Genetic Algorithms for a Multiagent Approach to the Game of Go

碁

by

Todd Blackman

B.S. Computer Engineering, University of Kansas, 2000

Submitted to the Department of Electrical Engineering and Computer Science and the Faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Dr. Arvin Agah, Professor in Charge

Dr. Victor Frost, Committee Member

Dr. Costas Tsatsoulis, Committee Member

Date Defended: April 25, 2003
Abstract

Many researchers have written or attempted to write programs that play the ancient Chinese board game called *go*. Though some of these programs play quite well compared to beginners, few play extremely well, and none of the best programs rely on soft computing AI techniques such as genetic algorithms or neural networks. This thesis explores the advantages and possibilities of using genetic algorithms to evolve a multiagent *go* player. We show how each individual agent plays poorly, while the agents working together play the game significantly better.
Acknowledgments

I would like to thank Dr. Arvin Agah for all of his advice and cajoling. His intelligence and humor made my research experience both enjoyable and rewarding. I would also like to thank Dr. Costas Tsatsoulis and Dr. Victor Frost, both committee members. Dr. Tsatsoulis has been a great mentor and teacher during my years at the University of Kansas while Dr. Frost has been an inspiration and has impacted my future in countless ways. I would also like to recognize the significant use of code from Genetic Algorithms in Search, Optimization, and Machine Learning [6] and Numerical Recipes in C: The Art of Scientific Computing [15].

On a more jovial note, I would like to thank Linus Torvalds for allowing me the opportunity of not using any Microsoft (TM) products including, but not limited to, Word (TM), Excel (TM), Powerpoint (TM), Internet Explorer (TM), and Windows (TM). Revision control was handled by CVS, and this document was typeset using \TeX. Hayley Chapnick and Mara Reichman both gave of their time to proofread this work. Finally, I would like to thank Kristy Blackman for dealing with all of the time I spent in my office working on my computer and Shawn Steiman for inspiring me to finish my thesis in a timely manner.
Contents

List of Figures .............................. x

List of Tables ............................ xii

1 Introduction ............................ 1
   1.1 Organization .......................... 5

2 Background and Related Work ........ 6
   2.1 The Game of Go ....................... 6
       2.1.1 Surrounding Territory ............ 8
       2.1.2 Capturing ........................ 8
       2.1.3 Eyes .............................. 10
       2.1.4 Live and Dead Stones ............ 10
       2.1.5 Rule of Ko ........................ 11
       2.1.6 Seki (Stalemate) .................. 12
       2.1.7 Scoring ............................ 13
       2.1.8 Other Board Sizes ................ 13
       2.1.9 Go Player Ranking ................ 14
   2.2 Relevant AI and Computational Techniques ......... 15
       2.2.1 Search Techniques .................. 15
       2.2.2 Neural Networks .................... 16
       2.2.3 Multiagent Systems ................. 17
       2.2.4 Genetic Algorithms ................ 19
       2.2.5 Thread Pools ....................... 20
2.3 AI and Games ........................................ 21
  2.3.1 Minimax Search ................................. 21
  2.3.2 AI and Other Games ......................... 23
2.4 AI and Go ........................................... 25
  2.4.1 Search Space .................................. 25
  2.4.2 Neural Network Techniques ................. 26
  2.4.3 Traditional Techniques in Go Programs ... 27
  2.4.4 Genetic Algorithm Techniques in Go Programs .. 29
  2.4.5 Other Techniques and Hybrids ............ 31

3 Methodology ........................................... 33
  3.1 Design Overview ................................ 33
  3.2 Stone, Board, and Game Classes ............ 34
  3.3 User Interfaces ................................ 36
  3.4 Genetic Algorithm ............................. 37
  3.5 Moderator ........................................ 38
    3.5.1 Probability Board ......................... 38
  3.6 Agent Network Architecture ................ 39
  3.7 Genetic Algorithm Trainer .................. 41
  3.8 Genetic Algorithm Player .................... 42
    3.8.1 GA Player Details ......................... 42
    3.8.2 Fitness Function ......................... 42
  3.9 Agents .......................................... 43
    3.9.1 Random Agent .............................. 43
    3.9.2 Follower Agent ............................ 43
    3.9.3 Opener Agent .............................. 44
    3.9.4 Capturer Agent ............................ 44
    3.9.5 Tiger's Mouth Agent ..................... 45
    3.9.6 Extender Agent ............................ 45
  3.10 Regressions .................................... 46
  3.11 Unimplemented Features .................... 47
4 Experiments and Results

4.1 Individual Agent Experiments ........................................... 49
   4.1.1 Opener Agent ......................................................... 49
   4.1.2 Single Random Agent ............................................... 51
   4.1.3 Extension Agent ..................................................... 53
   4.1.4 Capturer Agent ....................................................... 54
   4.1.5 Follower Agent ....................................................... 56
   4.1.6 Tiger’s Mouth Agent .................................................. 57
4.2 Multiagent Experiments ..................................................... 57
   4.2.1 Five Random Agents .................................................. 59
   4.2.2 Multiagent Configuration ............................................ 59
   4.2.3 Multiagent Configuration, Large Population ...................... 62
4.3 Summary ................................................................. 64

5 Conclusion

5.1 Contributions ............................................................ 65
5.2 Limitations .............................................................. 66
5.3 Future Work ............................................................ 67

Bibliography

Appendix A: Doxygen Code Reference

A.1 Cross-references .......................................................... 70
   A.1.1 Exodus Class Hierarchy ............................................. 70
   A.1.2 Exodus Compound List ............................................. 71
   A.1.3 Exodus File List ..................................................... 72
   A.1.4 Exodus Related Pages ............................................. 73
A.2 Exodus Class Documentation .............................................. 73
   A.2.1 Agent Class Reference .............................................. 73
   A.2.2 AgentShell Class Reference ....................................... 76
   A.2.3 Blackboard Class Reference ....................................... 77
   A.2.4 Board Class Reference .............................................. 78
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.2.5</td>
<td>DummyGenerator Class Reference</td>
<td>88</td>
</tr>
<tr>
<td>A.2.6</td>
<td>ExtenderAgent Class Reference</td>
<td>89</td>
</tr>
<tr>
<td>A.2.7</td>
<td>FollowerAgent Class Reference</td>
<td>92</td>
</tr>
<tr>
<td>A.2.8</td>
<td>Ga Class Reference</td>
<td>96</td>
</tr>
<tr>
<td>A.2.9</td>
<td>Game Class Reference</td>
<td>112</td>
</tr>
<tr>
<td>A.2.10</td>
<td>GaTrainerInterface Class Reference</td>
<td>123</td>
</tr>
<tr>
<td>A.2.11</td>
<td>GenAlgoGenerator Class Reference</td>
<td>126</td>
</tr>
<tr>
<td>A.2.12</td>
<td><code>global_data_t</code> Struct Reference</td>
<td>134</td>
</tr>
<tr>
<td>A.2.13</td>
<td>GoModemInterface Class Reference</td>
<td>135</td>
</tr>
<tr>
<td>A.2.14</td>
<td>GroupStatsAgent Class Reference</td>
<td>136</td>
</tr>
<tr>
<td>A.2.15</td>
<td>GUIS Interface Class Reference</td>
<td>138</td>
</tr>
<tr>
<td>A.2.16</td>
<td>IGS_Interface Class Reference</td>
<td>141</td>
</tr>
<tr>
<td>A.2.17</td>
<td>Individual Struct Reference</td>
<td>142</td>
</tr>
<tr>
<td>A.2.18</td>
<td>Interface Class Reference</td>
<td>144</td>
</tr>
<tr>
<td>A.2.19</td>
<td>Moderator Class Template Reference</td>
<td>146</td>
</tr>
<tr>
<td>A.2.20</td>
<td>move_t Struct Reference</td>
<td>148</td>
</tr>
<tr>
<td>A.2.21</td>
<td>msg_t Struct Reference</td>
<td>149</td>
</tr>
<tr>
<td>A.2.22</td>
<td>NeuralNetGenerator Class Reference</td>
<td>150</td>
</tr>
<tr>
<td>A.2.23</td>
<td>NNGS_Interface Class Reference</td>
<td>150</td>
</tr>
<tr>
<td>A.2.24</td>
<td>OpenerAgent Class Reference</td>
<td>151</td>
</tr>
<tr>
<td>A.2.25</td>
<td>Population Struct Reference</td>
<td>153</td>
</tr>
<tr>
<td>A.2.26</td>
<td>PreCodex Class Reference</td>
<td>154</td>
</tr>
<tr>
<td>A.2.27</td>
<td>ProbBoard Class Reference</td>
<td>155</td>
</tr>
<tr>
<td>A.2.28</td>
<td>RandomAgent Class Reference</td>
<td>159</td>
</tr>
<tr>
<td>A.2.29</td>
<td>Stone Class Reference</td>
<td>160</td>
</tr>
<tr>
<td>A.2.30</td>
<td>Subthread Class Reference</td>
<td>165</td>
</tr>
<tr>
<td>A.2.31</td>
<td>Subthread_test Class Reference</td>
<td>168</td>
</tr>
<tr>
<td>A.2.32</td>
<td>testCodex Class Reference</td>
<td>169</td>
</tr>
<tr>
<td>A.2.33</td>
<td>TextInterface Class Reference</td>
<td>170</td>
</tr>
<tr>
<td>A.2.34</td>
<td>TigersMouthAgent Class Reference</td>
<td>173</td>
</tr>
<tr>
<td>A.3</td>
<td>Exodus File Documentation</td>
<td>177</td>
</tr>
</tbody>
</table>
A.3.1  agent.cpp File Reference ........................................ 177
A.3.2  agent.h File Reference ........................................... 178
A.3.3  bdemo.cpp File Reference ......................................... 179
A.3.4  blackboard.cpp File Reference .................................... 181
A.3.5  blackboard.h File Reference ....................................... 182
A.3.6  board.cpp File Reference .......................................... 182
A.3.7  board.h File Reference ............................................ 183
A.3.8  config.h File Reference ........................................... 184
A.3.9  dummygenerator.cpp File Reference ............................ 185
A.3.10 exodus.h File Reference ........................................... 186
A.3.11 extenderagent.cpp File Reference ............................... 189
A.3.12 followeragent.cpp File Reference ............................... 190
A.3.13 ga.cpp File Reference ............................................. 191
A.3.14 ga.h File Reference ................................................ 192
A.3.15 gafunc.h File Reference ........................................... 193
A.3.16 game.cpp File Reference .......................................... 194
A.3.17 game.h File Reference ............................................. 196
A.3.18 gatypes.h File Reference .......................................... 197
A.3.19 genalgogenerator.cpp File Reference ............................ 198
A.3.20 ginterface.cpp File Reference .................................... 199
A.3.21 groupstatsagent.cpp File Reference ............................. 200
A.3.22 iinterface.cpp File Reference .................................... 201
A.3.23 interface.cpp File Reference ..................................... 202
A.3.24 interface.h File Reference ....................................... 202
A.3.25 main.cpp File Reference ......................................... 203
A.3.26 moderator.t File Reference ....................................... 210
A.3.27 move.cpp File Reference ......................................... 211
A.3.28 move.h File Reference ........................................... 212
A.3.29 openeragent.cpp File Reference ................................. 213
A.3.30 outputgen.h File Reference ..................................... 214
A.3.31 probboard.cpp File Reference ................................... 215
Appendix B: Running the Program

Index
# List of Figures

1.1 Summation Network Architecture. .......................... 4

2.1 A Go Board. ............................................... 7
2.2 Surrounded Territory. ....................................... 8
2.3 White Stones to be Captured on Next Move (in Atari). ... 9
2.4 Capturing. .................................................. 9
2.5 Capturing II. ............................................... 10
2.6 Dead Group. ............................................... 11
2.7 Example of Go Without the Ko Rule. ....................... 12
2.8 Seki. ....................................................... 13

3.1 Board Locations For a 9 x 9 Board. ........................ 35
3.2 An ASCII Board. ........................................... 36
3.3 Graphical User Interface Screen Shot. ....................... 37
3.4 Tiger’s Mouth Formation. .................................. 45
3.5 Extensions. ................................................ 46

4.1 GA Data Plot With Opener Agent. .......................... 50
4.2 GA Data Plot With Randomly Playing Agent. ............... 52
4.3 GA Data Plot With Extension Agent. ........................ 54
4.4 GA Data Plot With Capturer Agent. ........................ 55
4.5 GA Data Plot With Follower Agent. ........................ 57
4.6 GA Data Plot With Tiger’s Mouth Agent. ................... 58
4.7 GA Data Plot With Five Random Agents. ................... 60
4.8 GA Data Plot With All Agents. ............................. 61
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9</td>
<td>GA Data Plot With All Agents (Large Population)</td>
<td>63</td>
</tr>
<tr>
<td>4.10</td>
<td>Agent Comparison</td>
<td>64</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Opener Agent Data. ........................................ 50
4.2 Randomly Playing Agent Data. .......................... 51
4.3 Extension Agent Data. ..................................... 53
4.4 Capturer Agent Data. ....................................... 55
4.5 Follower Agent Data. ....................................... 56
4.6 Tiger’s Mouth Agent Data. ................................. 58
4.7 Five Random Agent Data. ................................... 59
4.8 All Five Agents Data. ....................................... 60
4.9 All Five Agents Data (Large Population). ............. 63
Chapter 1

Introduction

Games have often been used to test new concepts in artificial intelligence because of their relative simplicity compared to other more complex possibilities such as simulations and real-world testing. Go has the potential to excel as a testbed for AI concepts because of the complexity of the tactics and strategies used to play the game well. These complexities resemble real-world problems better than most other games. Brute-force search cannot be used exclusively to play this game, as in other games, because of go’s huge branching factor which starts out at 361 at the beginning of a game and approximately decreases by one after each move.

With pure search ruled out as a viable method for playing go, one must turn to more intelligent methods such as pattern recognition or rule-based deduction. Complexity often plagues go programmers because of the intricacies of how a player must think about the game—often remote locations on the board influence
a local situation. Sometimes, what one would hastily consider the best move really reveals itself as the worst move because of global concerns on the board. Current go programs play at only the level of a skilled novice, and we believe that these limitations exist because of the programs’ architectures and their insistence on using only traditional methods such as pattern-matching, hard-coded rules in computer code, and minimax with alpha-beta pruning.

What we propose is that programs should play go using relatively simple agents that combine to play the game well. Traditional methods have their place in go programs, but to play an abstract and multi-faceted game one must use an abstract and multi-faceted approach. Genetic algorithms have been employed to play complex games, but these genetic algorithms often use evolved values that are too low-level to allow the program to attain the skill required to play well. By low-level, we mean that these values allow for the evolution of useful information such as patterns or algorithmic code, but to play the game on a professional level one would need too many of these individual pieces of information. Analogously, it would be like creating a neural network with \(3 \times 19^2\) inputs (representing the \(19^2\) board locations and the three possible states for each location: white, black, empty) and \(19^2\) outputs. Training an artificial neural network of this size will remain inconceivable for quite some time. Likewise, trying to evolve a set of rules using a genetic algorithm would fail in much the same way. Too many rules exist, and evolving them would take too long.
Our program and approach differ from most current go programs. Other programs are extremely complex, representing huge amounts of go knowledge. They eventually become unwieldy, difficult to maintain, hard to follow, and tricky to improve upon.

The motivation for this work is thus to study whether a set of relatively simple agents can each look at the problem from their own perspective after which genetic-algorithm-evolved weights will allow the agents’ solutions to be summed together to produce a final solution. This approach exchanges the ability to fine-tune the program with the ability to incorporate more agents, and thus more knowledge, in a consistent and scalable way. Our research will illustrate a novel multiagent approach to playing games that uses a multilayer network to suggest moves based on the moves suggested by each individual agent.

The problem is therefore to develop a set of agents that generate a value for each location on the board (higher values representing a more highly recommended move). These values are entered into a matrix such that each location on the go board corresponds to a place in the matrix. These matrices are then normalized and combined non-linearly using genetic-algorithm-evolved weights. The summation network architecture, shown in Figure 1.1, resembles a neural network in that the resulting matrix of values is generated from a weighted sum of a set of weighted sums. The resulting move to play will then be either the highest value in the matrix or chosen probabilistically with higher values receiving a greater
probability of being chosen. Thus, our goal is to:

1. Show that as the genetic algorithm evolves, the program plays better.

2. Show that as more agents are added and the chromosomes evolved, the program plays better.

3. Show that simple agents can be used to achieve better performing meta behavior.

4. Illustrate a novel multiagent approach to programming games.

These considerations are obviously limited by our ability to program knowledge about go; consequently, we have an additional goal of becoming better players so as to program the game better. Also, we tried to develop agents that were not exceedingly complex. They should have specific and well-defined expertise and a clear focus.
1.1 Organization

Chapter 2 begins with information about the game of go including the rules of the game, a few direct implications of the rules, basic concepts, how to score a game, and how players are ranked. This is followed by a description of some techniques relevant to this thesis. Following these, we consider how AI relates to games, and in particular, to the game of go.

The next chapter (Chapter 3) explores the design of the program written for this thesis. It explains all of the main components of the program and provides a top-level view of the program’s architecture.

In Chapter 4 we explain all of the experiments performed along with the results of these experiments. In addition to the results, some conclusions are drawn as to the relevance of the data and what the interpretation of the data is. The next chapter wraps up this thesis and summarizes the contributions, limitations, and future of this work.
Chapter 2

Background and Related Work

2.1 The Game of Go

The game of go (also called goe, igo, baduk, wei-chi, weichi, weiqi, wei-qi, etc.) is a board game of perfect information\(^1\) played by two opponents on a 19 \times 19 grid as shown in Figure 2.1. Each player takes turns placing his or her piece (called a *stone*) on an intersection starting with the individual playing black. The opponent, as white, places his or her stone, and the game continues until both players pass their turns in succession. No stone can be moved unless it is captured (as will be explained later), and all stones are completely equal in power unlike pieces in games such as chess.

\(^1\) *Perfect information* is a term used to describe games that allow the players to see the entire state of the game at all times. No guessing or probability is involved as in games such as backgammon or bridge which have uncertainty and hidden state respectively.
Notice that the letter $I$ is not used which is to prevent confusion with the letter $J$ when transcribing games.

Figure 2.1: A Go Board.
2.1.1 Surrounding Territory

The goal of the game is to surround more territory than your opponent with a secondary goal of capturing your opponent’s stones. Each surrounded intersection or captured stone is worth one point\(^2\). In general, surrounding territory is considered much more important by anyone versed in go. Figure 2.2 shows three examples of surrounded territory: The black group on the left surrounds nine points, the white group surrounds four points, and the right-most black group surrounds two points.

2.1.2 Capturing

As stated, there is also a secondary way to gain points—capturing the opponent’s stones. To capture a single stone, one must play on all adjacent intersection points that are at right angles to the stone(s) to be captured. Figure 2.3 shows three examples of a white stone about to be captured by a black stone if black were to play on the locations marked \(A\). The white stone is \textit{in atari}.

\(^2\)An exception exists when a stalemate condition arises as will be explained later
Figure 2.3: White Stones to be Captured on Next Move (in Atari).

Figure 2.4: Capturing.

To capture groups\(^3\) one must play on all the liberties\(^4\) of the group. Figure 2.2 illustrates three disjoint groups, and Figure 2.4 shows an example of a possible state of the board during a game. In this figure, if it were white’s turn, she could play at the location marked A to capture ten black stones. On the other hand, if it were black’s turn to play, he could also play at A to capture four white stones. Furthermore, if white were to play at location B in Figure 2.5, then white would only capture \textit{seven} stones as the two black stones at the top of the board are \textit{not} part of the black group below it.

\(^3\)A group is defined as a set of stones that connect adjacently to each other through the straight lines on the board (i.e., at right angles). \textit{Diagonals} do not count.

\(^4\)A liberty is simply an empty location adjacent to a group.
2.1.3 Eyes

It follows that any group is unconditionally safe if it can partition itself into at least two sections (also called eyes). Figure 2.2 illustrates multiple groups that have two eyes each—the two rightmost groups. These groups are unconditionally safe because if the opponent plays in an eye with a single intersection inside that eye, then it commits suicide (all of its liberties are taken). One can logically rationalize the unconditional safeness of a group with two or more eyes by imagining that to kill the group, one would have to play in all eyes at the same time (after surrounding the group first) which is of course illegal. If the eyes are too big, one’s opponent could create a living group inside an eye and then the eye could become useless. As an aside, the group on the left in Figure 2.2 has an uncertain living ability; it is neither alive nor dead as it stands.

2.1.4 Live and Dead Stones

Surrounding territory is crucial while playing go, but there is a very important twist that can make what seems like one’s territory actually one’s opponent’s.
If a group of stones (at the end of the game after both sides pass in succession) could not possibly survive an attack (i.e., it does not have two eyes or the ability to make them if pressed), then that group is removed from the board and given to the opponent. After playing many games, one soon learns how to identify hopelessly dead groups of stones—if a disagreement arises about whether a group is alive, then the game continues. Figure 2.6 shows a black group that is hopelessly dead and the resulting board fragment after it is removed.

### 2.1.5 Rule of Ko

The last important concept in go is that of the rule of ko. This rule states that no board state may be repeated. Stated another way, livelock is not allowed to occur. The sequence of plays in Figure 2.7 illustrates an example of what could happen if this rule were not in effect. The first move by white ($S_0 \rightarrow S_1$) captures
a black stone, while the second move \((S_1 \rightarrow S_2)\) captures a white stone. This second move \((S_1 \rightarrow S_2)\) is illegal, and black \textit{must} play somewhere else. One can see that without the ko rule, a livelock situation would arise and both sides would continually gain the same number of prisoners (stones).

\subsection*{2.1.6 Seki (Stalemate)}

Seki can be viewed as a localized stalemate condition. In the game of go, there are situations when neither side can count his or her territory because both sides would have dead groups \textit{if} they played first. Figure 2.8 illustrates this condition. If it were white’s turn, she could not play at \(A\) (suicide), while playing at \(C\) would fill in her own eye. The only option is for white to play at \(B\), but that would allow black to play at \(C\) on the next turn, capturing the white group. Likewise, if it
were black’s turn to play, C would be suicide and A would be filling in his own eye. A play by black at B would allow his small group to be killed with a white play at A. Thus, the three locations A, B, and C are not counted as territory.

2.1.7 Scoring

Many variations exist for scoring finished go games, but for simplicity a player’s score is calculated by first removing dead groups (which become prisoners), then counting the number of captured prisoners, and finally counting the number of intersections completely surrounded by one’s own color. In many games the person to play second may get additional points called a komi to compensate for going second which can vary from 0.5 to 5.5 points.

2.1.8 Other Board Sizes

Go can be played on boards of any size, but historically, games have been played on boards of size 19, 17, 13, and nine (for a total of 361, 289, 169, and 81 intersections respectively). Nine is used to teach beginners and is also often used in computer
go games because it has a smaller search space by orders of magnitude. Boards of size 17 are usually used when one wants to have the essence of a full game but does not have the time for a full $19 \times 19$ board. A $9 \times 9$ board, for example, has a very different character than a $17 \times 17$ or a $19 \times 19$ board.

### 2.1.9 Go Player Ranking

Go has a well standardized hierarchy of skill levels that allow players to compete on a fairly equal basis with standardized handicaps. A complete beginner that has played a game and knows the rules starts off at 30 kyu. The scale progresses to one kyu which is the best kyu ranking one can attain. After this, the scale continues at one dan up to nine dan, the highest amateur ranking possible. Thus, for the kyu ranks, lower values imply a better player, while for the dan ranks, higher values imply a better player. To confuse the issue, professionals rank themselves on the dan scale as well from one to nine, but their rankings are usually considered stronger. Thus, a five dan professional and a five dan amateur would not usually be able to play an equal game (with no handicap). Further complicating the matter, different countries and groups may not completely coincide with each other on strength. For example, a Korean eight kyu might not be equal to a British eight kyu. Computer programs are often given honorary diplomas of a certain level, but these can be misleading as the programs often play well but make horrible mistakes every once in a while.
2.2 Relevant AI and Computational Techniques

Numerous AI and computational techniques exist that find uses in go programs. Search techniques, neural networks, thread pools, multiagent systems, and genetic algorithms make up what can be considered relevant paradigms to programming the game of go [12, 19, 20].

2.2.1 Search Techniques

Many methods of search exist in the repertoire of most computer scientists. Quite a few are a variation on either breadth-first or depth-first search. These methods include uniform-cost, iterative deepening, bidirectional, and depth-limited searches. Canonical breadth-first search expands a new tree layer at every iteration while depth-first search expands one element from the next layer at every iteration. Uniform-cost search, a variant of the breadth-first search, expands the next cheapest node at every iteration, and iterative deepening search is simply breadth-first with the number of layers increased at each iteration. Bidirectional search attempts to search from both the goal and the starting state simultaneously. Finally, depth-limited search is a variant of depth-first search that includes a provision for the maximum depth that will be searched before backing up.

These search methods can be improved upon by incorporating knowledge about the problem space to help make the search more efficient. The simplest informed
method is a greedy search that always expands the node that appears closest to the solution. Another method is $A^*$ and its close relative $IDA^*$ (iterative deepening $A^*$) [19]. In $A^*$ the next node to be expanded is the node that has the lowest value for a variable $\phi$. This variable $\phi$ is defined as the cost from the initial node to the current node in question plus the estimated cost of the best path from that node to the goal.

2.2.2 Neural Networks

A neural network can be viewed as a random search method that yields a function that sometimes cannot be found by more traditional methods. Neural networks are often composed of multiple layers of neurons, and each neuron in each layer can be connected to all of the neurons in the immediately adjacent layer. Each connection has a weight associated with it, and each input neuron receives some part of an input signal which is passed through a non-linear activation function (which determines if the neuron will fire). A neuron that fires has its output signal multiplied by the aforementioned weights which is then directed into all the connected neurons in the next layer. Each neuron in this layer receives a signal from each input neuron, and these signals are then summed and once-again passed through an activation function to determine its output. This process continues for all neurons in all layers until an output is received on the output layer of the neural network.
Many paradigms exist for training neural networks, but the most common is the backpropagation technique which compares the output of the network with a correct training example. The delta between the expected and actual output is backpropagated through the network to modify the weights between the neurons. Other methods exist to modify the weights [16].

The usefulness of neural networks is heavily influenced by a number of factors: the number of neurons, the number of layers, the choice of whether to allow connections within a layer, the choice of whether to allow connections back to previous layers, the choice of training data, the learning rate, and the choice of the training method. These all can affect the quality of the resulting network. There is much trial-and-error involved in neural network design.

2.2.3 Multiagent Systems

As described in Gerhard Weiss's book, “An agent is a computer system that is situated in some environment, and that is capable of autonomous action in the environment in order to meet its design objectives” [20]. Though there is much disagreement on the exact definition of an agent, most agree that agents are indeed autonomous. Intelligent agents have the characteristics of agents but also have intelligent traits such as reactivity, pro-activeveness, and social ability [20]. A robot capable of interacting in its environment might be considered an intelligent agent, while the thermostat in one's home would not be.
There are many issues that must be resolved if one wants to create a multiagent system. One must consider what kind of information the agent will have about the environment that it exists in. A computer program running a single thread for each agent is quite different from a robot traversing ice-sheets at the poles of the earth. Is the environment real or virtual? Will the agent receive all inputs through a socket, via shared memory, or by way of an external bump sensor? All of these questions are important to designing a multiagent system. Closely related to this concept of an agent’s environment is the idea of an ontology which is, “...a specification of the objects, concepts, and relationships in an area of interest” [20]. All agents must exist within some framework.

Agents that simply exist in an environment that they can sense are different from cases where a method of communication becomes important to consider. If the agents exist in a virtual world on a single computer they might communicate via shared memory or via sockets, while distributed agents might communicate over a wireless network or even with physical means such as actual speech.

Being able to communicate is of little use if one does not have a common language and protocol for exchanging messages, thus these concerns arise and must be considered by a multiagent system designer. Many languages and protocols exist for agent communication [8], but the important idea is that these issues must be carefully considered for the application at hand.

A high-level goal must exist that enables the agents to actually achieve some-
thing useful. Even if no single entity has established a clear plan or goal, often a
goal is inherent in the behavior of each single agent. For example, one could argue
that a goal of human society is to survive. This goal is exemplified in each indi-
vidual human-being’s actions. Analogously, computer agents interact together to
produce some type of useful output. Agents can be cooperative or self-interested
which both lead to different types of agent interactions. Cooperative agents may
actually negotiate an agreed upon goal, while self-interested agents might achieve
their own personal goals leading to a net benefit to at least themselves.

2.2.4 Genetic Algorithms

Genetic algorithms are search methods that approximate biological genetics (i.e.,
simulate evolution) in an attempt to find a solution or goal for a particular problem
[11, p. 1–6]. Essentially, a genetic algorithm (or GA) begins with the creation
of a set of entirely random chromosomes of alleles. These “arrays of bits” are
translated into parameters or data that can then be tested to see how well they
approximate the solution that is being searched for. This function that finds how
well an individual chromosome performs is called a fitness function or objective
function.

GAs progress by first creating a population of randomly generated chromo-
somes. The fitness of each of these chromosomes is calculated and then pairs
of chromosomes are picked (with higher fitness values more likely to be picked).
These pairs possibly undergo crossover and/or mutation. The resulting chromosome then becomes a new member of the new population. The process repeats itself for each generation until a chromosome with some minimum cutoff fitness is found, or until a maximum number of generations have transpired.

GAs perform a form of directed random search. The efficacy of using this approach relies heavily on the choice of fitness function, the size of each generation, the maximum number of generations, the crossover rate, the mutation rate, and sometimes a scaling factor [6, p. 1–86].

Many variations have been proposed that modify the basic GA paradigm. For example, Rosin and Belew describe [18] a co-evolution method that evolves two separate populations that compete with each other after each generation. The idea is that as population $\alpha$ evolves an individual that can beat individuals in population $\beta$, population $\beta$ will have to evolve in order to keep up. This competition is analogous to different species competing in the wild.

### 2.2.5 Thread Pools

Though not an AI technique, thread pools are important for our program. Thread pools begin by starting a finite number of threads, each capable of doing some work. When work becomes available, it is given to a thread in the thread pool to do. If there is more work to do than threads available, then the work must simply wait to be done. This method saves some overhead because each time work must
be done, the operating system does not need to create and destroy a new thread which can take much time.

A problem with this method is that one must determine an optimal number of threads to start within the pool. The number of processors, the complexity of the problem, and the amount of work to be done all affect the usefulness and size of thread pools.

2.3 AI and Games

The field of artificial intelligence has been applied to the development of game playing programs, particularly two-person games of perfect information. One of the most important ideas is minimax search along with alpha-beta pruning.

2.3.1 Minimax Search

Minimax search involves enumerating all possible moves for one of the two players followed by enumerating every possible response to each of these initial choices. This process is continued until all the leaves of the tree represent final game states. A tree such as this allows the program to play a game perfectly, but for all but the most trivial games this approach requires too much time and memory.

As a compromise, programmers might only allow the tree to expand to a certain level and then assign an estimate of the quality of the game state with an
evaluation function. At any given point in the game, the minimax tree allows the programmer to pick the best move assuming the evaluation function is accurate. The problem with this approach is that this evaluation function only approximates the quality of a position and may take quite some time to evaluate. Additionally, if the evaluation function is accurate, then there would be no need to have any tree in the first place.

**Alpha-Beta Pruning** As an improvement to using an evaluation function and limiting the number of levels, one can use alpha-beta pruning which keeps two values: an \( \alpha \) value and a \( \beta \) value. These values store the highest and lowest evaluation values. Low values are good for one side, while high values are good for the other player. Assuming that players always play the best move possible, one can prune tree branches that are worse than some other possible move that the algorithm has seen before. For example, while performing a depth-first search one finds that one player can achieve an evaluation of 10. Then, deeper in the search, a possible sequence of moves leads to a value of five. The subtree with the value five will be pruned and no further moves from that path will be considered. This method always returns the same value as pure minimax search with depth-limitation, so this pruning method is almost always used along with minimax search.
2.3.2 AI and Other Games

Other games of perfect information such as Reversi, Pente, checkers, chess, and go-moku can derive benefits from AI techniques just as go can. The main difference, in our opinion, lies in their branching factors and the manner in which each piece affects other pieces. Reversi, checkers, and chess all have relatively small branching factors making them much more conducive to traditional approaches such as alpha-beta search with move-ordering and other advanced pruning techniques. Pente and go-moku, on the other hand, have similar branching factors to go, but have much simpler interactions between the pieces.

Peter Norvig [13] discusses in depth the construction and refinement of a Lisp program to play Othello. The evaluation function and some of the details are useless for programming go, but the work has much to contribute relative to efficiency issues, searching, and other miscellaneous topics. For example, Norvig uses minimax search with alpha-beta pruning, but he also suggests improvements upon this method. One improvement is to order the moves at each node in the search tree in an attempt to allow pruning to remove more nodes. This ordering can be accomplished if certain locations on the board are more advantageous than others, i.e., better moves are placed first.

Another method is to find the evaluation value for each successor of the current node and then proceed to traverse these, not in an uninformed depth-first manner, but rather in order by evaluation value so that the best successor node is searched
next. This potentially allows for a greater number of pruned nodes as well.

Another improvement to playing games of perfect information is to keep track of *killer* moves. This method would have the programmer store moves that were exceedingly bad (moves which were discounted while performing minimax search). If these moves show up later in the search then they are placed first, before other nodes in the minimax tree.

Norvig continues with the idea of generating abstract heuristic values that are relevant to the game such as mobility in the game of Othello or pawn structure in the game of chess. Go, for example, has potential candidates for approaches of this nature such as thickness and good shape which both describe abstract concepts that relate to good moves. One usually wants to build thickness and to make good shape.

Another method is *forward pruning* which requires a function that removes obviously poor moves from the search. It is difficult to do and very subjective. While out of favor as a rigorous method, it is a necessity for games with large branching factors.

Programs that think while the opponent is playing can gain some advantage, and the use of board hashing and opening book databases can help programs’ strengths as well. Also, exhaustive searching near the end of a game can be an option for some games such as Othello, but may not be feasible in go.
2.4 AI and Go

2.4.1 Search Space

The number of possible states $S$ on a go board of size 19 is $S = 3^{19^2} \approx 1.74 \times 10^{172}$. There are $19 \times 19 = 19^2$ intersections on the board, and each location has three possible states: black, white, or empty. Though many of the states are very unlikely to occur, one can appreciate the size of the complete search tree. Even accounting for symmetries such as color-inversion symmetry, rotational symmetry, reflection symmetry, and the fact that the number of stones of each color are usually roughly the same, the number is huge.

One of the greatest difficulties in programming go is the immense branching factor in the game. The first move in a game of go can be any one of $19^2 = 361$ moves, while chess has only 20 initial moves. Reversi only has four moves possible at the onset. Though the number of possible moves fluctuates as play progresses, these games cannot be compared to the order of magnitude difference in the branching factor of go. Chess has $20 \times 20 = 400$ game states after the initial two moves, while go has $19^2 \times (19^2 - 1) = 129,960$ game states after the initial two moves! Including a third move brings chess up to approximately $400 \times 25 = 10,000$ states, while go has $129,960 \times (19^2 - 2) = 46,655,640$ states. This example illustrates why nobody has (and possibly ever will) play go well by brute force—there are already three orders of magnitude difference in the size of the search
space after just three moves. One can prune many moves throughout the game, but even the ability to prune three-fourths of the moves would result in a huge search space.

2.4.2 Neural Network Techniques

Many programmers and researchers have used neural networks to attempt to play go well including neural networks with GA-evolved weights. For example, Markus Enzenberger [4] created an architecture for his program NeuroGo that evaluated the board using a neural network with backpropagation and temporal difference learning. The network received its input from a feature expert, while a relation expert controlled the connections between the layers of the neural network. In addition, there existed an external expert that could override the neural network’s output for a small class of problems. The idea of using experts to extract features from the gobans\(^5\) is interesting, but few details of the inner workings of NeuroGo were available.

Paul Donnelly et al. [3] studied neural networks that were evolved using genetic algorithms. They used a 9 × 9 board along with a three layer non-recurrent network. They also postulated that recurrent networks with more than a single hidden layer might be better suited for the non-linearities of go. Their experiments consisted of creating a population of 32 networks that all played each other. The

\(^5\)A goban is simply another term for a board for playing go.
network winning the most games overwrote the network that lost the most at
the end of the cycle. They used the networks to evaluate the quality of a given
position which was accomplished via a single output neuron and input neurons
that derived their inputs directly from the goban. Each location on the board
corresponded to three individual input neurons (one each for white, black, and
empty positions). The resulting input layer thus had $9 \times 9 \times 3 = 243$ neurons.
The authors found that the networks slowly got better, but the network still
played poorly compared to modern go programs. This approach theoretically has
merit, but to implement this architecture on a full board using only two hidden
layers each analogous to those in the paper, one would need $19 \times 19 \times 3 = 1083$
input neurons and $19 \times 19 = 361$ neurons in each hidden layer. This results in
$1083 \times 361 + 361 \times 361 + 361 = 521,645$ connections. A network of this caliber
could be constructed, but recurrent connections might be required for it to play
well, and training time would be prohibitive.

2.4.3 Traditional Techniques in Go Programs

Some go programs do not use any soft computing techniques, i.e., they do not
rely on learning, genetic algorithms, neural networks, cellular automaton, or
other similar approaches.

In *Computer Go as a Sum of Local Games: An Application of Combinatorial
Game Theory* [10], Müller studied methods of playing go that generate moves by
first enumerating possible moves based on small, local views of the goban. These
moves are filtered, ordered, checked, and refiltered. The best move is executed.
If a ko ensues, a special ko module is called. If no move survives this process,
the program passes. At the core of this approach is a pattern matching database
that uses Patricia trees which is a method normally used to search large text
databases such as dictionaries. This program contained about 3000 patterns, and
pattern matching was its chief element. This reflects a very prominent trend
across many go programs: they often rely heavily on vast databases of patterns
that have been built by hand. These pattern databases make these programs
better, and the implementation of these databases is not a trivial task. The use of
large databases proves nothing about a program’s “intelligence” since it becomes
in essence a sophisticated lookup table. There does remain the possibility of
learning patterns as the program plays, but techniques such as these would not
fall under the category of traditional techniques.

Another prominent program, *The Many Faces of Go* [5], as of 1993 had an open-
ing move database that contained around 45,000 moves and a pattern database
of about 1,000 patterns. This program contained a rule-based expert system with
around 200 rules that were used to suggest moves to look into further. Addition-
ally, dynamic knowledge was stored about the state of the board which was
generated with algorithmic C-code [5].

Though this investigation of traditional techniques has been very cursory, these
programs represent some prominent themes in most strong go programs: they construct meta-data based on the state of the board and use this meta-data along with large databases of patterns to decide what move to play next. Rarely does learning or extensive minimax-style search play a role in the skill of these games.

2.4.4 Genetic Algorithm Techniques in Go Programs

Many attempts have been made to create a program that plays go by using genetic algorithms. None have been successful at creating a world-class player, but nobody has accomplished this feat without genetic algorithms either. What follows is a perusal of some attempts to use genetic algorithms to play this game.

Da Silva [2] used GAs to evolve a go evaluation function for $7 \times 7$ boards. The evaluation function worked by attempting to translate a given board into a new board that represented how the final configuration of the game would be. The evaluation function then looked at who won to calculate the fitness. Essentially, the genetic algorithm attempted to evolve an evaluation function that could be used in minimax searches with alpha-beta pruning. The evolved parameters were a set of low-level functions that performed simple calculations based on the board state. These functions, organized as the chromosome dictates, produced what the author called an $S$-expression, which is a significant component of calculating a board evaluation and consequently the fitness. Da Silva’s approach yielded a player that on average never beat a defacto opponent called Wally, a freely
available public domain go program.

Jeffrey Greenberg has written a program using genetic algorithms to play go [7]. He feels that go represents a good test-bed that approaches the complexity of real problems while not being as complex as a commercial application. One could argue with this premise since go is easily as complex to program as a modern commercial software package—why else would modern go programs remain such poor players? Aside from this point of view, he wrote a GA engine in C++ independent of go. Knowledge in this program is represented entirely by triples reminiscent of Prolog predicates such as $IfPointAt(x, y, z)$. These statements can be nested. Detailed descriptions were scant, but it appears that each variable (x, y, or z) is comprised of a board location, the color (white, black, or empty), and the action to take (move, pass, or resign). If the parameter x is satisfied, then y is checked, otherwise z is checked. Through this possibly layered traversal of the statements, moves are chosen. The program, “...was very poor at breeding individuals that could match. And when it did, the individual would often resign after but a few moves” [7].

In [9], the researchers used genetic programming and the game of go to create genetic algorithms that incorporate qualities of true human experts. One inclusion was to incorporate useful but infrequently used rules, and another was to model ecological systems. The ecological models dictate that many species coexist. Their ideas revolve around the fact that species live together in an environment, yet
they can be radically different. Rules, in their system, increase in number and eat virtual *food.* Rules whose activations decrease to zero, die, while rules whose activations become too high split into the original rule and a more specific rule. A training datum is considered food, which is eaten by a rule that matches it; the activation value of the rule then increases. These researchers used their genetic algorithm entirely to evolve rules based on patterns found on the board. The authors did not report the playing skill of their program, but they did present the rules that the program generated to go experts. These experts decided that 41.6% were good, 21.1% were average, and 37.3% were bad [9].

### 2.4.5 Other Techniques and Hybrids

In [17], the authors describe their *SANE* architecture that evolves neural networks to play go by using genetic algorithms. The program starts with no prior go knowledge at all. The process involves evolving individual neurons using crossover mutations and random point mutations. Each neuron is defined as a set of bits that describe connections and the connection weights. Each neuron has a fixed number of connections, but each connection can be attached to either the output or the input layer. Network blueprints are also evolved along with the individual neurons. Entire networks are evolved based on the final state of the game rather than assigning credit to individual moves, which the authors state is unreasonable; however, it could be argued that one can simply assume that game records between
two masters represent on average the best move at each point in the game. This may not actually be true, but it is a close enough approximation.

In [14], the researchers discuss the evolution of neural networks on a variant of the SANE architecture that evolves individual neurons, but evaluates the fitness of entire networks. In addition, blueprints (i.e., sets of neurons that work well together) are evolved. The neurons in question are only for the single hidden layer of the network. SANE has been shown to work well in continuous domains and games with hidden state information. The authors describe their EuSANE architecture:

"The core idea of EuA is to select every allele of the offspring separately, based on explicit analysis of the allele fitness distributions in the population. It furthermore contains a restriction operator that focuses the analysis on members of the population most relevant for determining the next allele. In every generation only one new individual is generated, implementing a steady-state replacement.” [14]
Chapter 3

Methodology

Our approach consists of a three layer summation network with each layer fully connected to its adjacent layers. Each connection is characterized by an integer weight, and each node sums arrays. These arrays each contain an element that corresponds to locations on the board (i.e., it is a one-to-one mapping). The cornerstone of our design is to evolve these weights using our genetic algorithm, thus each chromosome specifies a set of integer weights for the summation network. The initial inputs to the network are the outputs from the individual agents.

3.1 Design Overview

*Exodus*, as the program we wrote is called, provides the end user with the ability to run regressions, evolve a GA player using stored game training sets, and play a
human player with extensibility in mind to allow IGS\textsuperscript{1} and gomodem\textsuperscript{2} connectivity in the future.

Exodus was designed in a highly object-oriented fashion as will be described in this chapter. It consists of a moderator that allows two move generation classes (called interfaces) to play against each other. Through this abstract interface class, we have developed an ASCII text player that interfaces with a human user, a simple Perl/Tk\textsuperscript{3} interface that also interfaces with a human user, a GA player that will be described in detail below, and a GA trainer that is designed to play against the GA player in order to calculate the fitness of the GA player. The interface's simplicity allows for the potential future development of interfaces that can play go over the Internet or interfaces that communicate over a serial line, i.e., as used in competitions.

### 3.2 Stone, Board, and Game Classes

The stone class represents a single location on the goban, which was implemented with speed as the primary concern. It uses bit operations to test various traits of a location such as if the location has a black stone or a white stone. This feature eliminates many potential modulo operations that would be necessary otherwise.

\textsuperscript{1}Internet Go Server
\textsuperscript{2}A protocol for serial communication between two computers, each playing go.
\textsuperscript{3}Tk is a graphical package, originally implemented for use by the language TCL, that provides basic graphic capabilities such as window creation, buttons, frames, text boxes, etc.
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>4</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
</tr>
<tr>
<td>3</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
</tr>
<tr>
<td>2</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
</tr>
<tr>
<td>1</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 3.1: Board Locations For a $9 \times 9$ Board.

It also has functions that test if the location is on an edge. This class can be found in section A.2.29.

The next layer of abstraction encapsulates the concept of a board, which is simply a one-dimensional array of stones. A one-dimensional array was chosen in an attempt to speed up board manipulations by reducing the need for pointer arithmetic that is required in multiarray offset calculations. Figure 3.1 shows how a $9 \times 9$ board is represented and shows how the two-dimensional structure is mapped onto a one-dimensional array. Stones on the edges are marked as such to allow tests such as $\text{stone}[9].\text{left()}$ or $\text{stone}[32].\text{nottop()}$.

On top of the board abstraction there is a game class which stores a linked list of boards and keeps track of which side’s turn it is. The game class enforces certain optional rules such as whether or not to allow suicide. It also provides functions such as $\text{play\_move}$ and $\text{legal}$, both of which have obvious uses. The source code for this class is found in section A.2.9.
3.3 User Interfaces

The Exodus program contains two distinct user interfaces, each of which inherits from a superclass Interface found in section A.2.18: a text interface and a graphical interface. The text interface displays the goban using ASCII characters with a # representing black and an o representing white. Figure 3.2 shows an ASCII board for a 9 × 9 game. This interface is useful when visual appeal is not an issue (i.e. testing code, not directly related to the output of the board).

Another user interface is a GUI interface that uses an external Perl/Tk program to display the goban. Figure 3.3 shows a screen shot. This interface is important for playing games against the program (a graphical board is easier to interact with). This interface was also useful while developing the board and game classes as it made debugging easier. A graphical user interface allowed for a quicker way to play with the program in an attempt to find problems or bugs.
3.4 Genetic Algorithm

The code for performing genetic algorithms was originally taken from David E. Goldberg [6]. The code in this text was converted to C++ and syntactically modified to better suit an object-oriented approach like ours. The code can be found in section A.2.8.

The GA code was made as generic as possible and supplies a member function called \texttt{set\_codex} which allows the reception of a pointer to a class of type \texttt{PreCodex}, which is a superclass of any class that wishes to supply a fitness function.

The program keeps statistics on the performance of the GA and tracts the minimum, maximum, sum, average, variance, and standard deviation of the fitnesses from each generation. In addition to these, F-test and T-test values are computed for each generation, comparing the statistics to the initial generation. The F-test value calculates whether two distributions have significantly different variances. The T-test (Student’s T-test) measures whether two distributions have
significantly different means. Two versions of the T-test were used. One version is used for distributions with statistically different variances and the other for distributions with statistically identical variances. These numbers allow us to better determine the significance of the data.

3.5 Moderator

The moderator class, found in section A.2.19, essentially loads two move generators which can manifest themselves as anything from a user interface to a random move generator or a genetic algorithm player. There also exists a genetic algorithm trainer which is described in section 3.7.

The moderator class is multi-threaded, allowing a thread for each move generator. This design allows both sides to have processing time throughout the entire game—not just during a side’s turn. Another feature of this class is that it was implemented as a template, which allows the two players to be specified when one instantiates a moderator class. Message passing is used to allow communication between the moderator and the two move generators.

3.5.1 Probability Board

The probability board class, found in section A.2.27, is a conceptually simple abstraction that stores an array of values which correspond with the locations on
the goban. The semantics are such that the values at each element represent how
highly that location is valued as a possible next move. Each agent constructs one
of these, and to facilitate the aforementioned use of this class, a spin function was
implemented (to choose a move probabilistically), and a normalize function was
implemented to facilitate the addition of two or more of the boards together, each
from a potentially difference source. Also, the ability to multiply each board by
a scalar value was added (whose use will become apparent in section 3.6).

3.6 Agent Network Architecture

The GA player uses a thread pool to run multiple agents that each generate a
numerical value for each board location. These arrays are multiplied by GA-
evolved weights, added together, normalized, and fed through a second layer of
summation nodes. The resulting array is then normalized. The highest value in
this result array then becomes the move played. Figure 1.1 shows a graphical
representation of this process which is described algorithmically as follows:

1. Each of \( N \) agents computes a value for each location on the goban.

   This probability board is a vector and shall be denoted as \( \beta_n \), where
   \( n \) is the agent number.
2. Each of the $K$ second level nodes $\gamma_k$ sum all $\beta_n$ values multiplied by a scalar value $w_{k,n}$. Thus,

$$\gamma_k = \sum_{n=0}^{N} w_{k,n} \cdot \beta_n$$

3. These $\gamma_k$ are vectors that are then normalized so that the values add up to 1 in each vector unless all of the values in a vector are zero, in which case they are left that way.

4. These normalized $\gamma$-vectors are then multiplied by a second set of weights and added together:

$$\epsilon = \sum_{k=0}^{K} w_k \cdot \gamma_k$$

5. This final vector, $\epsilon$, is normalized and represents a distribution of which move to play. To make training the GA simpler, we simply choose the first highest value rather than choosing the move to play probabilistically, though either way is possible.

This approach theoretically allows for a large number of agents, limited primarily by the size of the thread pool and the number of processors available to the program. A major goal of this project was to create a design that was scalable and could benefit from a highly parallel machine. Though scalability was not tested,
the possibility of adding more agents could easily be realized. Figuring out what each agent would do could become a significant bottleneck, though.

3.7 Genetic Algorithm Trainer

This move generator was designed to play against the genetic algorithm player. It reads a sequence of moves from a data file (which were derived from recorded games of professionals in the public domain). It sets up the board and then allows the genetic algorithm player to play. After the GA player has played, the trainer resets the game state to whatever the professional actually played in the game record. The colors on the board are flipped, and the GA player is allowed to play again. The colors are flipped to allow the GA player, which plays a single color, to gain benefit from the entire game record and not just from the plays of a single color. After all, the recorded games are from two professionals playing, and each player can be assumed to be playing well. The usefulness of this GA trainer player, which is shown in section A.2.10, will become apparent in section 3.8.2, which describes the fitness function in detail.
3.8 Genetic Algorithm Player

3.8.1 GA Player Details

This code, found in section A.2.11, loads the parameters for the weights in the summation network (described in section 3.6) and computes the move to play by running the agents, filtering their values through the summation network, and then picking either the first highest value or normalizing and then choosing the move probabilistically. This class also inherits from PreCodex which implies that it provides a fitness function (that the GA uses).

3.8.2 Fitness Function

The fitness is calculated by setting up a goban as dictated by stored games from the Internet. If the GA player chooses the correct next move, an accumulator is incremented. The fitness is then simply the percentage of moves correctly played. Many other GA go programs calculate fitness by using some variation of attempting to guess how the current board configuration relates to the final division of points at the end of the game. Our approach sidesteps this difficulty which relates closely with the difficulty of simply scoring a finished game. The fitness function code is shown in section A.2.11.
3.9 Agents

We have designed and implemented six different agents that each choose moves in significantly different ways. Currently, there is a random agent that plays random legal moves, a follower agent that tries to play close to the enemy, an opener agent that plays in the locations usually played in at the beginning of a game, a capture agent that attempts to kill groups by reducing other groups’ liberties, an agent that attempts to create a strong configuration known as a tiger’s mouth, and an extension agent that favors moves close to friendly stones.

3.9.1 Random Agent

An agent that plays random legal moves was developed to allow the testing of code that directly uses the agents and to allow the testing of the code that lets the agents interact. Additionally, the random agent was used as a baseline with which the other agents can compare themselves. For example, the standard by which the success of the genetic algorithms is judged is the set of five random agents. Section A.2.28 contains the code for the random agent.

3.9.2 Follower Agent

The follower agent, found in section A.2.7, values playing on locations adjacent to enemy stones. As is often found in games of go, many good moves are often near
enemy stones, i.e., attacking them. Playing close to enemy stones not only attacks them, but also attempts to push the enemy group in the opposite direction.

3.9.3 Opener Agent

This agent, found in section A.2.24, suggests moves around the perimeter of the board near the third or fourth row. The values decrease the further the game progresses. The reasoning behind this type of agent is that at the very beginning of most games, stones are played near the edges and sides because this is where it is easiest to make territory. In a corner, one only has to worry about attacks from two directions. On a side, attacks are only possible from three directions, while in the middle, attacks can be made from all directions. These considerations are what justifies having an opener agent.

3.9.4 Capturer Agent

This agent attempts to capture enemy stones by filling in their last liberties. It has no knowledge of living or dead groups, thus it plays simply by calculating which groups have one or two liberties left and then plays in those liberties. Located in section A.2.14 and called GroupStatsAgent, this agent does not take into account moves that would reduce a friendly group’s liberty count down to one. What this means is that this agent would be perfectly content to play a move that reduced an enemy’s group to a single liberty while that very same move would allow the
enemy to capture a friendly group on the next turn.

### 3.9.5 Tiger’s Mouth Agent

The tiger’s mouth agent (section A.2.34) attempts to create a powerful configuration called a *tiger’s mouth*. Figure 3.4 shows what a tiger’s mouth looks like. This formation retains the same name regardless of all symmetries. This configuration is considered strong because it allows three stones to NOT be connected while retaining the ability to become connected by playing in the center location. Another strength of this configuration is that if an enemy stone tries to keep these stones from connecting, that enemy stone can be captured on the next turn if it is not part of another group. The versatility of this formation provides justification for the inclusion of this agent.

### 3.9.6 Extender Agent

This agent plays many of the common extensions. Each type of extension has a different weight (or value) which is derived from the GA chromosome. It is also
Extensions starting at the upper-left and continuing clockwise. 1. Extension • 2. One-point extension • 3. Two-point extension • 4. Three-point extension • 5. Shoulder extension • 6. Knight’s move • 7. Large Knight’s move

Figure 3.5: Extensions.

the only agent that uses alleles from the chromosomes to set these internal values (in this case, the alleles specify the relative value of each of the extension types).

These types of extensions are shown in Figure 3.5.

3.10 Regressions

We have written extensive amounts of code to test the validity and accuracy of much of the code. Nearly every function in every class has some sort of regression. The regressions for each source file are located within that file. One cannot guarantee program correctness by running the regressions, but the regressions do serve to instill a greater feeling of security that incremental changes to the code do not break anything coded previously.
3.11 Unimplemented Features

Over the course of designing and implementing this program many ideas that were designed into the original program were not actually implemented, though integration of these parts would be relatively simple given enough time. These unimplemented features include:

- The ability to have the program play against real people on IGS (Internet Go Server).

- A blackboard architecture for agent communication.

- The ability to play another program via a protocol called the *gomodem protocol*.

- The ability to track time during games.

- The modification of agents to allow the use of time when the opponent is thinking to do useful work.

- The incorporation of interagent communication.

- The ability to score finished games.
Chapter 4

Experiments and Results

The experiment descriptions and results that follow attempt to show the successful evolution of summation network weights for a multiagent approach to playing go. The key point is that we want to illustrate that though each individual agent may play poorly, the agents playing together actually play better. We wish to further show that random search (using a GA), finds weights for the summation network that improve over multiple generations.

The fitness function for our experiments uses recorded games; we used recorded $9 \times 9$ games between professionals from the public domain that occurred between 1995 and 2000 on an international go server [1].
4.1 Individual Agent Experiments

The genetic algorithm was run for eight generations with the program configured to use only a single agent. These agents were the random-move-generating agent, the extension agent, the follower agent, the capture agent, the opening move agent, and the tiger's eye agent. In each case, the populations contained ten individuals. Such a small population and small number of generations were used because of the large amount of time it took to run the GAs even with this configuration. These runs took around three days on a dual-processor, 1.2GHz machine. Additionally, most of the single-agent configurations do not benefit from the GA on their own, making the time required for a larger population or a larger number of generations of questionable use. The crossover percentage was 40% with a mutation probability of 0.0333. Additionally, the F-multiplier was two. The random agent was used as a baseline. After evolution by the genetic algorithm, the best individual in the final population was used to play against a testing data-set representing game records that were different than those used to train during the run of the genetic algorithm.

4.1.1 Opener Agent

Table 4.1 shows the results of the genetic algorithm run using only the opener agent. These data (fitness values) are also shown in Figure 4.1. Since the genetic
<table>
<thead>
<tr>
<th>Generation</th>
<th>Max</th>
<th>Min</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sumfitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0</td>
<td>0.114</td>
</tr>
<tr>
<td>1</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0</td>
<td>0.114</td>
</tr>
<tr>
<td>2</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0</td>
<td>0.114</td>
</tr>
<tr>
<td>3</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0</td>
<td>0.114</td>
</tr>
<tr>
<td>4</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0</td>
<td>0.114</td>
</tr>
<tr>
<td>5</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0</td>
<td>0.114</td>
</tr>
<tr>
<td>6</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0</td>
<td>0.114</td>
</tr>
<tr>
<td>7</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0</td>
<td>0.114</td>
</tr>
<tr>
<td>8</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0.0114</td>
<td>0</td>
<td>0.114</td>
</tr>
</tbody>
</table>

Table 4.1: Opener Agent Data.

Figure 4.1: GA Data Plot With Opener Agent.
<table>
<thead>
<tr>
<th>Generation</th>
<th>Max</th>
<th>Min</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sumfitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>4.91e-10</td>
<td>0.0622</td>
</tr>
<tr>
<td>1</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>7.39e-06</td>
<td>0.0622</td>
</tr>
<tr>
<td>2</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>7.39e-06</td>
<td>0.0622</td>
</tr>
<tr>
<td>3</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.000906</td>
<td>0.0622</td>
</tr>
<tr>
<td>4</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.000906</td>
<td>0.0622</td>
</tr>
<tr>
<td>5</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.01</td>
<td>0.0622</td>
</tr>
<tr>
<td>6</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.01</td>
<td>0.0622</td>
</tr>
<tr>
<td>7</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.0334</td>
<td>0.0622</td>
</tr>
<tr>
<td>8</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.0334</td>
<td>0.0622</td>
</tr>
</tbody>
</table>

Table 4.2: Randomly Playing Agent Data.

algorithm-evolved weights are not effective if a single agent is used, one would expect that the fitness values would not change, which is exactly what appears to have happened here. The best chromosome (which incidentally is arbitrary) of the last generation chose 1.14% of the training moves correctly and 1.55% of the testing moves correctly. Considering that this agent was designed to play opening moves, this is not a surprise that it fared so poorly. The F-test and T-test (described in section 3.4) have little use here in a straight-forward example such as this. Each distribution of each generation is clearly identical to each other, so nothing was gained from the genetic algorithm.

4.1.2 Single Random Agent

The single randomly playing agent did not fare well as shown in the data (Table 4.2 and Figure 4.2). Since the random agents always at least pick legal moves, the number of possible moves near the end of any game becomes smaller, which increases the likelihood that a random guess would be correct. These considerations
Figure 4.2: GA Data Plot With Randomly Playing Agent.

aside, one should note that the random agents are not actually randomly choosing locations to play, but rather assigning the same value for every legal position to move to. The final resulting probability board (see section 3.5.1 above) contains an array of values (which in this case would all be the same). The program can be configured to either pick the first highest or to pick one probabilistically. For this experiment (and all of the others as well), the first, more deterministic path was taken. The result of this is that the first legal move is always chosen, which ends up being correct a static number of times. We hypothesize that the 0.622% of the moves that the best chromosome of the eighth generation got correct was a result of this effect. If one keeps choosing the same legal location as one’s guess, it
<table>
<thead>
<tr>
<th>Generation</th>
<th>Max</th>
<th>Min</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sumfitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0415</td>
<td>0.0301</td>
<td>0.0365</td>
<td>0.00428</td>
<td>0.365</td>
</tr>
<tr>
<td>1</td>
<td>0.0689</td>
<td>0</td>
<td>0.0354</td>
<td>0.0319</td>
<td>0.354</td>
</tr>
<tr>
<td>2</td>
<td>0.0777</td>
<td>0.0037</td>
<td>0.0389</td>
<td>0.0347</td>
<td>0.389</td>
</tr>
<tr>
<td>3</td>
<td>0.0788</td>
<td>0.0056</td>
<td>0.0394</td>
<td>0.0641</td>
<td>0.394</td>
</tr>
<tr>
<td>4</td>
<td>0.0753</td>
<td>2.76e-10</td>
<td>0.0409</td>
<td>0.0667</td>
<td>0.409</td>
</tr>
<tr>
<td>5</td>
<td>0.0821</td>
<td>8.1e-09</td>
<td>0.0436</td>
<td>0.0899</td>
<td>0.436</td>
</tr>
<tr>
<td>6</td>
<td>0.0797</td>
<td>5.57e-09</td>
<td>0.0458</td>
<td>0.0893</td>
<td>0.458</td>
</tr>
<tr>
<td>7</td>
<td>0.0669</td>
<td>0</td>
<td>0.0468</td>
<td>0.102</td>
<td>0.468</td>
</tr>
<tr>
<td>8</td>
<td>0.0861</td>
<td>0</td>
<td>0.0449</td>
<td>0.105</td>
<td>0.449</td>
</tr>
</tbody>
</table>

Table 4.3: Extension Agent Data.

Eventually becomes correct. An interesting feature of these data are that the testing data yielded 0.62% correct moves which is not surprising given the previous explanation.

4.1.3 Extension Agent

The extension agent is more interesting, in that there are internal parameters to the agent that derive their values from the evolved chromosomes. Table 4.3 shows the results of the genetic algorithm run using only the extension agent, and the data are also shown graphically in Figure 4.3. The genetic algorithm-evolved weights still do not matter for this single agent, but this agent has internal parameters that could benefit from evolution. As one would hope, as the generations progressed, the mean fitness and the maximum fitness rose. Additionally, the minimum fitness had a net decrease of 0.0301 by the end of the eighth generation. To back up these observations, the F-test predicts that the first and the final generations have insignificantly different variances which allowed us to use
Figure 4.3: GA Data Plot With Extension Agent.

the T-test to predict with a probability of 99.9642% that the improvement is real and not a result of chance. The decrease of the minimum is not a concern due to the increase of the maximum and the mean. The best chromosome of the last generation chose 5.18% of the training moves correctly and 3.88% of the testing moves correctly.

4.1.4 Capturer Agent

The data for this agent (shown in Table 4.4 and graphically in Figure 4.4) shows the same lack of improvement as other individual agents because this agent has no internal parameters that might benefit from evolution. An interesting feature,
<table>
<thead>
<tr>
<th>Generation</th>
<th>Max</th>
<th>Min</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sumfitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0777</td>
<td>7.85e-09</td>
<td>0.777</td>
</tr>
<tr>
<td>1</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0777</td>
<td>2.95e-05</td>
<td>0.777</td>
</tr>
<tr>
<td>2</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0777</td>
<td>2.95e-05</td>
<td>0.777</td>
</tr>
<tr>
<td>3</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.00181</td>
<td>0.777</td>
</tr>
<tr>
<td>4</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.00181</td>
<td>0.777</td>
</tr>
<tr>
<td>5</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0142</td>
<td>0.777</td>
</tr>
<tr>
<td>6</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0142</td>
<td>0.777</td>
</tr>
<tr>
<td>7</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0397</td>
<td>0.777</td>
</tr>
<tr>
<td>8</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0777</td>
<td>0.0397</td>
<td>0.777</td>
</tr>
</tbody>
</table>

Table 4.4: Capturer Agent Data.

Figure 4.4: GA Data Plot With Capturer Agent.


<table>
<thead>
<tr>
<th>Generation</th>
<th>Max</th>
<th>Min</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sumfitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.0415</td>
<td>3.93e-09</td>
<td>0.415</td>
</tr>
<tr>
<td>1</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.0415</td>
<td>2.09e-05</td>
<td>0.415</td>
</tr>
<tr>
<td>2</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.0415</td>
<td>2.09e-05</td>
<td>0.415</td>
</tr>
<tr>
<td>3</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.00152</td>
<td>0.415</td>
</tr>
<tr>
<td>4</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.00152</td>
<td>0.415</td>
</tr>
<tr>
<td>5</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.013</td>
<td>0.415</td>
</tr>
<tr>
<td>6</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.013</td>
<td>0.415</td>
</tr>
<tr>
<td>7</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.038</td>
<td>0.415</td>
</tr>
<tr>
<td>8</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.0415</td>
<td>0.038</td>
<td>0.415</td>
</tr>
</tbody>
</table>

Table 4.5: Follower Agent Data.

though, is the change in standard deviation, which probably resulted from numerical round-off errors that resulted from the summation network calculations. The best chromosome from the last generation got 7.77% of the moves correct and 4.03% of the testing moves correct.

### 4.1.5 Follower Agent

The follower agent followed in the footsteps of the other single agents with its lack of improvement. No internal parameters for the genetic algorithm were used. The best chromosome of the last generation got 4.15% of the training moves correct, while it got 3.26% of the testing moves correct. The data can be found in Table 4.5 and in Figure 4.5.
4.1.6 Tiger’s Mouth Agent

The last single-agent configuration’s data are shown in Table 4.6 and in Figure 4.6. The tiger’s eye agent showed no improvement due to a lack of internal genetic algorithm parameters. The best chromosome from the final generation got 2.38% of the moves correct, while it got 2.17% of the moves correct on the testing data.

4.2 Multiagent Experiments

The multiagent experiments closely mirrored the individual agent experiments with the exception that in these cases the program was run with all of the agents
<table>
<thead>
<tr>
<th>Generation</th>
<th>Max</th>
<th>Min</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sumfitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.238</td>
</tr>
<tr>
<td>1</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.238</td>
</tr>
<tr>
<td>2</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.238</td>
</tr>
<tr>
<td>3</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.238</td>
</tr>
<tr>
<td>4</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.238</td>
</tr>
<tr>
<td>5</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.238</td>
</tr>
<tr>
<td>6</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.238</td>
</tr>
<tr>
<td>7</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.238</td>
</tr>
<tr>
<td>8</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.0238</td>
<td>0.238</td>
</tr>
</tbody>
</table>

Table 4.6: Tiger’s Mouth Agent Data.

![Fitness By Generation (Tiger’s Mouth Agent)](image)

Figure 4.6: GA Data Plot With Tiger’s Mouth Agent.
<table>
<thead>
<tr>
<th>Generation</th>
<th>Max</th>
<th>Min</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sumfitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>4.91e-10</td>
<td>0.0622</td>
</tr>
<tr>
<td>1</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>7.39e-06</td>
<td>0.0622</td>
</tr>
<tr>
<td>2</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>7.39e-06</td>
<td>0.0622</td>
</tr>
<tr>
<td>3</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.000906</td>
<td>0.0622</td>
</tr>
<tr>
<td>4</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.000906</td>
<td>0.0622</td>
</tr>
<tr>
<td>5</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.01</td>
<td>0.0622</td>
</tr>
<tr>
<td>6</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.01</td>
<td>0.0622</td>
</tr>
<tr>
<td>7</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.0334</td>
<td>0.0622</td>
</tr>
<tr>
<td>8</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.00622</td>
<td>0.0334</td>
<td>0.0622</td>
</tr>
</tbody>
</table>

Table 4.7: Five Random Agent Data.

at once excluding the random-move-generating agent. A separate run that used five random-move-generating agents was used as a baseline.

4.2.1 Five Random Agents

Not surprisingly, the genetic algorithm configured with five identical random legal move generating agents performed rather poorly. The results were nearly identical to those of the single random agent above. The results are shown in Table 4.7 and in Figure 4.7.

4.2.2 Multiagent Configuration

Table 4.8 and Figure 4.8 show the results of evolving the genetic algorithm using five agents: Opener, Extension, GroupStats, Follower, and TigersMouth agents. Three hidden-layer nodes were used, and each generation had 10 individuals. Initially, the maximum fitness was 0.0881 and the mean fitness was 0.0537. By
Figure 4.7: GA Data Plot With Five Random Agents.

<table>
<thead>
<tr>
<th>Generation</th>
<th>Max</th>
<th>Min</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sumfitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0881</td>
<td>0.0435</td>
<td>0.0537</td>
<td>0.0138</td>
<td>0.537</td>
</tr>
<tr>
<td>1</td>
<td>0.108</td>
<td>0.0245</td>
<td>0.0539</td>
<td>0.0454</td>
<td>0.539</td>
</tr>
<tr>
<td>2</td>
<td>0.121</td>
<td>0.0106</td>
<td>0.0604</td>
<td>0.0575</td>
<td>0.604</td>
</tr>
<tr>
<td>3</td>
<td>0.119</td>
<td>3.49e-10</td>
<td>0.0694</td>
<td>0.0865</td>
<td>0.694</td>
</tr>
<tr>
<td>4</td>
<td>0.105</td>
<td>3.15e-09</td>
<td>0.0812</td>
<td>0.0867</td>
<td>0.812</td>
</tr>
<tr>
<td>5</td>
<td>0.109</td>
<td>2.82e-09</td>
<td>0.0848</td>
<td>0.103</td>
<td>0.848</td>
</tr>
<tr>
<td>6</td>
<td>0.108</td>
<td>2.15e-09</td>
<td>0.0877</td>
<td>0.103</td>
<td>0.877</td>
</tr>
<tr>
<td>7</td>
<td>0.134</td>
<td>0</td>
<td>0.087</td>
<td>0.113</td>
<td>0.87</td>
</tr>
<tr>
<td>8</td>
<td>0.14</td>
<td>0</td>
<td>0.0798</td>
<td>0.118</td>
<td>0.798</td>
</tr>
</tbody>
</table>

Table 4.8: All Five Agents Data.
the final generation, the maximum fitness had risen to 0.14 and the mean fitness had risen to 0.0798. The question then becomes one of deciding if this difference should be attributed to chance or to legitimate improvement. Using the F-test, the difference in the variances was not significant. The T-test value of the final generation was $-4.23$ which implied a probability of $0.000504$ that these results were from chance and not from a different population as the initial population, i.e., the confidence interval was 99.95% that the difference in the means was significant. The best chromosome from the final generation got 10.2% of the moves correct while it got 5.558% of the testing set correct.

The agents were loaded in the following order: OpenerAgent, TigersMouthA-
gent, GroupStatsAgent (capturer), FollowerAgent, and ExtenderAgent. The final best network configuration had weights from the agents to the second layer of the network as...

\[
Weights = \begin{pmatrix}
12 & 15 & 13 \\
11 & 8 & 15 \\
10 & 15 & 14 \\
10 & 3 & 1 \\
5 & 0 & 13
\end{pmatrix}
\]

where each row corresponds to an agent and each column corresponds to a node in the next layer. The weights from this next layer to the output node is...

\[
\begin{pmatrix}
11 \\
15 \\
3
\end{pmatrix}
\]

4.2.3 Multiagent Configuration, Large Population

Table 4.9 and Figure 4.9 show the results after seven generations of the multiagent configuration with a population size of 100. All parameters were the same as the smaller multiagent configuration except for the population size. These data support the results from the smaller multiagent experiment.
<table>
<thead>
<tr>
<th>Generation</th>
<th>Max</th>
<th>Min</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sumfitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0995</td>
<td>0.0321</td>
<td>0.0549</td>
<td>0.0142</td>
<td>5.49</td>
</tr>
<tr>
<td>1</td>
<td>0.115</td>
<td>0.0203</td>
<td>0.0573</td>
<td>0.0273</td>
<td>5.73</td>
</tr>
<tr>
<td>2</td>
<td>0.126</td>
<td>0.0198</td>
<td>0.063</td>
<td>0.0296</td>
<td>6.3</td>
</tr>
<tr>
<td>3</td>
<td>0.134</td>
<td>4.88e-10</td>
<td>0.069</td>
<td>0.0352</td>
<td>6.9</td>
</tr>
<tr>
<td>4</td>
<td>0.143</td>
<td>0</td>
<td>0.073</td>
<td>0.0458</td>
<td>7.3</td>
</tr>
<tr>
<td>5</td>
<td>0.137</td>
<td>3.93e-09</td>
<td>0.0762</td>
<td>0.043</td>
<td>7.62</td>
</tr>
<tr>
<td>6</td>
<td>0.135</td>
<td>4.03e-09</td>
<td>0.0782</td>
<td>0.0371</td>
<td>7.82</td>
</tr>
<tr>
<td>7</td>
<td>0.14</td>
<td>0</td>
<td>0.0786</td>
<td>0.0409</td>
<td>7.86</td>
</tr>
</tbody>
</table>

Table 4.9: All Five Agents Data (Large Population).

![Fitness By Generation (All Agents) Population of 100](image)

Figure 4.9: GA Data Plot With All Agents (Large Population).
Figure 4.10: Agent Comparison.

4.3 Summary

Figure 4.10 shows a comparison of the best fitnesses achieved by all of the agent configurations. The randomly playing agent played the poorest, and the two configurations that could benefit from the genetic algorithm (extension agent and all of the agents combined) actually did. The testing data shows some variability, and in some cases an agent that performed better on the training data did worse on the testing data (compared to the other agents). Mostly, though, there appears to be a benefit of using the genetic algorithm to evolve go players.
Chapter 5

Conclusion

5.1 Contributions

We have found that a multiagent approach using a summation network does indeed yield a viable go player. Furthermore, improvement was gained over the course of multiple generations. In addition to these results, a unique approach to playing go was illustrated. As far as we know, nobody has written a program that plays go using probabilistic methods incorporating multiple agents whose interactions (the summation network) have been evolved or learned in some way. This approach shows that it may be possible to break down certain large intractable problems and use genetic algorithms to combine multiple sources of information without knowing exactly how the information interacts to form a solution. This architecture exemplifies the possibility of trading the ability to fine-tune the be-
behavior of a system with the ability to scale the system indefinitely, limited mainly by the number of processing nodes.

5.2 Limitations

This approach to playing go has many potential limitations. Foremost, it relies heavily on the ability of the programmer to create agents that contribute to the skill of the program. As we are not go experts, creating good agents was a challenge.

Another limitation is that genetic algorithms take long periods of time to run. Larger training sets, larger testing sets, larger populations, more intricate summation networks, and more generations could all help improve the program, but unfortunately all of these would contribute to a significantly slower program.

Though scalability was an important goal, the realization of a massively parallel multiagent go program must be quelled by the prohibitive cost and the scarcity of machines with dozens of processors. The future may not hold a limitation such as this, but currently it is a very real limitation to increasing the number of agents extensively.

Yet another restrictive aspect of this work was the use of only $9 \times 9$ boards for all experiments. This enabled us to complete the research in a reasonable amount of time. Trying to use $19 \times 19$ boards would have likely taken too long.
The program does not play go very well, though the GA does allow the program to improve which was one of the goals of this project. Many authors often compare their programs to standard programs such as one called Wally, but our program does not yet have an interface that would allow automatic matches. Though this is a limitation, not playing well may not matter as much as showing that our program improves. Clearly, the program that we developed does not play go better than its peers.

5.3 Future Work

The future of this approach remains unclear, but additional research to test larger networks utilizing a larger number of agents could yield positive results. Scalability was a secondary goal—a goal that seems within reach given the prominence and proliferation of multiprocessor machines. Perhaps in the decades to come someone will create a go program that can play at the level of the masters. This is a goal that many await patiently.
Bibliography


Appendix A: Doxygen Code Reference

The code index was generated automatically using a tool called Doxygen that parses the source files’ comments...

A.1 Cross-references

A.1.1 Exodus Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Agent ................................................................. 73
   ExtenderAgent ............................................... 89
   FollowerAgent ............................................... 92
   GroupStatsAgent ............................................. 136
   OpenerAgent ................................................. 151
   RandomAgent .................................................. 159
   TigersMouthAgent .......................................... 173
Blackboard .......................................................... 77
Board ............................................................... 78
Ga ................................................................. 96
Game ................................................................. 112
global_data_t ..................................................... 134
Individual ........................................................... 142
Moderator ........................................................... 146
move_t .............................................................. 148
msg_t ............................................................... 149
Population .......................................................... 153
PreCodex ............................................................ 154
   GenAlgoGenerator ............................................ 126
testCodex .......................................................... 169
ProbBoard ................................. 155
Stone ...................................... 160
Subthread .................................. 165
AgentShell .................................. 76
Interface ................................... 144
  DummyGenerator .......................... 88
  GaTrainerInterface ..................... 123
  GenAlgoGenerator ....................... 126
  GoModemInterface ....................... 135
  GUIInterface ............................ 138
  IGS_Interface ........................... 141
  NeuralNetGenerator ..................... 150
  NNGS_Interface .......................... 150
  TextInterface ........................... 170
Subthread_test ......................... 168

A.1.2 Exodus Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

Agent (Defines the basic structure of an agent) ...................... 73
AgentShell (Represents a single thread in a thread pool) .......... 76
Blackboard (This class contains globally relavent information) .... 77
Board (Defines a goban abstraction) ................................. 78
DummyGenerator (A dummy move generator that generates random
    legal moves) ................................ 88
ExtenderAgent (Suggests moves that extend from friendly stones) .. 89
FollowerAgent (Suggests moves near opponent’s last move) ......... 92
Ga (Defines a Genetic Algorithm) ................................ 96
Game (A class that defines a series of boards) ..................... 112
GaTrainerInterface (Used to train a GA to work correctly) ....... 123
GenAlgoGenerator (A genetic algorithm move generator) ......... 126
global_data_t (Global data structure) .............................. 134
GoModemInterface (Go modem interface) ............................ 135
GroupStatsAgent (Agent (p. 73) to calculate group information) .. 136
GUIInterface (Graphical User Interface (p. 144)) ................. 138
IGS_Interface (Internet Go Server (IGS) Interface (p. 144)) ...... 141
Individual (An individual in a population of a GA) ............... 142
Interface (The interface between a move generator (outside) and the
    inside of the program) ................................ 144
Moderator (Encapsulates two interfaces and has them play together) 146
move_t (A single move on the goban) ............................... 148
msg.t (A message to or from a thread) .......................... 149
NeuralNetGenerator (A Neural Network move generator) .......... 150
NNGS_Interface (No Name Go Server Interface (p.144)) .......... 150
OpenerAgent (Suggests good opening moves) .................. 151
Population (A single population within a GA) ................. 153
PreCodex (Allows other classes to profile a fitness function) .... 154
ProbBoard (Agent (p.73)’s probability output board) .......... 155
RandomAgent (Suggests random legal moves) ................ 159
Stone (Defines a point (stone) on the board) .................. 160
Subthread (Defines a sub-thread) .................................. 165
Subthread_test (For debugging) .................................. 168
testCodex (A testing fitness function provider) ................ 169
TextInterface (Text Interface (p.144)) ....................... 170
TigersMouthAgent (Tries to make tiger’s mouths) ............. 173

### A.1.3 Exodus File List

Here is a list of all documented files with brief descriptions:

- **agent.cpp** (Implementation of Agent (p.73) and AgentShell (p.76) classes) ........................................ 177
- **agent.h** (Header file for Agent (p.73) related classes) ........ 178
- **bdemo.cpp** (Prints a demo board for numerical reference) .. 179
- **blackboard.cpp** (Implementation of Blackboard (p.77) class) 181
- **blackboard.h** (Header file for the Blackboard (p.77) class) 182
- **board.cpp** (Implementation for Board (p.78) class) ........ 182
- **board.h** (Header file for board class) ....................... 183
- **config.h** (System configuration definitions) .................. 184
- **dummygenerator.cpp** (Implementation of random move generator called DummyGenerator (p.88)) .................. 185
- **exodus.h** (Global constants declarations) .................. 186
- **extenderagent.cpp** (Implementation of an ExtenderAgent (p.89) that attempts to extend from friendly stones) ........ 189
- **followeragent.cpp** (Implementation of FollowerAgent (p.92) which plays moves close to opponent) .................. 190
- **ga.cpp** (Implementation for Ga (p.96) class) ............... 191
- **ga.h** (Header file for genetic algorithm related classes) .......... 192
- **gafunc.h** (Header file for GA testing and aux. functions) .... 193
- **game.cpp** (Implementation of the Game (p.112) class) ........ 194
- **game.h** (Header file for game class) ....................... 196
- **gatypes.h** (Header file for genetic algorithm types and defaults) ........ 197
- **genalgogenerator.cpp** (A genetic algorithm player using agents) .. 198
ginterface.cpp (Implementation of a GUI interface) .......................... 199
groupstatsagent.cpp (Provides an agent to calculate group information) 200
iinterface.cpp (Implementation of IGS interface class (IGS_Interface (p.141))) .................................................. 201
interface.cpp (Implementation for abstract Interface (p.144) classes) 202
interface.h (Header file for interfaces) ........................................ 202
main.cpp (Main, cmd-line, init-file functions) ................................ 203
moderator.t (Implementation and definition of Moderator (p.146) template) .................................................. 210
move.cpp (Implementation of the move_t (p.148) struct) ............ 211
move.h (Describes a Move struct) ........................................... 212
openeragent.cpp (Opening move agent) ....................................... 213
outputgen.h (Header file for GenAlgoGenerator (p.126), NeuralNetGenerator (p.150), and DummyGenerator (p.88) classes) ............. 214
probboard.cpp (The implementation for the probability board) .... 215
probboard.h (Probability matrix for an agent’s next move) .......... 216
randomagent.cpp (Random agent implementation) ....................... 217
stone.cpp (Implementation of the Stone (p.160) class) ............... 218
stone.h (Header file for Stone (p.160) class) ............................ 219
subthread.cpp (Implementation for abstract class Subthread (p.165)) 220
subthread.h (Defines virtual class for a running sub-thread) ........... 221
testcodex.cpp (Stub code for fitness function for GAs) ................... 223
tigersmouthagent.cpp (Implementation of tiger’s mouth class) ...... 224
tinterface.cpp (Implementation of text interface) ......................... 225
tools.cpp (Utilities) .................................................................. 226
tools.h (Defines useful utilities) ............................................. 233
traingainterface.cpp (Implementation of Trainer class for GAs) ..... 240

A.1.4 Exodus Related Pages

Here is a list of all related documentation pages:

Todo List .................................................................................. 241
Bug List .................................................................................... 241

A.2 Exodus Class Documentation

A.2.1 Agent Class Reference

Defines the basic structure of an agent.
#include <agent.h>

Inheritance diagram for Agent:

[Diagram: Agent -> ExtenderAgent, FollowerAgent, GroupStatsAgent, OpenerAgent, RandomAgent, TigersMouthAgent]

Public Methods

- **Agent ()**
  
  Constructor.

- virtual ~**Agent ()**
  
  Destructor.

- void **set_id** (int id)
  
  Sets agent ID.

- int **get_id** (void)
  
  Gets agent ID.

- void **set_bb_ptr** (Blackboard *bb_p)
  
  Set the blackboard pointer.

- void **set_pb_ptr** (ProbBoard *pb_p)
  
  Set the probboard pointer.

- virtual void **force** (void)=0
  
  Force agent to make a move.

- virtual void **update** (Game *)=0
  
  Updates the game for the agent. Refresh agent with a new game state.

- virtual bool **dowork** (void)=0
  
  Main agent work function.

- virtual void **notify** (void *)=0
  
  Tell the agent something.
• virtual unsigned int **query_bits_needed_from_GA** (void)=0
  
  *Ask the agent how many bits it needs from GA.*

• virtual void **send_bits** (chromosome_t chrom, int start)=0
  
  *Allows the agent to get the bits it needs.*

**Protected Attributes**

• int **ID**
  
  *Unique agent identification number.*

• Blackboard* **bb_ptr**
  
  *Blackboard (p. 77).*

• ProbBoard* **pb_ptr**
  
  *Probability board.*

• Game theGame
  
  *The game in question.*

A.2.1.1 **Detailed Description**

Defines the basic structure of an agent.

**Warning:**

This is an abstract class.

A.2.1.2 **Member Function Documentation**

void Agent::set_id (int id)  

Sets agent ID.

Sets the agent ID number which can be used to uniquely order all agents to prevent dealocks due to possible future agent dependences and the use of thread pools. Thread pools only allow a finite number of agents to run at a time, and if the first agents to run depend on another agent that isn’t running, then deadlock will occur. Agent IDs prevent this.
Warning:
IDs are currently not used, but in the event that they do become used, then it shall be expected that agents with higher IDs have dependences on only agents with lower IDs (if any at all).

0074 { this->ID = id; }

A.2.2 AgentShell Class Reference

Represents a single thread in a thread pool.

#include <agent.h>

Inheritance diagram for AgentShell::

```
Subthread
   |
   +---------------------
   |                     |
   +--- AgentShell ---+

Public Methods

• AgentShell ()
  
  Constructor.

• ~AgentShell ()
  
  Destructor.

Private Methods

• void processing (void)
  
  Represents a single thread of the thread pool’s main processing loop.

• void init (void)
  
  Performs any initialization that is needed.

76
Private Attributes

- Agent* theAgent

Agent (p. 73) identity to assume.

A.2.2.1 Detailed Description

Represents a single thread in a thread pool.

This class shall be able to "turn" into any of the agents via the proper messages.

A.2.2.2 Constructor & Destructor Documentation

AgentShell::AgentShell () Constructor.

Note:
Stores a copy of the game, not a pointer.

0080 {
0081  theAgent = NULL;
0082 }

A.2.2.3 Member Function Documentation

void AgentShell::init (void) [private] Performs any initialization that is needed.

Note:
Currently, this function is a stub.

0092 {

A.2.3 Blackboard Class Reference

This class contains globaly relavent information.

#include <blackboard.h>
Public Methods

- void set_game_ptr (Game *gamePtr)
  Tell the blackboard what game to look at.

- void update (void)
  Instructs Blackboard to update internal data-structures.

Static Private Attributes

- Game* g_ptr
  Points to the current game.

A.2.3.1 Detailed Description

This class contains globaly relavent information.

A blackboard is a paradigm whereby agents write information or data to a single localized location. This class provides an interface to this global scratchpad.

Warning:

Set gptr before doing anything. After this is set, all function calls are undefined until update is called at least once. Note that this class is a stub and currently provides no actual functionality.

A.2.3.2 Member Function Documentation

void Blackboard::update (void) Instructs Blackboard to update internal data-structures.

This function causes the blackboard class to regenerate all data-structures it stores locally.

0041 {
0042 }

A.2.4 Board Class Reference

Defines a goban abstraction.

#include <board.h>
Public Types

- enum flags_t { FUNKOWN, FSAFE, FEMPTY }
  
  Flags for board capture state-machine.

Public Methods

- Board ()
  
  Constructor I.

- Board (const Board &other)
  
  Copy Constructor.

- ~Board ()
  
  Destructor.

- bool valid_location (loc_t) const
  
  Tells if the location is playable.

- loc_t get_bsize (void) const
  
  Gets board size.

- Stone* get_goban (void) const
  
  Gets goban array.

- string raw_output (void)
  
  Output function for GUI.

- pair<usi_t, usi_t> play_move (loc_t, color_t)
  
  Plays a move on the board.

- void invert (void)
  
  Inverts the stones' colors.

- bool operator== (Board &)
  
  Equality operator.

- bool operator!= (const Board &)
Inequality operator.

• Board **operator** = (Board)
  Assignment operator.

• **color_t get_color_played** (void)
  Returns the color played for this board.

• **loc_t get_move_played** (void)
  Gets the move that was played for this board.

• **color_t operator[]** (loc_t location)
  Offset operator.

Static Public Attributes

• const **usi_t PASS** = 0xFFFF
  Offset into board array of PASS is used to represent a pass.

• **usi_t BSIZE**
  Default board size.

• **usi_t HANDICAP**
  Size of handicap.

• **list<usi_t> HANDICAP_PLACES**
  Force handicap locations.

Private Methods

• void **del_stone** (loc_t)
  Removes a stone from board.

• **usi_t del_group** (color_t)
  Removes groups with no liberties.
• void **setup** ()
  
  *Bulk of the constructors’ logic.*

• void **fill_safety** (vector< flags_t > &, int, color_t)
  
  *Recursive flood for finding safe stones.*

• void **put_stone** (loc_t, color_t)
  
  *Sets a board location to a specific color.*

**Private Attributes**

• loc_t **loc_played**
  
  *Location played to make this board.*

• Stone* **goban**
  
  *Actual board.*

• color_t **color_played**
  
  *Who’s turn is it?*

• loc_t **actual_size**
  
  *size of goban vector (BSIZE^2).*

**Static Private Attributes**

• bool **PRINTEXTRA**
  
  *Print extra right and bottom information on board.*

**Friends**

• ostream& **operator<<** (ostream &strm, Board &aBoard)
  
  *Output operator.*
A.2.4.1 Detailed Description

Defines a goban abstraction.

This class stores a board as an array of **Stone** (p.160) classes. It provides all functions that would be expected from a board abstraction.

A.2.4.2 Constructor & Destructor Documentation

**Board::~Board** () Destructor.

Deallocates array of **Stone** (p.160) classes

```cpp
0112 {
0113 //cerr << "pre " << flush;
0114 delete goban;
0115 //cerr << "post" << endl;
0116 }
```

A.2.4.3 Member Function Documentation

**usi_t Board::del_group** (**color_t color**) [private] Removes groups with no liberties.

**Parameters:**

- **color** The color of the groups to remove

```cpp
0238 {
0239  TAU_PROFILE("Board::del_group()");
0240 0241  color_t enemy_color = INV(color);
0242  vector<flags_t> scratch(actual_size, FUNKOWN);
0243 0244  // Mark enemy stones as safe
0245  for (int x=0; x<actual_size; ++x) {
0246   if (enemy_color==goban[x].getcolor()) scratch[x]=FSAFE;
0247  }
0248 0249 0250  // Do a flood fill on each empty spot, filling over friendly
0251  // stones but not passing enemy stones.
0252  for (int x=0; x<actual_size; ++x) {
0253   if (EMPTY==goban[x].getcolor()) {
0254    fill_safety(scratch, x, color);
0255   }
0256  }
0257
```
// Remove "unknown" stones as they are now dead.
usi_t count=0;
for (int x=0; x<actual_size; ++x) {
    if (FUNKNOWN==scratch[x]) {
        ++count;
        goban[x].setcolor(EMPTY);
    }
}
return count;

void Board::fill_safety (vector< flags_t > & scratch, int loc, color_t color) [private] Recursive flood for finding safe stones.

Does a flood fill of all safe pieces. Any stone that is safe automatically (logically) gives its safeness to all adjacent stones of the same color.

Parameters:
scratch A pass-by-reference scratch-pad used in this algorithm to figure what stones are safe/dead
loc Location to start at when looking for safety.
color The color to check for safety.

TAU_PROFILE("Board::fill_safety()", ",", TAU_DEFAULT);
if (scratch[loc] != FSAFE) {
    scratch[loc] = FSAFE;
    if (goban[loc].notleft() &&
        (color=goban[loc-1].getcolor()))
        fill_safety(scratch, loc-1, color);
    if (goban[loc].notright() &&
        (color=goban[loc+1].getcolor()))
        fill_safety(scratch, loc+1, color);
    if (goban[loc].nottop() &&
        (color=goban[loc-BSIZE].getcolor()))
        fill_safety(scratch, loc-BSIZE, color);
    if (goban[loc].notbottom() &&
        (color=goban[loc+BSIZE].getcolor()))
        fill_safety(scratch, loc+BSIZE, color);
}
**loc_t Board::get_move_played (void)**  Gets the move that was played for this board.

**Precondition:**

play_move was called already for this board.

0330 { return loc_played; }

**void Board::invert (void)**  Inverts the stones’ colors.

This function makes all white stones black and all black stones white.

0478  
0479 {  
0480 TAU_PROFILE("Board::invert()", ",", TAU_DEFAULT);  
0481 for (int x=0; x<actual_size; ++x) {  
0482 if (goban[x].getcolor() != EMPTY) {  
0483 goban[x].setcolor(INV(goban[x].getcolor()));  
0484 }  
0485 }

**pair< usi_t, usi_t > Board::play_move<usi_t, usi_t> (loc_t offset, color_t color)**  Plays a move on the board.

**Parameters:**

*offset*  Move to play

*color*  Color to play

**Returns:**

A pair such that the second element is a count of the stones removed for called color and the first element is a count of the stones removed for the opposite of the called color. The first element is thus the most important.

**Precondition:**

color is BLACK or WHITE but not EMPTY

0211 {  
0212 TAU_PROFILE("Board::play_move()", ",", TAU_DEFAULT);  
0213 assert(offset < actual_size);  
0214 loc_played = offset;  
0215 }
void Board::put_stone (loc_t loc, color_t color) [private] Sets a board location to a specific color.

Parameters:
    loc location as a single-dimension array offset
    color The color of the stone to place

Warning:
    Does not check for captures or suicide

string Board::raw_output (void) Output function for GUI.
    Raw board output

0218 goban[offset].setcolor(color);
0219
0220 // Check and delete for dead of opposite color
0221 usi_t them = del_group(INV(color));
0222
0223 // Check and delete for dead of our color
0224 usi_t us = del_group(color);
0225
0226 // Record with this board the color of the move just played
0227 color_played = color;
0228
0229 return make_pair(them, us);
0230 }

0341 {
0342 goban[loc].setcolor(color);
0343 }

0307 {
0308 string tmp;
0309
0310 //parsable board
0311 tmp += "board ";
0312 for (int loc=0; loc<actual_size; ++loc) {
0313     switch (goban[loc].getcolor()) {
0314         case BLACK: tmp += "B"; break;
0315         case WHITE: tmp += "W"; break;
0316         case EMPTY: tmp += "N"; break;
0317     }
0318     if ( (loc != (actual_size-1)) &&
0319         ( ((BSIZE - 1) == (loc % BSIZE )) ) tmp += ":";
void Board::setup () [private]  Bulk of the constructors’ logic.

This function performs the actual setup of the board. It allocates the Stone (p.160) class array and sets variables to initial values.

Precondition:
size is a natural number, and all elements in the list handicapPlaces are less than size*size. A board smaller than three or four probably is not useful as well.

Postcondition:
All variables are initialized and goban especially is setup. The exception is the variable loc_played which is undefined.

Warning:
loc_played is defined upon exit as the last of the setup moves played. If the board starts with a move at A13 then H2 as a handicap, then H2 is the logical value stored here.

{  
    // Allocate board and define its size
    actual_size = BSIZE * BSIZE;
    //bsize=size;
    goban = new Stone [actual_size];
    // Check memory allocation
    if (!goban) {
        LOG("-BRD -E- Goban memory allocation failed.");
        cerr << "-E- Goban memory allocation failed.";
        exit(1);
    }
    PRINTEXTRA = false;
    // Setup which turn. If a non-handicap game, black plays first on this
    // board which means that the "next turn" is white. On the other hand,
    // if there is a handicap, white plays first as black’s handicap was his
    // virtual first move.
}
0158 if (HANDICAP_PLACES.empty()) {
0159   color_played=WHITE;
0160 } else {
0161   color_played=BLACK;
0162 }
0163
0164 // Setup each spot in goban as empty (also setup column/row information)
0165 Stone tmp_stone;
0166 int col, row;
0167 for (int x=0; x < actual_size; ++x) {
0168   tmp_stone.clear();
0169
0170   // Offset mod the board size yield column number
0171   col=x%BSIZE;
0172
0173   // Offset divided by board size yields row number when truncated
0174   row=static_cast<usi_t>(x / BSIZE);
0175
0176   tmp_stone.setcol(col);
0177   tmp_stone.setrow(row);
0178   if (col == (BSIZE - 1)) tmp_stone.setlastcol();
0179   if (row == (BSIZE - 1)) tmp_stone.setlastrow();
0180
0181   goban[x] = tmp_stone;
0182 }
0183
0184 // Setup handicaps
0185 std::list<loc_t>::iterator pos;
0186 for (pos=HANDICAP_PLACES.begin(); pos != HANDICAP_PLACES.end(); ++pos) {
0187   // Only black gets handicap stones
0188   goban[*pos].setcolor(BLACK);
0189   loc_played = *pos;
0190 }
0191 }

bool Board::valid_location (loc_t loc) const  Tells if the location is playable.

This function takes no rules into account other than ”cannot play on an already
taken spot.”

Parameters:
loc offset into board vector

Warning:
Doesn’t check for loc less than zero, but it’s unsigned so it doesn’t matter.

0466 {
0467   TAU_PROFILE("Board::valid_location()", ",", TAU_DEFAULT);

87
A.2.5 DummyGenerator Class Reference

A dummy move generator that generates random legal moves.

```c
#include <outputgen.h>
```

Inheritance diagram for DummyGenerator::

```
public:
  DummyGenerator ();
  ~DummyGenerator ();

  int processing (void);

private:
  int rdbuf;
```

A.2.5.1 Detailed Description

A dummy move generator that generates random legal moves.
A.2.6 ExtenderAgent Class Reference

Suggests moves that extend from friendly stones.

#include <agent.h>

Inheritance diagram for ExtenderAgent:

```
Agent

ExtenderAgent
```

Public Methods

- **ExtenderAgent** ()
  
  Constructor.

- **~ExtenderAgent** ()
  
  Destructor.

- void **force** (void)
  
  Force this agent to move.

- void **update** (Game *)
  
  Updates the game for the agent. Refresh agent with a new game state.

- bool **dowork** (void)
  
  work thread.

- void **notify** (void *)
  
  Tell the agent something.

- unsigned int **query_bits_needed_from_GA** (void)
  
  Asks the agent how many bits it needs in the GA.

- void **send_bits** (**chromosome** cout chrom, int start)
  
  Sends to this agent the bits it needs from the GA.
Private Methods

- void attempt (int value, int locations[], int start)
  
  A helper function.

- unsigned int getval (chromosome_t chrom, int start)
  
  Calculates extension value from chromosome.

Private Attributes

- unsigned int bits_per_value
  
  Number of GA bits to use for each of the extension types.

- unsigned int num_values
  
  Number of extension types below.

- int extendValue
  
  Simple extension value.

- int extendLocations [5]
  
  Locations.

- int onePointExtendValue
  
  1-point extension value.

- int onePointExtendLocations [5]
  
  Locations.

- int twoPointExtendValue
  
  2-point extension value.

- int twoPointExtendLocations [5]
  
  Locations.

- int threePointExtendValue
  
  3-point extension value.

- int threePointExtendLocations [5]
Locations.

- int shoulderValue
  
  *Shoulder extension value.*

- int shoulderLocations [5]
  
  Locations.

- int knightValue
  
  *Knight's move value.*

- int knightLocations [9]
  
  Locations.

- int largeKnightValue
  
  *Large knight's move value.*

- int largeKnightLocations [9]
  
  Locations.

A.2.6.1 Detailed Description

Suggests moves that extend from friendly stones.

A.2.6.2 Member Function Documentation

```cpp
unsigned int ExtenderAgent::getval (chromosome_t chrom, int start)
[private]  Calculates extension value from chromosome.
```

**Parameters:**

- **chrom** The chromosome
- **start** Offset in chromosome where ExtenderAgent parameters are stored.

**Returns:**

The value read from the chromosome as an integer
```
0184 {  
0185     int sum = 0;
0186
0187     for(unsigned int x=start; x<start+bits_per_value; x++) {
0188         sum += static_cast<int>(chrom[x] * pow(2, x-start));
0189     }
0190
0191     return sum;
0192 }
```

```c
unsigned int ExtenderAgent::query_bits_needed_from_GA (void) [virtual]  
```

Asks the agent how many bits it needs in the GA.

**Returns:**
Number of bits needed in GA chromosome

Reimplemented from **Agent** (p. 75).

```
0170 {  
0171     return bits_per_value * num_values;
0172 }
```

### A.2.7 FollowerAgent Class Reference

Suggests moves near opponent’s last move.

```c
#include <agent.h>
```

Inheritance diagram for **FollowerAgent**:

```
Agent  
  
FollowerAgent
```

**Public Methods**

- **FollowerAgent ()**
  
  *Constructor.*

- **~FollowerAgent ()**
Destructor.

- void **force** (void)
  
  *Force the agent to make its move.*

- void **update** (Game *)
  
  *Update this agent with the latest state of the game.*

- bool **dowork** (void)
  
  *work thread.*

- void **notify** (void *)
  
  *Tell the agent something.*

- unsigned int **query_bits_needed_from_GA** (void)
  
  *Tells how many bits this agent needs from the GA.*

- void **send_bits** (chromosome_t chrom, int start)
  
  *Sends to this agent the bits it needs from the GA.*

Private Methods

- void **imprint** (loc_t loc, Board &b)
  
  *Helper function for internal algorithm.*

A.2.7.1 Detailed Description

Suggests moves near opponent’s last move.

A.2.7.2 Member Function Documentation

**void FollowerAgent::imprint** (loc_t loc, Board & b) [private] *Helper function for internal algorithm.*

  Adds a probability to a location based on how close it is to enemy stones

**Parameters:**

  - **loc** The location to check
b The go board to consult

```c
0095 {
0096     Stone *stones = b.get_goban();
0097     Stone s = stones[loc];
0098     color_t color = theGame.get_turn();
0099     if (color == BLACK) {
0100         if (s.notleft()) {
0101             // Due left
0102                 if (stones[loc-1].white()) (*pb_ptr)[loc] += 1.0;
0104         }
0105         // Top left
0106         if (s.nottop() && stones[loc-Board::BSIZE-1].white()) {
0107             (*pb_ptr)[loc] += 0.5;
0108         }
0109     }
0110     // Bottom left
0111     if (s.notbottom() &&
0112         stones[loc+Board::BSIZE-1].white()) {
0113         (*pb_ptr)[loc] += 0.5;
0114     }
0115     // Level 2: TODO
0116 }
0117 }
0118     if (s.nottop()) {
0119         // Right top
0120         if (stones[loc+1].white()) (*pb_ptr)[loc] += 1.0;
0121         // Top right
0122         if (s.nottop() &&
0123             stones[loc-Board::BSIZE+1].white()) {
0125         (*pb_ptr)[loc] += 0.5;
0126     }
0127     // Bottom right
0128     if (s.notbottom() &&
0129         stones[loc+Board::BSIZE+1].white()) {
0131         (*pb_ptr)[loc] += 0.5;
0132     }
0133     // Level 2: TODO
0134 }
0135 }
0136     if (s.nottop()) {
0137         // Due top
0138         if (stones[loc-Board::BSIZE].white())
0139             (*pb_ptr)[loc] += 1.0;
0140     // level 2
0141 }
0142 }
0143     if (s.notbottom()) {
```
// Due bottom
if (stones[loc+Board::BSIZE].white())
    (*pb_ptr)[loc] += 1.0;
// level 2
}
else {
    if (s.notleft()) {
        // Due left
        if (stones[loc-1].black()) (*pb_ptr)[loc] += 1.0;
        // Top left
        if (s.nottop() && stones[loc-Board::BSIZE-1].black()) {
            (*pb_ptr)[loc] += 0.5;
        }
        // Bottom left
        if (s.notbottom() &&
            stones[loc+Board::BSIZE-1].black()) {
            (*pb_ptr)[loc] += 0.5;
        }
        // Level 2: TODO
    }
    if (s.notright()) {
        // Due right
        if (stones[loc+1].black()) (*pb_ptr)[loc] += 1.0;
        // Top right
        if (s.nottop() && stones[loc+Board::BSIZE+1].black()) {
            (*pb_ptr)[loc] += 0.5;
        }
        // Bottom right
        if (s.notbottom() && stones[loc+Board::BSIZE+1].black()) {
            (*pb_ptr)[loc] += 0.5;
        }
        // Level 2: TODO
    }
    if (s.nottop()) {
        // Due top
        if (stones[loc-Board::BSIZE].black()) (*pb_ptr)[loc] += 1.0;
        // level 2
    }
    if (s.notbottom()) {
        // Due bottom
        if (stones[loc+Board::BSIZE].black()) (*pb_ptr)[loc] += 1.0;
        // level 2
    }
}
A.2.8 Ga Class Reference

Defines a Genetic Algorithm.

`#include <ga.h>`

Public Methods

- `Ga ()`
  
  Constructor.

- `~Ga ()`
  
  Destructor.

- `void init ()`
  
  Initialize first generation.

- `void set_codex (PreCodex *)`
  
  Tells GA what class has the fitness function to use.

- `int loadpop (string name="")`
  
  Loads a generation from disk.

- `int savepop (string name="")`
  
  Saves a generation to disk.

- `int savebest (string name="")`
  
  Saves the best chromosome to disk.

- `void start (void)`
  
  Starts the GA process.

Static Public Attributes

- `usi_t MAXGEN`
  
  Maximum number of generations.
• **usi_t** POPSIZE  
  *Size of the population.*

• **float** FITNESS_CUTOFF  
  *Unused.*

• **float** PCROSS  
  *Probability of crossover.*

• **float** PMUTATION  
  *Probability of mutation.*

• **float** FMULTIPLE  
  *Linear scaling parameter.*

• **string** FILENAME_IN  
  *For loading generations from disk.*

• **string** FILENAME_OUT  
  *For saving generations to disk.*

• **string** BEST_FILENAME_OUT  
  *For saving a chromosome.*

• **string** TRAIN_FILE  
  *SGF-derived data for training.*

**Private Methods**

• **void** ftest (float *f, float *prob)  
  *Compute the F-test.*

• **void** ttest (float *t, float *prob)  
  *Compute (Student’s) T-test.*

• **void** tutest (float *t, float *prob)
Compute the T-test (Student’s) if the variances aren’t the same.

- **int select (Population &pop)**
  
  Selects a chromosome.

- **bool flip (float)**
  
  Bernoulli probability.

- **allele mutation (allele allelemal)**
  
  Mutates an allele.

- **void crossover (chromosome_t &parent1, chromosome_t &parent2, chromosome_t &child1, chromosome_t &child2, usi_t &jcross)**
  
  Crosses the two parents to create the children.

- **void generation (void)**
  
  Increments the generation.

- **float scale (float, float, float)**
  
  Scales the fitness.

- **void scalepop (void)**
  
  Scales the fitness of the population.

- **void prescale (float &, float &)**
  
  Calculates linear parameters.

Private Attributes

- **pthread_mutex_t interrupt_watcher**
- **Population* oldpop**
  
  Pointer to old population.

- **Population* newpop**
  
  Pointer to new population.

- **usi_t lchrom**
Length of a chromosome.

- **usi_t gen**
  
  Generation counted.

- **bool stop**
  
  Interrupt flag.

- **usi_t nmutation**
  
  Number of mutations performed.

- **usi_t ncross**
  
  Number of crossovers performed.

- **float osumfitness**
  
  1st generation fitness sum.

- **float oavg**
  
  Average fitness in the first generation.

- **float omax**
  
  Maximum fitness in the first generation.

- **float omin**
  
  Minimum fitness in the first generation.

- **float ostdev**
  
  Standard deviation in the first generation.

- **float ovar**
  
  Variance in the first generation.

- **unsigned int rdbuf**
  
  Used for thread-safe random number generation.

- **PreCodex* leader**
  
  Points to the class that has the fitness function get_fitness().
A.2.8.1 Detailed Description

Defines a Genetic Algorithm.

The genetic algorithm needs a fitness function, thus one must make a call to \texttt{set\_codex()} (p. 108) before this class can be used.

A.2.8.2 Member Function Documentation

\textbf{bool Ga::flip (float val) [private]} Bernoulli probability.

\textbf{Returns:}

True if val is greater or equal to a uniform pseudo-random variable generated over [0,1]

\begin{verbatim}
0435 {  
0436   TAU_PROFILE("Ga::flip()", "", TAU_DEFAULT);  
0437   float res = static_cast<float>(rand_r(&rdbuf)) /  
0438       static_cast<float>(RAND_MAX);  
0439   return (res <= val) ? true : false;  
0440 }
\end{verbatim}

\textbf{void Ga::ftest (float * f, float * prob) [private]} Compute the F-test.

Computes statistics that help evaluate whether two distributions have different variances.

\textbf{Author:}

Numerical Recipes in C, modified by Todd Blackman, page 619

\begin{verbatim}
0735 {  
0736  
0737   if (ovar > newpop->var) {  
0738     *f = ovar / newpop->var;  
0739   if (fabs(newpop->var - 0.0) < 0.0000001) {  
0740     if (!gl\_global\_data.reg\_on) {  
0741       cerr << "-E- Bad variance of zero." << endl;  
0742       LOG("-E- Bad variance of zero.");  
0743     }  
0744     *f = -1;  
0745     *prob = -1;  
0746     return;  
0747   }  
0748 } else {  
0749   *f = newpop->var / ovar;
\end{verbatim}
if (fabs(ovar - 0.0) < 0.0000001) {
    if (!global_data.reg_on) {
        cerr << "-E- Bad variance of zero." << endl;
        LOG("-E- Bad variance of zero.");
    }
    *f = -1;
    *prob = -1;
    return;
}
float df = P0PSIZE - 1;

*prob = 2.0 * betai(0.5*df, 0.5*df, df/(df + df * (*f)));
if (*prob > 1.0) {
    *prob = 2.0 - *prob;
}

void Ga::init (void)  Initialize first generation.
    This function creates the newpop and oldpop structures which are both identical after this function finishes.
    
    {
        assert(leader != 0);
        //LOG("-- GA --M- Entered init().");
        oldpop->sumfitness = 0.0;
        oldpop->stdev = 0.0;
        oldpop->var = 0.0;
        oldpop->max = 0.0;
        oldpop->min = 10000;
        gen = 0;
        // Loop through each individual and initialize it
        for (int i=0; i<P0PSIZE; ++i) {
            Individual indv;
            assert(indv.chrom.empty());
            for (int a=0; a<lchrom; ++a) {
                indv.chrom.push_back(flip(0.5));
            }
            indv.parent1=0;
            indv.parent2=0;
            indv.xsite=0;
            }
indv.fitness = leader->get_fitness(indv.chrom);
indv.ofitness = indv.fitness;
assert(indv.fitness >= 0.0);
oldpop->individuals.push_back(indv);
oldpop->sumfitness += indv.fitness;
if ((*oldpop).individuals[i].fitness > oldpop->max) {
  oldpop->max = (*oldpop).individuals[i].fitness;
  oldpop->whichmax = i;
}
if ((*oldpop).individuals[i].fitness < oldpop->min) {
  oldpop->min = (*oldpop).individuals[i].fitness;
  oldpop->whichmin = i;
}
// This is to make the other vector know its size
newpop->individuals.push_back(indv);
}

//assert(oldpop->sumfitness > 0.0);
//assert(oldpop->max > 0.0);
oldpop->avg = oldpop->sumfitness / POPSIZE;

// Calculate the standard deviation of the values
float diffsquare;
for (int i=0; i<POPSIZE; ++i) {
diffsquare = (*oldpop).individuals[i].fitness - oldpop->avg;
diffsquare *= diffsquare;
oldpop->var += diffsquare;
}
oldpop->var /= POPSIZE - 1;
oldpop->stdev = sqrt(oldpop->var);
newpop->avg = oldpop->avg;
newpop->max = oldpop->max;
newpop->min = oldpop->min;
newpop->sumfitness = oldpop->sumfitness;
newpop->stdev = oldpop->stdev;
newpop->var = oldpop->var;
newpop->whichmax = oldpop->whichmax;
newpop->whichmin = oldpop->whichmin;

osumfitness = newpop->sumfitness;

oavg = newpop->avg;
omin = newpop->min;
int Ga::loadpop (string name = ")" Loads a generation from disk.

Parameters:
name The file name containing the population

Author:
Todd Blackman

```c
0195  ostdev = newpop->stddev;
0196  ovar = newpop->var;
0197 }
```

```c
0263 {
0264  int chrom_length;
0265  int pop_size;
0266  chromosome_t tmpchrome;
0267  Individual tmpindev;
0268  char c;
0269  //int x=0;
0270  
0271  if (leader == NULL) {
0272     cout << "E- use set_codex() first." << endl;
0273     exit(1);
0274  }
0275  
0276  if (name == ")") name = Ga::FILENAME_IN;
0277  oldpop->individuals.clear();
0278  newpop->individuals.clear();
0279  newpop->max = 0.0;
0280  newpop->min = 10000;
0281  newpop->sumfitness = 0.0;
0282  
0283  //LOG("- GA -M- Loading.");
0284  ifstream fin(name.c_str());
0285  if (fin) {
0286     fin >> pop_size;
0287     fin >> chrom_length;
0288  
0289  if (pop_size != POPSIZEx) {
0290     cerr << "E- datafile population size doesn't match." << endl;
0291     cerr << POPSIZEx << endl;
0292     cerr << pop_size << endl;
0293     LOG("- GA -E- Invalid popsize or chromosome_t length read");
0294     return -1;
0295  }
0296  
0297  }
0298  if (chrom_length != 1chrom) {
```
cerr << "-E- datafile chromosome length doesn’t match." << endl;
cerr << 1chrn << endl;
cerr << chrom_length << endl;
LOG("- GA  -E- Invalid popsize or chromosome_t length read");
return -1;
}

int i=0;
fin.get(c); // newline grab
while (!fin.eof()) {
    fin.get(c);
    if (c == 't') tmpchrome.push_back(true);
    if (c == 'f') tmpchrome.push_back(false);
    if (c == '\n') {
        i++;
        ttmpindev.chrom = tmpchrome;
        //LOG("- GA  -M- chromosome is " << ttmpchrome);
tmpindev.parent1 = 0;
tmpindev.parent2 = 0;
tmpindev.xsite = 0;

    // Fitness and stats calculate
    ttmpindev.fitness =
    leader->get_fitness(ttmpindev.chrom);
tmpindev.ofitness = ttmpindev.fitness;
assert(ttmpindev.fitness >= 0.0);
newpop->sumfitness += ttmpindev.fitness;
    if (ttmpindev.fitness > newpop->max) {
        newpop->max = ttmpindev.fitness;
        newpop->whichmax = i;
    }
    if (ttmpindev.fitness < newpop->min) {
        newpop->min = ttmpindev.fitness;
        newpop->whichmin = i;
    }
}
//oldpop->individuals[x++] = ttmpindev;
    if (tmpchrome.size() > 0) {
        newpop->individuals.push_back(ttmpindev);
    oldpop->individuals.push_back(ttmpindev);
    }
    //LOG("- GA  -M- iter10."");
tmpchrome.clear();
    }
fin.close();

newpop->avg = newpop->sumfitness / P0PSIZE;
scalepop();

104
void Ga::prescale (float & a, float & b) [private] Calculates linear parameters.

Parameters:
  a Slope
  b Intercept

Note:
  Page 79
int Ga::savebest (string name = "") Saves the best chromosome to disk.

Warning:
Untested
float Ga::scale (float obj, float a, float b) [private] Scales the fitness.

Note that the code in the text does not cut off at zero

Parameters:

obj objective value to scale

Note:
See page 79 of "Genetic Algorithms in Search, Optimization, and Machine Learning"

int Ga::select (Population & pop) [private] selects a chromosome.

This function selects a chromosome based on a roulette wheel paradigm

Note:
Taken from page 63.
float randn;  //<! Point on roulette wheel
float partsum = 0.0;  //<! Accumulator
int j=0;  //<! LCV (population index)

//LOG("- GA   -M- Entered select()");

// Wheel location
randn = static_cast<float>(rand_r(&rndbuf)) /
    static_cast<float>(RAND_MAX) * pop.sumfitness;

//LOG("- GA   -M- randn = " << randn);
//LOG("- GA   -M- sumfitness = " << pop.sumfitness);
//LOG("- GA   -M- rndbuf = " << rndbuf);
//LOG("- GA   -M- RAND_MAX = " << RAND_MAX);

assert(randn <= pop.sumfitness);
assert(pop.sumfitness >= 0);

// Find which individual it landed on.
do {
    partsum += pop.individuals[j++].fitness;
    //} while ((partsum < randn) && (j != POPSIZE));
} while ((partsum < randn) && (j < POPSIZE));

//LOG("- GA   -M- Leaving select() with value of " << j-1);

assert(j-1 >= 0);
assert(j-1 < POPSIZE);

// Return the index of the individual
return (j-1);

void Ga::set_codedx (PreCodex * f)  Tells GA what class has the fitness function to use.

This function also obtains the length of the chromosome.

Parameters:
    f A pointer to a class of type PreCodex (p.154) through inheritance.

Author:
    Todd Blackman

0104 {
0105   leader = f;
0106   lchrom = leader->get_chrom_size();
0107 }
void Ga::start (void)  Starts the GA process.

Author:
   Todd Blackman

pthread_mutex_lock(&interrupt_waiter);
stop=false;
pthread_mutex_unlock(&interrupt_waiter);

/\ could put a mutex in loop, but so what if we read the wrong value. On
// the next loop iteration it will read the correct one.

// Compute statistics
float t,f;
float tprob, fprob;
ttest(&t, &fprob);
string tsig, fsig;

if (fprob < SIGCUTOFF) {
   ttest(&t, &fprob);
   fsig = "diff";
} else {
   ttest(&t, &fprob);
   fsig = "same";
}

if (tprob < SIGCUTOFF) {
   tsig = "diff";
} else {
   tsig = "same";
}
// Output statistics
LOG("- GA -M " << setprecision(3)
<< setw(4) << gen << " 
<< setw(10) << (newpop->max) << " 
<< setw(10) << (newpop->min) << " 
<< setw(10) << (newpop->avg) << " 
<< setw(10) << (newpop->stdev) << " 
<< setw(10) << (newpop->sumfitness) << " 
" << fsig " " 
<< setw(10) << fprob << " 
" << t " " 
<< setw(10) << tprob << " 
);

do {
  generation();

  // Compute statistics
  ftest(&f, &fprob);

  if (fprob < SIGCUTOFF) {
    ttest(&t, &tprob);
    fsig = "diff";
  } else {
    ttest(&t, &tprob);
    fsig = "same";
  }

  if (tprob < SIGCUTOFF) {
    tsig = "diff";
  } else {
    tsig = "same";
  }

  // Output statistics
  LOG("- GA -M " << setprecision(3)
<< setw(4) << gen << " 
<< setw(10) << (newpop->max) << " 
<< setw(10) << (newpop->min) << " 
<< setw(10) << (newpop->avg) << " 
<< setw(10) << (newpop->stdev) << " 
<< setw(10) << (newpop->sumfitness) << " 
" << fsig " " 
<< setw(10) << fprob << " 
" << t " " 
<< setw(10) << tprob << " 
);

  pthread_mutex_lock(&log_mutex);
  assert(thread_count <= MAX_THREADS);
  assert(thread_count == 1);
  pthread_mutex_unlock(&log_mutex);
void Ga::ttest (float * t, float * prob) [private]  Compute (Student’s) T-test.

Computes statistics that help evaluate whether two distributions have different means

**Author:**
Numerical Recipes in C, modified by Todd Blackman, page 616

0779 {
0780   float df,svar;
0781   df=POPSIZE+POPSIZE-2;
0782   // Compute pooled variance
0783   svar = ((POPSIZE-1)*ovar+(POPSIZE-1)*newpop->var)/df;
0784   if ( fabs(svar - 0.0) < 0.0000001) {
0785       *t = -1;
0786       *prob = -1;
0787       return;
0788   }
0789   *t = (oavg-newpop->avg)/sqrt(svar*(1.0/POPSIZE+1.0/POPSIZE));
0790   *prob=betai(0.5*df,0.5,df/((df+(*t)*(*t)));
0791 }

void Ga::tutest (float * t, float * prob) [private]  Compute the T-test (Student’s) if the variances aren’t the same.

Computes statistics that help evaluate whether two distributions have different means

**Author:**
Numerical Recipes in C, modified by Todd Blackman, page 617-8

0806 {
0807   float df;
0808   *t = (oavg-newpop->avg) /
0810       sqrt(ovar/POPSIZE + newpop->var/POPSIZE);
0811 // Degrees of freedom calculation
0812 df=SQR(ovar/P0PSIZE + newpop->var/P0PSIZE) /
0813 (SQR(oover/P0PSIZE)/(P0PSIZE-1) +
0814 SQR(newpop->var/P0PSIZE)/(P0PSIZE-1));
0816 *prob=betai(0.5*df, 0.5, df/(df+SQR(*t)));
0818 }

A.2.8.3 Member Data Documentation

pthread_mutex_t Ga::interrupt_watcher [private] MUTEX for interrupting GA

Warning:
(unused)

A.2.9 Game Class Reference

A class that defines a series of boards.

#include <game.h>

Public Methods

- **Game** (void)
  
  Constructor.

- **Game** (Game &other)
  
  Copy Constructor.

- ~**Game** ()
  
  Destructor.

- void **reset** (void)

- void **play_move** (loc_t l)
  
  Play a move.

- void **play_move** (int x, int y)
  
  Plays a move given (x,y) coordinates.
• void `play_move (move_t m)`
  *Plays a move given a `move_t` (p.148) struct.*

• void `retract (usi_t num)`
  *Retracts moves.*

• `move_t last` (void)
  *Returns the last move made.*

• bool `legal (loc_t)`
  • bool `legal (int, int)`
    *Is the move legal?*

• bool `is_over` (void)
  *Is the game over yet?*

• `Board get_board ()` const
  *Returns the current board.*

• `usi_t get_bsize ()` const
  *Returns the board size.*

• `color_t wturn ()`
  *Whose turn is it?*

• `color_t get_turn ()`
  *Whose turn is it?*

• void `set_turn (color_t c)`
  *Override game conventions and just set whose turn it is.*

• list<loc_t> `enumerate_legal_locations` (void)
  *Returns legal locations.*

• void `invert_board` (void)
  *Changes black to white and vice versa.*

• void `lock` (void)
- void `unlock` (void)
  \textit{Unlocks the class.}

- int `get_captures` (color\_t \textit{col})
  \textit{Stub.}

- int `move_num` (void)

- bool `operator==` (const Game &)
  \textit{Equality operator.}

- bool `operator!=` (const Game &)

- Game `operator=` (Game)
  \textit{Assignment operator.}

\textbf{Static Public Methods}

- void `set_super_ko` (bool \textit{a})
  \textit{Set super KO checking.}

- void `set_suicide` (bool \textit{a})
  \textit{Set suicide checking.}

\textbf{Static Public Attributes}

- bool `SUPER_KO`
  \textit{Is superko rule in affect?}

- bool `SUICIDE`
  \textit{Is suicide allowed?}

- float `KOMI`
  \textit{Komi points to give.}

- `usi_t` `INITIAL_TIME`
  \textit{Initial game time.}
• **usi_t BYOMI.TIME**
  Time per byomi period.

• **usi_t BYOMI.STONES**
  Stones per byomi period.

Private Methods

• void **inv_turn** (void)
  Change whose turn it is.

• void **setup** ()
  Initializes things.

Private Attributes

• list<**Board**> **theGame**
  Actual list of boards.

• **color_t whose_turn**
  Color whose turn it is.

• list<**Board**>::iterator **currentBoard**
  Iterator pointing to board.

• list<**usi_p**> **capStones**
  captured stones. 1st is black; 2nd white.

• bool **enum_memoize_flag**
  Used for memoizing legal moves.

• pthread_mutex_t **mutex**
  MUTEX for operating on internal structures.
Static Private Attributes

- **bool super_ko**
  
  *Is superko rule in affect?*

- **bool suicide**
  
  *Is suicide allowed?*

- **float komi**
  
  *Points to white for having to play second.*

Friends

- **ostream& operator<<** (ostream & strm, Game & aGame)
  
  *Stream operator.*

A.2.9.1 Detailed Description

A class that defines a series of boards.

This class stores the game as a linked-list of **Board** (p. 78) classes.

A.2.9.2 Member Function Documentation

**list< loc_t > Game::enumerate_legal_locations<loc_t> (void)** Returns legal locations.

**Returns:**

- a list of integers (locations)

```
0245 { 
0246   TAU_PROFILE("Game::enumerate_legal_locations()", ",", TAU_DEFAULT); 
0247 
0248   list<loc_t> tmp; 
0249 
0250   //int lcnt = 0; 
0251 
0252   //LOG("-GAM -M- Entered enumerate_legal_locations()."); 
0253 
0254   for (loc_t x=0; x<(Board::BSIZE * Board::BSIZE); ++x) { 
```
if (legal(x)) {
    tmp.push_back(x);
    //lcnt++;
}

//LOG("-GAM -M- Did enumerate_legal_locations(): There were "
//    << lcnt << " legal moves excluding passing.");

return tmp;
}

int Game::getcaptures (color_t col)  Stub.

Returns how many stones captured by "col"

Parameters:

  col  Color that has captured the other color's stones

void Game::invert_board (void)  Changes black to white and vice versa.

This function does not alter whose turn it is nor the number of stones captured
semantics. The board is inverted using a call to the Board (p.78) class invert
function.
0139 { 
0140       TAU_PROFILE("Game::invert_board()", ",", TAU_DEFAULT);  
0141 
0142       //Board b = *theGame.rbegin();  
0143       Board b = theGame.back();  
0144       theGame.pop_back();  
0145       b.invert();  
0146       theGame.push_back(b);  
0147 }

bool Game::is_over (void)  Is the game over yet?
   
This is determined by there being two passes (two identical boards in a row)

0273         {  
0274         TAU_PROFILE("Game::is_over()", ",", TAU_DEFAULT);  
0275         list<Board>::reverse_iterator pos1;  
0276         list<Board>::reverse_iterator pos2;  
0277         list<Board>::reverse_iterator pos3;  
0278 
0279         pos1 = theGame.rbegin();  
0280         pos2 = theGame.rbegin();  
0281         pos2++;  
0282         
0283         if ((theGame.empty()) ||  
0284            (pos1 == theGame.rend()) ||  
0285            (pos2 == theGame.rend())) {  
0286             return false;  
0287         } else {  
0288             pos3 = pos2;  
0289             pos3++;  
0290             // Only two moves in record  
0291             if (pos3 == theGame.rend()) {  
0292                 return false;  
0293             } else {  
0294                 return (*pos1 == *pos2) && (*pos1 == *pos3));  
0295             }  
0296         }  
0297         }

move_t Game::last (void)  Returns the last move made.

Warning:
   Games with no moves yet return an undefined value.

0165 {  
0166       TAU_PROFILE("Game::last()", ",", TAU_DEFAULT);  
0167 
0168       //
0169 
016118
static move_t mv;
if (theGame.size() == 1) {
    cout << "E- No moves made yet. Cannot get last move." << endl;
    return mv;
} else {
    Board b = *theGame.rbegin();
    mv.loc = b.get_move_played();
    mv.color = b.get_color_played();
    if (mv.loc == Board::PASS) { mv.pass = true; }
    if (theGame.size() == 2) {
        mv.newboard = true;
    } else {
        mv.newboard = false;
    }
    mv.setup_phase = false;
    //mv.bsize = this->bsize;
    return mv;
}

bool Game::legal (int x, int y)  Is the move legal?

Warning:
Passes are always legal. This function does not accept semantics of "pass"

return legal(y * Board::BSIZE + x); }

bool Game::legal (loc_t loc)  Tests if a move is legal.

Todo:
Add memoizability-> store vector of legal/not-legal that is updated as moves
are made.
void Game::lock (void)  Locks the class

    {
    pthread_mutex_lock(&mutex);
    }

int Game::movenum (void)  Tells the current move number

    This function calculates this value based on the number of boards in the game.
bool Game::operator!=(const Game & other)  // Inequality operator
{
    return !(this == other);
}

void Game::play_move(loc_t l)  // Play a move.
{
    // A location to play a move on (must be legal!)
    // Does not do error checking or validity checking

    TAU_PROFILE("Game::play_move()", ",", TAU_DEFAULT);
    Board b;
    pair<usi_t, usi_t> captures;
    captures.first = 0;
    captures.second = 0;
    // Copy the last board onto the end of the list
    b=*currentBoard;
    theGame.push_back(b);
    ++currentBoard;
    //enum_memoize_flag = false;
    if (loc != Board::PASS) {
        captures = currentBoard->play_move(loc, whose_turn);
        if (whose_turn==WHITE) {
            swap(captures.first, captures.second);
        }
    }
    capStones.push_back(captures);
    // Make it the other color's turn
    inv_turn();
void Game::reset (void)  Totally clears and resets the game to initial state.

0481 {  
0482   Board b;  
0483   theGame.clear();  
0484   theGame.push_back(b);  
0485   capStones.clear();  
0486   usi_p capturedStones;  
0487   capturedStones.first=0;  
0488   capturedStones.second=0;  
0489   capStones.push_back(capturedStones);  
0490   whose_turn = BLACK;  
0491   //enum_memoize_flag = false;  
0492   currentBoard = theGame.begin();  
0493 }  

void Game::retract (usi_t num)  Retracts moves.

This function completely destroys all record of the previous num moves. Since the moves/board-states are stored in a list, retracting is a very simple matter.

Parameters:

num The number of moves to retract

0107 {  
0108   TAU_PROFILE("Game::retract()", ",", TAU_DEFAULT);  
0109   // Protect against retracting past first move.  
0110   if (theGame.size() <= num) {  
0111     num = theGame.size() - 1;  
0112   }  
0113 }  
0114  
0115  // Remove last "num" boards  
0116   for (int x=0; x<num; ++x) { theGame.pop_back(); }  
0117  // Remove last "num" captured stones pairs  
0118   for (int x=0; x<num; ++x) { capStones.pop_back(); }  
0120   assert(theGame.size() == capStones.size());  
0122  // Set whose turn it is.  
0124   if (odd(num)) { inv_turn(); }  
0125
// Set current board iterator as the last board in the list
currentBoard = theGame.end();
--currentBoard;
}

A.2.9.3 Friends And Related Function Documentation

ostream & operator<< (ostream & strm, Game & aGame) [friend]
Stream operator.
This is used to output the latest state of the game

Board b = aGame.get_board();
strm << b;
return strm;

A.2.10 GaTrainerInterface Class Reference

Used to train a GA to work correctly.
#include <interface.h>

Inheritance diagram for GaTrainerInterface:

Public Methods

• GaTrainerInterface ()
  Constructor.

• ~GaTrainerInterface ()
  Destructor.

• void load (string filename)
  Loads into memory the training data.
- float get_percentage ()
  
  Figures fraction of correct guesses.

Private Methods

- void processing (void)
  
  Main processing loop.

- void handle_move (void)
  
  Handles modifications to setup board for opponent given the correct move.

- void init (void)
  
  Initializes Interface (p. 144).

Private Attributes

- int totalmoves
  
  Moves played.

- int movesGuessed
  
  Moves played correctly.

- list<move_t> movestream
  
  The correct moves (from recorded games).

- list<move_t>::iterator movestream_iter
  
  Iterator for movestream.

A.2.10.1 Detailed Description

Used to train a GA to work correctly.
A.2.10.2 Member Function Documentation

float GaTrainerInterface::get_percentage () Figures fraction of correct guesses.

This function looks at the number of moves in the game record and the number of moves guessed correctly and calculates the fraction of the moves guessed correctly.

Returns:

Fraction of the recorded game moves guessed correctly

0222 {
0223     if (totalmoves == 0) { return static_cast<float>(0); };
0224     float perc = static_cast<float>(movesGuessed) /
0225         static_cast<float>(totalmoves);
0226     log("-GAT -M- Got " << perc << " right.");
0227     return perc;
0228 }

void GaTrainerInterface::init (void) [private] Initializes Interface (p.144).

Precondition:

movestream has been loaded with data via the load() (p.125) function call.

0074 {
0075     // Point to data start
0076     movestream_iter = movestream.begin();
0077 }

void GaTrainerInterface::load (string fname) Loads into memory the training data.

The format is a space delimited record with records marked with newline characters PASS MOVE LOCATION COLOR IGNORE IGNORE IGNORE BOARDSIZE.
all fields are one character except MOVE LOCATION which is three MOVE - LOCATION is an offset into a single-dimension array.
A.2.11 GenAlgoGenerator Class Reference

A genetic algorithm move generator.

```c++
#include <outputgen.h>

Inheritance diagram for GenAlgoGenerator:
```
Public Methods

- **GenAlgoGenerator ()**
  Constructor.

- **~GenAlgoGenerator ()**
  Destructor.

- void **load** (string filename="")
  Loads GA parameters from a file on disk.

- void **printweights** (void)
  Prints the weights for the network to the log file.

- void **decode** (const chromosome_t &chrom)
  Decodes chromosome.

- float **get_fitness** (const chromosome_t &chrom)
  Objective Function.

- void **summary** (Population *newpop)
  Outputs the best chromosome and the weights.

Private Methods

- void **init** (void)
  Sets up agent/generator connections.

- loc.t **get_move** (void)
  Generate a probability board and etc.
• **void** **processing** (void)
  
  *Main processing loop.*

**Private Attributes**

• **int** **weights** [SECONDLEVELNODES][MAX_AGENTS]
  
  *Weights to optimize.*

• **int** **secondLevelWeights** [SECONDLEVELNODES]
  
  *Second level weights.*

• **Agent** *theAgents* [MAX_AGENTS]
  
  *Pointers to all agents we use.*

• **AgentShell** *theThreads* [MAX_THREADS]
  
  *The threads for working.*

• **ProbBoard** **results** [MAX_AGENTS]
  
  *Agents put results here.*

• **Blackboard** **bb**
  
  *Information viewable by all agents.*

• **int** **num_agents**
  
  *Number of active agents.*

• **int** **num_threads**
  
  *Number of running threads for agents.*

• **unsigned int** **rndbuf**
  
  *State var used for random number generation.*

• **bool** **weights_loaded**
  
  *Are the weights set yet?*

• **unsigned int** **bits_per_weight**
  
  *Bits used for each weight in the net.*
• unsigned int num_second_level_nodes
  
  Number of nodes in level 2.

• unsigned int total_bits
  
  Total number of bits in chromosome.

A.2.11.1 Detailed Description

A genetic algorithm move generator.

A.2.11.2 Member Function Documentation

void GenAlgoGenerator::decode (const chromosome_t & chrom)  Decodes chromosome.

This function reads chromosome as a string of three integer weights encoded as four bits each (12 bits total)

Author:
  
  Todd Blackman

0406 {
0407     TAUPROFILE("GenAlgo::decode()", ",", TAU_DEFAULT);
0408
0409     int z=0;
0410     int tmpweight=0;
0411
0412     set_chrom_size(total_bits);
0413
0414     //cout "-M- " chrom.size() " " total_bits endl;
0415
0416     assert(chrom.size() == total_bits);
0417
0418     LOG("-GAG " setw(7) childt "-M- Loading chromosome: "
0419         << chrom);
0420
0421     unsigned int count=0;
0422     unsigned int count2=0;
0423     int endOfWeights = num_agents * bits_per_weight * num_second_level_nodes +
0424          num_second_level_nodes * bits_per_weight;
0425     for (int x=0; x<endOfWeights; ++x) {
0426         if ((x>0) && (x % bits_per_weight))) {
0427
129
float GenAlgoGenerator::get_fitness (const chromosome_t & chrom) [virtual]  Objective Function.

This objective function is exceedingly complex though one cannot tell just by looking here. This function creates a moderator with two players, the GA player and a GA-trainer player. This is then allowed to run until the trainer exhausts its series of moves to play.
Returns:

Fitness value

Reimplemented from Precodex (p. 155).

0486 {  
0487   TAU_PROFILE("GenAlgo::get_fitness()", ",", TAU_DEFAULT);  
0488   float fitness;  
0489  
0490  // LOG("-GAG " << setw(7) << childt << "-M- Entered get_fitness().");  
0491  
0492  // Connect the trainer and trainee with a moderator  
0493  Moderator<GenAlgoGenerator, GaTrainerInterface> trainpair;  
0494  
0495  cerr << "." << flush;  
0496  
0497  // Setup the trainer and trainee (Do we need this at all?)  
0498  // Interface *trainees;  
0499  GenAlgoGenerator *trainee;  
0500  // Interface *trainer;  
0501  GaTrainerInterface *trainer;  
0502  
0503  trainee = static_cast<GenAlgoGenerator *>(trainpair.get_IO());  
0504  trainer = static_cast<GaTrainerInterface *>(trainpair.get_IO());  
0505  
0506  // Adjust weights based on chromosome  
0507  trainee->decode(chrom);  
0508  
0509  // prepare the trainer  
0510  //static_cast<GaTrainerInterface *>(trainer)->load(".test.dat");  
0511  static_cast<GaTrainerInterface *>(trainer)->load(Ga::TRAIN_FILE);  
0512  
0513  LOG("-GAG " << setw(7) << childt  
0514       << ":-M- About to call mainloop from get_fitness().");  
0515  
0516  // Run moderator till trainer says done.  
0517  trainpair.mainloop();  
0518  
0519  LOG("-GAG -M- Done with mainloop in fitness-finding function.");  
0520  
0521  // Get stat from trainer interface.  
0522  fitness = static_cast<GaTrainerInterface *>(trainer)->get_percentage();  
0523  
0524  LOG("-GAG -M- Fitness has been calculated.");  
0525  
0526  // Return percentage of moves correctly guessed.  
0527  return fitness;  
0528 }
loc_t GenAlgoGenerator::get_move (void) [private] Generate a probability board and etc.

This function calls all agents to the task of creating a probability board. This function then proceeds to sum these boards as defined by the genetic algorithm weight parameters.

0305 {
0306    TAU_PROFILE("GenAlgo::get_move()", ",", TAU_DEFAULT);
0307
0308    loc_t loc;
0309    ProbBoard pbtmp;
0310    ProbBoard pbresult;
0311    msg_t msg;
0312
0313    // LOG("-GAG " << setw(7) << childt << ",-M- Entered get_move()");
0314
0315    // Fill up as many threads as we have.
0316    for (int x=0; x<num_threads; x++) {
0317        assert(theAgents[x] != NULL);
0318
0319        // Load the next agent
0320        msg.id = LOAD;
0321        msg.data = static_cast< void > *(theAgents[x]);
0322        theThreads[x].send_msg(msg);
0323
0324        // Update the agent
0325        msg.id = UPDATE;
0326        // pthread_mutex_lock(&update_mutex);
0327        msg.data = static_cast< void > *(gptr);
0328        // pthread_mutex_unlock(&update_mutex);
0329        theThreads[x].send_msg(msg);
0330    }
0331
0332    int count=0;
0333    while(count < num_threads) {
0334        msg = theThreads[count].get_msg_nb();
0335        if (msg.id == FINISHED) count++;
0336    }
0337
0338    // LOG("-GAG " << setw(7) << childt << ",-M- Got all finished messages.");
0339
0340    // Compute probability board
0341    pbresult.clear();
0342    for (unsigned int lcv=0; lcv<num_second_level_nodes; ++lcv) {
0343        pbtmp.clear();
0344        // Calculate results second level results.
0345        for (int x=0; x<num_agents; ++x) {
0346            pbtmp += results[x] * weights[lcv][x];
0347
0348                // cout << "results[" << x << "]=" << endl << results[x]
0349                // << endl;
0350
0351            pbresult += pbtmp;
0352        }
0353
0354        // pbresult = pbtmp;
0355    }
0356    return pbresult;
0357}
void GenAlgoGenerator::load (string filename = "") Loads GA parameters from a file on disk.

Author:
Todd Blackman

{ ifstream fin;
  chromosome_t tmpchr;
  unsigned int chrom_length;
  char c;
}
0175 if (name == "") name = Ga::BEST_FILENAME_OUT;
0177 fin.open(name.c_str());
0178
0179 if (fin) {
0180     fin >> chrom_length;
0181     // Check that datafile has correct cromosome length
0182     if (chrom_length != total_bits) {
0183         cerr << "-E- datafile chromosome length doesn't match: ";
0184         cerr << total_bits << "!=" << chrom_length << endl;
0185         LOG("-GAG -E Invalid chromosome_t length read");
0186         return;
0187     }
0188 }
0189
0190     while (fin) {
0191         fin.get(c);
0192         if (c == '1') tmpchrome.push_back(true);
0193         if (c == '0') tmpchrome.push_back(false);
0194     }
0195 fin.close();
0196
0197 if (tmpchrome.size() != total_bits) {
0198     cerr << "-E- datafile chromosome length doesn't match: ";
0199     cerr << total_bits << "!=" << tmpchrome.size() << endl;
0200     LOG("-GAG -E Invalid chromosome_t length read");
0201     return;
0202 }
0203 }
0204
0205 } else {
0206     LOG("-GAG -E Cannot load Genetic Algorithm data from disk.");
0207 }
0208 }
0209 decode(tmpchrome);
0210
0211
0212 }

A.2.12 global_data_t Struct Reference

global data structure.

#include <exodus.h>

Public Attributes

- bool welcome
Show welcome screen to user.

- **bool train**
  
  Train GA or not.

- **int verbosity**
  
  How much output to output (unused).

- **char* resume**
  
  File name to resume a GA run.

- **bool version**
  
  Show version number or not.

- **bool help**
  
  Show help message or not.

- **bool reg_on**
  
  Run regression or not.

- **color_t my_color**
  
  Human player’s color (untested).

- **char* handicap_placement**
  
  Where to place the handicap stones.

### A.2.12.1 Detailed Description

global data structure.

Make this as small as possible

### A.2.13 GoModemInterface Class Reference

Go modem interface.

```c
#include <interface.h>
```

Inheritance diagram for GoModemInterface:
Public Methods

- **GoModemInterface ()**
  
  *Constructor.*

A.2.13.1 Detailed Description

Go modem interface.

A.2.14 GroupStatsAgent Class Reference

**Agent** (p. 73) to calculate group information.

```c
#include <agent.h>
```

Inheritance diagram for GroupStatsAgent:

- **Agent**
- **GroupStatsAgent**

Public Methods

- **GroupStatsAgent ()**
  
  *Constructor.*

- **~GroupStatsAgent ()**
  
  *Destructor.*

- **void force (void)**
  
  *Force the agent to move.*
• void **update** (Game *)
  
  *Updates the agent with the latest state of the game.*

• bool **dowork** (void)
  
  *work thread. Kills groups.*

• void **notify** (void *)
  
  *Tell the agent something.*

• unsigned int **query_bits_needed_from_GA** (void)
  
  *Asks the agent for the number of bits it needs from GA.*

• void **send_bits** (chromosome_t chrom, int start)

**Private Methods**

• void **printScratch** (void)
  
  *Prints the group scratchpad to STDOUT.*

• void **recurse** (Stone goban[], int loc, int gnum)
  
  *Recursive function to label a group by number.*

**Private Attributes**

• int **scratch** [19 *19]
  
  *Holds a bitmap that shows the group numbers. Each group is uniquely identified by a number.*

• int **numgroups**
  
  *Number of groups.*

• bool **dead** [MAXGROUPS]
  
  *Which groups are dead and which alive.*

• **color_t** **grpcolor** [MAXGROUPS]
  
  *Color of the group.*
• unsigned int **size [MAXGROUPS]
  Size of the group.

• unsigned int liberties [MAXGROUPS]
  Number of liberties.

• bool liberty_locations [MAXGROUPS][19 *19]
  Liberty locations as a bitmap.

• list<loc_t> liberty_locations_list [MAXGROUPS]
  Liberty locations as a list of locations.

A.2.14.1 Detailed Description

Agent (p.73) to calculate group information.

A.2.14.2 Member Function Documentation

void GroupStatsAgent::send_bits (chromosome_t chrom, int start)
[virtual] Sends to this agent the bits it needs from the GA
  Reimplemented from Agent (p.75).

0069 { 
0070   //int end = start + query_bits_needed_from_GA();
0071 
0072 }

A.2.15 GUIInterface Class Reference

Graphical User Interface (p.144).

#include <interface.h>

Inheritance diagram for GUIInterface::
Public Methods

- **GUIInterface (usi_t size=19)**
  
  Constructor.

- **GUIInterface (string thepath, usi_t size=19)**
  
  Constructor.

- **~GUIInterface ()**
  
  Destructor.

Static Public Attributes

- **string GPATH**
  
  Path to GUI frontend.

Private Methods

- **loc.t get_move (void)**
  
  Gets move from GUI.

- **void send_board (Board b, color_t whose_turn)**
  
  Sends the current board to the interface from engine.

- **void init (void)**
  
  Forks off the gui.

- **void figure_path (void)**
  
  Sets path to gui.
• void processing (void)
  
  Main logic loop of the interface.

Private Attributes

• int m2s [2]
  
  Master to slave flow (interface to outside).

• int s2m [2]
  
  Slave to master flow (outside to interface).

• int pid
  
  Child Process ID.

• uso_t bsize
  
  Board (p.78) size.

• string path
  
  Path to gui program.

Static Private Attributes

• const int READ = 0
  
  Constant.

• const int WRITE = 1
  
  Constant.

A.2.15.1 Detailed Description

Graphical User Interface (p.144).
A.2.16  IGS_Interface Class Reference

Internet Go Server (IGS) Interface (p.144).

#include <interface.h>

Inheritance diagram for IGS_Interface:

```
+------------------+
| Subthread        |
+------------------+
|                  +------------------+
| Interface        | Interface           |
|                  +------------------+
|                  |                  +------------------+
| IGS_Interface    |                  | IGS_Interface       |
```

Public Methods

- **IGS_Interface ()**  
  Constructor.

- **~IGS_Interface ()**  
  Destructor.

Private Methods

- void **setup (void)**  
  Initializes the class.

Private Attributes

- string **host1**  
  First IGS server to try.

- string **host2**  
  Second IGS server to try.

- usi_t **port1**  
  Port on first host.
• usi_t port2
  
  *Port on second host.*

• sockaddr my_addr
  
  *Local IP address.*

• int sfd

  *File descriptor for socket connection.*

A.2.16.1 Detailed Description

Internet Go Server (IGS) Interface (p. 144).

**Warning:**

This class is a stub.

A.2.17 Individual Struct Reference

An individual in a population of a GA.

#include <ga.h>

**Public Methods**

• bool operator== (const Individual&) const
  
  *Equality operator.*

• bool operator!= (const Individual&) const
  
  *Inequality operator.*

• Individual operator= (Individual)
  
  *Assignment operator.*

**Public Attributes**

• chromosome_t chrom

  *The chromosome that represents this individual.*


- float ofitness
  
  *Original fitness.*

- float fitness
  
  *Fitness after scaling.*

- usi_t parent1
  
  *First parent chromosome.*

- usi_t parent2
  
  *Second parent chromosome.*

- usi_t xsite
  
  *Site of crossover.*

A.2.17.1 Detailed Description

An individual in a population of a GA.

A.2.17.2 Member Function Documentation

bool Individual::operator==(const Individual & other) const

Equality operator.

**Warning:**

Not used for now. This will be useful when the algorithm is multi-threaded which it currently isn’t.

```c
void Ga::interrupt() {  
  pthread_mutex_lock(&interrupt_watcher);  
  stop=true;  
  pthread_mutex_unlock(&interrupt_watcher);  
}
```

0959 {
0960     return(this->chrom == other.chrom);
0961 }

143
A.2.18  Interface Class Reference

The interface between a move generator (outside) and the inside of the program.

```cpp
#include <interface.h>
```

Inheritance diagram for Interface:

```
Subthread
  Interface
    DummyGenerator
    GaTrainerInterface
    GenAlgoGenerator
    GoModemInterface
    GUIInterface
    IGS_Interface
    NeuralNetGenerator
    NNGS_Interface
    TextInterface
```

**Public Methods**

- **Interface ()**
  
  *Constructor.*

- **~Interface ()**
  
  *Destructor.*

- void **set_my_turn_on** (void)
  
  *Moderator* (p. 146) *class uses these to control who is "active" meaning.*

- bool **get_made_a_move** (void)

144
Moderator (p. 146) class uses these to control who is "active" meaning "whose turn is it?".

- void `set_my_color (const color_t col)`

Protected Attributes

- bool `my_turn`  
  Is it my turn?

- bool `made_a_move`  
  Have I made my move for this round?

- Game* `gptr`  
  Points to current Game (p. 112).

- color_t `my_color`  
  My color.

- color_t `their_color`  
  Opponent’s color.

A.2.18.1 Detailed Description

The interface between a move generator (outside) and the inside of the program.

Warning:
This is an abstract class

`set_die_ptr` of the Subthread class must be set before using any 
interface. In addition to this, `gptr` and `resign_ptr` need to be set before sub-
thread’s start() (p. 165) function is called.

A.2.18.2 Member Function Documentation

void Interface::set_my_color (const color_t col) [inline] Sets interface’s color

```c++
0070 { my_color = col;
0071     their_color = INV(col); }
```
A.2.19 Moderator Class Template Reference

Encapsulates two interfaces and has them play together.

Public Methods

- **Moderator ()**
  
  Constructor.

- **~Moderator ()**
  
  Destructor.

- **Interface* get_I0 (void)**
  
  Retrieves the first interface.

- **Interface* get_I1 (void)**
  
  Retrieves the second interface.

- **Game* get_game (void)**
  
  Retrieves the game.

- **void mainloop (void)**
  
  Lets the interfaces play with each other.

- **void swap_interfaces (void)**
  
  Swaps semantics of I0 and I1.

Private Attributes

- **I0.* I0**
  
  First Interface (p.144) (0).

- **I1.* I1**
  
  Second Interface (p.144) (1).

- **bool whose_turn**
  
  Which interface’s turn is it?
• Game theGame
  
  The game.

A.2.19.1 Detailed Description

template<class I0_t, class I1_t> class Moderator Encapsulates two interfaces and has them play together.

  The first interface, I0, is the black player and thus I1 receives the handicap by definition. To alter this, one needs only to swap the two interfaces using the method swap_interfaces() (p.146).

Warning:

  The functions get_I0() (p.146), get_I1() (p.146), and get_game() (p.146) should be used wisely. They are sources of error and potential faults, but I’ll trust myself and potential future programmers to not screw the semantics up like making two interfaces think the board is a different size than it is.

Todo:

  Add time-keeping code.

A.2.19.2 Constructor & Destructor Documentation

template<class I0_t, class I1_t> Moderator< I0_t, I1_t >::Moderator< I0_t, I1_t > () Constructor.

Warning:

  I0 always goes first

0110 {
0111
0112   I0 = new(I0_t);
0113   I1 = new(I1_t);
0114   if ((!I0) || (!I1)) {
0115     LOG("-MOD -E- Interface memory allocation failed.");
0116     cerr << "-E- Interface memory allocation failed.";
0117     exit(1);
0118   }
0119 
0120   // Set Game class for two opponents
0121   msg_t msg;
0122   msg.id = SET_GAME_PTR;

147
0123 msg.data = static_cast<void *>(&theGame);
0124 10->send_msg(msg);
0125 11->send_msg(msg);
0126 }

A.2.20 move_t Struct Reference

A single move on the goban.

#include <move.h>

Public Methods

- bool operator==(move_t &m)
  equality operator.

- move_t operator=(move_t other)
  assignment operator.

- void regression (void)
  Unused.

Public Attributes

- bool pass
  Is the move a pass?

- loc_t loc
  Where the move is played.

- color_t color
  Color of the move played.

- bool newboard
  Is this the first of a new board?

- bool setup_phase
  Still setting up handicaps?
- int bsize
  
  *Size of the board.*

A.2.20.1 Detailed Description

A single move on the goban.

A.2.21 msg_t Struct Reference

A message to or from a thread.

```c
#include <subthread.h>
```

Public Methods

- msg_t `operator=(msg_t)`
  
  *Assignment operator.*

Public Attributes

- msg_id_t `id`
  
  *Message ID (type).*

- void* `data`
  
  *Message payload.*

A.2.21.1 Detailed Description

A message to or from a thread.

A.2.21.2 Member Function Documentation

```c
msg_t msg_t::operator=(msg_t tmpmsg)
```

*Assignment operator.*

*Equality operator for a message.*
A Neural Network move generator.

#include <outputgen.h>

Inheritance diagram for NeuralNetGenerator:

```
subthread
  |   |
  |   |  
  |   |  
interface
  |   |
  |   |  
  |   |  
networkgenerator
```

Public Methods

- **NeuralNetGenerator ()**
  
  *Constructor.*

A.2.22.1 Detailed Description

A Neural Network move generator.

**Warning:**
This is a STUB

A.2.23 NNGS_Interface Class Reference

No Name Go Server Interface (p. 144).

#include <interface.h>

Inheritance diagram for NNGS_Interface::
Public Methods

- **NNGS_Interface ()**
  
  *Constructor.*

### A.2.23.1 Detailed Description

No Name Go Server **Interface** (p. 144).

**Warning:**

This class is a STUB.

### A.2.24 OpenerAgent Class Reference

Suggests good opening moves.

```cpp
#include <agent.h>
```

Inheritance diagram for OpenerAgent:

```
    Agent
     |
     |
  OpenerAgent
```

Public Methods

- **OpenerAgent ()**
  
  *Constructor.*

- **~OpenerAgent ()**
  
  *Destructor.*
• void **force** (void)
  
  *Force agent to move.*

• void **update** (Game *)
  
  *Update the agent with the current game.*

• bool **dowork** (void)
  
  *work thread.*

• void **notify** (void *)
  
  *Tell the agent something.*

• unsigned int **query_bits_needed_from_GA** (void)
  
  *Ask the agent how many bits it needs in GA.*

• void **send_bits** (chromosome_t chrom, int start)

**Private Attributes**

• ProbBoard pb19
  
  *Stores choices for 19x19 board.*

• ProbBoard pb17
  
  *Stores choices for 17x17 board.*

• ProbBoard pb9
  
  *Stores choices for 9x9 board.*

**A.2.24.1 Detailed Description**

*Suggests good opening moves.*

**A.2.24.2 Member Function Documentation**

void OpenerAgent::**send_bits** (chromosome_t chrom, int start)
  
  [virtual] Sends to this agent the bits it needs from the GA
  
  Reimplemented from Agent (p. 75).
0122 {  
0123     //int end = start + query_bits_needed_from_GA();  
0124 }

A.2.25 Population Struct Reference

A single population within a GA.

#include <ga.h>

Public Methods

- bool operator==(const Population & const Population)
  
  Equality operator.

- bool operator!=(const Population & const Population)
  
  Inequality operator.

- Population operator=(Population)
  
  Population assignment operator.

Public Attributes

- vector<Individual> individuals
  
  The individuals in the chromosome.

- float sumfitness
  
  Sum of all fitness values in this generation.

- float avg
  
  Average fitness in this generation.

- float max
  
  Maximum fitness in this generation.

- float min
  
  Minimum fitness in this generation.
• float stdev
  Standard deviation in this generation.

• float var
  Variance deviation in this generation.

• int whichmax
  Which in population is max.

• int whichmin
  Which in population is min.

Friends

• ostream& operator<<(ostream &strm, const Population &pop)
  Population output operator.

A.2.25.1 Detailed Description

A single population within a GA.

Every GA has two populations: An old one and a new one.

A.2.26 PreCodex Class Reference

Allows other classes to provide a fitness function.

#include <gafunc.h>

Inheritance diagram for PreCodex::

```
    PreCodex
     ^
    |  
GenAlgoGenerator  testCodex
```

Public Methods

• void set_chrom_size (int s)
Sets the chromosome size to s.

- int get_chrom_size (void)
  
  Gets the current chromosome size.

- virtual float get_fitness (const chromosome_t &chrom)=0
  
  Decoder and Objective function.

- virtual void summary (Population *newpop)=0
  
  Outputs testing results after training is done.

Protected Attributes

- int lchrom
  
  Length of chromosome in bits (alleles).

A.2.26.1 Detailed Description

Allows other classes to provide a fitness function.

A.2.27 ProbBoard Class Reference

Agent (p. 73)'s probability output board.

```c
#include <probboard.h>
```

Public Methods

- ProbBoard ()
  
  Constructor.

- ~ProbBoard ()
  
  Destructor.

- void set_val (int offset, float value)
  
  Sets weight.
• float \texttt{get\_val} (int offset)  
  \textit{Gets weight.}

• void \texttt{normalize} (void)  
  \textit{Normalizes the weights.}

• int \texttt{spin} (void)  
  \textit{Chooses a random offset in probability board based on probabilities.}

• \texttt{loc\_t maxloc} (void)  
  \textit{Choose the location with the highest value.}

• void \texttt{clear} (void)  
  \textit{Clears board.}

• ProbBoard \texttt{operator=} (ProbBoard)  
  
  Assignment \texttt{operator.}

• bool \texttt{operator==} (ProbBoard)  
  \textit{equality overloaded \texttt{operator.}}

• bool \texttt{operator!=} (ProbBoard)  
  \textit{Inequality \texttt{operator.}}

• bool \texttt{operator+=} (ProbBoard)  
  \textit{Addition assignment \texttt{operator.}}

• ProbBoard \texttt{operator\*} (float) const  
  \textit{Multiplication \texttt{operator.}}

• ProbBoard \texttt{operator+=} (ProbBoard)  
  \textit{Addition \texttt{operator.}}

• float\& \texttt{operator[]} (loc\_t location)  
  \textit{Offset and Array-use \texttt{operator.}}
Private Attributes

- int actualSize
  
  Size of internal array.

- float internal_board [19 *19+1]
  
  Single dimension array.

- unsigned int rndbuf
  
  Seed for random number generator.

Friends

- ostream& operator<< (ostream &strm, ProbBoard &aBoard)
  
  Stream operator.

A.2.27.1 Detailed Description

Agent (p. 73)’s probability output board.

A.2.27.2 Member Function Documentation

loc_t ProbBoard::maxloc (void) Choose the location with the highest value. Heuristic would have the FIRST of any tie values chosen.

0184 {  
0185   TAU_PROFILE("ProbBoard::maxloc()", ",", TAU_DEFAULT);  
0186   float max=0;  
0187   loc_t loc=Board::PASS;  
0188  
0189   for(loc_t j=0; j<actualSize; j++) {  
0190     if (internal_board[j] > max) {  
0191       max = internal_board[j];  
0192       loc = j;  
0193     }  
0194   }  
0195 }  
0196 return loc;  
0197 }
void ProbBoard::normalize (void)  Normalizes the weights.
Normalization makes the sum of all weights equal to 1.

0080  {
0081     TAU_PROFILE("ProbBoard::normalize()", ",", TAU_DEFAULT);
0082
0083     float sum = 0.0;
0084
0085     // Find the sum
0086     for (int lcv = 0; lcv < actualSize; ++lcv) sum += internal_board[lcv];
0087
0088     // Convert board into percentage board (normalize)
0089     if (sum != static_cast<float>(0.0)) {
0090         for (int lcv = 0; lcv < actualSize; ++lcv)
0091             internal_board[lcv] = internal_board[lcv] / sum;
0092     }
0093 }

int ProbBoard::spin (void)  Chooses a random offset in probability board based on probabilities.

Precondition:
The sum of the probability board locations is very close to 1 or else the sum is zero.

Returns:
The chosen location

0208  {
0209     TAU_PROFILE("ProbBoard::spin()", ",", TAU_DEFAULT);
0210     float target;
0211     float current = 0.0;
0212     int j = 0;
0213
0214     // Pick a number between zero and one
0215     target = static_cast<float>(rand_r(&rdbuf)) / static_cast<float>(INT_MAX);
0216
0217     assert(target >= 0);
0218     assert(target <= 1.0);
0219
0220     // todo: Rewrite (simplify, it’s easy)
0221     // while (((current < target) && (j < actualSize)) ||
0222     //    ((internal_board[j] == 0) && (j < actualSize))) {
0223         while (j < actualSize) &
0224             (current < target) || ((j>0) && (internal_board[j-1] == 0))) {
0225             current += internal_board[j++];
0226         }

158
0227 j--; 
0228 0229 //if (j == actualSize) { --j; } 
0230 0231 assert(j < actualSize); 
0232 0233 return j; 
0234 }

A.2.28 RandomAgent Class Reference

Suggests random legal moves.

#include <agent.h>

Inheritance diagram for RandomAgent::

```
Agent
    RandomAgent
```

Public Methods

- **RandomAgent ()**
  
  *Constructor.*

- **~RandomAgent ()**

  *Destructor.*

- **void force (void)**

  *Force the Random agent to make its move.*

- **void update (Game *)**

  *Updates the game for the agent. Refresh agent with a new game state.*

- **bool dowork (void)**

  *work thread.*

- **void notify (void *)**

  *Tell the agent something.*
• unsigned int query_bits_needed_from_GA (void)
   
   Asks the agent how many bits it needs in the GA.

• void send_bits (chromosome*, chrom, int start)
   
   Sends to this agent the bits it needs from the GA.

A.2.28.1 Detailed Description

Suggests random legal moves.

A.2.28.2 Member Function Documentation

void RandomAgent::force (void) [virtual] Force the Random agent to
make its move.

This function is just here for completeness.

Reimplemented from Agent (p.74).

0054 {};

unsigned int RandomAgent::query_bits_needed_from_GA (void) [virtual] Asks the agent how many bits it needs in the
GA.

Returns:
   Number of bits needed in GA chromosome

Reimplemented from Agent (p.75).

0069 { return 7; }  

A.2.29 Stone Class Reference

Defines a point (stone) on the board.

#include <stone.h>
Public Methods

- **Stone ()**
  
  Constructor I.

- **Stone (int)**
  
  Constructor II.

- **Stone (const Stone &other)**
  
  Copy Constructor III.

- **bool white (void) const**
  
  Is stone white?

- **bool black (void) const**
  
  Is stone black?

- **bool empty (void) const**
  
  Is there a stone?

- **bool notempty (void) const**
  
  Is there no stone?

- **bool notblack (void) const**
  
  Is there a stone that isn’t black (empty or white)?

- **bool notwhite (void) const**
  
  Is there a stone that isn’t white (empty or black)?

- **bool notleft (void) const**
  
  Not the leftmost column.

- **bool notright (void) const**
  
  Not the rightmost column.

- **bool nottop (void) const**
  
  Not the topmost column.

- **bool notbottom (void) const**
Not the bottommost column.

• int getrow (void) const
  Get row stone is in.

• int getcol (void) const
  Get column stone is in.

• int lastrow (void) const
  Is stone in last row.

• int lastcol (void) const
  Is stone in last column.

• color_t getcolor (void) const
  Gets the color of the stone.

• void setrow (stone_t)
  Set stone’s row.

• void setcol (stone_t)
  Set stone’s column.

• void setcolor (color_t)
  Set stone’s color.

• void setlastrow (void)
  Stone is in last row.

• void setlastcol (void)
  Stone is in last column.

• void clearlastrow (void)
  Stone is not in last row.

• void clearlastcol (void)
  Stone is not in last column.
• void clear (void)
  clear stone’s bits.

• char stoneOut (void)
  text board output.

• bool operator==(const Stone&) const
  Stones are the same color (or empty).

• bool operator!=(const Stone&) const
  Stones are not the same color (or empty).

• Stone operator= (Stone)
  Overload the assignment operator.

Private Attributes

• stone_t theStone
  Bit-map representing a stone.

Static Private Attributes

• const stone_t WHITE_BIT = 0x0001
  0000 0000 0000 0001.

• const stone_t BLACK_BIT = 0x0002
  0000 0000 0000 0010.

• const stone_t BW_BITS = 0x0003
  0000 0000 0000 0011.

• const stone_t ROW_BITS = 0x007C
  0000 0000 0111 1100.

• const stone_t COL_BITS = 0xF80
  0000 1111 1000 0000.
• const stone_t LROW_BIT = 0x1000
  0001 0000 0000 0000.

• const stone_t LCOL_BIT = 0x2000
  0010 0000 0000 0000.

Friends

• ostream& operator<< (ostream &strm, Stone &aStone)
  
  *Overload the printing operator.*

A.2.29.1 Detailed Description

Defines a point (stone) on the board.

A.2.29.2 Member Function Documentation

bool Stone::operator!=(const Stone & other) const  Stones are not the same color (or empty).

**Warning:**

The stones are compared via color only. The other bits are ignored

```
0163       {
0164           return ((other.theStone & BW_BITS) !=
0165               (this->theStone & BW_BITS)) ? true : false;
0166       }
```

bool Stone::operator==(const Stone & other) const  Stones are the same color (or empty).

**Warning:**

The stones are compared via color only. The other bits are ignored

```
0153       {
0154           return ((other.theStone & BW_BITS) ==
0155               (this->theStone & BW_BITS)) ? true : false;
0156       }
```

164
A.2.30 Subthread Class Reference

Defines a sub-thread.

#include <subthread.h>

Inheritance diagram for Subthread:

```
Subthread
  AgentShell  Interface  Subthread_test
    DummyGenerator
    GaTrainerInterface
    GenAlgoGenerator
    GoModemInterface
    GUIInterface
    IGS_Interface
    NeuralNetGenerator
    NNGS_Interface
    TextInterface
```

Public Methods

- **Subthread ()**
  
  *Constructor.*

- virtual ~**Subthread ()**
  
  *Destructor.*

- void **start** (void)
  
  *Starts the thread running.*

- void **kill** (void)
 Stops the thread from running (kills it).

- void **send_msg** (msg_t msg)
  queues a message for this thread.

- msg_t **get_msg_nb** (void)
  Gets a message and returns if not there.

- void **join** (void)
  Do a thread join on this thread.

Public Attributes

- pthread_t **childt**
  Processing thread.

Protected Methods

- void **inside_send_msg** (msg_t msg)
  Allows thread to send out a message.

- msg_t **inside_get_msg_nb** (void)
  Allows thread to get a message (non-blocking).

- msg_t **inside_get_msg_b** (void)
  Allows thread to get a message (blocking).

- virtual void **processing** (void)=0
  Main logic loop of the interface.

- void **tell_message** (int, char *)
  Used to log message types (transform from number to enum text).

166
Protected Attributes

- `pthread_cond_t` **block_cond**
  
  *Condition variable for blocking receiving of messages.*

- `pthread_mutex_t` **message_queue_mutex**
  
  *Mutex for both queues.*

- `queue<msg_t>` **tothreadq**
  
  *Sending and receiving queues.*

- `queue<msg_t>` **fromthreadq**
  
  *Sending and receiving queues.*

Friends

- `void* CALL_processing (void *tmp_obj)`
  
  *Calls processing thread.*

A.2.30.1 Detailed Description

Defines a sub-thread.

This class supplies a child class with the ability to run in the background as a separate thread.

**This is a virtual class, but for all classes that inherit**

from this class, one must be sure to call the `set_die_ptr()` function before calling the function `start()` (p. 165).

A.2.30.2 Constructor & Destructor Documentation

**Subthread::~Subthread () [virtual]** Destructor.

**Warning:**

This function does not destroy the subthread. This is because the user will nicely kill the subthread via a QUIT message.
0093 {
0094   pthread_mutex_destroy(&message_queue_mutex);
0095   pthread_cond_destroy(&block_cond);
0096 }

A.2.30.3 Member Function Documentation

void Subthread::join (void)  Do a thread join on this thread.

Joins this thread.
This function joins the thread that this class created before.

0147   {
0148       assert(childt != 0);
0149       if (pthread_join(childt, NULL)) {
0150           cout << "E- Error in joining child." << endl;
0151           exit(823);
0152       } else {
0153           pthread_mutex_lock(&log_mutex);
0154           thread_count--;
0155           pthread_mutex_unlock(&log_mutex);
0156           childt = static_cast<pthread_t>(0);
0157       }
0158   }

A.2.31 Subthread_test Class Reference

For debugging.

#include <subthread.h>

Inheritance diagram for Subthread_test:

```
Subthread
   |
   v
Subthread_test
```

Public Methods

- ~Subthread_test ()
  Destructor.

- void regression (void)
Regression.

Protected Methods

- void processing (void)
  
  Main logic loop of the interface.

A.2.31.1 Detailed Description

For debugging.

A.2.32 testCodex Class Reference

A testing fitness function provider.

```cpp
#include <gafunc.h>
```

Inheritance diagram for testCodex::

```
    PreCodex
     |
     V
testCodex
```

Public Methods

- testCodex ()
  
  Constructor.

- float get_fitness (const chromosome_t &chrom)
  
  Finds the fitness (objective) function value.

- void summary (Population *newpop)
  
  Outputs testing results after training is done.

A.2.32.1 Detailed Description

A testing fitness function provider.
A.2.32.2 Member Function Documentation

float testCodex::get_fitness (const chromosome_t & chrom) [virtual]
Finds the fitness (objective) function value.

Parameters:
  chrom A chromosome to decode and then find the fitness of

Returns:
  fitness value

Reimplemented from Precodex (p.155).

0052 {  
0053   float res;
0054  
0055   // Decode
0056   int sum = 0;
0057   for(int x=0; x<lchrom; ++x) {
0058       sum += static_cast<long int>(pow(2, x) *
0059           static_cast<long int>(chrom[x]));
0060   }
0061  
0062   // Calculate fitness
0063   res = static_cast<float>(sum) /
0064       static_cast<float>(pow(2, lchrom) - 1.0);
0065  
0066   res *= 10.0;
0067  
0068   assert(res <= 10.0);
0069  
0070   if (res < 0) res = 0;
0071  
0072   return res;
0073 }

A.2.33 TextInterface Class Reference

Text Interface (p.144).

#include <interface.h>

Inheritance diagram for TextInterface:
Public Methods

- **TextInterface ()**
  
  *Constructor.*

Private Methods

- **loc_t get_user_input ()**
  
  *Gets user input.*

- **void processing (void)**
  
  *Main processing of interface.*

Private Attributes

- **string msg**
  
  *Text message before asking user for input.*

- **string prompt**
  
  *Prompt for user input.*

**A.2.33.1 Detailed Description**

Text Interface (p.144).

**A.2.33.2 Member Function Documentation**

**void TextInterface::processing (void) [private, virtual]**  
Main processing of interface.
This function, which the base class Interface (p. 144) defines as a pure virtual function, provides the main bulk of the logic of the interface.

Warning:

The interface may have outside signals that need to be seen. Thus, the function should finish (return) periodically. It will then be recalled as it is inside an infinite loop that checks for signals then calls this function again.

Reimplemented from Subthread (p. 166).

0067 {
0068   loc_t loc;
0069   Board tb;
0070   msg_t packetmsg;
0071
0072
0073   while (true) {
0074     if (my_turn) {
0075       packetmsg = inside_get_msg_nb();
0076     } else {
0077       // Nothing to do here but wait for a message (block).
0078       packetmsg = inside_get_msg_b();
0079     }
0080   }
0081
0082   if (packetmsg.id == QUIT) {
0083     LOG("-TIN " << setw(7) << childt
0084       << ", Text Interface exiting.");
0085     pthread_exit(0);
0086   } else if (packetmsg.id == RESIGN) {
0087     LOG("-TIN " << setw(7) << childt
0088       << ", Got RESIGN message.");
0089     pthread_exit(0);
0090   } else if (packetmsg.id == FORCE ) {
0091     LOG("-TIN " << setw(7) << childt
0092       << ", Forcing a move not allowed.");
0093   } else if (packetmsg.id == SET_GAME_PTR) {
0094     gptr = static_cast<Game*>(packetmsg.data);
0095   } else if (packetmsg.id == TURN) {
0096     my_turn = true;
0097   } else if (my_turn) {
0098       // Redisplay board (even if board hasn’t changed)
0100
0101       // Get a move and play it.
0102       loc = get_user_input();
0103       while (!gptr->legal(loc) && (loc != Board::PASS)) {
0104         cerr << ", That was an illegal move. " << endl;
0105       loc = get_user_input();
0106     }
0107
0108     }
0106    
0107    gp->play_move(loc);
0108
0109
0110    // Redisplay board
0111    tb = gp->get_board();
0112    cerr << tb << endl;
0113
0114    my_turn = false;
0115
0116
0117    // Tell moderator that I'm done.
0118    packetmsg.id = TURN;
0119    inside_send_msg(packetmsg);
0120
0121    }
0122    }
0123

A.2.34  TigersMouthAgent Class Reference

Tries to make tiger's mouths.

#include <agent.h>

Inheritance diagram for TigersMouthAgent::

```
Agent

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TigersMouthAgent</td>
</tr>
</tbody>
</table>
```

Public Methods

- **TigersMouthAgent ()**
  
  *Constructor.*

- **~TigersMouthAgent ()**
  
  *Destructor.*

- **void force (void)**
  
  *Force the agent to make its move.*

- **void update (Game *)**
Update this agent with the latest state of the game.

- bool **dowork** (void)
  Work thread.

- void **notify** (void *)
  Tell the agent something.

- unsigned int **query_bits_needed_from_GA** (void)
  Tells how many bits this agent needs from the GA.

- void **send_bits** (**chromosome_t** chrom, int start)
  Sends to this agent the bits it needs from the GA.

**Private Methods**

- bool **findtiger** (**loc_t** loc, **Board** &b)
  Find tiger’s mouths.

**A.2.34.1 Detailed Description**

Tries to make tiger’s mouths.

**A.2.34.2 Member Function Documentation**

**bool** **TigersMouthAgent::findtiger** (**loc_t** **loc**, **Board** & **b**) [private]
Find tiger’s mouths.

This function takes the current board, and a location on the board. It returns true if by playing at this location the player would make at least one tiger’s eye.

**Parameters:**
- **loc** Location to check
- **b** The board to consult

0117 {
0118     Stone *stones = b.get_goban();
0119
if ((stones[loc].notleft()) && (stones[loc-1].notleft()) &&
    (stones[loc].nottop())) {
    return (stones[loc-2].black() &&
            stones[loc-1*Board::BSIZE].black());
}

if ((stones[loc].notleft()) && (stones[loc].nottop()) &&
    (stones[loc-Board::BSIZE].nottop())) {
    return (stones[loc-1*Board::BSIZE].black() &&
            stones[loc-2*Board::BSIZE].black());
}

if (stones[loc].notright() && stones[loc].nottop() &&
    stones[loc-Board::BSIZE].nottop()) {
    return (stones[loc+1*Board::BSIZE].black() &&
            stones[loc+1-Board::BSIZE].black());
}

if (stones[loc].nottop() && stones[loc].notright() &&
    stones[loc+1].notright()) {
    return (stones[loc+2].black() &&
            stones[loc+1+Board::BSIZE].black());
}

if (stones[loc].notright() && stones[loc+1].notright() &&
    stones[loc].notbottom()) {
    return (stones[loc+2].black() &&
            stones[loc+1+Board::BSIZE].black());
}
0171 // . #
0172 // ? .
0173 // ?.
0174 if (stones[loc].notleft() && stones[loc].notbottom() &&
0175 stones[loc+Board::BSIZE].notbottom()) {
0176 return (stones[loc-1+Board::BSIZE].black() &&
0177 stones[loc+2*Board::BSIZE].black());
0178 }
0179
0180 // ? . #
0181 // . ?.
0182 if (stones[loc].notleft() && stones[loc-1].notleft() &&
0183 stones[loc].notright()) {
0184 return (stones[loc-2].black() &&
0185 stones[loc-1+Board::BSIZE].black());
0186 }
0187
0188 // ? . ?
0189 // . #.
0190 if (stones[loc].nottop() && stones[loc].notleft() &&
0191 stones[loc].notright()) {
0192 return (stones[loc-1-Board::BSIZE].black() &&
0193 stones[loc+1-Board::BSIZE].black());
0194 }
0195
0196 // . ?
0197 // # .
0198 // . ?
0199 if (stones[loc].notright() && stones[loc].nottop() &&
0200 stones[loc].notbottom()) {
0201 return (stones[loc+1+Board::BSIZE].black() &&
0202 stones[loc+1-Board::BSIZE].black());
0203 }
0204
0205 // . # .
0206 // ? . ?
0207 if (stones[loc].notbottom() && stones[loc].notright() &&
0208 stones[loc].notleft()) {
0209 return (stones[loc-1+Board::BSIZE].black() &&
0210 stones[loc+1+Board::BSIZE].black());
0211 }
0212
0213 // ? .
0214 // . #
0215 // ? .
0216 if (stones[loc].notleft() && stones[loc].nottop() &&
0217 stones[loc].notbottom()) {
0218 return (stones[loc-1+Board::BSIZE].black() &&
0219 stones[loc-1-Board::BSIZE].black());
0220 }
0221
A.3 Exodus File Documentation

A.3.1 agent.cpp File Reference

Implementation of Agent (p. 73) and AgentShell (p. 76) classes.

Defines

- \#define LOG(x)
  
  *Macro for outputing to the log file.*

Variables

- char rcsid [ ]
  
  *Source code identifier.*

A.3.1.1 Detailed Description

Implementation of Agent (p. 73) and AgentShell (p. 76) classes.

Revision:

1.21

Date:

2003/04/23 21:42:59

Author:

Todd Blackman

A.3.1.2 Variable Documentation

char rcsid [static] Initial value:


Source code identifier.

A.3.2 agent.h File Reference

Header file for Agent (p.73) related classes.

Compounds

- class Agent
  
  Defines the basic structure of an agent.

- class AgentShell
  
  Represents a single thread in a thread pool.

- class ExtenderAgent
  
  Suggests moves that extend from friendly stones.

- class FollowerAgent
  
  Suggests moves near opponent’s last move.

- class GroupStatsAgent
  
  Agent (p.73) to calculate group information.

- class OpenerAgent
  
  Suggests good opening moves.

- class RandomAgent
  
  Suggests random legal moves.

- class TigersMouthAgent
  
  Tries to make tiger’s mouths.
Defines

- `#define MAX_AGENTS 5`
  
  Maximum number of agents allowed.

- `#define MAXSTONE 10`
  
  Highest logical value for probability board element.

- `#define MAXGROUPS 50`
  
  Maximum number of distinct groups.

A.3.2.1 Detailed Description

Header file for Agent (p. 73) related classes.

This file contains headers for the Agent (p. 73) class, the AgentShell (p. 76) class, and all of the individual agents.

Revision:

1.22

Date:

2003/04/30 01:57:59

Author:

Todd Blackman


A.3.2.2 Define Documentation

`#define MAXSTONE 10` Highest logical value for probability board element.

All probability boards generated by all agents shall output a value of MAXSTONE for highly suggested values and 0 for unsuggested values. No agent shall make an element of a probability board larger than this value.

A.3.3 bdemo.cpp File Reference

Prints a demo board for numerical reference.
Functions

- int main (int argc, char *argv[])
  
  Main function for printing demo boards.

A.3.3.1 Detailed Description

Prints a demo board for numerical reference.

Revision:
1.5

Date:
2003/04/23 21:42:59

Author:
Todd Blackman


A.3.3.2 Function Documentation

int main (int argc, char * argv[])  Main function for printing demo boards.

Demo boards are just ASCII representations of the goban that has at each
location of the board the number representing the offset into a single dimension
array. For example, a 9x9 board’s leftmost value for the second row from the top
is ”8.” The third row from the top would be ”18.”

0027
0028   if (argc == 2) {
0029     int val = atoi(argv[1]);
0030     if (val < 1) {
0031       cout << "Invalid board size." << endl;
0032     } else {
0033       print_demo(val);
0034     }
0035   } else {
0036     cout << "Please give board size as single parameter."
0037     << endl;
0038   }
0039   return 0;
0040 }
A.3.4 blackboard.cpp File Reference

Implementation of Blackboard (p. 77) class.

Defines

- `#define LOG(x)
  
  Macro for outputing to log file.

Variables

- `char rcsid []
  
  Source code identifier.

A.3.4.1 Detailed Description

Implementation of Blackboard (p. 77) class.

Revision:

1.9

Date:

2003/04/23 21:42:59

Author:

Todd Blackman


A.3.4.2 Variable Documentation

char rcsid [static] Initial value:

"$Id: blackboard.cpp,v 1.9 2003/04/23 21:42:59 blackman Exp $"

Source code identifier.
A.3.5 blackboard.h File Reference

Header file for the Blackboard (p. 77) class.

Compounds

- class Blackboard
  
  *This class contains globally relevant information.*

A.3.5.1 Detailed Description

Header file for the Blackboard (p. 77) class.

Revision:

1.9

Date:

2003/04/30 01:57:59

Author:

Todd Blackman


A.3.6 board.cpp File Reference

Implementation for Board (p. 78) class.

Defines

- #define LOG(x)

  *Macro for outputting to log file.*

Functions

- ostream& operator<< (ostream &strm, Board &aBoard)

  *Output operator.*
Variables


Source code identifier.

A.3.6.1 Detailed Description

Implementation for Board (p.78) class.

Revision:
1.22

Date:
2003/04/23 21:42:59

Author:
Todd Blackman


A.3.7 board.h File Reference

Header file for board class.

Compounds

- class Board

  Defines a goban abstraction.

Typedefs

- typedef usi_t loc_t

  Offset into board 1D array.
Functions

- ostream& operator<<(ostream &strm, Board &aBoard)
  Output operator.

A.3.7.1 Detailed Description

Header file for board class.

Revision:
1.14

Date:
2003/04/30 01:57:59

Author:
Todd Blackman


A.3.8 config.h File Reference

System configuration definitions.

Defines

- #define STDC_HEADERS 1
- #define HAVE_FCNTL_H 1
- #define HAVE_MALLOC_H 1
- #define MAX_THREADS 10
  Maximum number of threads that can be active at the same time.

- #define MAX_AGENT_THREADS 5
  Maximum number of threads in the thread pool. This value only includes threads used by the agents and does not include other supplemental threads.

- #define AGENTS_USED {"RandomAgent", "end"}
  Agents used in this compilation of the program.
• #define PERLTK 1
  This tells that the Perl/Tk GUI will be used.

• #define NDEBUG 1
  This deactivates assertions.

A.3.8.1 Detailed Description
System configuration definitions.

Warning:
  Automatically generated by ../configure script

A.3.8.2 Define Documentation

#define HAVE_FCNTL_H 1 Define if you have the <fcntl.h> header file.

#define HAVE_MALLOC_H 1 Define if you have the <malloc.h> header file.

#define MAX_THREADS 10 Maximum number of threads that can be active at the same time.
  This value includes all threads running.

#define STDC_HEADERS 1 Define if you have the ANSI C header files.

A.3.9 dummygenerator.cpp File Reference
Implementation of random move generator called DummyGenerator (p.88).

Defines

• #define LOG(x)
  Macro for outputing to log file.
Variables

- char rcsid []
  
  Source code identifier.

A.3.9.1 Detailed Description

Implementation of random move generator called DummyGenerator (p. 88).

Revision:
1.11

Date:
2003/04/23 21:42:59

Author:
Todd Blackman


A.3.9.2 Variable Documentation

char rcsid [static] Initial value:


Source code identifier.

A.3.10 exodus.h File Reference

Global constants declarations.

Compounds

- struct global_data_t
  
  global data structure.
Defines

- `#define NUM_THREADS 5`
  
  Number of threads in the thread pool.

- `#define VERSION "R016.000B"
  
  Program version.

- `#define LOG(x)
  
  Log file macro.`

Typedefs

- `typedef unsigned short int usi_t
  
  unsigned short int.`

Enumerations

- `enum color_t { EMPTY, WHITE, BLACK }`
  
  Stone (p.160) color type.

Variables

- `global_data_t global_data
  
  Holds the minimal amount of global data required for this program.

- ofstream log_cout
  
  Output stream for log file.

- `pthread_mutex_t log_mutex
  
  MUTEX for writing to the log file.

- int thread_count
  
  Number of threads currently open.`
A.3.10.1 Detailed Description

Global constants declarations.

Conventions used in this project:

1. Function braces are all in far-left
2. Loop start brace is on same line; end brace is far-left
3. Classes start with a capital letter
4. Classes with name inside have each new word in caps
5. vars use underscores
6. types end in _t
7. globals in all caps ??? maybe end in _g
8. defines in all caps
9. abbreviations:
   - tmp......temporary
   - ptr......pointer
   - func......function
10. Functions start with lowercase letter (unless constructor, etc.)
11. Each new word in a function is delimited with underscores

Revision:
   1.27

Date:
   2003/04/30 01:57:59

Todo:
   Agents need to be able to communicate for complex situations.

Author:
   Todd Blackman

A.3.10.2 Typedef Documentation

typedef unsigned short int usi_t  unsigned short int.

Used so much, this makes code neater

A.3.11 extenderagent.cpp File Reference

Implementation of an ExtenderAgent (p. 89) that attempts to extend from friendly stones.

Defines

- #define LOG(x)
  
  *Macro for outputing to log file.*

Variables

- char rcsid []

  *Source code identifier.*

A.3.11.1 Detailed Description

Implementation of an ExtenderAgent (p. 89) that attempts to extend from friendly stones.

Revision:

1.6

Date:

2003/04/30 01:54:48

Author:

Todd Blackman

A.3.11.2 Variable Documentation

char rcsid [static] Initial value:

"$Id: extenderagent.cpp,v 1.6 2003/04/30 01:54:48 blackman Exp $"

Source code identifier.

A.3.12 followeragent.cpp File Reference

Implementation of FollowerAgent (p. 92) which plays moves close to opponent.

Defines

- `#define LOG(x)`
  
  _Macro for outputing to log file._

Variables

- char rcsid []
  
  _Source code identifier._

A.3.12.1 Detailed Description

Implementation of FollowerAgent (p. 92) which plays moves close to opponent.

Revision:

1.8

Date:

2003/04/23 21:42:59

Author:

Todd Blackman

A.3.12.2 Variable Documentation

char rcsid [static]  Initial value:

"$Id: followeragent.cpp,v 1.8 2003/04/23 21:42:59 blackman Exp $"

Source code identifier.

A.3.13 ga.cpp File Reference

Implementation for Ga (p.96) class.

Defines

- \#define LOG(x)

  *Macro for outputing to log file.*

Functions

- ostream& operator<< (ostream &strm, const Population &pop)
  
  Population (p.153) *output operator.*

- bool strchrom (chromosome_t chrom, string chrom2)

  *Simple comparison function.*

Variables

- char *rcsid [] = "$Id: ga.cpp,v 1.28 2003/04/30 01:54:48 blackman Exp $"

  *Source code identifier.*

A.3.13.1 Detailed Description

Implementation for Ga (p.96) class.

Revision:

1.28
Date:
2003/04/30 01:54:48

WORKS CITED:


Author:
Todd Blackman


A.3.13.2 Function Documentation

bool strchrom (chromosome_t chrom, string chrom2) [static] Simple comparison function.

This function allows the programmer to compare a string representation of a chromosome with the datastructure representation.

1049 {
1050     unsigned int x;
1051
1052     //cout << endl;
1053     //cout << chrom << endl;
1054     //cout << chrom2.length() << endl;
1055     for (x=0; x<chrom2.length(); x++) {
1056         bool b;
1057         b = (chrom2[x] == '1') ? true : false;
1058         //cout << b << flush;
1059         if (chrom[x] != b) break;
1060     }
1061     //cout << endl;
1062     if (chrom2.length() == x) return true;
1063     return false;
1064 }

A.3.14 ga.h File Reference

Header file for genetic algorithm related classes.
Compounds

- class Ga
  
  Defines a Genetic Algorithm.

- struct Individual
  
  An individual in a population of a GA.

- struct Population
  
  A single population within a GA.

Defines

- #define MAXPOP 10000
  
  Maximum size of population.

A.3.14.1 Detailed Description

Header file for genetic algorithm related classes.

This file also includes some statistics functions and definitions for Ga (p.96), Population (p.153), and Individual (p.142)

Revision:
1.16

Date:
2003/04/30 01:57:59

Todo:

- Get rid of vectors and replace with arrays

Author:

- Todd Blackman


A.3.15 gafunc.h File Reference

Header file for GA testing and aux. functions.
Compounds

- class **PreCodex**
  
  *Allows other classes to provide a fitness function.*

- class **testCodex**
  
  *A testing fitness function provider.*

A.3.15.1 Detailed Description

Header file for GA testing and aux. functions.

This file provides **PreCodex** (p. 154) and **testCodex** (p. 169)

**Revision:**

1.12

**Date:**

2003/04/30 01:57:59

**Author:**

Todd Blackman


A.3.16 game.cpp File Reference

Implementation of the **Game** (p. 112) class.

**Defines**

- `#define LOG(x)`
  
  *Macro for outputing to log file.*

**Functions**

- `ostream& operator<< (ostream &strm, Game &aGame)`
  
  *Stream operator.*
Variables

- char rcsid []

Source code identifier.

A.3.16.1 Detailed Description

Implementation of the Game (p.112) class.

Revision:
1.28

Date:
2003/04/23 21:42:59

Bug:
super-ko does not take rotation and symmetry into account.

Author:
Todd Blackman


A.3.16.2 Function Documentation

ostream & operator<< (ostream & strm, Game & aGame) Stream operator.
This is used to output the latest state of the game

0412       {
0413       Board b = aGame.get_board();
0414       strm << b;
0415       return strm;
0416     }

A.3.16.3 Variable Documentation

char rcsid [static] Initial value:


Source code identifier.
A.3.17 game.h File Reference

Header file for game class.

Compounds

- class Game
  *A class that defines a series of boards.*

Defines

- #define SUICIDE_CHECK 1  
  *Enable checking for suicide.*

- #define SUPERKO_CHECK 0  
  *Enable checking for superko.*

Typedefs

- typedef pair<usi_t, usi_t> usi_p
  *Type to make code simpler.*

Functions

- ostream& operator<< (ostream &strm, Game &aGame)  
  *Stream operator.*

A.3.17.1 Detailed Description

Header file for game class.

Revision:
  1.21

Date:
  2003/04/30 01:57:59
Author:
   Todd Blackman


A.3.17.2 Function Documentation

ostream& operator<< (ostream & strm, Game & aGame) Stream operator.
   This is used to output the latest state of the game

   0412       {
   0413       Board b = aGame.get_board();
   0414       strm << b;
   0415       return strm;
   0416     }

A.3.18 gatypes.h File Reference

Header file for genetic algorithm types and defaults.

Typedefs

- typedef bool allele

  Allele type.

- typedef vector<allele> chromosome_t

  Chromosome type.

Functions

- ostream& operator<< (ostream & strm, const chromosome_t &chrom)

  Chromosome output operator.

A.3.18.1 Detailed Description

Header file for genetic algorithm types and defaults.
A.3.19 genalgogenerator.cpp File Reference

A genetic algorithm player using agents.

Defines

- `#define LOG(x)
  
  *Macro for outputing to log file.*

Variables

- `char rcsid []`
  
  *Source code identifier.*

A.3.19.1 Detailed Description

A genetic algorithm player using agents.
A.3.19.2 Variable Documentation

char rcsid [static] Initial value:

"$Id: genalgogenerator.cpp,v 1.31 2003/04/30 01:54:48 blackman Exp "$

Source code identifier.

A.3.20 ginterface.cpp File Reference

Implementation of a GUI interface.

Defines

- \#define LOG(x)  
  
  *Macro for outputing to log file.*

Variables

  
  *Source code identifier.*

A.3.20.1 Detailed Description

Implementation of a GUI interface.

This file provides **GUIInterface** (p. 138)

**Revision:**

1.24

**Date:**

2003/04/23 21:42:59

**Author:**

Todd Blackman

A.3.21  groupstatsagent.cpp File Reference

Provides an agent to calculate group information.

Defines

- #define LOG(x)

  *Macro for outputing to log file.*

Variables

- char rcsid []

  *Source code identifier.*

A.3.21.1 Detailed Description

Provides an agent to calculate group information.

This agent (GroupStatsAgent (p.136)) calculates a unique number for each
group on the board. Planned for this agent are the following tasks:

*) Assign a unique number to each group

*) Count the liberties of each group

*) Calculate a safety value

This agent also acts as a "killer" agent that tries to capture enemy stones

Todo:

Make this more memoized

Revision:

1.7

Date:

2003/04/23 21:42:59

Author:

Todd Blackman

A.3.21.2 Variable Documentation

char rcsid [static] Initial value:


Source code identifier.

A.3.22 iinterface.cpp File Reference

Implementation of IGS interface class (IGS_Interface (p.141)).

Defines

- \#define LOG(x)
  
  *Macro for outputing to log file.*

Variables

  
  *Source code identifier.*

A.3.22.1 Detailed Description

Implementation of IGS interface class (IGS_Interface (p.141)).

Revision:

1.11

Date:

2003/04/23 21:42:59

Author:

Todd Blackman

A.3.23 interface.cpp File Reference

Implementation for abstract Interface (p.144) classes.

Defines

- `#define LOG(x)`

  *Macro for outputing to log file.*

A.3.23.1 Detailed Description

Implementation for abstract Interface (p.144) classes.

This file also includes all code for all abstract classes below Interface (p.144), but above an actual interface or generator.

Revision:
1.16

Date:
2003/04/23 21:42:59

Author:
Todd Blackman


A.3.24 interface.h File Reference

Header file for interfaces.

Compounds

- class `GaTrainerInterface`

  *Used to train a GA to work correctly.*

- class `GoModemInterface`

  *Go modem interface.*
• class **GUIInterface**
  *Graphical User Interface* (p.144).

• class **IGS_Interface**
  *Internet Go Server (IGS) Interface* (p.144).

• class **Interface**
  *The interface between a move generator (outside) and the inside of the program.*

• class **NGGS_Interface**
  *No Name Go Server Interface* (p.144).

• class **TextInterface**
  *Text Interface* (p.144).

### A.3.24.1 Detailed Description

Header file for interfaces.

This file provides interfaces between the outside world and the program. The first virtual class, **Interface** (p.144), defines the functionality of the various interfaces that the program will have to be aware of. View this first class as a two-way pipe. All interfaces have a public method called `\texttt{start}` that spawns a thread and then returns. Other Interfaces can be found in outputgen.cpp

**Revision:**

1.20

**Date:**

2003/04/30 01:57:59

**Author:**

Todd Blackman


### A.3.25 main.cpp File Reference

Main, cmd-line, init-file functions.
Defines

- `#define LOG(x)`
  
  *Macro for outputing to log file.*

Functions

- `int main (int argc, char *argv[])`
  
  *Main function.*

- `void start_training (void)`
  
  *Start a GA run.*

- `void start_play ()`
  
  *Starts a game between GA and a human.*

- `void initialize ()`
  
  *Runs regressions Initializes some aspects.*

- `void parse_cmd_line_options (int argc, char **argv)`
  
  *Reads command-line parameters.*

- `void parse_rc_file ()`
  
  *parses .exodusrc file.*

- `void assign_global (const char *var, const char *val)`
  
  *Assign global variable values.*

- `void print_welcome (void)`
  
  *Prints a welcome message.*

- `void print_help (void)`
  
  *Prints a help message.*
Variables

- char rcsid []
  \(\textit{Source code identifier.}\)

- \texttt{global\_data\_t global\_data}
  \(\textit{Holds the minimal amount of global data required for this program.}\)

- int thread\_count = 1
  \(\textit{Number of threads currently open.}\)

- \texttt{pthread\_mutex\_t log\_mutex}
  \(\textit{MUTEX for writing to the log file.}\)

- ofstream log\_cout
  \(\textit{Output stream for log file.}\)

A.3.25.1 Detailed Description

Main, cmd-line, init-file functions.

Main program, command-line parsing, init-file parsing

Revision:

1.43

Date:

2003/04/23 21:42:59

Author:

Todd Blackman


A.3.25.2 Function Documentation

\texttt{void assign\_global (const char \* var, const char \* val)} Assign global variable values.

Assign the value \texttt{val} to the global variable \texttt{var} with error checking
Parameters:

*var* What global variable to assign a value to

*val* The value to assign to the global variable specified in the other parameter

Precondition:

var and val point to valid strings.

```cpp
0486     
0487     }  
0488     //cout << "-D- Doing " << var << " with " << val << endl;  
0489     //General parameters
0490     if (!strcmp(var, "welcome")) {
0491         global_data.welcome = atoi(val);  
0492     } else if (!strcmp(var, "train")) {
0493         global_data.train = atoi(val);  
0494     } else if (!strcmp(var, "gui")) {
0495         GUIInterface::GPATH = val;  
0496     } else if (!strcmp(var, "verbosity")) {
0497         global_data.verbosity = atoi(val);  
0498         if ((global_data.verbosity < 0) ||
0499             (global_data.verbosity > 10)) {
0500             global_data.verbosity=0;  
0501             cout << "-W- Verbosity must be between 0 and 10 inclusive. "
0502             << "Set to 0." << endl;  
0503         }
0504     } else if (!strcmp(var, "version")) {
0505         global_data.version = atoi(val);  
0506     } else if (!strcmp(var, "help")) {
0507         global_data.help = atoi(val);  
0508     } else if (!strcmp(var, "thread_count")) {
0509         global_data.thread_count = atoi(val);  
0510     } else if (!strcmp(var, "MAX_THREADS")) {
0511         global_data.thread_count=1;  
0512     } else if (!strcmp(var, "reg_on")) {
0513         global_data.reg_on = static_cast<bool>(atoi(val));  
0514     } else if (!strcmp(var, "bsize")) {
0515         Board::BSIZE = atoi(val);  
0516     } else {
0517     }
0518     // Go specific parameters
0519     } else if (!strcmp(var, "bsize")) {
0520         Board::BSIZE = atoi(val);  
0521     } else if ((Board::BSIZE < 3) || (Board::BSIZE > 19)) {
0522         Board::BSIZE=19;  
0523     cout << "-W- bsize must be between 3 and 19"  
0524     << " inclusive. Set to 19." << endl;
```
}  
#else if (!strcmp(var, "super_ko")) {
  Game::SUPER_KO = static_cast<bool>(atoi(val));
#endif
#else if (!strcmp(var, "komi")) {
  Game::KOMI = atof(val);
  if ((Game::KOMI < -15.0) || (Game::KOMI > 15.0)) {
    cout << "-W- KOMI value is strange: " << Game::KOMI << endl;
  }
#endif
#else if (!strcmp(var, "initial_time")) {
 .Game::INITIAL_TIME = atof(val);
#endif
#else if (!strcmp(var, "byomi_time")) {
  Game::BYOMI_TIME = atof(val);
#endif
#else if (!strcmp(var, "byomi_stones")) {
  Game::BYOMI_STONES = atof(val);
#endif
#else if (!strcmp(var, "suicide")) {
  Game::SUICIDE = static_cast<bool>(atoi(val));
#endif
#else if (!strcmp(var, "my_color")) {
#endif
#else if (!strcmp(var, "num_handicap")) {
  Board::HANDICAP = atof(val);
  if (Board::HANDICAP > 9) cout << "-W- Handicap is strange: "
    << Board::HANDICAP << endl;
#endif
#else if (!strcmp(var, "handicap_placement")) {
  cout << "-W- handicap placement not implemented yet." << endl;
#endif
  // GA parameters
#endif
#else if (!strcmp(var, "resume")) {
  // Resume implies train
  global_data.train=1;
#endif
#else if (!strcmp(var, "train_file")) {
  Ga::TRAIN_FILE = val;
#endif
#else if (!strcmp(var, "output")) {
  Ga::FILENAME_OUT = val;
#endif
#else if (!strcmp(var, "best")) {
  Ga::BEST_FILENAME_OUT = val;
#endif
#else if (!strcmp(var, "popsize")) {
  Ga::POPSIZE = atof(val);
#endif
#else if (!strcmp(var, "maxgen")) {
  Ga::MAXGEN = atof(val);
#endif
#else if (!strcmp(var, "fitness_cutoff")) {
  Ga::FITNESS_CUTOFF = atof(val);
#endif
#else if (!strcmp(var, "pcross")) {
  Ga::PCROSS = atof(val);
#endif
#else if (!strcmp(var, "pmutation")) {
  Ga::PMUTATION = atof(val);
#endif
#else if (!strcmp(var, "fmutation")) {
  Ga::FMUTATION = atof(val);
#endif
#else {
  cout << "-W- Invalid parameter: " << var << endl;
}
}
void parse_rc_file () parses .exodusrc file.

Looks for a .exodusrc file in the current directory. It parses the file if it is found.

Precondition:

None, but .exodusrc must be in current directory for it to be found.

Postcondition:

variables/parameters specified in rc file have been communicated to this pro-
gram and are stored in global variables. All paths equal either the null string
or a valid path to a valid file when finished.

Returns:

none

0382  
0383  void assign_global(const char *, const char *);
0384  
0385  char line[256];
0386  char var[100], val[100];
0387  char *ptr;
0388  
0389  //give global variables initial values.
0390  global_data.welcome=1;
0391  global_data.train=1;
0392  
0393  GUIInterface::GPATH = "";
0394  
0395  global_data.verbosity=0;
0396  global_data.version=0;
0397  global_data.help=0;
0398  //global_data.thread_count=1;
0399  global_data.reg_on=0;
0400  
0401  Board::BSIZE=9;
0402  
0403  Game::SUPER_KO = false;
0404  Game::KOMI = 0.5;
0405  Game::SUICIDE = false;
0406  
0407  Game::INITIAL_TIME=300;
0408  Game::BYOMI_TIME=300;
0409  Game::BYOMI_STONES=10;
0410  
0411  // GA related
0412  Ga::MAXGEN = 10;
0413  Ga::POPSIZE = 20;
0414  Ga::FITNESS_CUTOFF = 1.0;
0415  Ga::PCROSS = 0.4;
Ga::PMUTATION = 0.0333;
Ga::FMULTIPLE = 2.0;
Ga::FILENAME_IN = "";
Ga::FILENAME_OUT = "ga.save";
Ga::BEST_FILENAME_OUT = "ga.best.save";
Ga::TRAIN_FILE = "ga.train";

//Construct string containing path to rc file (which might not be there).
//current_dir = get_current_dir_name();
//rcfile = (char *) malloc((sizeof current_dir) + 15);
//sprintf(rcfile, "%s/.exodusrc", current_dir);
//cout << rcfile << endl;

//Attempt to open the rc file and parse it.
ifstream fin(".exodusrc");
if (fin) {
  while ( !fin.eof() ) {
    fin.getline(line, 255);
    filter_whitespace(line);
    if ((line[0] != '#') && (strlen(line) != 0)) {
      //cout << line << endl;
      ptr = strpbrk(line, "=");
      if ((ptr != NULL) && (strlen(ptr) >= 2)) {
        ptr++;
        if (*ptr == ' ') ptr++;
        strcpy(val, ptr);
      } else {
        //default to true
        strcpy(val, "1");
      }

      //Construct var value from line
      ptr = strpbrk(line, "=");
      if (ptr != NULL) {
        *ptr = '\0';
        if (*ptr - 1 == ' ') *(ptr - 1) = '\0';
      }
      strcpy(var, line);
      //cout << var << " = " << val << endl;
      assign_global(var, val);
    }
  }
}
void print_welcome (void)  Prints a welcome message.

Author:
Anonymous Web program (large text)

A.3.25.3  Variable Documentation
char rcsid [static]  Initial value:


Source code identifier.

A.3.26  moderator.t File Reference

Implementation and definition of Moderator (p.146) template.
Compounds

- class Moderator
  
  Encapsulates two interfaces and has them play together.

Defines

- #define LOG(x)
  
  Macro for outputing to log file.

Variables

- char rcsidm [] = "$Id: moderator.t,v 1.20 2003/04/23 21:42:59 blackman
  Exp $"
  
  Source code identifier.

A.3.26.1 Detailed Description

Implementation and definition of Moderator (p. 146) template.

Revision:

1.20

Date:

2003/04/23 21:42:59


Author:

Todd Blackman

A.3.27 move.cpp File Reference

Implementation of the move.t (p. 148) stuct.
Defines

- \#define LOG(x)  
  \textit{Macro for outputing to log file.}

Variables

- char rcsid []  
  \textit{Source code identifier.}

\subsection{A.3.27.1 Detailed Description}

Implementation of the move.t (p.148) struct.

\subsubsection{Revision}

1.9

\subsubsection{Date}

2003/04/23 21:42:59

\subsubsection{Author}

Todd Blackman


\subsection{A.3.27.2 Variable Documentation}

\texttt{char rcsid} \texttt{[static]} \texttt{Initial value:}


Source code identifier.

\subsection{A.3.28 move.h File Reference}

Describes a Move struct.
Compounds

- struct move_t
  
  A single move on the goban.

A.3.28.1 Detailed Description

Describes a Move struct.

Revision:

1.4

Date:

2003/04/30 01:57:59

Author:

Todd Blackman


A.3.29 openeragent.cpp File Reference

Opening move agent.

Defines

- #define LOG(x)
  
  Macro for outputing to log file.

Variables

- char rcsid []
  
  Source code identifier.
A.3.29.1 Detailed Description

Opening move agent.

Revision:
1.16

Date:
2003/04/23 21:42:59

Author:
Todd Blackman


A.3.29.2 Variable Documentation

char rcsid [static] Initial value:


Source code identifier.

A.3.30 outputgen.h File Reference

Header file for GenAlgoGenerator (p.126), NeuralNetGenerator (p.150), and DummyGenerator (p.88) classes.

Compounds

- class DummyGenerator
  
  A dummy move generator that generates random legal moves.

- class GenAlgoGenerator
  
  A genetic algorithm move generator.

- class NeuralNetGenerator
  
  A Neural Network move generator.
Defines

- `#define BITSPERWEIGHT 4`
  
  *Each network weight is an integer represented in this number of bits.*

- `#define SECONDLEVELNODES 3`
  
  *Number of nodes at the second level of the network.*

**A.3.30.1 Detailed Description**

Header file for `GenAlgoGenerator` (p.126), `NeuralNetGenerator` (p.150), and `DummyGenerator` (p.88) classes.

Output generators take the agents and a blackboard and use them to generate the next move. Essentially, this class takes a move as input and outputs a move.

**Revision:**

1.19

**Date:**

2003/04/30 01:57:59

**Author:**

Todd Blackman


**A.3.31 probboard.cpp File Reference**

The implementation for the probability board.

Defines

- `#define LOG(x)`

  *Macro for outputting to log file.*
Functions

- `ostream& operator<<(ostream &strm, ProbBoard &aBoard)
  
  Stream operator.

Variables

  
  Source code identifier.

A.3.31.1 Detailed Description

The implementation for the probability board.

This file provides the `ProbBoard` (p. 155) class.

Revision:
1.22

Date:
2003/04/23 21:42:59

Author:
Todd Blackman


A.3.32 probboard.h File Reference

Probability matrix for an agent’s next move.

Compounds

- class `ProbBoard`
  
  `Agent` (p. 73)’s probability output board.
Defines

- \#define SPIN 0
  
  Spin or choose first highest location on probboard.

A.3.32.1 Detailed Description

Probability matrix for an agent’s next move.

Revision:
1.15

Date:
2003/04/30 01:57:59

Author:
Todd Blackman


A.3.32.2 Define Documentation

\#define SPIN 0 Spin or choose first highest location on probboard.

- If 1, then spin a wheel for choosing location on the board.
  - If 0, just choose the last location with the maximum value.

A.3.33 randomagent.cpp File Reference

Random agent implementation.

Defines

- \#define LOG(x)
  
  Macro for outputing to log file.
Variables

- char rcsid [] = "$Id: randomagent.cpp,v 1.5 2003/04/23 21:42:59 blackman
  Exp $"
  
  Source code identifier.

A.3.33.1 Detailed Description

Random agent implementation.

This file contains the implementation of the RandomAgent (p. 159) class

Revision:

1.5

Date:

2003/04/23 21:42:59

Author:

Todd Blackman


A.3.34 stone.cpp File Reference

Implementation of the Stone (p. 160) class.

Defines

- #define LOG(x)

  Macro for outputing to log file.

Functions

- ostream& operator<<(ostream &strm, Stone &aStone)

  Overload the printing operator.
Variables

- char rcsid []
  
  Source code identifier.

A.3.34.1 Detailed Description

Implementation of the Stone (p. 160) class.

Revision:
1.11

Date:
2003/04/23 21:42:59

Author:
  Todd Blackman


A.3.34.2 Variable Documentation

char rcsid [static] Initial value:


Source code identifier.

A.3.35 stone.h File Reference

Header file for Stone (p. 160) class.

Compounds

- class Stone
  
  Defines a point (stone) on the board.
Defines

- \#define \texttt{INV(x)} (x==\texttt{BLACK}) ? \texttt{WHITE} : \texttt{BLACK} \\
  \textit{Macro for inverting the color.}

Typedefs

- typedef unsigned short int \texttt{stone.t} \\
  \texttt{Stone (p.160) bit-map type.}

Functions

- \texttt{ostream\& operator\ll (ostream \&\texttt{strm, Stone \&}\texttt{aStone})} \\
  \textit{Overload the printing operator.}

A.3.35.1 Detailed Description

Header file for \texttt{Stone (p.160) class.} \\
Anticipating that this code will be called quite a bit, the implementation in this file sacrifices clarity for efficiency. Still, it should be fairly strait forward. It’s just a collection of bit-masks.

Precondition:

unsigned short int is 2 bytes or more.

Warning:

Endianess of architecture may matter here.

Author:

Todd Blackman


A.3.36 subthread.cpp File Reference

Implementation for abstract class \texttt{Subthread (p.165).}
Defines

- #define LOG(x)
  
  *Macro for outputing to log file.*

Functions

- void* CALL_processing (void *tmp_obj)
  
  *Calls processing thread.*

Variables

  Exp $”
  
  *Source code identifier.*

A.3.36.1 Detailed Description

Implementation for abstract class Subthread (p. 165).

Revision:

1.14

Date:

2003/04/23 21:42:59

Author:

Todd Blackman


A.3.37 subthread.h File Reference

Defines virtual class for a running sub-thread.
Compounds

- struct msg_t
  
  A message to or from a thread.

- class Subthread
  
  Defines a sub-thread.

- class Subthread_test
  
  For debugging.

Enumerations

- enum msg_id_t { NOMSG, SET_GAME_PTR, RESIGN, FORCE, QUIT, ERROR, TURN, FINISHED, LOAD, UPDATE, NOTIFY, SET_BB_PTR, SET_PROB_PTR }

  Message types for thread communication.

A.3.37.1 Detailed Description

Defines virtual class for a running sub-thread.

Revision:

1.13

Date:

2003/04/30 01:57:59

Author:

Todd Blackman


A.3.37.2 Enumeration Type Documentation

enum msg_id_t  Message types for thread communication.

Enumeration values:

  NOMSG  No message.
**SET_GAME_PTR**  Data is a points to a game class.

**RESIGN**  An interface wishes to resign.

**FORCE**  Force interface to make a move ”soon”.

**QUIT**  quit program (usually only recvd by thread).

**ERROR**  Some error occurred.

**TURN**  Signals that it is the receiver’s turn. When sent by an interface, it signals that the turn is finished.

**FINISHED**  Agent (p. 73) finished its work.

**LOAD**  AgentShell (p. 76) is instructed to do work as the specified agent.

**UPDATE**  Board (p. 78) has changed. Take note of this.

**NOTIFY**  Tell agent that information is available at bb that it might need. (experimental feature).

**SET_BB_PTR**  Set blackboard repository area pointer.

**SET_PROB_PTR**  Set the result location for probability board.

0021    {
0022    NOMSG,
0023    SET_GAME_PTR,
0024    RESIGN,
0025    FORCE,
0026    QUIT,
0027    ERROR,
0028    TURN,
0029
0030
0031
0032    // Agent related
0033    FINISHED,
0034    LOAD,
0035
0036    UPDATE,
0037    NOTIFY,
0038
0039
0040    SET_BB_PTR,
0041    SET_PROB_PTR
0042 } msg_id_t;

**A.3.38  testcodex.cpp File Reference**

Stub code for fitness function for GAs.
Functions

- `ostream& operator<<(ostream &strm, const chromosome_t &chrom)`
  
  Output stream operator.

Variables

  
  Source code identifier.

A.3.38.1 Detailed Description

Stub code for fitness function for GAs.

This file provides `testCodex` (p. 169) and `PreCodex` (p. 154) classes

Revision:

1.8

Date:

2003/04/23 21:42:59

Author:

Todd Blackman


A.3.39 tigersmouthagent.cpp File Reference

Implementation of tiger’s mouth class.

Defines

- `#define LOG(x)`
  
  Macro for outputing to log file.
Variables

- char rcsid []
  
  Source code identifier.

A.3.39.1 Detailed Description

Implementation of tiger’s mouth class.
This file provides TigersMouthAgent (p. 173) which attempts to make a
tiger’s mouth which is a good formation in go.

Revision:
1.3

Date:
2003/04/23 21:42:59

Author:
Todd Blackman


A.3.39.2 Variable Documentation

char rcsid [static] Initial value:

"$Id: tigersmouthagent.cpp,v 1.3 2003/04/23 21:42:59 blackman Exp $"

Source code identifier.

A.3.40 tinterface.cpp File Reference

Implementation of text interface.

Defines

- #define LOG(x)
  
  Macro for outputing to log file.
Variables

  
  Source code identifier.

A.3.40.1 Detailed Description

Implementation of text interface.

This file provides the class TextInterface (p. 170).

Revision:

1.25

Date:

2003/04/23 21:42:59

Precondition:

The class has never been instantiated

Warning:

Only two interface classes may be instantiated at a time.

Author:

Todd Blackman


A.3.41 tools.cpp File Reference

Utilities.

Functions

- float gammln (float xx)
  
  Computes gamma function.

- float betacf (float a, float b, float x)
Evaluates continued fraction for incomplete beta function by modified Lentz's method.

• float \texttt{betai} (float \(a\), float \(b\), float \(x\))
  \textit{Computes the incomplete beta function} \(I_x(a,b)\).

• void \texttt{filter_whitespace} (char line[\])
  \textit{Removes whitespace.}

• bool \texttt{odd} (int \(n\))
  \textit{Finds if the number is odd.}

• void \texttt{print_demo} (int size)
  \textit{Prints a helpful diagram.}

• int \texttt{pipe_getline} (int \(fd\), string &\texttt{buf}, char endchar=’\n’)
  \textit{Read from file descriptor to endchar.}

• void \texttt{loopy} (int whichone=0, bool done=false, char *\texttt{msg}=”...DONE”)
  \textit{Prints a “waiting” rotating character.}

Variables

• char \texttt{rcsid} [\] = ”$Id: tools.cpp,v 1.17 2003/04/30 01:54:48 blackman Exp$"
  \textit{Source code identifier.}

A.3.41.1 Detailed Description

Utilities.

Contains function to perform simple utility operations.

WORKS CITED:

\texttt{title=Numerical Recipes in C: The Art of Scientific Computing}
\texttt{publisher=The Press Syndicate of the University of Cambridge edition=Second year=1997}
\texttt{pages=227,616–619}

\$Revision$
A.3.41.2 Function Documentation

float betacf (float a, float b, float x) Evaluates continued fraction for incomplete beta function by modified Lentz’s method.

Author:
Numerical Recipes in C, page 227-8

```c
0066 {  
0067     int m,m2;  
0068     float aa,c,d,del,h,qab,qam,qap;  
0069     qa = = a + b;  
0070     qa = = a + 1.0;  
0071     qa = = a - 1.0;  
0072     c = = 1.0;  
0073     d = = 1.0 - qab*x/qap;  
0074     if (fabs(d) < FPMIN) d= FPMIN;  
0075     d = = 1.0/d;  
0076     h = = d;  
0077     for (m = = 1;m = = MAXIT;m++) {  
0079       m2 = = 2*m;  
0080       aa = = m*(b-*m)*x/((qam+m2)*(a+m2));  
0081       d = = 1.0 + aa*d;  
0082       if (fabs(d) < FPMIN) d= FPMIN;  
0083       c = = 1.0 + aa/c;  
0084       if (fabs(c) < FPMIN) c= FPMIN;  
0085       d = = 1.0/d;  
0086       h = = d*c;  
0087       aa = = -(a+m)*(qab+m)*x/((a+m2)*(qap+m2));  
0088       d = = 1.0 + aa*d;  
0089       if (fabs(d) < FPMIN) d= FPMIN;  
0090       c = = 1.0 + aa/c;  
0091       if (fabs(c) < FPMIN) c= FPMIN;  
0092       d = = 1.0/d;  
0093       del = = d*c;  
0094       h = = del;  
0095       if (fabs(del-1.0) < EPS) break;  
0096 }  
0097 // if ((m > MAXIT) && (!global_data.reg_on)) {  
0098 if (m > MAXIT) {  
0099     cerr << "-E- a or b too big, or MAXIT too "
```
float beta (float a, float b, float x)  Computes the incomplete beta function \( I_x(a,b) \).

Author:
Numerical Recipes in C, page 227

void filter_whitespace (char line[])  Removes whitespace.

Removes leading whitespace, trailing whitespace, and compresses internal whitespace.

Invariant:
*line will not get larger

Precondition:
line points to a null terminated string of chars
Postcondition:
line points to a null terminated string of chars that is either smaller or the same size as the original value.

Parameters:
line A pointer to a string to remove excess whitespace from

Returns:
none

Author:
Todd Blackman

```c
0151     {  
0152     char *tmp;
0153     char tmp2[256];
0154     int offset=0;
0155     int offset2;
0156     //Leading whitespace skip
0157     while ( ((line[offset] == '\t') || (line[offset] == ' ')) &&
0158             (line[offset] != '\0')) offset++;
0159     tmp = &line[offset];
0160     //cycle through chars, looking for two whitespaces in a row. As we go,
0161     //copy char by char into tmp2. When two whitespace in a row found, copy
0162     //a single space into tmp2, and ignore rest of white space.
0163     offset=offset2=0;
0164     while (tmp[offset] != '\0') {
0165         //More than one whitespace char in a row
0166         while ( (tmp[offset+1] != '\0') &&
0167             ((tmp[offset] == '\t') || (tmp[offset] == ' ')) &&
0168             ((tmp[offset+1] == '\t') || (tmp[offset+1] == ' ')))
0169             {
0170                 if (tmp[offset] == '\t') tmp[offset]= ' ';  
0171                 offset++;
0172             }
0173     }
0174     tmp2[offset2++] = tmp[offset++];
0175     offset2 = tmp2[offset2];
0176     if (offset2 > 0) && (tmp2[offset2-1] == ' ')) tmp2[offset2-1] = '\0';
0177     //Copy tmp2 into line
```
float gammln (float xx)  Computes gamma function.

Author:
    Numerical Recipes in C, page 214

int pipe_getline (int fd, string & buf, char endchar = '\n')  Read from
file descriptor to endchar.

Parameters:
    fd  File descriptor
    buf Buffer to read into
    endchar Flag character to stop at

Warning:
    This function is blocking. It returns when either a pipe read error occurs or
the end of a line is reached.

Returns:
    The number of characters read from the pipe
```c
0250 {
0251    int x=0;
0252    int res;
0253    char tempbuf[500];
0254
0255    if (-1 == (res=read(fd, static_cast<void *>(tempbuf[x]), 1))) {
0256        return -1;
0257    }
0258    while (tempbuf[x] != endchar) {
0259        if (res == 1) {
0260            // cout << endl << "Read a char: " << tempbuf[x] << endl;
0261            ++x;
0262        }
0263        if (-1 == (res=read(fd, static_cast<void *>(tempbuf[x]), 1))) {
0264            return -1;
0265        }
0266    }
0267    tempbuf[x+1]=0;
0268    buf = tempbuf;
0269    return x;
0270 }
```

**void print_demo (int size)**  Prints a helpful diagram.

This function prints an ASCII representation of the board of the given size.

**Parameters:**

- **size** The size of the board to print

```c
0274 {
0275    cout << "   ";
0276    for (char x='A'; x<'A'+size; ++x) {
0277        if (x < 'I') {
0278            cout << "---" << x << "\n";
0279        } else {
0280            cout << "---" << static_cast<char>(x+1) << "\n";
0281        }
0282    }
0283    cout << "_.\n";
0284    cout << endl;
0285    // Rows
0286    int count=0;
0287    for (int x=size; x>0; --x) {
0288        cout << setw(3) << x << "|\n";
0289    }
0290    for (int z=0; z<size; ++z) {
0291        cout << " " << setw(3) << count++ << "\n";
0292 }
```
A.3.42 tools.h File Reference

Defines useful utilities.

Defines

- \#define SIGCUTOFF 0.01
  
  Level of significance for mean and variance comparisons required for acceptance.

- \#define SQR(x) ((x) * (x))
  
  Find the square of a number macro.

- \#define MAXIT 100
  
  Statistic constant.

- \#define EPS 3.0e-7
  
  Statistic constant.

- \#define FPMIN 1.0e-30
  
  Statistic constant.

Functions

- float gammln (float xx)
  
  Computes gamma function.
• float \texttt{betacf} (float a, float b, float x)

\textit{Evaluates continued fraction for incomplete beta function by modified Lentz's method.}

• float \texttt{betai} (float a, float b, float x)

\textit{Computes the incomplete beta function }\texttt{I}_\alpha(a,b).

• void \texttt{filter\_whitespace} (char line[])

\textit{Removes whitespace.}

• void \texttt{print\_demo} (int size)

\textit{Prints a helpful diagram.}

• int \texttt{pipe\_getline} (int fd, string &buf, char endchar=’\n’)

\textit{Read from file descriptor to endchar.}

• void \texttt{loopy} (int which, bool done=false, char *msg=’...DONE’)

\textit{Prints a "waiting" rotating character.}

• bool \texttt{odd} (int n)

\textit{Finds if the number is odd.}

\textbf{A.3.42.1 Detailed Description}

Defines useful utilities.

\textbf{Revision:}

1.10

\textbf{Date:}

2003/04/30 01:57:59

\textbf{Author:}

Todd Blackman

A.3.42.2 Define Documentation

#define SIGCUTOFF 0.01 Level of significance for mean and variance comparisons required for acceptance.

A.3.42.3 Function Documentation

float betacf (float a, float b, float x) Evaluates continued fraction for incomplete beta function by modified Lentz’s method.

Author:
   Numerical Recipes in C, page 227-8

0066 {
0067     int m,m2;
0068     float aa,c,d,del,h,qab,qam,qap;
0069     qab=a+b;
0070     qap=a+1.0;
0071     qam=a-1.0;
0072     c=1.0;
0073     d=1.0-qab*x/qap;
0074     if (fabs(d) < FPMIN) d=FPMIN;
0075     d=1.0/d;
0076     h=d;
0077     for (m=1;m<MAXIT;m++) {
0078         m2=2*m;
0079         aa=m*(b-m)*x/((qam+m2)*(a+m2));
0080         d=1.0+aa*d;
0081         if (fabs(d) < FPMIN) d=FPMIN;
0082         c=1.0+aa/c;
0083         if (fabs(c) < FPMIN) c=FPMIN;
0084         d=1.0/d;
0085         h *= d*c;
0086         aa = -(a+m)*(qab+m)*x/((a+m2)*(qap+m2));
0087         d=1.0+aa*d;
0088         if (fabs(d) < FPMIN) d=FPMIN;
0089         c=1.0+aa/c;
0090         if (fabs(c) < FPMIN) c=FPMIN;
0091         d=1.0/d;
0092         del=d*c;
0093         h *= del;
0094         if (fabs(del-1.0) < EPS) break;
0095     }
0096 }
0097 //if ((m > MAXIT) && (!global_data.reg_on)) {
0098     if (m > MAXIT) {
0099         cerr << "-E- a or b too big, or MAXIT too "
0100             << "small in betacf" << endl;

float betai (float a, float b, float x) Computes the incomplete beta function Lx(a,b).

Author:
Numerical Recipes in C, page 227

void filter_whitespace (char line[]) Removes whitespace.

Removes leading whitespace, trailing whitespace, and compresses internal whitespace.

Invariant:
*line will not get larger

Precondition:
line points to a null terminated string of chars
Postcondition:
line points to a null terminated string of chars that is either smaller or the same size as the original value.

Parameters:

*line* A pointer to a string to remove excess whitespace from

Returns:
none

Author:
Todd Blackman

```c
{
char *tmp;
char tmp2[256];
int offset=0;
int offset2;

//Leading whitespace skip
while ((line[offset] == '\t') || (line[offset] == ' ')) &&
       (line[offset] != '\0')) offset++;

//cycle through chars, looking for two whitespaces in a row. As we go,
//copy char by char into tmp2. When two whitespace in a row found, copy
//a single space into tmp2, and ignore rest of white space.
offset=offset2=0;
while (tmp[offset] != '\0') {
  //More than one whitespace char in a row
  while ( (tmp[offset+1] != '\0') &&
         ((tmp[offset] == '\t') || (tmp[offset] == ' ')) &&
         ((tmp[offset+1] == '\t') || (tmp[offset+1] == ' '))
  ) {
    if (tmp[offset] == '\t') tmp[offset]=' ';
    offset++;
  }
  tmp2[offset2++] = tmp[offset++];
}
tmp2[offset2] = tmp[offset];

//If last char is a space, delete it from tmp2.
if ((offset2 > 0) && (tmp2[offset2-1] == ' ')) tmp2[offset2-1] = '\0';

//Copy tmp2 into line
```
float gammln (float xx)  Computes gamma function.

Author:
   Numerical Recipes in C, page 214

int pipe_getline (int fd, string & buf, char endchar = '\n')  Read from
file descriptor to endchar.

Parameters:
   fd  File descriptor
   buf Buffer to read into
   endchar Flag character to stop at

Warning:
   This function is blocking. It returns when either a pipe read
error occurs or the end of a line is reached.

Returns:
   The number of characters read from the pipe

0187   strcpy(line, tmp2);
0188
0189   return;
0190 }

0045 {
0046   double x,y,tmp,ser;
0047   static double cof[6]={76.18009172947746, -86.505322787565, 24.0150862873205, -1.0000000000000190, 0.500000000000000, 9.5479966187}, ser=1.000000000000000;
0049   for (j=0;j<5;j++) ser *= cof[j]/++y;
0057   return -tmp+log(2.5066282746310005*ser/x);
0058 }

238
0250  {
0251    int x=0;
0252    int res;
0253    char tempbuf[500];
0254    if (-1 == (res=read(fd, static_cast<void*>(&tempbuf[x]), 1))) {
0255      return -1;
0256    }
0257    while (tempbuf[x] != endchar) {
0258      if (res == 1) {
0259        //cout << endl << "Read a char: " << tempbuf[x] << endl;
0260        ++x;
0261      }
0262      if (-1 == (res=read(fd, static_cast<void*>(&tempbuf[x]), 1))) {
0263        return -1;
0264      }
0265    }
0266  }
0267  tempbuf[x+1]=0;
0268  buf = tempbuf;
0269  return x;
0270  }
0271

void print_demo (int size)  Prints a helpful diagram.

  This function prints an ASCII representation of the board of the given size.

Parameters:
  size The size of the board to print

0244  {
0245    cout << "   ";
0246    for (char x='A'; x<\'A\'+size; ++x) {
0247      if (x < 'I') {
0248        cout << "--" << x << " ";
0249      } else {
0250        cout << "--" << static_cast<char>(x+1) << " ";
0251      }
0252    }
0253    cout << " _ ";
0254    cout << endl;
0255    //Rows
0256    int count=0;
0257    for (int x=size; x>0; --x) {
0258      cout << setw(3) << x << " | ";
0259    }
0260    for (int z=0; z<size; ++z) {
0261      cout << " " << setw(3) << count++ << " ";
0262    }
0263  }
239
A.3.43  traininterface.cpp File Reference

Implementation of Trainer class for GAs.

Defines

- `#define LOG(x)`
  
  *Macro for outputing to log file.*

Functions

- istream& operator>>(istream &strm, move_t &tmove)
  
  *Used to read a move_t (p.148) structure from a stream.*

Variables

- char rcsid []
  
  *Source code identifier.*

A.3.43.1  Detailed Description

Implementation of Trainer class for GAs.

This file provides the GaTrainerInterface (p.123) class
Revision:
1.22

Date:
2003/04/23 21:42:59

Author:
Todd Blackman

Warning:
Train only on boards that start empty with black moving first. Ensure that this interface is SECOND! Not first.


A.3.43.2 Variable Documentation

char rcsid [static] Initial value:


Source code identifier.

A.4 Exodus Page Documentation

A.4.1 Todo List

Class Moderator Add time-keeping code.

File exodus.h Agents need to be able to communicate for complex situations.

File ga.h Get rid of vectors and replace with arrays

File groupstatsagent.cpp Make this more memoized

Member Game::legal(loc_t) Add memoizability--store vector of legal/not-
legal that is updated as moves are made.

A.4.2 Bug List

File game.cpp super-ko does not take rotation and symmetry into account.
Appendix B: Running the Program

The program was written in ANSI C++, and simply typing ./configure in the exodus directory followed by make in the exodus/src directory will create the executable called exodus. Command-line options can be viewed by typing exodus -h. Default options can be given in a file called .exodusrc when this file is located in the current directory. Important options follow:

-`-reg_on` Run the regressions.
-`-bsize=x` Set the board size to x.
-`-train=x` If x is non-zero put the program in training mode otherwise put the program in playing mode.
-`-output=name` Sets the file that stores the latest genetic algorithm generation to name.
-`-best=name` Sets the file that stores the best chromosome from the last generation of the genetic algorithm to name. This parameter is also used to initialize the genetic algorithm player when it plays against a human player.
-`-train_file=name` Sets the name of the data file that contains recorded games to name.
-`-resume=name` If the program has been set to training mode, then this option tells the genetic algorithm to run the GA starting with the generation specified in name.
-maxgen=x Set the maximum generation to x.
-popsize=x Set the population size to x.
-pmutation=x Set the mutation probability to x.
-pcross=x Set the crossover probability to x.
-fmultiple=x Set the F multiplier (in GA) to x.
Glossary

atarī The single liberty left for a group.
baduk Another name of go.
dan A high ranking.
eye An open space inside of a group of stones. Two eyes make a group unconditionally alive if they are small enough.
goban A go playing board.
good shape The abstract concept that describes a set of stones that is in a formation conducive to being able to form a living group in the future.
group A set of stones that are connected to each other by being adjacent to at least one member of the group through a line on the board (i.e., not adjacent via diagonals).
liberty The adjacent locations to a stone that contain no other stones.
ko A situation involving the possible repetition of the state of the board which is not allowed.
kōmi Points of compensation given to white (always the second player) to equalize the effect of having to move second when the game begins. Usually it is 0.5 or 5.5 points so that a tie is impossible.
kyu A lower ranked go player.
seki Localized stalemate situation characterized by two groups sharing at least one liberty such that if one player were to play there first, his or her group would die. Neither side should play first, and neither side gains points for the territory surrounded by the two groups in seki.
thickness The abstract concept that describes how much a set of stones radiates influence, usually in a specific direction.
Index

A*, 16
A* search, see search techniques
~Agent
   Agent, 72
~AgentShell
   AgentShell, 74
~Board
   Board, 80
~ExtenderAgent
   ExtenderAgent, 87
~FollowerAgent
   FollowerAgent, 90
~GUIInterface
   GUIInterface, 137
~Ga
   Ga, 94
~GaTrainerInterface
   GaTrainerInterface, 121
~Game
   Game, 110
~GenAlgoGenerator
   GenAlgoGenerator, 125
~GroupStatsAgent
   GroupStatsAgent, 134
~IGS_Interface
   IGS_Interface, 139
~Interface
   Interface, 142
~Moderator
   Moderator, 144
~OpenerAgent
   OpenerAgent, 149
~ProbBoard
   ProbBoard, 153
~RandomAgent
   RandomAgent, 157
~Subthread
   Subthread, 165
~Subthread_test
   Subthread_test, 166
~TigersMouthAgent
   TigersMouthAgent, 171
actual_size
   Board, 79
actualSize
   ProbBoard, 155
Agent, 71
   ~Agent, 72
   Agent, 72
   bb_ptr, 73
dowork, 72
   force, 72
   get_id, 72
   ID, 73
   notify, 72
   pb_ptr, 73
   query_bits_needed_from_GA, 73
   send_bits, 73
   set_bb_ptr, 72
   set_id, 73
   set_pb_ptr, 72
   theGame, 73
   update, 72
agent.cpp, 175
   LOG, 175
   resid, 176
agent.h, 176
   MAX_AGENTS, 177
   MAXGROUPS, 177
MAXSTONE, 177
agents, 17
AGENTS_USED
config.h, 182
AgentShell
~AgentShell, 74
AgentShell, 75
processing, 74
theAgent, 75
AgentShell, 74
AgentShell, 75
init, 75
allele
gatypes.h, 195
alpha-beta, see alpha-beta pruning
alpha-beta pruning, 22
alpha/beta, see alpha-beta pruning
assign_global
main.cpp, 203
attempt
ExtenderAgent, 88
avg
Population, 151
baduk, 6
bb
GenAlgoGenerator, 126
bb_ptr
Agent, 73
bdemo.cpp, 177
main, 178
BEST_FILENAME_OUT
Ga, 95
betaCF
 tools.cpp, 226
tools.h, 233
betai
 tools.cpp, 227
tools.h, 234
bidirectional search, see search techniques
bits_per_value
ExtenderAgent, 88
bits_per_weight
GenAlgoGenerator, 126
BITSPERWEIGHT
outputgen.h, 213
black
Stone, 159
BLACK_BIT
Stone, 161
Blackboard, 75
g_ptr, 76
set_game_ptr, 76
update, 76
blackboard.cpp, 179
LOG, 179
rcsid, 179
blackboard.h, 180
block_cond
Subthread, 165
Board, 76
~Board, 80
actual_size, 79
Board, 77
BSIZE, 78
color_played, 79
del_group, 80
del_stone, 78
fill_safety, 81
get_bsize, 77
get_color_played, 78
get_goban, 77
get_move_played, 81
goban, 79
HANDICAP, 78
HANDICAP_PLACES, 78
invert, 82
loc_played, 79
operator<<, 79
operator=, 78
operator==, 77
operator[], 78
PASS, 78
play_move, 82
PRINTEXTRA, 79
put_ stone, 83
raw_output, 83
setup, 84
valid_location, 85
board.cpp, 180
LOG, 180
operator<<, 180
rcsid, 181
board.h, 181
loc_t, 181
operator<<, 182
breadth-first search, see search techniques
BSIZE
Board, 78
bsize
GUIInterface, 138
move_t, 147
BW_BITS
Stone, 161
BYOMI_STONES
Game, 113
BYOMI_TIME
Game, 113
CALL_processing
Subthread, 165
subthread.cpp, 219
capStones
Game, 113
Capturing, 8
checkers, 23
cycle, 23, 25
child
Subthread, 164
chrom
Individual, 140
chromosome_t
gatypes.h, 195
clear
ProbBoard, 154
Stone, 161
clearlastcol
Stone, 160
clearlastrow
Stone, 160
COL_BITS
Stone, 161
color
move_t, 146
color_played
Board, 79
config.h, 182
AGENTS_USED, 182
HAVE_FCNTL_H, 183
HAVE_MALLOC_H, 183
MAX_AGENT_THREADS, 182
MAX_THREADS, 183
NDEBUG, 183
PERLTK, 183
STDC_HEADERS, 183
crossover
Ga, 96
currentBoard
Game, 113
don, 14
data
msg_t, 147
dead
GroupStatsAgent, 135
decode
GenAlgoGenerator, 127
del_group
Board, 80
del_stone
Board, 78
depth-first search, see search techniques
depth-limited search, see search techniques
dowork
Agent, 72
ExtenderAgent, 87
FollowerAgent, 91
GroupStatsAgent, 135
OpenerAgent, 150
RandomAgent, 157
TigersMouthAgent, 172

DummyGenerator
DummyGenerator, 86
processing, 86
rndbuf, 86
DummyGenerator, 86
dummygenerator.cpp, 183
LOG, 183
rcsid, 184

empty
Stone, 159
tools.h, 231
enumerate_legal_locations
Game, 114

ERROR
subthread.h, 221
exodus.h, 184
global_data, 185
LOG, 185
log_out, 185
log_mutex, 185
NUM_THREADS, 185
thread_count, 185
usL, 187

ExtennderAgent
~ExtennderAgent, 87
attempt, 88
bits_per_value, 88
dowork, 87
ExtennderAgent, 87
extendLocations, 88
extendValue, 88
force, 87
knightLocations, 89
knightValue, 89
largeKnightLocations, 89
largeKnightValue, 89
notify, 87
num_values, 88
onePointExtendLocations, 88
onePointExtendValue, 88
send_bits, 87
shoulderLocations, 89
shoulderValue, 89
treePointExtendLocations, 88
treePointExtendValue, 88
twoPointExtendLocations, 88
twoPointExtendValue, 88
update, 87
ExtennderAgent, 87
getval, 89
query_bits_needed_from_GA, 90
tools.cpp, 227
tools.h, 234

figure_path
GUIInterface, 137

FILENAME_IN
Ga, 95
FILENAME_OUT
Ga, 95

call_safety
Board, 81

filter_whitespace
subthread.h, 221

fitness
Individual, 141

fitness function, 42

248
FITNESS_CUTOFF
Ga, 95
flip
Ga, 98
FMULTIPLE
Ga, 95
FollowerAgent
~FollowerAgent, 90
dowork, 91
FollowerAgent, 90
force, 91
notify, 91
query_bits_needed_from_GA, 91
send_bits, 91
update, 91
FollowerAgent, 90
imprint, 91
followeragent.cpp, 188
LOG, 188
rcsid, 189
FORCE
subthread.h, 221
force
Agent, 72
ExtenderAgent, 87
FollowerAgent, 91
GroupStatsAgent, 134
OpenerAgent, 150
RandomAgent, 158
TigersMouthAgent, 171
forward pruning, 24
FPMIN
tools.h, 231
fromthreadq
Subthread, 165
ftest
Ga, 98
g.ptr
Blackboard, 76
GA, see Genetic Algorithms
Ga, 94
~Ga, 94
BEST_FILENAME.OUT, 95
crossover, 96
FILENAME_IN, 95
FILENAME_OUT, 95
FITNESS_CUTOFF, 95
flip, 98
FMULTIPLE, 95
ftest, 98
Ga, 94
gen, 97
generation, 96
init, 99
interrupt watcher, 110
lchrom, 96
leader, 97
loadpop, 101
MAXGEN, 94
mutation, 96
ncross, 97
newpop, 96
nmutation, 97
oavg, 97
oldpop, 96
omax, 97
omin, 97
ostdev, 97
osumfitness, 97
ovar, 97
PCROSS, 95
PMUTATION, 95
POPSIZE, 95
prescale, 103
rndbuf, 97
savebest, 104
savepop, 94
scale, 105
scalepop, 96
select, 105
set кодекс, 106
start, 107
stop, 97
TRAIN FILE, 95
GenAlgoGenerator
~GenAlgoGenerator, 125
bb, 126
bits_per_weight, 126
GenAlgoGenerator, 125
init, 125
num_agents, 126
num_second_level_nodes, 127
num_threads, 126
printweights, 125
processing, 126
results, 126
rdbuf, 126
secondLevelWeights, 126
summary, 125
theAgents, 126
theThreads, 126
total_bits, 127
weights, 126
weights_loaded, 126
GenAlgoGenerator, 124
decode, 127
get_fitness, 128
get_move, 129
load, 131
genalgogenerator.cpp, 196
LOG, 196
rcsid, 197
generation
Ga, 96
Genetic Algorithms, 19
get_board
Game, 111
get_bsize
Board, 77
Game, 111
get_captures
Game, 115
get_chrom_size
PreCodex, 153
get_color_played
Board, 78
get_fitness
GenAlgoGenerator, 128
PreCodex, 153
testCodex, 168
get_game
Moderator, 144
get_goban
Board, 77
get_J0
Moderator, 144
get_J1
Moderator, 144
get_jd
Agent, 72
get_move
GenAlgoGenerator, 129
GUIInterface, 137
get_move_played
Board, 81
get_msg_nb
Subthread, 164
get_percentage
GaTrainerInterface, 123
get_turn
Game, 111
get_user_input
TextInterface, 169
get_val
ProbBoard, 154
getcol
Stone, 160
getcolor
Stone, 160
getrow
Stone, 160
getval
ExtenderAgent, 89
ginterface.cpp, 197
LOG, 197
rcsid, 197
global_data
  exodus.h, 185
  main.cpp, 203

global_data_t, 132
  handicap_placement, 133
  help, 133
  my_color, 133
  region, 133
  resume, 133
  train, 133
  verbosity, 133
  version, 133
  welcome, 132

go-moku, 23

goban
  Board, 79

goel, 6

GoModemInterface
  GoModemInterface, 134
  GoModemInterface, 133

GPATH
  GUIInterface, 137

gptr
  Interface, 143

GroupStatsAgent
  ~GroupStatsAgent, 134
  dead, 135
  dowork, 135
  force, 134
  GroupStatsAgent, 134
  grpcolor, 135
  gsize, 136
  liberties, 136
  liberty_locations, 136
  liberty_locations_list, 136
  notify, 135
  numgroups, 135
  printScratch, 135
  query_bits_needed_from_GA, 135
  recurse, 135
  scratch, 135
  update, 135

GroupStatsAgent, 134
  send_bits, 136
  groupstatsagent.cpp, 198
  LOG, 198
  rcsid, 199
  grpcolor
    GroupStatsAgent, 135
  gsize
    GroupStatsAgent, 136

GUIInterface, 136
  ~GUIInterface, 137
  bsize, 138
  figure_path, 137
  get_move, 137
  GPATH, 137
  GUIInterface, 137
  init, 137
  m2s, 138
  path, 138
  pid, 138
  processing, 138
  READ, 138
  s2m, 138
  send_board, 137
  WRITE, 138

HANDICAP
  Board, 78
  handicap_placement
    global_data_t, 133

HANDICAP_PACES
  Board, 78
  handle_move
    GaTrainerInterface, 122

HAVE_FCNTL_H
  config.h, 183

HAVE_MALLOC_H
  config.h, 183
  help
    global_data_t, 133
  host1
    IGS_Interface, 139
  host2
IGS Interface, 139

10
  Moderator, 144

11
  Moderator, 144

ID
  Agent, 73

id
  msg_t, 147

igo, 6

IGS Interface, 139
  ~IGS Interface, 139
  host1, 139
  host2, 139
  IGS Interface, 139
  my_addr, 140
  port1, 139
  port2, 140
  setup, 139
  sfd, 140

interface.cpp, 199
  LOG, 199
  rcsid, 199

imprint
  FollowerAgent, 91

Individual, 140
  chrom, 140
  fitness, 141
  ofitness, 141
  operator=, 140
  operator==, 141
  parent1, 141
  parent2, 141
  xsite, 141

individuals
  Population, 151

init
  AgentShell, 75
  Ga, 99
  GaTrainerInterface, 123
  GenAlgoGenerator, 125
  GUI Interface, 137

INITIAL_TIME
  Game, 112

initialize
  main.cpp, 202
    inside_get_msg_b
      Subthread, 164
    inside_get_msg_nb
      Subthread, 164
    inside_send_msg
      Subthread, 164

intelligent agents, 17

Interface, 142
  ~Interface, 142
    get_make_a_move, 142
    gpstr, 143
    Interface, 142
    made_a_move, 143
    my_color, 143
    my_turn, 143
    set_my_color, 143
    set_my_turn_on, 142
    their_color, 143

interface.cpp, 200
  LOG, 200

interface.h, 200

internal_board
  ProbBoard, 155

interruptWatcher
  Ga, 110

INV
  stone.h, 218

inv_turn
  Game, 113

invert
  Board, 82

invert_board
  Game, 115

is_over
  Game, 116

iterative deepening, see search techniques

join
Subthread, 166

kill
Subthread, 163
killer moves, 24
knightLocations
  ExtenderAgent, 89
knightValue
  ExtenderAgent, 89
KOMI
  Game, 112
komi
  Game, 114
kyu, 14

largeKnightLocations
  ExtenderAgent, 89
largeKnightValue
  ExtenderAgent, 89
last
  Game, 116
lastcol
  Stone, 160
lastrow
  Stone, 160
lchrom
  Ga, 96
  PreCodex, 153
LCOL_BIT
  Stone, 162
leader
  Ga, 97
legal
  Game, 117
liberties, 9
liberties
  GroupStatsAgent, 136
libertyLocations
  GroupStatsAgent, 136
libertyLocations_list
  GroupStatsAgent, 136
Lisp, 23
livelock, see ko

LOAD
  subthread.h, 221
load
  GaTrainerInterface, 123
  GenAlgoGenerator, 131
loadpop
  Ga, 101
loc
  move_t, 146
locPlayed
  Board, 79
loc_t
  board.h, 181
lock
  Game, 118
LOG
  agent.cpp, 175
  blackboard.cpp, 179
  board.cpp, 180
  dummygenerator.cpp, 183
  exodus.h, 185
  extenderagent.cpp, 187
  followeragent.cpp, 188
  ga.cpp, 189
  game.cpp, 192
  genalgogenerator.cpp, 196
  ginterface.cpp, 197
  groupstatsagent.cpp, 198
  iinterface.cpp, 199
  interface.cpp, 200
  main.cpp, 202
  moderator_t, 209
  move.cpp, 210
  openeragent.cpp, 211
  probboard.cpp, 213
  randomagent.cpp, 215
  stone.cpp, 216
  subthread.cpp, 219
  tigersmouthagent.cpp, 222
  tinterface.cpp, 223
  traininterface.cpp, 238
log_cout
operator==, 146
operator==, 146
pass, 146
regression, 146
setup_phase, 146
movenum
    Game, 118
movesGuessed
    GaTrainerInterface, 122
movestream
    GaTrainerInterface, 122
movestream_iter
    GaTrainerInterface, 122
msg
    TextInterface, 169
msg jd_t
    subthread.h, 220
msg t, 147
data, 147
id, 147
operator==, 147
multiagent systems, 17
mutation
    Ga, 96
mutex
    Game, 113
my_addr
    IGS_Interface, 140
my_color
    global data t, 133
    Interface, 143
my turn
    Interface, 143
ncross
    Ga, 97
NDEBUG
    config.h, 183
neural networks, 16, 26
NeuralNetGenerator
    NeuralNetGenerator, 148
NeuralNetGenerator, 148
newboard
    move t, 146
newpop
    Ga, 96
nmutation
    Ga, 97
NGS Interface, 148
    NGS Interface, 149
NOMSG
    subthread.h, 220
normalize
    ProbBoard, 155
notblack
    Stone, 159
notbottom
    Stone, 159
notempty
    Stone, 159
NOTIFY
    subthread.h, 221
notify
    Agent, 72
    ExtenderAgent, 87
FollowerAgent, 91
GroupStatsAgent, 135
OpenerAgent, 150
RandomAgent, 157
TigersMouthAgent, 172
notleft
    Stone, 159
notright
    Stone, 159
nottop
    Stone, 159
notwhite
    Stone, 159
num_agents
    GenAlgoGenerator, 126
num_second_level_nodes
    GenAlgoGenerator, 127
NUM_THREADS
    exodus.h, 185
num_threads
GenAlgoGenerator, 126
num_values
    ExtenderAgent, 88
numgroups
    GroupStatsAgent, 135
oavg
    Ga, 97
objective function, 42
odd
    tools.cpp, 225
tools.h, 232
ofitness
    Individual, 141
oldpop
    Ga, 96
omax
    Ga, 97
omin
    Ga, 97
onePointExtendLocations
    ExtenderAgent, 88
onePointExtendValue
    ExtenderAgent, 88
OpenerAgent
    ~OpenerAgent, 149
dowork, 150
force, 150
notify, 150
OpenerAgent, 149
    pb17, 150
    pb19, 150
    pb9, 150
query_bits_needed_from_GA, 150
    update, 150
OpenerAgent, 149
    send_bits, 150
openeragent.cpp, 211
    LOG, 211
rcsid, 212
operator *
    ProbBoard, 154
operator!=
    Board, 77
    Game, 119
    Individual, 140
    Population, 151
    ProbBoard, 154
    Stone, 162
operator+
    ProbBoard, 154
operator+=
    ProbBoard, 154
operator<<
    Board, 79
    board.cpp, 180
    board.h, 182
    ga.cpp, 189
    Game, 121
game.cpp, 193
game.h, 195
gatypes.h, 195
    Population, 152
    ProbBoard, 155
    probboard.cpp, 214
    Stone, 162
    stone.cpp, 216
    stone.h, 218
testcodex.cpp, 222
operator=
    Board, 78
    Game, 112
    Individual, 140
    move_t, 146
msg_t, 147
    Population, 151
    ProbBoard, 154
    Stone, 161
operator==
    Board, 77
    Game, 112
    Individual, 141
    move_t, 146
    Population, 151
    ProbBoard, 154
    Stone, 161
Stone, 162
operator>>
  traingainterface.cpp, 238
operator[]
  Board, 78
  ProbBoard, 154
ostdev
  Ga, 97
osumfitness
  Ga, 97
Othello, 23
outputgen.h, 212
  BITSPERWEIGHT, 213
  SECONDLEVELNODES, 213
ovar
  Ga, 97
parent1
  Individual, 141
parent2
  Individual, 141
parse_cmd_line_options
  main.cpp, 202
parse_rc_file
  main.cpp, 205
PASS
  Board, 78
pass
  move_t, 146
path
  GUIInterface, 138
Patricia tree, 28
pb17
  OpenerAgent, 150
pb19
  OpenerAgent, 150
pb9
  OpenerAgent, 150
pb_ptr
  Agent, 73
PCROSS
  Ga, 95
Pente, 23
PERLTK
  config.h, 183
pid
  GUIInterface, 138
piece, see stone
pipe_getline
  tools.cpp, 229
  tools.h, 236
play_move
  Board, 82
  Game, 110, 111, 119
PMUTATION
  Ga, 95
POPSIZE
  Ga, 95
Population, 151
  avg, 151
  individuals, 151
  max, 151
  min, 151
  operator<<, 152
  operator=, 151
  operator==, 151
  stdev, 152
  sumfitness, 151
  var, 152
  whichmax, 152
  whichmin, 152
port1
  IGS_Interface, 139
port2
  IGS_Interface, 140
PreCodex
  get_chrom_size, 153
  get_fitness, 153
  lchrom, 153
  set_chrom_size, 152
  summary, 153
PreCodex, 152
prescale
  Ga, 103
print_demo

258
tools.cpp, 230
tools.h, 237
print_help
main.cpp, 202
print_welcome
main.cpp, 208
PRINTEXTRA
Board, 79
printScratch
GroupStatsAgent, 135
printweights
GenAlgoGenerator, 125
ProbBoard
~ProbBoard, 153
actualSize, 155
clear, 154
get_val, 154
internal_board, 155
operator *, 154
operator+, 154
operator+=, 154
operator<<, 155
operator=, 154
operator==, 154
operator[], 154
ProbBoard, 153
rdbuf, 155
set_val, 153
ProbBoard, 153
maxloc, 155
normalize, 155
spin, 156
probboard.cpp, 213
LOG, 213
operator<<, 214
rcsid, 214
probboard.h, 214
SPIN, 215
processing
AgentShell, 74
DummyGenerator, 86
GaTrainerInterface, 122
GenAlgoGenerator, 126
GUIInterface, 138
Subthread, 164
Subthread_test, 167
TextInterface, 169
prompt
TextInterface, 169
put_stone
Board, 83
query_bits_needed_from_GA
Agent, 73
ExtenderAgent, 90
FollowerAgent, 91
GroupStatsAgent, 135
OpenerAgent, 150
RandomAgent, 158
TigersMouthAgent, 172
QUIT
subthread.h, 221
RandomAgent
~RandomAgent, 157
dowork, 157
notify, 157
RandomAgent, 157
send_bits, 158
update, 157
RandomAgent, 157
force, 158
query_bits_needed_from_GA, 158
randomagent.cpp, 215
LOG, 215
rcsid, 216
ranking, 14
raw_output
Board, 83
rcsid
agent.cpp, 176
blackboard.cpp, 179
board.cpp, 181
dummygenerator.cpp, 184
extenderagent.cpp, 188
followeragent.cpp, 189
ga.cpp, 189
game.cpp, 193
genalgogenerator.cpp, 197
ginterface.cpp, 197
groupstatsagent.cpp, 199
iinterface.cpp, 199
main.cpp, 208
move.cpp, 210
openeragent.cpp, 212
probboard.cpp, 214
randomagent.cpp, 216
stone.cpp, 217
subthread.cpp, 219
testodex.cpp, 222
tigersmouthagent.cpp, 223
tinterface.cpp, 224
tools.cpp, 225
traininginterface.cpp, 239
rcsidm
moderator.t, 209
READ
GUIInterface, 138
recurse
GroupStatsAgent, 135
reg_on
global_data_t, 133
regression
move_t, 146
Subthread_test, 166
reset
Game, 119
RESIGN
subthread.h, 221
results
GenAlgoGenerator, 126
resume
global_data_t, 133
retract
Game, 120
Reversi, 23
reversi, 25
rndbuf
DummyGenerator, 86
Ga, 97
GenAlgoGenerator, 126
ProbBoard, 155
ROW_BITS
Stone, 161
s2m
GUIInterface, 138
SANE, 31, 32
savebest
Ga, 104
savepop
Ga, 94
scalability, 40
scale
Ga, 105
scalepop
Ga, 96
scoring, 13
scratch
GroupStatsAgent, 135
search, see search techniques
search space, 25
search techniques, 15
SECONDLEVELNODES
outputgen.h, 213
secondLevelWeights
GenAlgoGenerator, 126
Seki, 12
select
Ga, 105
send_bits
Agent, 73
ExtenderAgent, 87
FollowerAgent, 91
GroupStatsAgent, 136
OpenerAgent, 150
RandomAgent, 158
TigersMouthAgent, 172
send_board
GUIInterface, 137
send_msg
Subthread, 164
SET_BB_PTR
subthread.h, 221
set_bb_ptr
Agent, 72
set_chrom_size
PreCodex, 152
set_code
Ga, 106
SET_GAME_PTR
subthread.h, 220
set_game_ptr
Blackboard, 76
set_id
Agent, 73
set_my_color
Interface, 143
set_my_turn_on
Interface, 142
set_pb_ptr
Agent, 72
SET_PROB_PTR
subthread.h, 221
set_suicide
Game, 112
set_super_ko
Game, 112
set_turn
Game, 111
set_val
ProbBoard, 153
setcol
Stone, 160
setcolor
Stone, 160
setlastcol
Stone, 160
setlastrow
Stone, 160
setrow
Stone, 160
setup
Board, 84
Game, 113
IGS_Interface, 139
setup_phase
move_t, 146
sfd
IGS_Interface, 140
shoulderLocations
ExtenderAgent, 89
shoulderValue
ExtenderAgent, 89
SIGCUTOFF
tools.h, 233
sizes, board, 13
SPIN
probboard.h, 215
spin
ProbBoard, 156
SQR
tools.h, 231
Stalemate, see Seki
start
Ga, 107
Subthread, 163
start_play
main.cpp, 202
start_training
main.cpp, 202
STDC_HEADERS
config.h, 183
stdev
Population, 152
Stone, 158
black, 159
BLACK_BIT, 161
BW_BITS, 161
clear, 161
clearlastcol, 160
clearlastrow, 160
COL_BITS, 161
empty, 159
getcol, 160
getcolor, 160
getrow, 160
lastcol, 160
lastrow, 160
LCOL_BIT, 162
L_ROW_BIT, 162
notblack, 159
notbottom, 159
notempty, 159
notleft, 159
notright, 159
nottop, 159
notwhite, 159
operator!=, 162
operator<<, 162
operator==, 161
operator==, 162
ROW_BITS, 161
setcol, 160
setcolor, 160
setlastcol, 160
setlastrow, 160
setrow, 160
Stone, 159
stoneOut, 161
theStone, 161
white, 159
WHITE_BIT, 161
stop
Ga, 97
strchrom
ga.cpp, 190
Subthread, 163
~Subthread, 165
block_cond, 165
CALL_processing, 165
childt, 164
fromthreadq, 165
get_msg_nb, 164
inside_get_msg_b, 164
inside_get_msg_nb, 164
inside_send_msg, 164
join, 166
kill, 163
message_queue_mutex, 165
processing, 164
send_msg, 164
start, 163
Subthread, 163
tell_message, 164
tothreadq, 165
subthread.cpp, 218
CALL_processing, 219
LOG, 219
rcsid, 219
subthread.h, 219
ERROR, 221
FINISHED, 221
FORCE, 221
LOAD, 221
msg_id_t, 220
NOMSG, 220
NOTIFY, 221
QUIT, 221
RESIGN, 221
SET_BB_PTR, 221
SET_GAME_PTR, 220
SET_PROB_PTR, 221
TURN, 221
UPDATE, 221

262
Subthread_test, 166
~Subthread_test, 166
processing, 167
regression, 166
SUICIDE
Game, 112
suicide
Game, 114
SUICIDE_CHECK
game.h, 194
sum_fitness
Population, 151
summary
GenAlgoGenerator, 125
PreCodex, 153
testCodex, 167
SUPER_KO
Game, 112
super_ko
Game, 114
SUPERK0_CHECK
game.h, 194
swap_interfaces
Moderator, 144
tell_message
Subthread, 164
testCodex
summary, 167
testCodex, 167
testCodex, 167
get_fitness, 168
testcodex.cpp, 221
operator<<, 222
rcsid, 222
TextInterface
get_user_input, 169
msg, 169
prompt, 169
TextInterface, 169
TextInterface, 168
processing, 169
theAgent
AgentShell, 75
theAgents
GenAlgoGenerator, 126
theGame
Agent, 73
Game, 113
Moderator, 145
their_color
Interface, 143
theStone
Stone, 161
theThreads
GenAlgoGenerator, 126
thread pools, 20
thread_count
exodus.h, 185
main.cpp, 203
threePointExtendLocations
ExtenderAgent, 88
threePointExtendValue
ExtenderAgent, 88
TigersMouthAgent
~TigersMouthAgent, 171
dowork, 172
force, 171
notify, 172
query_bits_needed_from_GA, 172
send_bits, 172
TigersMouthAgent, 171
update, 171
TigersMouthAgent, 171
findtiger, 172
tigersmouthagent.cpp, 222
LOG, 222
rcsid, 223
tinterface.cpp, 223
LOG, 223
rcsid, 224
token, see stone
tools.cpp, 224
betacf, 226
betal, 227
filter_whitespace, 227
gammln, 229
loopy, 225
odd, 225
pipe_getline, 229
print_demo, 230
rcsid, 225
tools.h, 231
betacl, 233
betai, 234
EPS, 231
filter_whitespace, 234
FPMIN, 231
gammeln, 236
loopy, 232
MAXIT, 231
odd, 232
pipe_getline, 236
print_demo, 237
SIGCUTOFF, 233
SQR, 231
total_bits
   GenAlgoGenerator, 127
totalmoves
   GaTrainerInterface, 122
tothreadq
   Subthread, 165
train
   global_data_t, 133
TRAIN_FILE
   Ga, 95
traininterface.cpp, 238
   LOG, 238
   operator>>, 238
   rcssid, 239

ttest
   Ga, 109
TURN
   subthread.h, 221
tutest
   Ga, 109
twoPointExtendLocations
   ExtenderAgent, 88
twoPointExtendValue
   ExtenderAgent, 88
uniform-cost search, see search techniques
unlock
   Game, 112
UPDATE
   subthread.h, 221
update
   Agent, 72
   Blackboard, 76
   ExtenderAgent, 87
   FollowerAgent, 91
   GroupStatsAgent, 135
   OpenerAgent, 150
   RandomAgent, 157
   TigersMouthAgent, 171
usi_p
   game.h, 194
usi_t
   exodus.h, 187
valid_location
   Board, 85
var
   Population, 152
verbosity
   global_data_t, 133
VERSION
   exodus.h, 185
version
   global_data_t, 133
wei-chi, 6
wei-qi, 6
weights
   GenAlgoGenerator, 126
weights_loaded
   GenAlgoGenerator, 126
welcome
   global_data_t, 132
whichmax
   Population, 152
whichmin
   Population, 152
white
   Stone, 159
WHITE_BIT
   Stone, 161
whose_turn
   Game, 113
   Moderator, 144
WRITE
   GUIInterface, 138
wtturn
   Game, 111
xsite
   Individual, 141