 Professor John Schlipf

John S. Schlipf received his Ph.D. in Mathematics from the University of Wisconsin (Madison) in 1975, specializing in mathematical logic, working mostly in model theory. In the 1980's he drifted to computer science, working primarily in (1) semantics of logic programming and deductive databases, (2) complexity and undecidability results and algorithms in those areas, and (3) algorithms for propositional satisfiability. His most-cited paper is "The well-founded semantics for general logic programming," co-authored with Allen Van Gelder and Kenneth Ross.

2.4 Timeline

The below charts show what tasks were accomplished throughout the course of this project.

2.4.1 Winter Quarter 2004

Winter Quarter 2004, 5 January - 19 March

Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11
Define Requirements	4 Weeks										
Research Topics			9 Weeks								
Design Application									3 Weeks		
Implementation											
Testing											
Learning											
Analysis											
Documentation		3 Weeks									

2.4.2 Spring Quarter 2004

Spring Quarter 2004, 29 March - 4 June

Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11
Define Requirements											
Research Topics	1 Weeks										
Design Application	2 Weeks										
Implementation			3 Weeks								
Testing					2 Weeks						
Learning						3 Weeks					
Analysis							3 Weeks				
Documentation									3 Weeks		

Appendix A

User Manual

A.1 Playing cgoban

To play cgoban you type 'cgoban' into your `xterm` while you are in `xwindows`.

A window will launch as shown allowing you so load a game, play a game vs a human opponent or to play against a computer opponent.

To play another human player simply click the 'New Game' button. In next screen you will be able to set your game parameters. Click ok to begin your games.

A.2 Playing GNU Go

To play GNU Go you need to type 'GNU Go' into an `xterm`.

The board and the current game setting will be displayed once you start the game. To move you simply type the co-ordinate on the board you wish to move your piece and 'pass' when you wish to pass.

A.3 Playing SapioGo

To learn about the different ways that SapioGo can be played. Please type '`./sapiogo -h`' into the `xterm` to review all the flags that can be set.

A.4 Playing SapioGo vs. Human

Start cgoban as shown earlier. This time click the lower right 'Go Modem' button. Select which color you would like to play, on the program side you will need to type the path to the executable, you will also need to specify the -gmp flag, a -threshold flag with a recommended value of 28000 += 1000. Also, if you would like to play against a matured neural network, be sure to specify a net to load by flagging -inNet foo.net

A.5 Playing SapioGo vs. GNU Go

To play against GNU Go. from the command line typo './sapiogo'. If you would like to have it bread use the -learn flag. And if you would like it to start with an existing network use the -inNet foo.net flag to load in that network.

A.6 Playing SapioGo vs. SapioGo

To play evolve against inself, from the command line typo './sapiogo-coevo'. If you would like it to start with an existing network use the -inNet foo.net flag to load in that network.

Appendix B

Implementation

B.1 License

The source code for this project is released under the following BSD-style license.

```
Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

   * Redistributions of source code must retain the above copyright
     notice, this list of conditions and the following disclaimer.
   * Redistributions in binary form must reproduce the above copyright
     notice, this list of conditions and the following disclaimer in the
     documentation and/or other materials provided with the distribution.
   * The name of the authors may not be used to endorse or promote
     products derived from this software without specific prior written
     permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

B.2 Source Code

The sections below contain all of the source code written for this project. The following is a brief overview of the files and the order they appear in:

- Go Text Protocol Library Source Code

  Makefile Build script for the GTP library

  gtp.h Gtp ADT definition

  gtp.c Gtp ADT implementation

  gogmp.h GoGmp ADT definition

  gogmp.c GoGmp ADT implementation

- Neural Network Library Source Code

  Makefile Build script for the Neural Network library

  Neurode.h Neurode ADT definition

  Neurode.cpp Neurode ADT implementation

  Network.h Neural Network ADT definition

  Network.cpp Nueral Network ADT implementation

- SANE Network Library Source Code

  Makefile Build script for the SANE Neural Network library

  SaneNetwork.h SANE Network ADT definition

  SaneNetwork.cpp SANE Network ADT implementation

- SapioGo Source Code

  Makefile Build script for the `sapiogo` and `sapiogo-coevo` programs

  errors.h A set of error constants used throughout the application

  GameSettings.h GameSettings ADT definition

  GameSettings.cpp GameSettings ADT implementation

  GameMove.h A set of routines for deciding what move the SapioGo program should make.

  GameMove.cpp Implementation of `GameMove.h` routines.

  sapiogo.cpp The SapioGo program

  sapiogo-coevo.cpp The co-evolving SapioGo program

B.2.1 Go Text Protocol Library Source Code

```
CPP=g++
CFLAGS=-Wall -O2 -ansi -c
LIB_DIR=/usr/local/lib
INC_DIR=/usr/local/include

INCS=-I/usr/local/include/gtp
LIBS=-L/usr/local/lib -lgtp

gtp.o: gtp.c gtp.h gogmp.o
$(CPP) $(CFLAGS) -o $@ gtp.c

gogmp.o: gogmp.c gogmp.h
$(CPP) $(CFLAGS) -o $@ gogmp.c

### the default build targets
libs: libgtp.a libgtp.so

### create static library
libgtp.a: gtp.o
ar cru $@ gtp.o gogmp.o

### create shared object library
libgtp.so: gtp.o
$(CPP) $(FLAGS) -fpic -shared -o $@ gtp.o gogmp.o

### install libs and includes into std system directories
install: libgtp.a libgtp.so
mkdir -p $(INC_DIR)/gtp
cp gtp.h $(INC_DIR)/gtp/
cp gogmp.h $(INC_DIR)/gtp/
cp libgtp.a $(LIB_DIR)/
cp libgtp.so $(LIB_DIR)/
chmod 444 $(INC_DIR)/gtp/*
chmod 444 $(LIB_DIR)/libgtp.a
chmod 555 $(LIB_DIR)/libgtp.so

### uninstall libs and includes from system directories
uninstall:
rm -rf $(LIB_DIR)/gtp
rm -f $(LIB_DIR)/libgtp.a
rm -f $(LIB_DIR)/libgtp.so

### remove backups
clean:
@echo "removing all object files and the
rm -f *.o *.so *.a *~
```

```
/* gtp.h: declaration of gtp ADT*/
#ifndef GTP_H
#define GTP_H

#include "gogmp.h"

using namespace std;

/*gtp CLASS INTERFACE (DEFINITION)*/
class Gtp
{
public:
//save game to sgf
//gtp_finish_sgftrace
//finish_sgftrace
//gtp_printsgf
//printsgf
//return type for gnugo_move


   Gtp();
   Gtp(int boardsize, int handicap, float komi, int isWhite);
   ~Gtp();
   void Connect(bool &mycolor, unsigned int &myboardsize);
   void playGame();
   void playgtp();
   void setboardsize(int size);
   void showboard();
   void sethandicap(int handicap);
   void MakeGtpMove(int x, int y, int iswhite);
   //void setkomi(char *s);
   void setkomi(float komi);
   void setcolor(int newcolor);
   void finish_sgftrace(char *filename);
   void printsgf(char *filename);
   void clearboard();
   void kill();
   int whatcolor(int x, int y);
   int captures(int iswhite);
   int gnugo_move();//returns a 0 if gnugo moved, else returns a 1 if gnugo passes
   int numberstones(int iswhite);
   int waitandmakemove();//returns a 0 if there was a move or a  1 if there was a pass
   int getnumberofmoves();
   void sendpass(int iswhite);
   int islegal(int x,int y,int iswhite);
   float newscore(int isWhite);
   void endgame();
   int countlib(int x, int y);
private:
   float myKomi;
```

```
   float GetScore(int isWhite);
   void PrintBuffer();
   char* lastparttochar(int y);
   char* firstparttochar(int x);
   int chartolocation(char x);
   int verbose,length,boardsize;
   bool color;/*1 for white, 0 for black*/
   int pfd_a[2];
   int pfd_b[2];
   int himFirst;
   int numbermoves;
   bool gmp;
   GoGmp  myGmp;
   void tell_gnugo(char* command);
   void ask_gnugo(char* gnugo_line);
   void clean_return_pipe();
   FILE *to_gnugo_stream, *from_gnugo_stream;
   void error(const char *msg);
   char gnugo_line[256];
};

#endif
```

```
/*IMPLEMENTATION OF Gtp CLASS*/
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <stdio.h>
#include "gtp.h"
#include "gogmp.h"


//estimate_score
//experimental_score
//final_score
//gtp_accuratelib
//gtp_accurate_approxlib
//gtp_worm_data
//gtp_worm_stones

using namespace std;

Gtp::Gtp(int boardsize, int handicap, float komi, int isWhite){
gmp = false;
numbermoves = 0;
  if (pipe(pfd_a) == -1)
    error("can't open pipe a");
  if (pipe(pfd_b) == -1)
    error("can't open pipe b");
  int forkid = fork();
  switch(forkid) {
  case -1:
    error("fork failed (try chopsticks)");
  case 0:
/* Attach pipe a to stdin */
     if (dup2(pfd_a[0], 0) == -1)
      error("dup pfd_a[0] failed");
/* attach pipe b to stdout" */
    if (dup2(pfd_b[1], 1) == -1)
      error("dup pfd_b[1] failed");
    execlp("gnugo", "gnugo", "--level", "1", "--mode", "gtp", "--quiet", "--boardsize", "9", NULL);
    error("execlp failed");
printf("execlp return:\n");
  }
  /* We use stderr to communicate with the client since stdout is needed. */
  /* Attach pipe a to to_gnugo_stream  */
  to_gnugo_stream = fdopen(pfd_a[1], "w");
  /* Attach pipe b to from_gnugo_stream */
  from_gnugo_stream = fdopen(pfd_b[0], "r");

color = isWhite;
setboardsize(boardsize);
```

```
setkomi(komi);
sethandicap(handicap);
}
Gtp::Gtp(){
gmp = false;
  numbermoves = 0;
  if (pipe(pfd_a) == -1)
    error("can't open pipe a");
  if (pipe(pfd_b) == -1)
    error("can't open pipe b");
  int forkid = fork();
  switch(forkid) {
  case -1:
    error("fork failed (try chopsticks)");
  case 0:
/* Attach pipe a to stdin */
     if (dup2(pfd_a[0], 0) == -1)
      error("dup pfd_a[0] failed");
/* attach pipe b to stdout" */
    if (dup2(pfd_b[1], 1) == -1)
      error("dup pfd_b[1] failed");
    execlp("gnugo", "gnugo", "--level", "1", "--mode", "gtp", "--quiet","--boardsize", "6", "--depth",
    error("execlp failed");
printf("execlp return:\n");
  }
  /* We use stderr to communicate with the client since stdout is needed. */
  /* Attach pipe a to to_gnugo_stream  */
  to_gnugo_stream = fdopen(pfd_a[1], "w");
  /* Attach pipe b to from_gnugo_stream */
  from_gnugo_stream = fdopen(pfd_b[0], "r");
}

Gtp::~Gtp(){
/* not really sure what we want to do for the destructor for right now*/
kill();
}

void Gtp::kill(){
tell_gnugo("quit \n");
clean_return_pipe();
}

void Gtp::clearboard(){
numbermoves = 0;
tell_gnugo("clear_board \n");
clean_return_pipe();
}

int Gtp::gnugo_move(){//returns a 0 if there was a move or a  1 if there was a pass
numbermoves++;
```

```
length = 0;
int returnvalue = 0;
if (color == 1) { /*if we are white then have GTP move as black */
tell_gnugo("genmove black \n");}
else {/*else we are black, and gnugo should move at white*/
tell_gnugo("genmove white \n");}
 while (length != 1) {
ask_gnugo(gnugo_line);
length = strlen(gnugo_line);
//printf("%s",gnugo_line);
if (!strncmp(gnugo_line, "= PASS", 6)){
returnvalue = 1;}
else if (!strncmp(gnugo_line, "? unknown command", 17)){
returnvalue = 1;
printf(gnugo_line);
printf("\nBailing\n");
}
fflush(stdout);
}
return returnvalue;
}

float Gtp::GetScore(int isWhite){

length = 0;
const char delimiters[] = " \t\r\n";
char *token;

tell_gnugo("final_score \n");
ask_gnugo(gnugo_line);
float myscore;

     char test[128];
     char *sep = " +\r\n";
     char *word, *brkt;
printf("%s\n",gnugo_line);
     strcpy(test, gnugo_line);
     for (word = strtok_r(test, sep, &brkt); word; word = strtok_r(NULL, sep, &brkt))
     { if (strlen(word)>1){
if (sscanf(word, "%f", &myscore)!=1)
printf("error\n");
}
     }
token = strtok(gnugo_line, delimiters);
token = strtok(NULL, delimiters);
clean_return_pipe();
char colorthatwon = (char)*token;

if ((colorthatwon=='W') && (isWhite)){
return myscore;}
```

```
else if ((colorthatwon=='B') && (!(isWhite))){
return myscore;}
else {
return (myscore*-1);}

}

void Gtp::finish_sgftrace(char *filename){
length = 0;
tell_gnugo("finish_sgftrace ");
tell_gnugo(filename);
tell_gnugo(" \n");
clean_return_pipe();
}
void Gtp::printsgf(char *filename){
length = 0;
tell_gnugo("printsgf ");
tell_gnugo(filename);
tell_gnugo(" \n");
clean_return_pipe();
}

void Gtp::sethandicap(int handicap){
length = 0;
tell_gnugo("fixed_handicap ");
tell_gnugo(lastparttochar(handicap));
tell_gnugo(" \n");
clean_return_pipe();
}

void Gtp::setcolor(int newcolor){
color = newcolor;
}

void Gtp::setkomi(float komi){
myKomi = komi;
length = 0;
int tensplace = (int)komi;
int decimalplace = ((int)(komi*10) - (tensplace*10));
char s[5];
sprintf(s, "%d.%d", tensplace,decimalplace);
tell_gnugo("komi ");
tell_gnugo(s);
tell_gnugo(" \n");
clean_return_pipe();
}

int Gtp::getnumberofmoves(){
return numbermoves;
}
```

```
int Gtp::waitandmakemove(){//returns a 0 if there was a move or a  1 if there was a pass
int x, y;
int  passes = 0;
GmpResult  message;
const char  *error2;
do
       {
      message = myGmp.gmp_check(1, &x, &y, &error2);
    } while (message == gmp_nothing);
   if (message == gmp_move)
   {
      MakeGtpMove( x, y,!(color));
      passes = 0;
return 0;
   }
   else if (message == gmp_pass)
   {passes++;
numbermoves++;
return 1;
   }
   /* we received some return message that we don't recognize exit the program.*/
   else
   {  fprintf(stderr, "Got a \"%s\" command during game play.  Exiting.\n", message);
      exit(1);
return 0;
    }
}

void Gtp::Connect(bool &mycolor, unsigned int &myboardsize){
gmp = true;
myGmp.gmp_create(0, 1);
myGmp.gmp_startGame(-1, -1, 5.5, -1, -1);
myGmp.waitForNewGame();
boardsize = myGmp.gmp_size();
color = myGmp.gmp_iAmWhite();
mycolor = color;
setboardsize(boardsize);
myboardsize = boardsize;
}

char* Gtp::firstparttochar(int x){
if(x == 18){return "T";}
else if(x == 17){return "S";}
else if(x == 16){return "R";}
else if(x == 15){return "Q";}
else if(x == 14){return "P";}
else if(x == 13){return "O";}
else if(x == 12){return "N";}
else if(x == 11){return "M";}
```

```
else if(x == 10){return "L";}
else if(x == 9){return "K";}
else if(x == 8){return "J";}
else if(x == 7){return "H";}
else if(x == 6){return "G";}
else if(x == 5){return "F";}
else if(x == 4){return "E";}
else if(x == 3){return "D";}
else if(x == 2){return "C";}
else if(x == 1){return "B";}
else if(x == 0){return "A";}
else{
printf("x error %d:\n",x);
exit (0);return "_";
}
}

char* Gtp::lastparttochar(int y){
if(y == 18){return "19";}
else if(y == 17){return "18";}
else if(y == 16){return "17";}
else if(y == 15){return "16";}
else if(y == 14){return "15";}
else if(y == 13){return "14";}
else if(y == 12){return "13";}
else if(y == 11){return "12";}
else if(y == 10){return "11";}
else if(y == 9){return  "10";}
else if(y == 8){return  "9";}
else if(y == 7){return  "8";}
else if(y == 6){return  "7";}
else if(y == 5){return  "6";}
else if(y == 4){return  "5";}
else if(y == 3){return  "4";}
else if(y == 2){return  "3";}
else if(y == 1){return  "2";}
else if(y == 0){return  "1";}
else{printf("y error %d:\n",y);
exit (0);
return "0";}
}

int Gtp::chartolocation(char x){
return (int)x - 65;
}

float Gtp::newscore(int isWhite){
showboard();
int mycaptures = captures(isWhite);
int numstones = numberstones(isWhite);
```

```
float returnscore = mycaptures + numstones;
if (isWhite==1){
returnscore += myKomi;
}
/*tell_gnugo("list_stones white \n");
PrintBuffer();
tell_gnugo("list_stones black \n");
PrintBuffer();*/
//printf(" mc:%d ms:%d\n",mycaptures,numstones);
if (isWhite==1){
//tell_gnugo("final_status_list white_territory \n");
//PrintBuffer();
tell_gnugo("final_status_list white_territory \n");
}else{
//tell_gnugo("final_status_list black_territory \n");
//PrintBuffer();
tell_gnugo("final_status_list black_territory \n");
}
     char test[1024];
     char *sep = " \t=\r\n";
     char *word, *brkt,*token;
     const char delimiters[] = " \t\r\n";
  int myterritory=0;
length = 0;
//printf("start\n");
while (length != 1) {
ask_gnugo(gnugo_line);
//printf("%s",gnugo_line);
length = strlen(gnugo_line);
        strcpy(test, gnugo_line);
fflush(stdout);
for (word = strtok_r(test, sep, &brkt); word; word = strtok_r(NULL, sep, &brkt))
  {myterritory++;}
}



returnscore += myterritory;

//printf(" mc:%d mt:%d ms:%d :mscore:%f \n",mycaptures,myterritory,numstones,returnscore);

length = 0;
tell_gnugo("final_status_list dead \n");

// PrintBuffer();
length = 0;
char buf[1024];
strncpy(buf,"",sizeof(buf));//clear out the buffer
//printf("buf:%s",buf);
int k = 0;
```

```
while (length != 1) {
k++;
ask_gnugo(gnugo_line);
length = strlen(gnugo_line);
//printf("%s",gnugo_line);
//printf("bf:%s\n",buf);
//printf("d:%d\n",k);
strlcat(buf, gnugo_line,sizeof(buf));
//printf("bf:%s\n",buf);
fflush(stdout);
      }
//printf("d2:%d\n",k);
//printf("buf:%s",buf);
strcpy(test, buf);

for (word = strtok_r(test, sep, &brkt); word; word = strtok_r(NULL, sep, &brkt))
{
if ((strlen(word)>=2) && (strlen(word)<=3)) {
//printf("%s\n",word);
tell_gnugo("color ");
tell_gnugo(word);
       tell_gnugo(" \n");
  ask_gnugo(gnugo_line);
//printf("%s",gnugo_line);
token = strtok(gnugo_line, delimiters);
token = strtok(NULL, delimiters);
if ((!strncmp(token, "white", 5)) && (isWhite==1)) {
returnscore--;
}else if ((!strncmp(token, "black", 5)) && (isWhite==1)) {
returnscore++;
}else if ((!strncmp(token, "black", 5)) && (isWhite!=1)) {
returnscore--;
}else if ((!strncmp(token, "white", 5)) && (isWhite!=1)) {
returnscore++;
}else{
printf("FATAL ERROR (%s,%s,%s) 10\n",word,buf,token);
exit(10);
}
clean_return_pipe();
}
}
return returnscore;
}

int Gtp::numberstones(int isWhite){
if (isWhite==1){
tell_gnugo("list_stones white \n");}
else {
tell_gnugo("list_stones black \n");}
ask_gnugo(gnugo_line);
```

```
//printf("%s",gnugo_line);
     char test[128];
     strcpy(test, gnugo_line);
     clean_return_pipe();
     char *sep = " \r\n";
     char *word, *brkt;
     int i=0;
     for (word = strtok_r(test, sep, &brkt); word; word = strtok_r(NULL, sep, &brkt))
     {i++;} i--;

return i;
}

int Gtp::captures(int iswhite){
length = 0;
tell_gnugo("captures ");
if (iswhite==1){tell_gnugo("white ");
}else{tell_gnugo("black ");}
tell_gnugo(" \n");
        ask_gnugo(gnugo_line);
int tmp = 0;
while (length != 1) {
ask_gnugo(gnugo_line);
length = strlen(gnugo_line);
//printf("%s",gnugo_line);
if (atoi(gnugo_line+2)!=0){
tmp = atoi(gnugo_line+2);}
fflush(stdout);
}
return tmp;
}

int Gtp::countlib(int x, int y){/*returns number of liberties, if vertex is empty returns a 0*/
int vertexcolor = whatcolor(x,y);
int returnvalue = 0;
if (vertexcolor==3){
returnvalue = 0;
}else{
char *last =lastparttochar(y);
char *first = firstparttochar(x);
//char *token;
char total[5];
strncpy(total,"",sizeof(total));
strcat(total,first);
strcat(total,last);
length = 0;
tell_gnugo("countlib ");
tell_gnugo(total);
       tell_gnugo(" \n");
returnvalue = atoi(gnugo_line+2);
```

```
        clean_return_pipe();
}
return returnvalue;
}


int Gtp::whatcolor(int x, int y){/*returns 1 if white, 2 if black, 3 if empty, -1 if there is an error
char *last =lastparttochar(y);
char *first = firstparttochar(x);
const char delimiters[] = " \t\r\n";
char *token;
char total[5];
strncpy(total,"",sizeof(total));
strcat(total,first);
strcat(total,last);
length = 0;
tell_gnugo("color ");
tell_gnugo(total);
       tell_gnugo(" \n");
  ask_gnugo(gnugo_line);
token = strtok(gnugo_line, delimiters);
token = strtok(NULL, delimiters);
int returnint;
if (!strncmp(token, "white", 5)) {
returnint=1;
}else if (!strncmp(token, "black", 5)) {
returnint=2;
}else if (!strncmp(token, "empty", 5)) {
returnint=3;
}else{
printf("Fatal error, unknown space %d, %d %d:\n",x,y,total);
returnint=-1;
}
while (length != 1) {
ask_gnugo(gnugo_line);
length = strlen(gnugo_line);
fflush(stdout);
  }
return returnint;
}

int Gtp::islegal(int x,int y,int iswhite){
char *last =lastparttochar(y);
char *first = firstparttochar(x);
char total[5];
strncpy(total,"",sizeof(total));
strcat(total,first);
strcat(total,last);
length = 0;
tell_gnugo("is_legal ");
```

```
if (iswhite){tell_gnugo("white ");
}else{tell_gnugo("black ");}
  tell_gnugo(total);
        tell_gnugo(" \n");
        ask_gnugo(gnugo_line);
int tmp = atoi(gnugo_line+2);
        clean_return_pipe();
return tmp;
}

void Gtp::endgame(){
showboard();
length = 0;
tell_gnugo("final_score \n");
PrintBuffer();
length = 0;
tell_gnugo("new_score \n");
PrintBuffer();
length = 0;
tell_gnugo("experimental_score black \n");
PrintBuffer();
length = 0;
tell_gnugo("experimental_score white \n");
PrintBuffer();
length = 0;
tell_gnugo("estimate_score \n");
PrintBuffer();
}

void Gtp::MakeGtpMove(int x, int y, int iswhite){
numbermoves++;
char *last =lastparttochar(y);
char *first = firstparttochar(x);
char total[5];
strncpy(total,"",sizeof(total));
strcat(total,first);
strcat(total,last);
length = 0;
if (iswhite==1){tell_gnugo("white ");
}else{tell_gnugo("black ");}
tell_gnugo(total);tell_gnugo("\n");
//ask_gnugo(gnugo_line);
clean_return_pipe();
if ((gmp) && (iswhite==color)){
myGmp.gmp_sendMove(x, y);
}
}

void Gtp::sendpass(int iswhite){
numbermoves++;
```

```
if ((gmp) && (iswhite==color)){
myGmp.gmp_sendPass();
}
}

void Gtp::error(const char *msg)
{
  printf("fatal error: %s\n", msg);
  abort();
}

void Gtp::showboard(){length = 0;
      tell_gnugo("showboard \n");
      while (length != 1) {
ask_gnugo(gnugo_line);
length = strlen(gnugo_line);
fprintf(stderr,"%s", gnugo_line);
fflush(stdout);
      }
}

void Gtp::setboardsize(int size){
length = 0;
boardsize = size;
tell_gnugo("boardsize ");
tell_gnugo(lastparttochar(size-1));
tell_gnugo(" \n");
clean_return_pipe();
}

void Gtp::tell_gnugo(char* command){
length = 0;
/*if (verbose) {
printf("%s", command);
fprintf(stderr,"%s", command);}*/
        if (fprintf(to_gnugo_stream, "%s", command) < 0)
              error ("can't write command in to_gnugo_stream");
        fflush(to_gnugo_stream);
}

void Gtp::ask_gnugo(char* gnugo_line){
if (!fgets(gnugo_line, 256, from_gnugo_stream))
                         error("can't get response");
}

void Gtp::clean_return_pipe(){
length = 0;
while (length != 1) {
ask_gnugo(gnugo_line);
//printf("%s",gnugo_line);
```

```
length = strlen(gnugo_line);
fflush(stdout);}
}

void Gtp::PrintBuffer(){
length = 0;
while (length != 1) {
ask_gnugo(gnugo_line);
length = strlen(gnugo_line);
//printf("%s",gnugo_line);
fflush(stdout);
      }
}
```

```
/* src/gmp.h
 * Copyright (C) 1995-1996 William Shubert.
 * Parts of this file are taken from "protocol.c", which is covered under
 *   the copyright notice below.
 * Any code that is not present in the original "proocol.c" is covered under
 *   the copyright notice above.
  protocol.c 1.00
  JCGA Go Communication Protocol
m   copyright(c)    Shuji Sunaga   95.7.9
  original         Standard Go Modem Protocol 1.0
                   by David Fotland
 * Permission granted to use this code for any
 * commercial or noncommercial purposes as long as this
 * copyright notice is not removed.
 * This code was written for Borland C++ 4.0J
*****************************************************/
/*
 * You may use this code in any way you wish as long as you retain the
 *   above copyright notices.
 */

#include <errno.h>

#ifndef GOGMP_H
#define GOGMP_H

#define  GMP_TIMEOUTSECS      60
#define  GMP_MAXSENDSQUEUED   16


/*********************************************************************
 * Data types
 *********************************************************************/
typedef enum  {
  cmd_ack, cmd_deny, cmd_reset, cmd_query, cmd_respond, cmd_move,
  cmd_undo
} Command;

typedef enum  {
  query_game, query_bufSize, query_protocol, query_stones,
  query_bTime, query_wTime, query_charSet, query_rules, query_handicap,
  query_boardSize, query_timeLimit, query_color, query_who,
  query_max
} Query;

typedef enum  {
  gmp_nothing, gmp_move, gmp_pass, gmp_reset, gmp_newGame, gmp_undo, gmp_err
} GmpResult;

typedef struct Gmp_struct  {
```

```
  int  inFile, outFile;
  int  boardSize, sizeVerified;
  int  handicap, handicapVerified;
  float  komi;
  int  chineseRules, rulesVerified;
  int  iAmWhite, colorVerified;
  Query  lastQuerySent;
  int  recvSoFar, sendsQueued, sendFailures;
  int  waitingHighAck;
  time_t  lastSendTime;
  int  myLastSeq, hisLastSeq;
  unsigned char  recvData[4];
  unsigned char  sendData[4];
  struct  {
    int  cmd, val;
  } sendsPending[GMP_MAXSENDSQUEUED];
  int  earlyMovePresent;
  int  earlyMoveX, earlyMoveY;
} Gmp;

   static const char  *commandNames[] = {
  "ACK", "DENY", "RESET", "QUERY", "RESPOND", "MOVE", "UNDO"};

static const char  *queryNames[] = {
  "GAME", "BUFFER SIZE", "PROTOCOL", "STONES",
  "BLACK TIME", "WHITE TIME", "CHAR SET", "RULES", "HANDICAP",
  "BOARD SIZE", "TIME LIMIT", "COLOR", "WHO"};



//typedef struct Command
#ifndef  _GMP_C_
typedef struct Gmp_struct  Gmp;
#endif  /* _GMP_C_ */

using namespace std;

class GoGmp
{
public:
   GoGmp();
   void  gmp_create(int inFile, int outFile);
   void  gmp_sendMove(int x, int y);
   void  gmp_sendUndo(int numUndos);
   void  gmp_sendPass();
   void  gmp_startGame(int size, int handicap, float komi, int chineseRules, int iAmWhite);
   void  processQ();
   void  putCommand(Command cmd, int val);
   void  waitForNewGame();
   float gmp_komi();
```

```
   int   gmp_size();
   int   gmp_handicap();
   int   gmp_chineseRules();
   int   gmp_iAmWhite();
   GmpResult gmp_check(int sleep, int *out1, int *out2, const char  **error);
private:

   GmpResult gotQueryResponse(int val, const char **err);
   GmpResult parsePacket(int *out1, int *out2, const char **error);
   GmpResult processCommand(Command command, int val, int *out1, int *out2, const char **error);
   GmpResult respond(Query query);
   GmpResult getPacket(int *out1, int *out2,const char **error);
   void  askQuery();
   void  gmp_destroy();
   char* gmp_resultString(GmpResult result);
   unsigned char  checksum(unsigned char p[4]);
   Gmp   *ge;
   int   heartbeat();
   int   gmp_debug;
};

#endif
```

```
/* src/gmp.h
 * Copyright (C) 1995-1997 William Shubert.
 * You may use this code in any way you wish as long as you retain the
 *   above copyright notice.
 * This is based on David Fotland's Go Modem Protocol Code and the
 *   "protocol.Z" info file from Bruce Wilcox.  It has been pretty much
 *   completely rewritten now, though. */
#include <time.h>
#include <assert.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <string.h>
#include "gogmp.h"

using namespace std;

#define  GMP_TIMEOUTSECS      60
#define  GMP_MAXSENDSQUEUED   16

/* Get the forward declaration of externally visible functions. */

#define  gmp_verified(ge)  \
  ((ge)->sizeVerified && (ge)->colorVerified && \
   (ge)->handicapVerified && (ge)->rulesVerified)

GoGmp::GoGmp(){
             gmp_debug = 0;
    }

void GoGmp::gmp_create(int inFile, int outFile)  {
  ge = new Gmp();/*(Gmp*) malloc(sizeof(Gmp));*/
  ge->inFile = inFile;
  ge->outFile = outFile;
  ge->boardSize = -1;
  ge->sizeVerified = 0;
  ge->handicap = -1;
  ge->handicapVerified = 0;
  ge->komi = 0.0;
  ge->chineseRules = -1;
  ge->rulesVerified = 0;
  ge->iAmWhite = -1;
  ge->colorVerified = 0;
  ge->lastQuerySent = (Query) 0;
  ge->recvSoFar = 0;
  ge->sendsQueued = 0;
  ge->sendFailures = 0;
  ge->waitingHighAck = 0;
```

```
  ge->lastSendTime = 0;
  ge->myLastSeq = 0;
  ge->hisLastSeq = 0;
  ge->earlyMovePresent = 0;
  /*return(*ge);*/
}

GmpResult  GoGmp::processCommand(Command command, int val, int *out1, int *out2, const char **error)
int  s, x, y;
  switch(command)  {
  case cmd_deny:
    putCommand(cmd_ack, ~0);
    break;
  case cmd_query:
    return((GmpResult)(respond((Query)val)));
    break;
  case cmd_reset:  /* New game. */
    fprintf(stderr, "GMP: Resetted.  New game.\n");
    askQuery();
    return((GmpResult)gmp_reset);
    break;
  case cmd_undo:  /* Take back moves. */
    putCommand(cmd_ack, ~0);
    *out1 = val;
    return((GmpResult)gmp_undo);
    break;
  case cmd_move:
    s = val & 0x1ff;
    if (s == 0)  {
      x = -1;
      y = 0;
    } else if (s == 0x1ff)  {
      x = -2;
      y = 0;
    } else  {
      --s;
      x = (s % ge->boardSize);
      y = ge->boardSize - 1 - (s / ge->boardSize);
    }
    putCommand(cmd_ack, ~0);
    if (x == -1)
      return(gmp_pass);
    else  {
      if (gmp_verified(ge)) {
  *out1 = x;
  *out2 = y;
  return((GmpResult)gmp_move);
      } else {
  assert(ge->earlyMovePresent == 0);
  ge->earlyMovePresent = 1;
```

```
  ge->earlyMoveX = x;
  ge->earlyMoveY = y;
  askQuery();
      }
    }
    break;
  case cmd_respond:
    return((GmpResult)(gotQueryResponse(val, error)));
    break;
  default:  /* Don't understand command. */
    putCommand(cmd_deny, 0);
    break;
  }
  return(gmp_nothing);
}

GmpResult  GoGmp::parsePacket(int *out1, int *out2, const char **error)  {
  int  seq, ack, val;
  Command  command;
  GmpResult  result;

  seq = ge->recvData[0] & 1;
  ack = (ge->recvData[0] & 2) >> 1;
  if (ge->recvData[2] & 0x08)  {  /* Not-understood command. */
    if (gmp_debug) {
      fprintf(stderr, "GMP: Unknown command byte 0x%x received.\n",
        ge->recvData[2]);
    }
    return(gmp_nothing);
  }
  command = (Command)((ge->recvData[2] >> 4) & 7);
  val = ((ge->recvData[2] & 7) << 7) | (ge->recvData[3] & 0x7f);
  if (gmp_debug) {
    if (command == cmd_query) {
      if (val >= query_max) {
  fprintf(stderr, "GMP: Read in command: %s unkown value %d\n",
    commandNames[command], val);
      } else {
    /*printf(stderr, "GMP: Read in command: %s %s\n", commandNames[command], queryNames[val]);*/
      }
    } else {
      fprintf(stderr, "GMP: Read in command: %s\n",
        commandNames[command]);
    }
  }
  if (!ge->waitingHighAck)  {
    if ((command == cmd_ack) ||  /* An ack.  We don't need an ack now. */
  (ack != ge->myLastSeq))  {  /* He missed my last message. */
      fprintf(stderr, "GMP: Unexpected ACK.\n");
      return(gmp_nothing);
```

```
    } else if (seq == ge->hisLastSeq)  {  /* Seen this one before. */
      fprintf(stderr, "GMP: Received repeated message.\n");
      putCommand(cmd_ack, ~0);
    } else  {
      ge->hisLastSeq = seq;
      ge->sendFailures = 0;
      return(processCommand(command, val, out1, out2, error));
    }
  } else  {
    /* Waiting for OK. */
    if (command == cmd_ack)  {
      if ((ack != ge->myLastSeq) || (seq != ge->hisLastSeq))  {
  /* Sequence error. */
  fprintf(stderr, "Sequence error.\n");
  return(gmp_nothing);
      }
      ge->sendFailures = 0;
      ge->waitingHighAck = 0;
      if (!gmp_verified(ge)) {
  askQuery();
      }
      processQ();
    } else if ((command == cmd_reset) && (ge->iAmWhite == -1)) {
fprintf(stderr, "gmp/his last seq = %d\n", seq);
      ge->hisLastSeq = seq;
      ge->waitingHighAck = 0;
      return(processCommand(command, val, out1, out2, error));
    } else if (seq == ge->hisLastSeq)  {
      /* His command is old. */
    } else if (ack == ge->myLastSeq)  {
      ge->sendFailures = 0;
      ge->waitingHighAck = 0;
      ge->hisLastSeq = seq;
      result = processCommand(command, val, out1, out2, error);
      processQ();
      return(result);
    } else  {
      /* Conflict with opponent. */
      fprintf(stderr, "Sending conflict.\n");
      ge->myLastSeq = 1 - ge->myLastSeq;
      ge->waitingHighAck = 0;
      processQ();
    }
  }
  return(gmp_nothing);
}

GmpResult  GoGmp::gmp_check(int sleep, int *out1, int *out2, const char **error)  {
  fd_set  readReady;
  struct timeval  noTime;
```

```
  int  intDummy;
  const char  *charPtrDummy;
  GmpResult  result;
  if (out1 == NULL)
    out1 = &intDummy;
  if (out2 == NULL)
    out2 = &intDummy;
  if (error == NULL)
    error = &charPtrDummy;
  if (gmp_verified(ge) && ge->earlyMovePresent) {
    *out1 = ge->earlyMoveX;
    *out2 = ge->earlyMoveY;
    ge->earlyMovePresent = 0;
    if (gmp_debug) {
      fprintf(stderr, "GMP: Returning early move.\n");
    }
    return(gmp_move);
  }
  *out1 = 0;
  *out2 = 0;
  *error = NULL;
  do  {
    if (time(NULL) != ge->lastSendTime)  {
      if (!heartbeat())  {
  *error = "GMP Timeout";
  return(gmp_err);
      }
    }
    FD_ZERO(&readReady);
    FD_SET(ge->inFile, &readReady);
    noTime.tv_usec = 0;
    if (sleep)
      noTime.tv_sec = 1;
    else
      noTime.tv_sec = 0;
    select(ge->inFile + 1, &readReady, NULL, NULL, &noTime);
    if (!sleep && !FD_ISSET(ge->inFile, &readReady))
      return(gmp_nothing);
    result = getPacket(out1, out2, error);
  } while (result == gmp_nothing);
  return(result);
}

GmpResult  GoGmp::getPacket(int *out1, int *out2,const char **error)  {
  unsigned char  charsIn[4], c;
  int  count = 0, cNum;
  static char  errOut[200];
  count = read(ge->inFile, charsIn, 4 - ge->recvSoFar);
  if (count <= 0)  {
    sprintf(errOut, "System error %d.", errno);
```

```
    *error = errOut;
    return(gmp_err);
  }
  for (cNum = 0;  cNum < count;  ++cNum)  {
    c = charsIn[cNum];
    if (!ge->recvSoFar)  {
      /* idle, looking for start of packet */
      if ((c & 0xfc) == 0)  {  /* start of packet */
        ge->recvData[0] = c;
        ge->recvSoFar = 1;
      } else {
        if (gmp_debug) {
        fprintf(stderr, "GMP: Received invalid packet.\n");
        }
      }
    } else  {
      /* We're in the packet. */
      if ((c & 0x80) == 0)  {  /* error */
  if (gmp_debug) {
    fprintf(stderr, "GMP: Received invalid packet.\n");
  }
  ge->recvSoFar = 0;
  if ((c & 0xfc) == 0)  {
    ge->recvData[ge->recvSoFar++] = c;
  }
      } else  {
  /* A valid character for in a packet. */
  ge->recvData[ge->recvSoFar++] = c;
  if (ge->recvSoFar == 4)  {  /* check for extra bytes */
    assert(cNum + 1 == count);
    ge->recvSoFar = 0;
    if (checksum(ge->recvData) == ge->recvData[1])
      return(parsePacket(out1, out2, error));
  }
      }
    }
  }
  return(gmp_nothing);
}

GmpResult  GoGmp::respond(Query query)  {
  int  response;
  int  wasVerified;
  wasVerified = gmp_verified(ge);
  if (query & 0x200)  {
    /* Do you support this extended query? */
    response = 0;  /* No. */
  } else  {
    ge->waitingHighAck = 1;
    switch(query)  {
```

```
  case query_game:
    response = 1;  /* GO */
    break;
  case query_rules:
    if (ge->chineseRules == -1) {
response = 0;
    } else {
ge->rulesVerified = 1;
if (ge->chineseRules == 1)
  response = 2;
else
  response = 1;
    }
    break;
  case query_handicap:
    if (ge->handicap == -1)
response = 0;
    else {
ge->handicapVerified = 1;
response = ge->handicap;
if (response == 0)
  response = 1;
    }
    break;
  case query_boardSize:
    if (ge->boardSize == -1) {
response = 0;
    } else {
response = ge->boardSize;
ge->sizeVerified = 1;
    }
    break;
  case query_color:
    if (ge->iAmWhite == -1) {
response = 0;
    } else  {
ge->colorVerified = 1;
if (ge->iAmWhite)
  response = 1;
else
  response = 2;
    }
    break;
  default:
    response = 0;
    break;
  }
}
putCommand((Command)cmd_respond, response);
if (!wasVerified && gmp_verified(ge)) {
```

```
    fprintf(stderr, "GMP: New game ready.\n");
    return(gmp_newGame);
  } else {
    return(gmp_nothing);
  }
}

GmpResult  GoGmp::gotQueryResponse(int val, const char **err)  {
  static const char  *ruleNames[] = {"Japanese", "Chinese"};
  static const char  *colorNames[] = {"Black", "White"};
  static char errOut[200];
  switch(ge->lastQuerySent)  {
  case query_handicap:
    if (val <= 1)
      --val;
    if (ge->handicap == -1) {
      if (val == -1) {
  sprintf(errOut, "Neither player knows what the handicap should be.");
  *err = errOut;
  return((GmpResult)gmp_err);
      } else {
  ge->handicap = val;
  ge->handicapVerified = 1;
      }
    } else {
      ge->handicapVerified = 1;
      if ((val != -1) && (val != ge->handicap))  {
  sprintf(errOut, "Handicaps do not agree; I want %d, he wants %d.",ge->handicap, val);
  *err = errOut;
  return((GmpResult)gmp_err);
      }
    }
    break;
  case query_boardSize:
    if (ge->boardSize == -1)  {
      if (val == 0)  {
  sprintf(errOut, "Neither player knows what the board size should be.");
  *err = errOut;
  return((GmpResult)gmp_err);
      } else {
  ge->boardSize = val;
  ge->sizeVerified = 1;
      }
    } else {
      ge->sizeVerified = 1;
      if ((val != 0) && (val != ge->boardSize))  {
  sprintf(errOut, "Board sizes do not agree; I want %d, he wants %d.",  ge->boardSize, val);
  *err = errOut;
  return((GmpResult)gmp_err);
      }
```

```
  }
  break;
case query_rules:
  if (ge->chineseRules == -1) {
    if (val == 0) {
sprintf(errOut, "Neither player knows what rule set to use.");
*err = errOut;
return((GmpResult)gmp_err);
    } else {
ge->chineseRules = val - 1;
ge->rulesVerified = 1;
    }
  } else {
    ge->rulesVerified = 1;
    if (val != 0)  {
if (ge->chineseRules != (val == 2))  {
  sprintf(errOut, "Rule sets do not agree; I want %s, he wants %s.",  ruleNames[ge->chineseRules], r
  *err = errOut;
  return((GmpResult)gmp_err);
}
    }
  }
  break;
case query_color:
  if (ge->iAmWhite == -1)  {
    if (val == 0)  {
sprintf(errOut, "Neither player knows who is which color.");
*err = errOut;
return((GmpResult)gmp_err);
    } else {
ge->iAmWhite = !(val == 1);
ge->colorVerified = 1;
    }
  } else {
    ge->colorVerified = 1;
    if (val != 0)  {
if (ge->iAmWhite == (val == 1))  {
  sprintf(errOut, "Colors do not agree; we both want to be %s.",
  colorNames[ge->iAmWhite]);
  *err = errOut;
  return((GmpResult)gmp_err);
}
    }
  }
  break;
default:
  break;
}
if (!gmp_verified(ge)) {
  askQuery();
```

```
    return((GmpResult)gmp_nothing);
  } else {
    putCommand((Command)cmd_ack, ~0);
    fprintf(stderr, "GMP: New game ready.\n");
    return((GmpResult)gmp_newGame);
  }
}
unsigned char  GoGmp::checksum(unsigned char p[4]){
/*char*  GoGmp::checksum(unsigned char p[4])  {*/
/*  char* sum;
  sum = p[0] + p[2] + p[3];
  sum |= 0x80;
  return(sum);*/
  unsigned char sum;
  sum = p[0] + p[2] + p[3];
  sum |= 0x80;  /* set sign bit */
  return(sum);
}

char* GoGmp::gmp_resultString(GmpResult result)  {
  char  *names[] = {
    "Nothing", "Move", "Pass", "Reset", "New game", "Undo", "Error"};
  assert(result <= gmp_err);
  return(names[result]);
}

int  GoGmp::heartbeat()  {
  Command  cmd;
  if (ge->waitingHighAck)  {
    if (++ge->sendFailures > GMP_TIMEOUTSECS)  {
      return(0);
    } else  {
      if (gmp_debug) {
  cmd = (Command)((ge->sendData[2] >> 4) & 7);
  if (cmd == cmd_query)
    fprintf(stderr, "GMP: Sending command: %s %s (retry)\n", commandNames[cmd], queryNames[ge->sendDat
  else
    fprintf(stderr, "GMP: Sending command: %s (retry)\n", commandNames[cmd]);
      }
      write(ge->outFile, ge->sendData, 4);
    }
  }
  return(1);
}

int  GoGmp::gmp_size()  {
  return(ge->boardSize);
}

int  GoGmp::gmp_handicap()  {
```

```
  return(ge->handicap);
}

int  GoGmp::gmp_chineseRules()  {
  return(ge->chineseRules);
}

int  GoGmp::gmp_iAmWhite()  {
  return(ge->iAmWhite);
}

float  GoGmp::gmp_komi()  {
  return(ge->komi);
}

void GoGmp::waitForNewGame(){
   GmpResult  result;
   const char  *err;
   /* sit and wait sit */
   do
   { result = gmp_check(1, NULL, NULL, &err);
   } while ((result == gmp_nothing) || (result == gmp_reset));

   if (result != gmp_newGame)
   {  fprintf(stderr, "dummy: Error \"%s\" occurred.\n", err);
      exit(1);}
}

void  GoGmp::askQuery()  {
  if (!ge->rulesVerified) {
    ge->lastQuerySent = (Query)query_rules;
  } else if (!ge->sizeVerified) {
    ge->lastQuerySent = (Query)query_boardSize;
  } else if (!ge->handicapVerified) {
    ge->lastQuerySent = (Query)query_handicap;
  } else {
    assert(!ge->colorVerified);
    ge->lastQuerySent = (Query)query_color;
  }
  putCommand((Command)cmd_query, (int)(ge->lastQuerySent));
}

void  GoGmp::putCommand(Command cmd, int val)  {
  if (ge->waitingHighAck && (cmd != cmd_ack) && (cmd != cmd_respond) && (cmd != cmd_deny))  {
    if (ge->sendsQueued < 1024)  {
      ge->sendsPending[ge->sendsQueued].cmd = cmd;
      ge->sendsPending[ge->sendsQueued].val = val;
      ++ge->sendsQueued;
    } else  {
      fprintf(stderr, "GMP: Send buffer full.  Catastrophic error.");
```

```
      exit(1);
    }
    return;
  }
  if ((cmd == cmd_ack) && (ge->sendsQueued))  {
    ge->waitingHighAck = 0;
    processQ();
    return;
  }
  if (cmd != cmd_ack)
    ge->myLastSeq ^= 1;
  ge->sendData[0] = ge->myLastSeq | (ge->hisLastSeq << 1);
  ge->sendData[2] = 0x80 | (cmd << 4) | ((val >> 7) & 7);
  ge->sendData[3] = 0x80 | val;
  ge->sendData[1] = checksum(ge->sendData);
  ge->lastSendTime = time(NULL);
  if (gmp_debug) {
    if (cmd == cmd_query)
      fprintf(stderr, "GMP: Sending command: %s %s\n", commandNames[cmd], queryNames[val]);
    else
      fprintf(stderr, "GMP: Sending command: %s\n", commandNames[cmd]);
  }
  write(ge->outFile, ge->sendData, 4);
  ge->waitingHighAck = (cmd != cmd_ack);
  return;
}
void  GoGmp::gmp_startGame(int size, int handicap, float komi, int chineseRules, int iAmWhite)  {
  assert((size == -1) || ((size > 1) && (size <= 22)));
  assert((handicap >= -1) && (handicap <= 27));
  assert((chineseRules >= -1) && (chineseRules <= 1));
  assert((iAmWhite >= -1) && (iAmWhite <= 1));
  ge->boardSize = size;
  ge->sizeVerified = 0;
  ge->handicap = handicap;
  ge->handicapVerified = 0;
  ge->komi = komi;
  ge->chineseRules = chineseRules;
  ge->rulesVerified = 0;
  ge->iAmWhite = iAmWhite;
  ge->colorVerified = 0;
  ge->earlyMovePresent = 0;
  if (iAmWhite != 1) {
    putCommand(cmd_reset, 0);
  }
}

void  GoGmp::gmp_sendPass()  {
  int  arg;
  if (ge->iAmWhite)
    arg = 0x200;
```

```
  else
    arg = 0;
  putCommand(cmd_move, arg);
}

void  GoGmp::processQ()  {
  int  i;
  if (!ge->waitingHighAck && ge->sendsQueued)  {
    putCommand((Command)(ge->sendsPending[0].cmd),ge->sendsPending[0].val);
    --ge->sendsQueued;
    for (i = 0;  i < ge->sendsQueued;  ++i)  {
      ge->sendsPending[i] = ge->sendsPending[i + 1];
    }
  }
}

void  GoGmp::gmp_sendMove(int x, int y)  {
  int  val;
  val = x + ge->boardSize * (ge->boardSize - 1 - y) + 1;
  if (ge->iAmWhite)
    val |= 0x200;
  putCommand(cmd_move, val);
}

void  GoGmp::gmp_destroy()  {
  free(ge);
}

void  GoGmp::gmp_sendUndo(int numUndos)  {
  putCommand(cmd_undo, numUndos);
}
```

B.2.2 Neural Network Library Source Code

```
#
# Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are
# met:
#
#    *  Redistributions of source code must retain the above copyright
#       notice, this list of conditions and the following disclaimer.
#    *  Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in the
#       documentation and/or other materials provided with the distribution.
#    *  The name of the authors may not be used to endorse or promote
#       products derived from this software without specific prior written
#       permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
# LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.
#

CPP=g++
CFLAGS=-Wall -O2 -ansi -c
LIB_DIR=/usr/local/lib
INC_DIR=/usr/local/include

INCS=-I/usr/local/include/nnet
LIBS=-static -L/usr/local/lib -lnnet

### the default build targets
libs: libnnet.a libnnet.so


###
### the neural network structure is built below
###
Network.o: Network.cpp Network.h Neurode.o
$(CPP) $(CFLAGS) -o $@ Network.cpp
```

```
Neurode.o: Neurode.cpp Neurode.h
$(CPP) $(CFLAGS) -o $@ Neurode.cpp


###
### some test applications
###
tests: TestNetwork TestNetworkLoad StressNetwork StressNetworkLoad

### test simple use of neural network
TestNetwork: TestNetwork.o
$(CPP) -o $@ TestNetwork.o $(LIBS)

TestNetwork.o: TestNetwork.cpp
$(CPP) $(CFLAGS) $(INCS) -o $@ TestNetwork.cpp

### test loading in an existing network
TestNetworkLoad: TestNetworkLoad.o
$(CPP) -o $@ TestNetworkLoad.o $(LIBS)

TestNetworkLoad.o: TestNetworkLoad.cpp
$(CPP) $(CFLAGS) $(INCS) -o $@ TestNetworkLoad.cpp

### stress testing the size of neural networks
StressNetwork: StressNetwork.o
$(CPP) -o $@ StressNetwork.o $(LIBS)

StressNetwork.o: StressNetwork.cpp
$(CPP) $(CFLAGS) $(INCS) -o $@ StressNetwork.cpp

### stress testing the load of large neural networks
StressNetworkLoad: StressNetworkLoad.o
$(CPP) -o $@ StressNetworkLoad.o $(LIBS)

StressNetworkLoad.o: StressNetworkLoad.cpp
$(CPP) $(CFLAGS) $(INCS) -o $@ StressNetworkLoad.cpp


### create static library
libnnet.a: Network.o
ar cr $@ Network.o Neurode.o

### create shared object library
libnnet.so: Network.o
$(CPP) $(FLAGS) -fpic -shared -o $@ Network.o Neurode.o

### install libs and includes into std system directories
install: libnnet.a libnnet.so
mkdir -p $(INC_DIR)/nnet
cp Network.h $(INC_DIR)/nnet/
```

```
cp Neurode.h $(INC_DIR)/nnet/
cp libnnet.a $(LIB_DIR)/
cp libnnet.so $(LIB_DIR)/
chmod 444 $(INC_DIR)/nnet/*
chmod 444 $(LIB_DIR)/libnnet.a
chmod 555 $(LIB_DIR)/libnnet.so

### uninstall libs and includes from system directories
uninstall:
rm -rf $(LIB_DIR)/nnet
rm -f $(LIB_DIR)/libnnet.a
rm -f $(LIB_DIR)/libnnet.so

### remove backups
clean:
@echo "removing all object files and the target executable"
rm -f *.o TestNetwork TestNetworkLoad StressNetwork StressNetworkLoad
rm -f libnnet.a libnnet.so
rm -f *.net

### remove all binaries and any core files
clean-all: clean
rm -rf *.core
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * Neurode.h - Neurode (artificial neuron) ADT definition for Neural Networks
 */

#ifndef NEURODE
#define NEURODE

#include <stdio.h>   // for *printf, *scanf, etc
#include <stdlib.h>  // for exit

// NOTE: this is a very simple ADT with all public members
class Neurode
{
   public:
      // the below two vectors represent the Neurode's chromosome
      float *InputWeights;       // input weight vector
      float *OutputWeights;      // output weight vector

      unsigned int InputWeightsSize;    // size of input weight vector
      unsigned int OutputWeightsSize;   // size of output weight vector
```

```
      // default constructor
      Neurode(void);

      // this constructor specifies the sizes of the two weight vectors
      Neurode(int InputSize, int OutputSize);

      // proper destructor
      ~Neurode(void);

      // output neurode to stdout - used mostly for debugging
      void output(void) const;
};

#endif
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * Neurode.cpp - Neurode ADT implementation
 */

#include "Neurode.h"

////////////////////////////////////////////////////////////////////////////
// default constructor
Neurode::Neurode(void)
{  // create and empty neurode
   InputWeights = OutputWeights = NULL;
   InputWeightsSize = OutputWeightsSize = 0;
}

////////////////////////////////////////////////////////////////////////////
// simple constructor
Neurode::Neurode(int InputSize, int OutputSize)
{  // set size of weight arrays to the provided sizes
   InputWeightsSize = InputSize;
   OutputWeightsSize = OutputSize;
```

```
   // create new arrays of floats for the input/output chromosomes
   InputWeights = new float[InputWeightsSize];
   OutputWeights = new float[OutputWeightsSize];

   // ensure that the memory above was allocated
   if (InputWeights == NULL || OutputWeights == NULL)
   {  fprintf(stderr,"error: could not allocate requisite memory.\n");
      exit(1);
   }
}

////////////////////////////////////////////////////////////////////////////
// proper destructor
Neurode::~Neurode(void)
{  // delete the chromosomes that were allocated in the constructor
   if (InputWeights != NULL && OutputWeights != NULL)
   {  delete InputWeights;
      delete OutputWeights;
   }
}

////////////////////////////////////////////////////////////////////////////
// output neurode to stdout -- used only for debugging
void Neurode::output(void) const
{  // output input chromosome
   for (unsigned int i = 0; i < InputWeightsSize; i++)
      printf("I[%d]: %f\n", i, InputWeights[i]);

   // output output chromosome
   for (unsigned int i = 0; i < OutputWeightsSize; i++)
      printf("O[%d]: %f\n", i, OutputWeights[i]);
}
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * Network.h - 2-layer Feed-Forward Neural Network ADT definition
 */

#ifndef NETWORK
#define NETWORK

#include <stdio.h>      // for *printf, *scanf, etc
#include <stdlib.h>     // for rand, exit
#include <vector>       // for std::vector

// also need the Neurode ADT provided with this library
#include "Neurode.h"

// the following typedef's make creating input/output vectors for a neural
// network a bit easier
typedef std::vector<float> InputVector;
typedef std::vector<float> OutputVector;
```

```
class Network
{
   private:
      unsigned int InputSize;          // size of input layer
      unsigned int HiddenSize;         // number of neurodes in hidden layer
      unsigned int OutputSize;         // size of output layer

   protected:
      std::vector<Neurode*> Neurodes;  // neurode population

      // generate random floating pont value between 0 and upper bound
      float getRandomFloat(float upper) const;

      // generate random weight between 0 and upper bound (identical to the
      // above method)
      float getRandomWeight(float upper) const;

   public:
      // default constructor
      Network(void);

      // create network from specified file
      Network(char *filename);

      // construct network with given size parameters
      Network(unsigned int InputLayerSize, unsigned int HiddenLayerSize,
                                           unsigned int OutputLayerSize);

      // proper destructor
      ~Network(void);

      // retrieve dimensions of the network
      unsigned int getInputSize(void) const;    // return input layer size
      unsigned int getHiddenSize(void) const;   // return hidden layer size
      unsigned int getOutputSize(void) const;   // return output layer size
      Neurode* getNeurode(unsigned int) const;  // return pointer to neurode

      // clear any existing network and its dimensions
      void clear(void);

      // assign random weights to each neurode
      void randomize(float upper);

      // some standard methods to save / load a network to / from a file
      bool save(char *filename);
      bool load(char *filename);

      // this method accepts an input vector and produces an output vector
      void iterate(InputVector &input, OutputVector &output);
```

```
      // output the network to stdout -- used primarily for debugging
      void output(void) const;
};

#endif
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * Network.cpp - Neural Network ADT implementation
 */

#include "Network.h"

////////////////////////////////////////////////////////////////////////////
// default constructor - create an empty network
Network::Network(void)
: InputSize(0)
, HiddenSize(0)
, OutputSize(0)
{ }

////////////////////////////////////////////////////////////////////////////
// constructor - build neural network from given file
Network::Network(char *filename)
{ load(filename); }

////////////////////////////////////////////////////////////////////////////
```

```
// construct neural network with the given dimensions
Network::Network(unsigned int InputLayerSize, unsigned int HiddenLayerSize,
                                              unsigned int OutputLayerSize)
: InputSize(InputLayerSize)
, HiddenSize(HiddenLayerSize)
, OutputSize(OutputLayerSize)
{  // add specified number of new neurodes to the hidden layer
   for (unsigned int i = 0; i < HiddenSize; i++)
   {  // add a new neurode to the population with the given input/output size
      Neurodes.push_back(new Neurode(InputSize, OutputSize));

      // check to make sure we were able to allocate the requisite memory
      if (Neurodes[i] == NULL)
      {  fprintf(stderr, "error: unable to allocate enough memory.\n");
         exit(-1);
      }
   }
}

////////////////////////////////////////////////////////////////////////////
// proper destructor - clear any neural network structure
Network::~Network(void)
{ clear(); }

////////////////////////////////////////////////////////////////////////////
// generate random floating point value between 0 and the upper bound given
float Network::getRandomFloat(float upper) const
{  // this method is far more 'random' than using the mod (%) method - that
   // method focusses only on the lower few bits of the number returned
   double x = ((double) rand() / (double) RAND_MAX);

   // now, return r * upper (and make sure it's positive!)
   return (x < 0 ? x * upper * -1 : x * upper);
}

////////////////////////////////////////////////////////////////////////////
// generate random weight for a chromosome (just call above function)
float Network::getRandomWeight(float upper) const
{ return getRandomFloat(upper); }

////////////////////////////////////////////////////////////////////////////
// return various properties of the neural network
unsigned int Network::getInputSize(void) const { return InputSize; }
unsigned int Network::getHiddenSize(void) const { return HiddenSize; }
unsigned int Network::getOutputSize(void) const { return OutputSize; }
Neurode* Network::getNeurode(unsigned int i) const { return Neurodes[i]; }

////////////////////////////////////////////////////////////////////////////
// clear any existing network and its dimensions
void Network::clear(void)
```

```
{  // remove all neurodes
   for (unsigned int i = 0; i < Neurodes.size(); i++)
      delete Neurodes[i];

   // zero-out the neurodes list
   Neurodes.clear();

   // set all dimensions to zero
   InputSize = HiddenSize = OutputSize = 0;
}

//////////////////////////////////////////////////////////////////////////
// assign random weights to each neurode with the provided upper bound
void Network::randomize(float upper)
{  // step through neuron population and generate random weights with the
   // provided upperbound for both input and output connections
   for (unsigned int i = 0; i < Neurodes.size(); i++)
   {
      // generate set of random input weights
      for (unsigned int j = 0; j < Neurodes[i]->InputWeightsSize; j++)
         Neurodes[i]->InputWeights[j] = getRandomWeight(upper);

      // generate set of random output weights
      for (unsigned int k = 0; k < Neurodes[i]->OutputWeightsSize; k++)
         Neurodes[i]->OutputWeights[k] = getRandomWeight(upper);
   }
}

//////////////////////////////////////////////////////////////////////////
// save neural network to a file - return true iff save was successful
bool Network::save(char *filename)
{
   FILE  *fptr;

   // open file for writing -- if error, alert stderr and return false
   if (!(fptr = fopen(filename, "w")))
   {  fprintf(stderr, "error: unable to open '%s'\n", filename);
      return false;
   }

   // first, write dimensions of neural network to file
   fprintf(fptr, "%d %d %d\n", InputSize, HiddenSize, OutputSize);

   // next, write each neurode to the file
   for (unsigned int i = 0; i < Neurodes.size(); i++)
   {
      // first, write out the neurodes input weights
      for (unsigned int j = 0; j < Neurodes[i]->InputWeightsSize; j++)
      {
         if (fprintf(fptr, "I[%d]: %f\n", j, Neurodes[i]->InputWeights[j]) < 0)
```

```
         {  fprintf(stderr, "error: unable to write to '%s'\n", filename);
            return false;
         }
      }

      // next, write out the neurodes output weights
      for (unsigned int k = 0; k < Neurodes[i]->OutputWeightsSize; k++)
      {
         if (fprintf(fptr, "O[%d]: %f\n", k, Neurodes[i]->OutputWeights[k]) < 0)
         {  fprintf(stderr, "error: unable to write to '%s'\n", filename);
            return false;
         }
      }
   }

   // close file
   if (fclose(fptr))
   {  fprintf(stderr, "error: unable to succefully close '%s'\n", filename);
      return false;
   }

   return true;
}

////////////////////////////////////////////////////////////////////////////
// load in neural network from file - return true iff load was successful
bool Network::load(char *filename)
{
   FILE  *fptr;
   int    index;
   float  weight;

   // open file for reading -- if error, alert stderr and return false
   if (!(fptr = fopen(filename, "r")))
   {  fprintf(stderr, "error: unable to open '%s'\n", filename);
      return false;
   }

   // clear any existing network
   clear();

   // first read in dimensions of neural network
   if (fscanf(fptr, "%d %d %d\n", &InputSize, &HiddenSize, &OutputSize) < 3)
   {  fprintf(stderr, "error: '%s' is malformed\n", filename);
      return false;
   }

   // create the requisite number of neurodes and read-in the data for each
   for (unsigned int i = 0; i < HiddenSize; i++)
   {
```

```
      // add new neurode to our neurode population
      Neurodes.push_back(new Neurode(InputSize, OutputSize));

      // read-in the set of input weights
      for (unsigned int j = 0; j < InputSize; j++)
      {
         if (fscanf(fptr, "I[%d]: %f\n", &index, &weight) < 2)
         {  fprintf(stderr, "error: '%s' is malformed\n", filename);
            return false;
         }

         Neurodes[i]->InputWeights[j] = weight;
      }

      // read-in the set of output weights
      for (unsigned int k = 0; k < OutputSize; k++)
      {
         if (fscanf(fptr, "O[%d]: %f\n", &index, &weight) < 2)
         {  fprintf(stderr, "error: '%s' is malformed\n", filename);
            return false;
         }

         Neurodes[i]->OutputWeights[k] = weight;
      }
   }

   // close file
   if (fclose(fptr))
   {  fprintf(stderr, "error: unable to succefully close '%s'\n", filename);
      return false;
   }

   return true;
}

////////////////////////////////////////////////////////////////////////////
// do an iteration - that is, feed an input vector (iVector) into the neural
// network and return an output vector (oVector).  note that the vectors are
// both passed by reference.  this is to cut down on the memory overhead of
// copying the vectors for large applications (such as a game of Go!)
void Network::iterate(InputVector &iVector, OutputVector &oVector)
{
   std::vector<float> HiddenLayer;
   float input, output;

   // clear the output vector of any existing data
   oVector.clear();

   // if input vector is of incorrect size, error out of program
   if (iVector.size() != getInputSize())
```

```
   {  fprintf(stderr, "error: InputVector is of incorrect size.\n");
      exit(1);
   }

   // for each neurode, calculate it's input base on the InputVector and the
   // neurode's weights.   add the input to the hidden layer vector
   for (unsigned int i = 0, j; i < Neurodes.size(); i++)
   {  // multiply neurodes input chromosome against entire weight vector
      for (j = 0, input = 0; j < getInputSize(); j++)
         input += iVector[j] * Neurodes[i]->InputWeights[j];

      HiddenLayer.push_back(input);
   }

   // for each output neurode, calculate it's input based on the hidden layer
   // input (calculated above) and the weights on the output connections
   for (unsigned int i = 0, j; i < getOutputSize(); i++)
   {  // multiply each neurodes output (from above) by its output chromosome
      for (j = 0, output = 0; j < Neurodes.size(); j++)
         output += HiddenLayer[j] * Neurodes[j]->OutputWeights[i];

      oVector.push_back(output);
   }
}

////////////////////////////////////////////////////////////////////////////
// output neural network to stdout - used only for debugging
void Network::output(void) const
{  // cycle through the neurode population and output each neurode
   for (unsigned int i = 0; i < Neurodes.size(); i++)
   {  printf("Neurode %d\n", i);
      Neurodes[i]->output();
   }
}
```

B.2.3 SANE Network Library Soruce Code

```
#
# Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are
# met:
#
#    *  Redistributions of source code must retain the above copyright
#       notice, this list of conditions and the following disclaimer.
#    *  Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in the
#       documentation and/or other materials provided with the distribution.
#    *  The name of the authors may not be used to endorse or promote
#       products derived from this software without specific prior written
#       permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
# LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.
#

CPP=g++
CFLAGS=-Wall -O2 -ansi -c
LIB_DIR=/usr/local/lib
INC_DIR=/usr/local/include

BINCS=-I/usr/local/include/nnet
BLIBS=-L/usr/local/lib -lnnet

INCS=-I/usr/local/include/sane
LIBS=-static -L/usr/local/lib -lsane

### the default build targets
libs: libsane.a libsane.so


###
### the sane network structure is built below
###
```

```
SaneNetwork.o: SaneNetwork.cpp SaneNetwork.h
$(CPP) $(CFLAGS) $(BINCS) -o $@ SaneNetwork.cpp


###
### some test applications
###
tests: TestSaneNetwork StressSaneNetwork TestCoEvolution libs

### test simple use of a SANE neural network
TestSaneNetwork: TestSaneNetwork.o
$(CPP) -o $@ TestSaneNetwork.o $(LIBS) $(BLIBS)

TestSaneNetwork.o: TestSaneNetwork.cpp
$(CPP) $(CFLAGS) $(INCS) $(BINCS) -o $@ TestSaneNetwork.cpp

StressSaneNetwork: StressSaneNetwork.o
$(CPP) -o $@ StressSaneNetwork.o $(LIBS) $(BLIBS)

StressSaneNetwork.o: StressSaneNetwork.cpp
$(CPP) $(CFLAGS) $(INCS) $(BINCS) -o $@ StressSaneNetwork.cpp

TestCoEvolution: TestCoEvolution.o
$(CPP) -o $@ TestCoEvolution.o $(LIBS) $(BLIBS)

TestCoEvolution.o: TestCoEvolution.cpp
$(CPP) $(CFLAGS) $(INCS) $(BINCS) -o $@ TestCoEvolution.cpp


### create static library
libsane.a: SaneNetwork.o
ar cru $@ SaneNetwork.o

### create shared object library
libsane.so: SaneNetwork.o
$(CPP) $(FLAGS) -fpic -shared -o $@ SaneNetwork.o

### install libs and includes into std system directories
install: libsane.a libsane.so
mkdir -p $(INC_DIR)/sane
cp SaneNetwork.h $(INC_DIR)/sane/
cp libsane.a $(LIB_DIR)/
cp libsane.so $(LIB_DIR)/
chmod 444 $(INC_DIR)/sane/*
chmod 444 $(LIB_DIR)/libsane.a
chmod 555 $(LIB_DIR)/libsane.so

### uninstall libs and includes from system directories
uninstall:
rm -rf $(LIB_DIR)/sane
```

```
rm -f $(LIB_DIR)/libsane.a
rm -f $(LIB_DIR)/libsane.so

### remove backups
clean:
@echo "removing all object files and the target executable"
rm -f *.o TestSaneNetwork StressSaneNetwork TestCoEvolution
rm -f libsane.a libsane.so
rm -f *.net

### remove all binaries and any core files
clean-all: clean
rm -f *.core
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * SaneNetwork.h - Symbiotic Adaptive Neuro-Evolution ADT definition
 */

#ifndef SANENETWORK
#define SANENETWORK

#include <algorithm>    // for std::swap
#include <vector>       // for std::vector

// include the neural network library
#include "Network.h"

// the following typedef's make creating blueprints a bit easier.  a blueprint
// is a subset of the neuron population for a neural network.  rather than
// doing a vector of neurode pointers, we use a vector of indexs into the
// neurode list since this will transfer between networks easily and will also
// save/load to/from a file easier (if we ever decide to do that).
typedef std::vector<unsigned int> blueprint;
```

```
class SaneNetwork : public Network
{
   private:
      blueprint tempBlueprint;    // storage for last generated blueprint

      std::vector<unsigned int> frequency; // frequency of each neurodes usage
      std::vector<float> ratings;          // cumulative rating of each neurode

      unsigned int MutationRate;  // rate of mutation in the network
      unsigned int BlueprintSize; // size of each blueprint
      unsigned int MinimumUsage;  // min # of times all neurodes must be used

   protected:
      // check parameters and make sure they represent a valid SANE network
      void checkDimensions(void) const;

      // determine if we should mutate or not based on our mutation rate
      bool shouldMutate(void) const;

      // mutate the provided neuron
      void mutate(Neurode *n);

      // mate two neurodes together and return an offspring
      Neurode* mateNeurodes(Neurode *mother, Neurode *father);

      // the below three methods are for quicksort-ing the neurode population
      // based on their corresponding cumulative rating
      void sortNeurodes(void);
      void sortNeurodes(unsigned int first, unsigned int last);
      unsigned int splitNeurodes(unsigned int first, unsigned int last);

   public:
      // construct SANE network from a neural network file and the provided
      // SANE specific dimensions
      SaneNetwork(char *filename,
                  unsigned int rate,
                  unsigned int size,
                  unsigned int number);

      // construct SANE network based on provided dimensions
      SaneNetwork(unsigned int InputLayerSize,
                  unsigned int HiddenLayerSize,
                  unsigned int OutputLayerSize,
                  unsigned int rate,
                  unsigned int size,
                  unsigned int number);

      // proper destructor
      ~SaneNetwork(void);
```

```
        // retrieve dimensions of the SANE specific structure
        unsigned int getMutationRate(void) const;
        unsigned int getBlueprintSize(void) const;
        unsigned int getMinimumUsage(void) const;

        // randomly create a new blueprint, clearing any existing blueprint
        blueprint randomizeBlueprint(void);

        // breed the neurode population with itself
        void breed(void);

        // rate all of the neurodes in the current blueprint with given rating
        void rate(float rating, blueprint *bprint = NULL);

        // return true iff each neurode has been used the minimun # of times
        bool isMinUsageMet(void) const;

        // return sum of ratings vector
        float maxRating(void) const;

        // reset the frequency and ratings vectors to all zero
        void reset(void);

        // overload the iterate method to only use a blueprint (not the whole
        // neuron population - the key difference with SANE)
        void iterate(InputVector &input, OutputVector &output,
                     blueprint *print = NULL);
};

#endif
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * SaneNetwork.cpp - SANE Neural Network ADT implementation
 */

#include "SaneNetwork.h"

////////////////////////////////////////////////////////////////////////////
// construct SANE network from a neural network file and the provided SANE
// specific dimensions
SaneNetwork::SaneNetwork(char *filename,
                         unsigned int rate,
                         unsigned int size,
                         unsigned int number)
: Network(filename)     // use Network constructor to load in from file
, MutationRate(rate)
, BlueprintSize(size)
, MinimumUsage(number)
{
   // ensure valid parameters were specified
   checkDimensions();
```

```
   // resize rating and frequency matrix to be size of the neurode matrix
   ratings.resize(getHiddenSize(), 0);
   frequency.resize(getHiddenSize(), 0);
}

//////////////////////////////////////////////////////////////////////////
// construct SANE network based on provided dimensions
SaneNetwork::SaneNetwork(unsigned int InputLayerSize,
                         unsigned int HiddenLayerSize,
                         unsigned int OutputLayerSize,
                         unsigned int rate,
                         unsigned int size,
                         unsigned int number)
: Network(InputLayerSize, HiddenLayerSize, OutputLayerSize)
, MutationRate(rate)
, BlueprintSize(size)
, MinimumUsage(number)
{
   // ensure valid parameters were specified
   checkDimensions();

   // resize rating and frequency matrix to be size of the neurode matrix
   ratings.resize(getHiddenSize(), 0);
   frequency.resize(getHiddenSize(), 0);
}

//////////////////////////////////////////////////////////////////////////
// proper destructor
SaneNetwork::~SaneNetwork(void)
{  // clear the blueprint, ratings, and frequency vectors
   tempBlueprint.clear();
   ratings.clear();
   frequency.clear();
}

//////////////////////////////////////////////////////////////////////////
// check parameters and make sure they represent a valid SANE network
void SaneNetwork::checkDimensions(void) const
{
   // ensure valid mutation rate was specified (it's a percent!)
   if (getMutationRate() > 100)
   {  fprintf(stderr,"error: invalid mutation rate.\n");
      exit(-1);
   }

   // ensure valid blueprint size (it must be less than the hidden layer size)
   if (getBlueprintSize() >= getHiddenSize())
   {  fprintf(stderr,"error: blueprint size must be < 1/2 hidden size.\n");
      exit(-1);
```

```
   }

   // ensure hidden layer size is a multiple of 4 - this makes breeding a bit
   // simpler since we break the neurode population into quarters
   if (getHiddenSize() % 4 != 0)
   {  fprintf(stderr,"error: hidden size must be a multiple of 4.\n");
      exit(-1);
   }
}

////////////////////////////////////////////////////////////////////////
// determine if we should mutate or not based on our mutation rate
bool SaneNetwork::shouldMutate(void) const
{ return getRandomFloat(100) < (double) MutationRate; }

////////////////////////////////////////////////////////////////////////
// mutate the provided neurode
void SaneNetwork::mutate(Neurode *n)
{
   // re-randomize the input weights
   for (unsigned int i = 0; i < n->InputWeightsSize; i++)
      n->InputWeights[i] = getRandomWeight(1);

   // re-randomize the output weights
   for (unsigned int i = 0; i < n->OutputWeightsSize; i++)
      n->OutputWeights[i] = getRandomWeight(1);
}

////////////////////////////////////////////////////////////////////////
// breed two neurodes together and return an offspring
Neurode* SaneNetwork::mateNeurodes(Neurode *mother, Neurode *father)
{
   unsigned int isize = mother->InputWeightsSize;
   unsigned int osize = mother->OutputWeightsSize;

   // if mother and father differ in dimensions, abort!
   if (isize != father->InputWeightsSize || osize != father->OutputWeightsSize)
   {  fprintf(stderr,"error: breeding neurodes of different species.\n");
      exit(-1);
   }

   // create offspring of same dimensions as the parents
   Neurode *offspring = new Neurode(isize, osize);

   // if we mutation the offspring, then do so and return the mutated result.
   // otherwise, we continue with the breeding
   if (shouldMutate())
   {  mutate(offspring);
      return offspring;
   }
```

```
   // NOTE: here is where we do a 1-point crossover between the mother and
   // father's chromosomes

   // calculate the crossover points for the input and output weights
   unsigned int icrossover = (unsigned int) getRandomFloat(isize);
   unsigned int ocrossover = (unsigned int) getRandomFloat(osize);

   // copy input vector from mother and father
   for (unsigned int i = 0; i < isize; i++)
      if (i < icrossover)
         offspring->InputWeights[i] = mother->InputWeights[i];
      else
         offspring->InputWeights[i] = father->InputWeights[i];

   // copy output vector from mother and father
   for (unsigned int i = 0; i < osize; i++)
      if (i < ocrossover)
         offspring->OutputWeights[i] = mother->OutputWeights[i];
      else
         offspring->OutputWeights[i] = father->OutputWeights[i];

   return offspring;
}

////////////////////////////////////////////////////////////////////////////
// randomly create a new blueprint, clearing any existing blueprint
blueprint SaneNetwork::randomizeBlueprint(void)
{
   bool copy;

   // clear any existing blueprint
   tempBlueprint.clear();

   // create a blueprint of the appropriate size and randomly select each
   // neurode we add from our entire neurode population
   for (unsigned int i = 0, index; i < getBlueprintSize(); i++)
   {
      // here, we get and index that hasn't already been added to the blueprint
      do
      {  // randomly grab a neurode from the population
         index = (int) getRandomFloat(getHiddenSize());

         // check to see if the selected neurode is already in the blueprint
         copy = false;
         for (unsigned int j = 0; j < tempBlueprint.size(); j++)
            if (index == tempBlueprint[j]) copy = true;

      } while (copy);
```

```
      // add the selected neurode to our blueprint and increment the # of
      // times that blueprint has been used
      tempBlueprint.push_back(index);
      ++frequency[index];
   }

   return tempBlueprint;
}

////////////////////////////////////////////////////////////////////////
// the following is 1 of 3 methods used to quicksort the neurode population
// based on their corresponding cumulative rating
// -> split and order the set of neurodes (quicksort split)
unsigned int SaneNetwork::splitNeurodes(unsigned int first, unsigned int last)
{
   unsigned int pivot, index;
   float value;

   // setup first and pivot elements in the list
   value = ratings[first];
   pivot = first;

   // merge the elements between first+1 and last
   for (index = first + 1; index < last; index++)
   {  // if current element is less than the first, swap them
      if (ratings[index] > value)
      {  ++pivot;
         std::swap(ratings[pivot],   ratings[index]);
         std::swap(frequency[pivot], frequency[index]);
         std::swap(Neurodes[pivot],  Neurodes[index]);
      }
   }

   // swap the first and pivot elements, and return the pivot position
   std::swap(ratings[first],   ratings[pivot]);
   std::swap(frequency[first], frequency[pivot]);
   std::swap(Neurodes[first],  Neurodes[pivot]);
   return pivot;
}

////////////////////////////////////////////////////////////////////////
// the following is 2 methods are used to quicksort the neurode population
// based on their corresponding cumulative rating
// -> sort neurodes by their ratings
void SaneNetwork::sortNeurodes(void) { sortNeurodes(0, getHiddenSize()); }
void SaneNetwork::sortNeurodes(unsigned int first, unsigned int last)
{  // standard quicksort algorithm follows...
   if (first < last)
   {  unsigned int pivot = splitNeurodes(first, last);
      sortNeurodes(first, pivot);
```

```
      sortNeurodes(pivot + 1, last);
   }
}

////////////////////////////////////////////////////////////////////////
// retrieve dimensions of the SANE specific structure
unsigned int SaneNetwork::getMutationRate(void) const {return MutationRate;}
unsigned int SaneNetwork::getBlueprintSize(void) const {return BlueprintSize;}
unsigned int SaneNetwork::getMinimumUsage(void) const {return MinimumUsage;}

////////////////////////////////////////////////////////////////////////
// breed the neurode population with itself
void SaneNetwork::breed(void)
{
   Neurode *offspring;

   // first, sort the neurode population by their ratings
   sortNeurodes();

   // for the top 25% of neurons, breed each neurode with its neighbor and add
   // the offspring to the bottom of the list, replacing the bottom 25% of the
   // neurons
   for (unsigned int i = 0; i < getHiddenSize() / 4; i++)
   {  // breed the neurode with its neighbor and create an offspring
      offspring = mateNeurodes(Neurodes[i], Neurodes[i + 1]);

      // delete corresponding neurode in the bottom 25% and replace it with
      // the new offspring
      delete Neurodes[getHiddenSize() - (i + 1)];
      Neurodes[getHiddenSize() - (i + 1)] = offspring;
   }
}

////////////////////////////////////////////////////////////////////////
// rate all of the neurodes in the current blueprint with the given rating
void SaneNetwork::rate(float rating, blueprint *bprint)
{  // if the provided blueprint is null, use the temporary blueprint
   if (bprint == NULL) bprint = &tempBlueprint;

   // for each neurode in the blueprint, apply the rating
   for (unsigned int i = 0; i < bprint->size(); i++)
      ratings[(*bprint)[i]] += rating;
}

////////////////////////////////////////////////////////////////////////
// return true iff each neurode has been used the minimum # of times
bool SaneNetwork::isMinUsageMet(void) const
{  // if any of the neurodes has NOT been used enough, return false.  else true
   for (unsigned int i = 0; i < getHiddenSize(); i++)
      if (frequency[i] < getMinimumUsage()) return false;
```

```
   return true;
}

////////////////////////////////////////////////////////////////////////////
// maxRating returns the sum of the ratings vector
float SaneNetwork::maxRating(void) const
{
   float sum = 0;

   // sum each element in the ratings vector and return it
   for (unsigned int i = 0; i < ratings.size(); i++)
      sum += ratings[i];

   return sum;
}

////////////////////////////////////////////////////////////////////////////
// reset the frequency and ratings vectors to all zero
void SaneNetwork::reset(void)
{  // cycle through the vectors and zero each element out
   for (unsigned int i = 0; i < getHiddenSize(); i++)
   {  frequency[i] = 0;
      ratings[i] = 0;
   }
}

////////////////////////////////////////////////////////////////////////////
// overload the iterate method to only use a blueprint (subset) of the neurode
// population - this is the key difference between the SANE and basic neural
// network iteration
void SaneNetwork::iterate(InputVector &iVector, OutputVector &oVector,
                          blueprint* bprint)
{
   std::vector<float> HiddenLayer;
   float input, output;

   // clear output vector
   oVector.clear();

   // if input vector is of incorrect size, error out of program
   if (iVector.size() != getInputSize())
   {  fprintf(stderr, "error: InputVector is of incorrect size.\n");
      exit(1);
   }

   // if no blueprint was specified, use the temporary one
   if (bprint == NULL) bprint = &tempBlueprint;

   // for each neurode in the blueprint, calculate it's input based on the
```

```
   // InputVector and the neurode's weights.  add the input to the hidden
   // layer vector.
   for (unsigned int i = 0, j; i < bprint->size(); i++)
   {  // multiply neurodes input chromosome against entire input vector
      for (j = 0, input = 0; j < getInputSize(); j++)
         input += iVector[j] * getNeurode((*bprint)[i])->InputWeights[j];

      HiddenLayer.push_back(input);
   }

   // for each output neurode, calculate it's input based on the hidden layer
   // input (calculated above) and the weights on the output connections
   for (unsigned int i = 0, j; i < getOutputSize(); i++)
   {  // mulitply each neurodes output (from above) by its output chromosome
      for (j = 0, output = 0; j < bprint->size(); j++)
         output += HiddenLayer[j] * getNeurode((*bprint)[j])->OutputWeights[i];

      oVector.push_back(output);
   }
}
```

B.2.4 SapioGo Source Code

```
# Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are
# met:
#
#    *  Redistributions of source code must retain the above copyright
#       notice, this list of conditions and the following disclaimer.
#    *  Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in the
#       documentation and/or other materials provided with the distribution.
#    *  The name of the authors may not be used to endorse or promote
#       products derived from this software without specific prior written
#       permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
# LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.

CPP=g++
CFLAGS=-Wall -O2 -ansi -c
BIN_DIR=/usr/local/bin

# necessary headers
INCS=-I/usr/local/include/nnet -I/usr/local/include/sane -I/usr/local/include/gtp
# necessary libraries
LIBS=-L/usr/local/lib -lnnet -lsane -lgtp


### build both the stand-alone and sapiogo-coevolution game
all: sapiogo sapiogo-coevo

###
### main program
###
sapiogo: sapiogo.o GameSettings.o GameMove.o
$(CPP) $(LIBS) -o $@ sapiogo.o GameSettings.o GameMove.o

sapiogo.o: sapiogo.cpp GameSettings.o GameMove.o errors.h
```

```
$(CPP) $(CFLAGS) $(INCS) -o $@ sapiogo.cpp

###
### sapiogo-coevolution program
###
sapiogo-coevo: sapiogo-coevo.o GameSettings.o GameMove.o
$(CPP) $(LIBS) -o $@ sapiogo-coevo.o GameSettings.o GameMove.o

sapiogo-coevo.o: sapiogo-coevo.cpp GameSettings.o errors.h
$(CPP) $(CFLAGS) $(INCS) -o $@ sapiogo-coevo.cpp

###
### GameSettings ADT file
###
GameSettings.o: GameSettings.h GameSettings.cpp
$(CPP) $(CFLAGS) $(INCS) -o $@ GameSettings.cpp

###
### GameMoves auxillary files
###
GameMove.o: GameMove.h GameMove.cpp
$(CPP) $(CFLAGS) $(INCS) -o $@ GameMove.cpp


###
### some standard targets...
###

### install sapiogo and sapiogo-coevo in the local system
install: sapiogo sapiogo-coevo
cp sapiogo $(BIN_DIR)
cp sapiogo-coevo $(BIN_DIR)
chmod 555 $(BIN_DIR)/sapiogo
chmod 555 $(BIN_DIR)/sapiogo-coevo

### remove sapiogo and sapiogo-coevo from the local system
uninstall:
rm -f $(BIN_DIR)/sapiogo
rm -f $(BIN_DIR)/sapiogo-coevo

### clean all build files
clean:
@echo "removing all object files and the target executable"
rm -f *.o
rm -f sapiogo sapiogo-coevo

### clean all build files + any core files
clean-all: clean
rm -f *.core
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * errors.h - simple constants used for keeping track of error return codes
 */

#ifndef ERRORS_H
#define ERRORS_H

const int ERR_USAGE     = 1;  // improper usage of sapiogo
const int ERR_NETWORK   = 2;  // improper usage of neural network library
const int ERR_SANE      = 3;  // improper usage of sane network library
const int ERR_GTP       = 4;  // imporper usage of gtp/gmp library

#endif
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * GameSettings.h - GameSettings ADT definition
 */

#include <stdio.h>      // for printf,fprintf,sscanf,etc
#include <string.h>     // for strncmp
#include <stdlib.h>     // for exit
#include <time.h>       // for time

// include our home-grown errors file
#include "errors.h"

#ifndef GAME_SETTINGS
#define GAME_SETTINGS

// some global constants used to define upper bounds on the sizes of command
// line arguments.  these could be increased to any desired size, but for our
// purposes, a size of 20 for switch names and 100 for their arguments is fine.
const unsigned int MaxSwitchSize = 21;
const unsigned int MaxSwitchArgSize = 101;
```

```
// GameSettings object - a nice little way to encapsulate all of the various
// settings used in our game into one single object.  it contains both Go
// game settings as well as SANE AI settings.
struct GameSettings
{
   // Go settings
   unsigned int   boardsize;     // size of go board
   unsigned int   handicap;      // handicap for current user
   unsigned int   maxmoves;      // maximum # of moves to allow in the game
   float          komi;          // the # of points awarded to white for Ko
   bool           isWhite;       // true iff user is white
   char           sgfFile[MaxSwitchArgSize];   // where we save SGF file
   bool           gtp;           // true iff client speaks GTP

   // AI settings
   bool           learn;         // true iff we want to learn
   unsigned int   seed;          // seed used for random number generator
   unsigned int   netsize;       // # of neurodes in neural network
   unsigned int   blueprintsize; // size of blueprints
   unsigned int   minusage;      // min # of times a neurode must be used
   unsigned int   mutaterate;    // rate of mutation
   unsigned int   generations;   // # of generations to play
   unsigned int   passpercent;   // % of the time to pass
   unsigned int   topmoves;      // randomly choose from top N number of moves
   float          threshhold;    // min value for output to be considered
   char           outNet[MaxSwitchArgSize];  // where we save neural net to
   char           inNet[MaxSwitchArgSize];   // where we load neural net from
   char           workset[MaxSwitchArgSize]; // name of all workset files
   bool           loadInNet;     // true iff and inNet was provided
};

// PrintUsage prints the proper usage of this application along with all
// available command line switches to stderr.  If an exitVal other than zero
// is provided, it exits the program with that value.
void PrintUsage(int exitVal);

// ParseCommandLine parses the provided command line and strips out all
// specified settings, storing them in the provided GameSettings object
void ParseCommandLine(int argc, char *argv[], GameSettings &settings);

// ParseCommandArgument parses a variable typed value out of the command line
// and stores it in the value v.
template <class T>
void ParseCommandArgument(int argc, char *argv[], int &i, char *type, T &v);

// ValidateGameSettings checks the provided GameSettings object and makes
// sure all settings are of appropriate values.  If any are not, the program
// displays an error message and exits.
void ValidateGameSettings(GameSettings settings);
```

```
// SetGameDefaults sets all of the members of the provided GameSettings object
// to what we define as their 'default' values.
void SetGameDefaults(GameSettings &settings);

#endif
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * GameSettings.cpp - GameSettings ADT implementation
 */

#include "GameSettings.h"

////////////////////////////////////////////////////////////////////////////
// PrintUsage outputs the proper command line usage (with all available
// options) to stdout.  if a paramater other than 0 is provided, we exit the
// application with that value.
void PrintUsage(int exitVal = 0)
{  // output correct usage of application
   printf("usage: go [options]\n");
   printf("\t\t\tGo Game Settings\n");
   printf("\t-boardsize N\t\tset size of board (can be from 9 to 19)\n");
   printf("\t-handicap H\t\tset handicap (can be from 0 to 9)\n");
   printf("\t-maxmoves N\t\tset maximum # of moves allowed (0 for infinte)\n");
   printf("\t-komi K\t\t\taward white N number of points for Komi\n");
   printf("\t-white\t\t\twill play as the White user (default)\n");
   printf("\t-black\t\t\twill play as the Black user\n");
```

```
   printf("\t-sgf FILE\t\twill save game as an SGF to FILE\n");
   printf("\t-gtp\t\t\tspeak the GTP (Go Text Protocol - default)\n");
   printf("\t-gmp\t\t\tspeak the GMP (Go Modem Protocol)\n");
   printf("\n\t\tArtificial Intelligence Settings\n");
   printf("\t-learn\t\t\twill load SANE network\n");
   printf("\t-seed\t\t\tseed used by random number generator\n");
   printf("\t-netsize N\t\tuse N neurodes in neural network\n");
   printf("\t-threshold T\t\tthreshold for passing or moving\n");
   printf("\t-passpercent P\t\tif no output above threshold, pass P\% time\n");
   printf("\t-topmoves T\t\trandomly choose from top T # of moves\n");
   printf("\t-blueprint S\t\tuse blueprints of size S\n");
   printf("\t-minusage M\t\tensure each neurodd used in M blueprints\n");
   printf("\t-mutaterate R\t\tset the random mutation rate (0 - 100)\n");
   printf("\t-generations N\t\tplay N generations of go\n");
   printf("\t-outNet FILE\t\twhere to save neural network to\n");
   printf("\t-inNet FILE\t\twhere to load neural network from\n");
   printf("\n\t-help\t\t\tDisplay this help message\n");

   // if an exit value was provided, exit the program with that value
   if (exitVal != 0) exit(exitVal);
}

////////////////////////////////////////////////////////////////////////////
// ParseCommandLine cycles through the command line and strips out all
// specified settings, storing them in the provided GameSettings object.
void ParseCommandLine(int argc, char *argv[], GameSettings &settings)
{
   char           name[MaxSwitchSize];

   // cycle through the command line and parse each token individually
   for (int i = 1; i < argc; i++)
   {
      // strip the name out of the switch - note, all valid switch begin with
      // a single dash '-'
      if (sscanf(argv[i], "-%20s", name) != 1)
      {  // malformed switch was entered -- alert and exit
         fprintf(stderr, "error: invalid switch '%s'\n", argv[i]);
         PrintUsage(ERR_USAGE);
      }

      // do a case-by-case comparison on all of our defined switches
      if (strncmp(name, "boardsize", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%d", settings.boardsize);
      else if (strncmp(name, "handicap", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%d", settings.handicap);
      else if (strncmp(name, "maxmoves", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%d", settings.maxmoves);
      else if (strncmp(name, "komi", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%f", settings.komi);
      else if (strncmp(name, "white", MaxSwitchSize) == 0)
```

```
         settings.isWhite = true;
      else if (strncmp(name, "black", MaxSwitchSize) == 0)
         settings.isWhite = false;
      else if (strncmp(name, "sgf", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%s", settings.sgfFile);
      else if (strncmp(name, "gtp", MaxSwitchSize) == 0)
         settings.gtp = true;
      else if (strncmp(name, "gmp", MaxSwitchSize) == 0)
         settings.gtp = false;
      else if (strncmp(name, "learn", MaxSwitchSize) == 0)
         settings.learn = true;
      else if (strncmp(name, "seed", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%d", settings.seed);
      else if (strncmp(name, "netsize", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%d", settings.netsize);
      else if (strncmp(name, "threshold", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%f", settings.threshhold);
      else if (strncmp(name, "passpercent", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%d", settings.passpercent);
      else if (strncmp(name, "topmoves", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%d", settings.topmoves);
      else if (strncmp(name, "blueprint", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%d", settings.blueprintsize);
      else if (strncmp(name, "minusage", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%d", settings.minusage);
      else if (strncmp(name, "mutaterate", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%d", settings.mutaterate);
      else if (strncmp(name, "generations", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%d", settings.generations);
      else if (strncmp(name, "outNet", MaxSwitchSize) == 0)
         ParseCommandArgument(argc, argv, i, "%s", settings.outNet);
      else if (strncmp(name, "inNet", MaxSwitchSize) == 0)
      {  ParseCommandArgument(argc, argv, i, "%s", settings.inNet);
         settings.loadInNet = true;
      }
      else if (strncmp(name, "help", MaxSwitchSize) == 0)
         PrintUsage(ERR_USAGE);
      else
      {  // unknown switch entered -- alert and exit
         fprintf(stderr, "error: unknown switch '%s'\n", argv[i]);
         PrintUsage(ERR_USAGE);
      }
   }
}

////////////////////////////////////////////////////////////////////////////
// ParseCommandArgument parses an argument of any type T out of a command
// line (structured like the typical argc and *argv[]).  it works by first
// ensuring there is at least one more argument in the list that can be
// parsed.  Then, it parses the argv string using the provided type and
```

```
// stores it in the value v.
template <class T>
void ParseCommandArgument(int argc, char *argv[], int &i, char *type, T &v)
{
   // make sure we have at least one more argument in the list
   if (argc <= (i + 1))
   {  // missing parameter
      fprintf(stderr, "error: switch '%s' requires and argument\n", argv[i]);
      PrintUsage(ERR_USAGE);
   }

   // now parse the value out of the array
   if (sscanf(argv[++i], type, &v) != 1)
   {  // malformed input -- alert and exit
      fprintf(stderr, "error: invalid parameter for '%s'\n", argv[i - 1]);
      PrintUsage(ERR_USAGE);
   }
}

////////////////////////////////////////////////////////////////////////////
// ValidateGameSettings checks the provided GameSettings object and ensures
// that all of its members are of appropriate values.  If any of them are not
// then we provide some useful error message and exit.
void ValidateGameSettings(GameSettings settings)
{
   // TODO check input files to see if they exist
   // TODO maybe same for output files?  i.e. can we create specified file?

   // validate boardsize (we only handle boards from 9 to 19)
   if (settings.boardsize < 5 || settings.boardsize > 19)
   {  fprintf(stderr, "error: boardsize must be from 5 to 19\n");
      PrintUsage(ERR_USAGE);
   }

   // validate handicap (must be positive integer less than 9)
   if (settings.handicap < 0 || settings.handicap > 9)
   {  fprintf(stderr, "error: handicap must be from 0 to 9\n");
      PrintUsage(ERR_USAGE);
   }

   // validate blueprintsize (must be <= netsize) -- only necessary to check
   // if we are running in AI mode
   if (settings.learn && (settings.blueprintsize >= settings.netsize))
   {  fprintf(stderr, "error: blueprintsize must be less than the size of ");
      fprintf(stderr, "the network (-netsize)\n");
      PrintUsage(ERR_USAGE);
   }

   // validate mutation rate (it's a percentage -- so it must be from 0 to 100)
   if (settings.mutaterate < 0 || settings.mutaterate > 100)
```

```
   {  fprintf(stderr, "error: mutation rate must be from 0 to 100\n");
      PrintUsage(ERR_USAGE);
   }

   // validate netsize (must be a perfect multiple of 4) -- only necessary to
   // check if we are running in AI mode
   if (settings.learn && settings.netsize % 4 != 0)
   {  fprintf(stderr, "error: netsize must be a multiple of 4\n");
      PrintUsage(ERR_USAGE);
   }

   // make sure pass % is a % (0 - 100)
   if (settings.passpercent < 0 || settings.passpercent > 100)
   {  fprintf(stderr, "error: passpercent must be from 0 to 100\n");
      PrintUsage(ERR_USAGE);
   }
}

////////////////////////////////////////////////////////////////////////////
// SetGameDefaults sets all of the members of the provided GameSettings
// object to their default values.  to change a default value for some game
// setting, simply alter the values below
void SetGameDefaults(GameSettings &settings)
{
   // Go settings
   settings.boardsize = 9;
   settings.handicap = 0;
   settings.maxmoves = 0;
   settings.komi = 5.5;
   settings.isWhite = true;
   snprintf(settings.sgfFile, MaxSwitchArgSize, "go.sgf");
   settings.gtp = true;

   // AI settings
   settings.learn = false;
   settings.seed = time(NULL);
   settings.netsize = 4000;
   settings.blueprintsize = 500;
   settings.minusage = 5;
   settings.mutaterate = 1;
   settings.generations = 5;
   settings.passpercent = 80;
   settings.topmoves = 5;
   settings.threshhold = 29000;
   snprintf(settings.outNet,  MaxSwitchArgSize, "go.net");
   snprintf(settings.inNet,   MaxSwitchArgSize, "go.net");
   snprintf(settings.workset, MaxSwitchArgSize, "go");
   settings.loadInNet = false;
}
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * GameMove.h - GameMove ADT definition and support functions
 */

#include <vector>          // for std::vector
#include <cmath>           // for log10

#include <SaneNetwork.h>   // for OutputVector / InputVector
#include <gtp.h>           // for GTP connection

#include "GameSettings.h"  // for GameSettings ADT

#ifndef GAME_MOVE
#define GAME_MOVE

// Below is the GameMove ADT.  It encapsulates all of the informatino needed
// to represent a single move in the game of Go.   If the 'pass' member is
// true, then the move is interpretted as a pass.  Otherwsise, the 'x' and 'y'
// members are interpretted as the x and y coordinates on the board where a
// move will be made.
```

```
struct GameMove
{  unsigned int x;
   unsigned int y;
   bool pass;
};

// The below three methods are used to quicksort a list of outputs and their
// corresponding GameMove ADT equivalents.  Once this list of GameMoves is
// sorted by their corresponding output, we can randomly choose what move to
// make out of the top N number of moves (where N varies).
void sortOutput(OutputVector &ovector, std::vector<GameMove> &moves);
void sortOutput(OutputVector &ovector, std::vector<GameMove> &moves,
                unsigned int first, unsigned int last);

// This is simply the quicksort 'split' method.
unsigned int splitOutput(OutputVector &ovector, std::vector<GameMove> &moves,
                         unsigned int first, unsigned int last);

// getOutputMove returns a GameMove from the given output vector, GTP
// connection, and game settings.  It works as follows:
//    1) create a GameMove corresponding to each output in the output vector
//    2) sort the list of GameMoves by their corresponding output
//    3) if no output above THRESHOLD (a game setting) pass with Y probability
//       (also a game setting).  if we decide not to pass (or there was output
//       above the threshold) then...
//    4) choose GameMove randomly from the top X number of moves (another game
//       setting).
GameMove getOutputMove(OutputVector &ovector, Gtp &connection,
                       GameSettings settings, int isWhite);

// This method builds an input vector adaquate for the neural network.  It
// builds the vector from the given Go game connection and settings.
InputVector BuildInputVector(Gtp &connection, GameSettings settings,
                             int isWhite);

#endif
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * GameMove ADT and support functions implementation
 */

#include "GameMove.h"

////////////////////////////////////////////////////////////////////////////
// quicksort algorithm - this sorts a list of GameMoves 'moves' in descending
// order by their associated output in 'ovector'.  This method sorts the
// entire list of GameMoves.
void sortOutput(OutputVector &ovector, std::vector<GameMove> &moves)
{  // simply call the method below and sort the entire list
   sortOutput(ovector,moves,0,ovector.size());
}

////////////////////////////////////////////////////////////////////////////
// quicksort algorithm - this sorts a list of GameMoves 'moves' in descending
// order by their associated output in 'ovector'.  This method sorts only the
// portion of the list of GameMoves between the 'first' and 'last' indices.
void sortOutput(OutputVector &ovector, std::vector<GameMove> &moves,
```

```
                unsigned int first, unsigned int last)
{  // standard quicksort algorithm...
   if (first < last)
   {  unsigned int pivot = splitOutput(ovector, moves, first, last);
      sortOutput(ovector, moves, first, pivot);
      sortOutput(ovector, moves, pivot + 1, last);
   }
}

////////////////////////////////////////////////////////////////////////////
// split and order the set of GameMoves according to their corresponding
// output (quicksort split)
unsigned int splitOutput(OutputVector &ovector, std::vector<GameMove> &moves,
                         unsigned int first, unsigned int last)
{
   unsigned int pivot, index;
   float value;

   // setup first and pivot elements in the list
   value = ovector[first];
   pivot = first;

   // merge the elements between first+1 and last
   for (index = first + 1; index < last; index++)
   {  // if curent element is less than first, swap them
      if (ovector[index] > value)
      {  ++pivot;
         std::swap(ovector[pivot], ovector[index]);
         std::swap(moves[pivot],   moves[index]);
      }
   }

   // swap the first and pivot elements, and return the pivot position
   std::swap(ovector[first], ovector[pivot]);
   std::swap(moves[first],   moves[pivot]);
   return pivot;
}

////////////////////////////////////////////////////////////////////////////
// return a GameMove from the current boardstate and output vector.  This is
// where the game actually makes the decision of what move to make after it
// has returned from the neural network.
GameMove getOutputMove(OutputVector &ovector, Gtp &connection,
                       GameSettings settings, int isWhite)
{
   unsigned int x;
   unsigned int y;
   bool pass = true;

   // create a dummy game move object that, unless changed, will simply pass
```

```
GameMove mv;
mv.pass = 1;
mv.x = 0;
mv.y = 0;

// if we have exceeded the maximum # of moves allowed in a game, simply
// return the dummy move object we just created (which will result in a
// pass)
if (connection.getnumberofmoves() > (int)settings.maxmoves &&
    settings.maxmoves != 0)
   return mv;

vector< GameMove > moves;  // our list of valid game moves
vector< float > outputs;   // corresponding list of valid outputs

// if no output was above our threshold, then we pass Y percent of the time
if ((rand() % 100) > (int)settings.passpercent)
   pass = false;

// cycle through the entire board and for each legal move we encounter, add
// it to the list of valid outputs and create a new GameMove corresponding
// to that output.
for (x = 0; x < settings.boardsize; x++)
{  for (y=0; y < settings.boardsize; y++)
   {  if (ovector[x * settings.boardsize + y] > settings.threshhold || !pass)
      {  if (connection.islegal(x,y,isWhite))
         {  // valid move and we're above our threhold - add it to our list
            mv.pass = 0;
            mv.x = x;
            mv.y = y;
            moves.push_back(mv);             // add GameMove to our list
            outputs.push_back(ovector[x*settings.boardsize+y]);   // add output to our list
         }
      }
   }
}

// if there were no legal moves, simply pass
if (moves.size() == 0)
{  mv.pass = true;
   return mv;
}

// if we have more moves than X, sort the list and drop all but the top X
// # of moves
else if (moves.size() > settings.topmoves)
{  // sort the possible moves according to their associated output weight
   sortOutput(outputs, moves);

   // reduce the set of moves to the maximum we allow
```

```
      while (moves.size() > settings.topmoves) moves.pop_back();
   }

   // randomly chose one of the top X moves (X is a game setting)
   // moves is a vector< GameMove >
   mv = moves[rand() % moves.size()];

return mv;
}

////////////////////////////////////////////////////////////////////////////
// Construct and return an input vector from the current boardstate,
// accessible via the provided GTP connection.
InputVector BuildInputVector(Gtp &connection, GameSettings settings,
                             int isWhite)
{
InputVector input;
input.clear();
unsigned int rows,columns;

for (rows = 0; rows < settings.boardsize; rows++)
   {
for (columns = 0; columns < settings.boardsize; columns++)
      {
int boardstate = connection.whatcolor(rows,columns);

if (boardstate == 3){ /*is blank*/
input.push_back(1);
input.push_back(1);
}else if (isWhite){/*if white*/
if (boardstate == 1){
input.push_back(0);
input.push_back(1);
}
else if (boardstate == 2){
input.push_back(1);
input.push_back(0);
}
else
exit(-1);
}else if (!(isWhite)){/*if black*/
if (boardstate == 2){
input.push_back(0);
input.push_back(1);
}
else if (boardstate == 1){
input.push_back(1);
input.push_back(0);
}
else
```

```
exit(-1);
}

         float liberties = connection.countlib(rows, columns);
         if (liberties == 0)
            liberties = (1 / (settings.boardsize * settings.boardsize));
         else
            liberties = (1 / liberties);

         input.push_back(liberties);
float bs = (settings.boardsize/2);
if (rows<bs){input.push_back((rows+1)/bs);}
else{input.push_back((settings.boardsize-rows)/bs);}
if (columns<bs){input.push_back((columns+1)/bs);}
else{input.push_back((settings.boardsize-columns)/bs);}
}
}
int p1captures = connection.captures(isWhite);
int p2captures = connection.captures(!(isWhite));
int myscore = 0;//connection.captures(isWhite);
if (( p1captures+p2captures ) > 0) {myscore = p1captures/(p1captures+p2captures);}
input.push_back(myscore);

int theirscore = 0;//connection.captures(!(isWhite));
if (( p1captures+p2captures ) > 0) {theirscore = p2captures/(p1captures+p2captures);}
input.push_back(theirscore);

int mystones = connection.numberstones(isWhite);
input.push_back(mystones/(settings.boardsize*settings.boardsize));
int theirstones = connection.numberstones(!(isWhite));
input.push_back(theirstones/(settings.boardsize*settings.boardsize));

input.push_back(((mystones + theirstones)/2)/(settings.boardsize*settings.boardsize));
float nummoves = connection.getnumberofmoves();
if (nummoves<1){nummoves = 1;}
nummoves = log10(nummoves);
input.push_back(nummoves);
return input;
}
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * sapiogo.cpp - main source file for the sapiogo program
 */

#include <iostream>
#include <stdio.h>
#include <string>
#include <Network.h>       // neural network library
#include <SaneNetwork.h>   // sane neural network library
#include <gtp.h>           // Go Text Protocol library

#include "errors.h"        // a list of defined error constants
#include "GameSettings.h"  // a set of functions for obtaining game settings
#include "GameMove.h"      // a set of functions for handling game moves

// the following two functions are provided to make determinig the input size
// and output size of the neural network a bit easier.  since our input set
// is constantly changing, this provides an easy way to keep track of it.
unsigned int InputSize(GameSettings settings)
{ return ((settings.boardsize * settings.boardsize) * 5 + 6); }
```

```
// this remains constant, but we provide a function anyway
unsigned int OutputSize(GameSettings settings)
{ return (settings.boardsize * settings.boardsize); }


char*    BuildGenerationName(char *str, unsigned int gen);
Network* BuildNetwork(GameSettings settings);
float    PlayGo(GameSettings settings, Network *net, Gtp &connection);


////////////////////////////////////////////////////////////////////////////
// main
int main(int argc, char *argv[])
{
   float          score;         // game score
   char           genname[50];   // name of a generation file
   GameSettings   settings;      // encapsulate all game settings

   // create default game settings, parse the command line and load any
   // specified settings from that, and then validate those settings
   SetGameDefaults(settings);                // set game defaults
   ParseCommandLine(argc, argv, settings);   // parse command line
   ValidateGameSettings(settings);           // validate settings

   // seed the random number generator with the specified seed
   srand(settings.seed);

   //connection.showboard(); // output initial board sate (more for debug)

   // build an appropriate neural network and save it as our initial gen
   Network *net = BuildNetwork(settings);
   net->save(BuildGenerationName(genname, 0));

   // if we are in learning mode, we begin our generations here.  otherwise,
   // we simply play a game of go, as below
   if ((settings.learn) && (settings.gtp))
   {
      // for as many generations as we specified, play enough games of go to
      // meet the minimum usage of the SANE network.  once the min usage is
      // met, breed the SANE network and start a new generation.
      for (unsigned int gen = 0; gen < settings.generations; gen++)
      {
    // create a new GTP connection
 Gtp connection(settings.boardsize, settings.handicap, settings.komi, settings.isWhite);
         // reset the frequencies and ratings of the SANE network
         ((SaneNetwork*)net)->reset();

         // create as many blueprints and play as many games as it takes until
         // all blueprints have been used the minimum number of times
```

```
         do
         {  // create a random blueprint and play the game with that blueprint
            ((SaneNetwork*)net)->randomizeBlueprint();
            score = PlayGo(settings, net, connection);

            // rate the blueprint with the score
            ((SaneNetwork*)net)->rate(score);

            // keep cycling until min usage for each neurode is met
         } while(!((SaneNetwork*)net)->isMinUsageMet());

         // breed the network.  once finished, save the network to file
         ((SaneNetwork*)net)->breed();
         net->save(BuildGenerationName(genname, gen + 1));
         fprintf(stderr, "%c", 7);  // audible beep
      }
   }

   // otherwise, we just play a game of go (no learning)
   else{
     Gtp connection(settings.boardsize, settings.handicap, settings.komi, settings.isWhite);
     if (!(settings.gtp))
connection.Connect(settings.isWhite,settings.boardsize);
      PlayGo(settings, net, connection);}

   return 0;
}

////////////////////////////////////////////////////////////////////////////
// Build an appropriate string to be used for the name of an output
// (generation) file.
char* BuildGenerationName(char *str, unsigned int gen)
{  snprintf(str, 50, "generation.%d.net", gen);
   return str;
}

////////////////////////////////////////////////////////////////////////////
// Build an appropriate Neural Network according to the provided game
// settings.  If we are in learning mode, then we build and return a SANE
// network.  Otherwise, we are in play mode and build a simple neural network.
// Likewise, if we are building the network from an existing file, we load
// that in here.  Otherwise, we generate a completely random network.
Network* BuildNetwork(GameSettings settings)
{
   Network *net;  // this will point to our network object

   // if we are running in AI mode, we load a SANE network
   if (settings.learn)
   {  // if we are loading in an existing network, we do that here
      if (settings.loadInNet)
```

```
      {  net = new SaneNetwork(settings.inNet,
                               settings.mutaterate,
                               settings.blueprintsize,
                               settings.minusage);
      }
      else
      {  // else, we create a brand new SANE network with random weights
         net = new SaneNetwork(InputSize(settings),
                               settings.netsize,
                               OutputSize(settings),
                               settings.mutaterate,
                               settings.blueprintsize,
                               settings.minusage);
         net->randomize(1);   // randomize SANE network with upper bound of 1
      }
   }
   // else we are running in PLAY mode and just load a Neural Network
   else
   {  // if a network was specified, load in that network
      if (settings.loadInNet)
         net = new Network(settings.inNet);
      else
      {  // else, create a new random neural network
         net = new Network(InputSize(settings),
                           settings.netsize,
                           OutputSize(settings));
         net->randomize(1);   // randomize network with upper bound of 1
      }
   }
   return net;
}

////////////////////////////////////////////////////////////////////////////
// This function encapsulates playing an entire game of Go, with the provided
// settings, against the provided neural network.
float PlayGo(GameSettings settings, Network *net, Gtp &connection)
{
   InputVector input;
   OutputVector output;
   int pass = 0;

   // if we are white, make the first move
   if (settings.isWhite)
   {  if (settings.gtp) //(playing GnuGo)
         connection.gnugo_move();
      else              //(playing over cgoban)
         connection.waitandmakemove();
   }

   while (1)//(state == play)
```

```
   {  //we build our input vector
      connection.showboard();
      input = BuildInputVector(connection, settings,settings.isWhite);
      // if in AI mode, iterate a SANE network, else iterate a neural network
      if (settings.learn)
         ((SaneNetwork*)net)->iterate(input, output);
      else
         net->iterate(input, output);

      GameMove mymove = getOutputMove(output,connection,settings, settings.isWhite);
      float max = 0;
      for (unsigned int i = 0; i < output.size(); i++)
         if (output[i] > max) max = output[i];

      printf("max output: %f\n", max);

      if (mymove.pass){connection.sendpass(settings.isWhite);pass++;}
      else{pass = 0;connection.MakeGtpMove(mymove.x,mymove.y,settings.isWhite);}

      if (pass>=2){break;}
      // wait for opponents move
      int theirmove = 0;
      if (settings.gtp)//(playing GnuGo)
      {theirmove = connection.gnugo_move();}
      else //(playing over cgoban)
      {theirmove = connection.waitandmakemove();}
      // if in AI mode, iterate a SANE network, else iterate a neural network

      if (theirmove){pass++;}
      else {pass = 0;}
      if (pass>=2){break;}
   }//end loop
   connection.printsgf(settings.sgfFile);
   float ourscore = connection.newscore(settings.isWhite);
   float theirscore = connection.newscore(!settings.isWhite);
   float returnscore = ourscore - theirscore;

   printf("Game Over: ");
   if (returnscore<0)
      printf("we lost: modscore: %f, ours: %f, theirs: %f\n",
            returnscore, ourscore , theirscore);
   else
      printf("we won: modscore: %f, ours: :%f, theirs: %f\n",
            returnscore, ourscore , theirscore);

   connection.clearboard();

   //return returnscore;
   return ourscore;
}
```

```
/*
 * Copyright (c) 2004, Ryan Flannery & Kevin Upchurch
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 *    *  Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *    *  Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *    *  The name of the authors may not be used to endorse or promote
 *       products derived from this software without specific prior written
 *       permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * co-evo.cpp - the main source file for the co-evolving client of SapioGo
 */

#include <iostream>
#include <stdio.h>
#include <string>
#include <SaneNetwork.h>   // sane neural network library
#include <gtp.h>           // Go Text Protocol library

#include "errors.h"        // a list of defined error constants
#include "GameSettings.h"
#include "GameMove.h"

// the following structure is used to encapsulate the scoring system for a game
struct GameScore
{  float black;
   float white;
};
```

```
// the following two functions are provided to make determinig input size and
// output size of the neural network a bit easier.  since our input set is
// continually changing, this provides an easy way to keep track of it.
unsigned int InputSize(GameSettings settings)
{ return ((settings.boardsize * settings.boardsize) * 5 + 6); }

// this remains constant, but we provide a function anyway
unsigned int OutputSize(GameSettings settings)
{ return (settings.boardsize * settings.boardsize); }


char*        BuildGenerationName(char *str, unsigned int gen);
SaneNetwork* BuildNetwork(GameSettings settings);
GameScore    PlayGo(GameSettings settings, SaneNetwork *net,
                    blueprint &black, blueprint &white, Gtp &connection);

////////////////////////////////////////////////////////////////////////////
// main
int main(int argc, char *argv[])
{
   int            GameCount;     // counter for # of games
   char           genname[60];   // name of a generation file
   GameScore      score;         // game score
   GameSettings   settings;      // encapsulate all game settings

   // create default game settings, parse the command line, and load any
   // specified settings from that, and then validate those settings
   SetGameDefaults(settings);               // set game defaults
   ParseCommandLine(argc, argv, settings);  // parse command line
   ValidateGameSettings(settings);          // validate settings

   // seed the random number generator with the specified seed
   srand(settings.seed);

   // build an appropriate SANE network and two blueprints to be used with it
   SaneNetwork *net = BuildNetwork(settings);
   blueprint black;
   blueprint white;

   // for as many generations as specifieda, play enough games of go to meet
   // the minimum usage of the SANE network.  once the min usage is met, breed
   // the SANE network and start a new generation
   for (unsigned int gen = 0; gen < settings.generations; gen++)
   {  // some output to update any nerds closely watching the terminal...
      printf("entering generation %d of %d...\n", gen+1, settings.generations);

      // reset the frequencies and ratings of the SANE network
      net->reset();

      sleep(1);   // sleep and reset our GTP connection
```

```
      Gtp connection(settings.boardsize, settings.handicap, settings.komi, settings.isWhite);

      // create as many blueprints and play as many games as it takes until
      // all blueprints have been used the minimum number of times
      GameCount = 0;
      do
      {  // create a random blueprint for both the white and black player
         black = net->randomizeBlueprint();
         white = net->randomizeBlueprint();

         // play the game and retrieve the score
         score = PlayGo(settings, net, black, white, connection);

         // rate the blueprint with the score
         net->rate(score.black, &black);
         net->rate(score.white, &white);

         // more output to tease any nerds watching the terminal...
         printf("\n\ngeneration %d of %d\n", gen + 1, settings.generations);
         printf("finished game %d with score: ", ++GameCount);
         printf(" black %f\twhite %f\n", score.black, score.white);

         // keep cycling until min usage for each neurode is met
      } while(!net->isMinUsageMet());

      // breed the network.  once finished, save the network to file
      net->breed();
      net->save(BuildGenerationName(genname, gen + 1));
   }

   return 0;
}

////////////////////////////////////////////////////////////////////////////
// Build an appropriate string to be used for the name of an output
// (generation) file.
char* BuildGenerationName(char *str, unsigned int gen)
{  snprintf(str, 50, "generation.%d.net", gen);
   return str;
}

////////////////////////////////////////////////////////////////////////////
// Build an appropriate SANE network for our co-evolution.  Use the provided
// settings to build network.
SaneNetwork* BuildNetwork(GameSettings settings)
{
   SaneNetwork *net;  // this will point to our network object

   // if we load in an existing network (from a file) do so here
   if (settings.loadInNet)
```

```
   {  net = new SaneNetwork(settings.inNet,
                            settings.mutaterate,
                            settings.blueprintsize,
                            settings.minusage);
   }
   else
   {  // else, we create a brand new SANE network with random weights
      net = new SaneNetwork(InputSize(settings),
                            settings.netsize,
                            OutputSize(settings),
                            settings.mutaterate,
                            settings.blueprintsize,
                            settings.minusage);
      net->randomize(1);
   }

   return net;
}

////////////////////////////////////////////////////////////////////////////
GameScore PlayGo(GameSettings settings, SaneNetwork *net,
                 blueprint &black, blueprint &white, Gtp &connection)
{
   InputVector input;
   OutputVector output;
   GameMove mymove;
   int pass = 0;

   while (1)//(state == play)
   {
   /*-----------------------------black moving-------------------------*/
      input = BuildInputVector(connection, settings, false);
      net->iterate(input, output, &black);
      mymove = getOutputMove(output,connection,settings, false);
      if (mymove.pass){connection.sendpass(false);pass++;}

      else{pass = 0;connection.MakeGtpMove(mymove.x,mymove.y,false);}
      if (pass>=2){break;}
      /*------------------------end black moving-------------------------*/

      connection.showboard();

      /*--------------------------white moving--------------------------*/
      input = BuildInputVector(connection, settings, false);
      net->iterate(input, output, &white);
      mymove = getOutputMove(output,connection,settings, true);

      if (mymove.pass){connection.sendpass(true);pass++;}
      else{pass = 0;connection.MakeGtpMove(mymove.x,mymove.y,true);}
      if (pass>=2){break;}
```

```
      connection.showboard();
      /*------------------------end white moving-----------------------*/
   }//end loop

   GameScore score;
   score.white = connection.newscore(true);
   score.black = connection.newscore(false);


//printf("score %f:\n",returnscore);
   printf ("Captures white %d. Captures black %d. ",connection.captures(true),connection.captures(fals
   connection.showboard();
   connection.printsgf(settings.sgfFile);
   connection.clearboard();
   return score;
}
```

Bibliography

[1] Creavier, Daniel. *AI: The Tumultuous History of the Search for Artificial Intelligence*. Basic Books, New York, New York, 1993, ISBN 0-465-02997-3.

[2] DeLoura, Mark. *Game Programming Gems*. Charles River Media, Inc., Rockland, Massachusetts, 2000, ISBN 1-58450-049-2.

[3] Iwamoto, Kaoru. *GO for Beginners*. Pantheon Books, New York, New York, 1976, ISBN 0-394-73331-2.

[4] LaMothe, André. *A Neural-Net Primer*. Charles RiverMedia, Inc., Rockland, Massachusetts, 2000.

[5] Lubberts, Alex. *Co-Evolving a Go-Playing Neural Network*. The University of Texas at Austin, 2001.

[6] Mitchell, Tom. *Machine Learning*. McGraw-Hill, Boston, Massachusetts, 1991, ISBN 0-07-042807-7.

[7] Moriarty, David E. *Efficient Reinforcement Learning through Symbiotic Evolution*. Kluwer Academic Publishers, Boston, Massachusettes, 1996.

[8] Richards, Norman. *Evolving Neural Networks to Play Go*. The University of Texas at Austin, 1996.

[9] Sharkey, Amanda. *Combining Artificial Neural Nets*. Springer, Great Britain, 1999, ISBN 1-85233-004-X.