# Optimizing Boggle boards

## An evaluation of parallelizable techniques

Ankur Dave
ankurdave@gmail.com
Candidate #0844-028

Word Count: 3611

January 22, 2009

## Abstract

This paper's objective is to find efficient, parallelizable techniques of solving global optimization problems. To do this, it uses the specific problem of optimizing the score of a Boggle board.

Global optimization problems deal with maximizing or minimizing a given function. This has many practical applications, including maximizing profit or performance, or minimizing raw materials or cost.

Parallelization is splitting up an algorithm across many different processors in a way that allows many pieces of work to run simultaneously. As parallel hardware increases in popularity and decreases in cost, algorithms should be parallelizable to maximize efficiency.

Boggle is a board game in which lettered cubes are shaken onto a 4-by-4 grid. The objective is to find words spelled by paths through the grid. The function to maximize is the sum of the scores of all possible words in the board.

In this paper, the performance of two algorithms for global optimization are investigated: hill climbing and genetic algorithms. Genetic algorithms, which model evolution to find the fittest solutions, are found to be more efficient because they are non-greedy. In addition, a modified genetic algorithm called the coarse-grained distributed genetic algorithm (DGA) is investigated. This algorithm can take advantage of multiple computers, running several semi-independent copies of the algorithm in parallel to provide extra genetic diversity and better performance. The success of the coarse-grained DGA shows that global optimization problems can benefit significantly from parallelization.

Investigating these genetic algorithms revealed several modifications that are beneficial to global optimization. These modifications solve the problem of premature convergence (a loss of genetic diversity). Several techniques to solve this problem are investigated, notably incest prevention and migration control, revealing a very significant performance increase.

# Contents

**1 Introduction** **1**

**2 Boggle board data format** **2**

**3 Techniques for optimizing Boggle boards** **2**

3.1 Testing method . . . 3

3.2 Hill climbing . . . 3

3.3 Genetic algorithm . . . 4

3.3.1 Single-population genetic algorithm . . . 4

3.3.2 Coarse-grained distributed genetic algorithm . . . 6

**4 Conclusion** **7**

**References** **7**

**A Appendix** **9**

A.1 Board.java . . . 9

A.2 BoardTester.java . . . 14

A.3 ByBoardScore.java . . . 15

A.4 ByStringLength.java . . . 15

A.5 Dictionary.java . . . 16

A.6 DictionaryTester.java . . . 18

A.7 GenerationEmptyException.java . . . 18

A.8 GeneticClient.java . . . 18

A.9 GeneticClientTester.java . . . 20

A.10 GeneticClientThread.java . . . 20

A.11 HillClimber.java . . . 22

A.12 Population.java . . . 22

A.13 Server.java . . . 24

A.14 ServerTester.java . . . 26

A.15 ServerThread.java . . . 26

A.16 Util.java . . . 28

# 1 Introduction

In recent years, parallel computing has seen an enormous rise in interest. While supercomputers have used multiple processors for years, the rise of multi-core processors in desktop computers is a fairly recent phenomenon [1]. In addition, Google-style cluster computing, harnessing thousands of commodity computers to do the work of a few high-end servers at a much lower cost, is rising in popularity [2]. In order to take advantage of these new developments, computer scientists who need to solve problems must find ways to do so that lend themselves to parallelization—being split up across many different processors in a way that allows every piece of work to run simultaneously.

Global optimization problems are one class of problems in computer science that can benefit from the rise of parallel computing. Global optimization problems deal with finding the solution among a set of all possible solutions that maximizes a given function [3]. The set of all possible solutions is called the search space, and the function to evaluate each possible solution is called the objective function [4]. Global optimization problems have many practical applications; any problem that benefits from the maximization or minimization of a value—for example, raw materials used or cost—uses a optimization algorithm.

Such applications of global optimization problems are usually limited by the amount of time taken to find the optimal solution to the problem [4], so they must be interrupted after a specified time has elapsed and the best solution so far must be used. Therefore, faster techniques for solving global optimization problems would benefit users of such problems, because they would allow better solutions to be found in the same amount of time. If the algorithm for solving a global optimization problem were able to take advantage of parallelization, its speed would be greatly improved.

The objective of this paper is to find efficient, parallelizable techniques of solving global optimization problems. In order to test such techniques, a specific global optimization problem is used: the optimization of the score of a Boggle board.

Boggle is a board game published by Hasbro in which 16 cubes, each having a letter printed onto every side, are shaken onto a 4-by-4 grid. The objective of the game is to find words that are spelled out by paths through the grid. These paths must take only horizontally, vertically, or diagonally adjacent steps, and they must not use a cube more than once for the same word. Longer words score more points; the detailed scoring rules are given in Table 1.

Every Boggle board has a score that is calculated by finding all the possible words in the board and summing the scores of each of the words.

In order to frame the Boggle board problem as a global optimization problem, it is necessary to specify the key parameters of global optimization problems: the format of possible solutions, the

| Word length | Points |
|---:|---|
| ≤ 3 | 0 |
| 4 | 1 |
| 5 | 2 |
| 6 | 3 |
| 7 | 5 |
| ≥ 8 | 11 |

Table 1: Boggle scoring rules

search space of possible solutions, and the objective function with which to measure solutions. In the case of the Boggle board problem, any Boggle board is a possible solution. The search space is therefore the set of all possible Boggle boards, and the objective function—the function that we wish to maximize—is the score of a Boggle board.

In order to understand why optimizing the score of a Boggle board is not a trivial problem, it is helpful to look at the search space. A Boggle board has 16 cubes. For simplicity of implementation each cube is represented by a random letter from the alphabet rather than one of the six letters on the actual cube in that position. Therefore, each letter can be one of the 26 letters in the English alphabet. This means that there are $26^{16}$ (about 44 sextillion, or about $2^{75}$) possible Boggle boards. If each board takes 0.01 second to score, then it would take over 13 trillion years to score all the boards in order to find the highest-scoring one. Clearly, any algorithm to optimize the score of a Boggle board must take a more intelligent approach to the problem than simply scoring every single board. Section 3 discusses the pursuit of such a technique.

An efficient and parallelizable technique for optimizing the score of a Boggle board would yield insights that would allow many other global optimization problems more efficiently to take advantage of the increasingly common and powerful resource of parallel computing.

## 2 Boggle board data format

In order to optimize the score of Boggle boards, the first step is to define the storage format for any Boggle board. These boards can be represented most clearly as objects, so throughout this paper, Java, an object-oriented language, is used. The Boggle board is stored as a `Board` object (A.1) that has a two-dimensional array of `Letters` that stores the information for each letter on the board, a value for the score of the board, a value for the side length of the board (4 in the official version of Boggle), a list of words contained in the board, and a reference to a dictionary.

The score of the board is calculated by finding the sum of the scores of all the possible words in the board (A.1, page 11, line 190). To find all possible words in the board, a depth-first search algorithm is used. This algorithm starts at the first letter on the board and recursively explores as far as possible along one path, stopping and backtracking when there are no letters left. Because using the same letter more than once in the same word is not allowed, each letter has a flag indicating whether or not it has been already used in the current word; this flag is set before traversing through a letter and lowered after finishing traversal through the letter. To improve performance, the dictionary used to check for valid words is stored as a trie—a tree in which each node is a letter and words in the dictionary are represented by paths through the tree (A.5). This provides the benefit that dictionary lookups take constant time instead of logarithmic time, as they would with a flat dictionary. Pruning of the search tree (skipping traversal of branches that clearly will not contain any valid words) is also done using the trie to see if the current path begins any valid words (A.5, page 16, line 28).

## 3 Techniques for optimizing Boggle boards

Optimization algorithms fall into two classes: greedy and non-greedy. This section investigates stochastic hill climbing, a greedy algorithm, and genetic algorithms, a greedy algorithm. After finding that non-greedy algorithms are a better choice for optimization problems, it explores whether distributed algorithms provide an improvement over non-distributed algorithms for

optimization.

### 3.1 Testing method

Each algorithm is run through 100 trials. In each trial, the algorithm is allowed to run until one of the following conditions is met:

- the Boggle board score reaches 3500, or
- 1000 seconds elapse.

Non-distributed algorithms (those that run on only one computer) are tested on a Dell Precision M90 with the following specifications:

**CPU** Intel Core 2 Duo T7200, 2.00 GHz

**RAM** 2.00 GB

**OS** Ubuntu Linux 8.10

### 3.2 Hill climbing

A simple starting point for the task of finding high-scoring Boggle boards is hill climbing, an algorithm that starts with a random board and makes random, small changes, called mutation. If a change improves the board's score, it keeps the change; otherwise, it tries a different change. Hill climbing for Boggle is implemented in A.11.

Hill climbing is a greedy algorithm, which means it only makes changes that improve the score of the Boggle board. This can cause it to get stuck in a local optimum. If there is a high-scoring Boggle board with no higher-scoring boards that are a small change from the current board, then the hill climbing algorithm will not be able to improve on the current board.

The implementation of hill climbing to optimize Boggle boards can be found in A.11. This implementation was run according to the standard testing method in 3.1. To investigate the impact of the fact that hill climbing is a greedy algorithm, it is necessary to check which halting condition has happened first—the score has reached 3500, or 1000 seconds have elapsed. If the first halting condition is satisfied first, it means that the algorithm has performed well, and the time taken to reach the score of 3500 provides a means to compare hill climbing with other algorithms. If the second halting condition is satisfied first, however, it means that the algorithm has likely reached a local optimum—a consequence of the fact that hill climbing is a greedy algorithm—and will not provide any more improvement in the Boggle board score.

The low percentage of trials that reached a score of 3500—the success rate of the algorithm—is likely caused by the algorithm reaching local optima and ceasing to improve the score. This mediocre overall performance because of local optima is due to the fact that hill climbing is a greedy algorithm. To overcome it, it is necessary to choose an algorithm that is non-greedy—willing to follow paths that do not, in the short term, improve the Boggle board's score. This will allow the algorithm to escape local optima in search of the global optimum. One such algorithm is the genetic algorithm.

## 3.3 Genetic algorithm

A different algorithm for optimizing the score of a Boggle board is the genetic algorithm. The genetic algorithm is a non-greedy algorithm, so it is not restricted to following paths that always improve the Boggle board score. It is able to take "risks" in order to escape local optima.

Genetic algorithms attempt to solve optimization problems by modeling evolution. They simulating a population of candidate solutions, called chromosomes. The population goes through multiple generations. In each generation the score, or fitness, of each chromosome is calculated. The more fit a chromosome, the more likely it is to be selected to reproduce.

After selecting a portion of the population to reproduce, the chromosomes in the population are paired up and combined using the crossover and mutation operators. The crossover operator takes parts of each parent chromosome and joins them to form an offspring. The mutation operator adds an element of randomness to the simulation to help escape local optima. It is helpful when most of the chromosomes in the population are similar; in this situation, it introduces variety to continue improving the score. It randomly replaces a part of the offspring chromosome with a random piece of genetic material.

### 3.3.1 Single-population genetic algorithm

The basic version of the genetic algorithm works as described above, with no modifications made to allow it to run on multiple computers or take advantage of multiple cores in a computer. Beginning with the simple, non-parallelized version reveals whether parallelizing the algorithm yields significant improvements or not.

The implementation for the single-population genetic algorithm is contained in the `Population` class (A.12). This class stores the current population of Boggles, and each time the `Population.evolve` method (A.12, page 23, line 53) is called, it advances the population to the next generation by pairing up parents, creating children using crossover and mutation (A.1, page 13, line 270), and replacing the parent generation with the child generation.

Testing this implementation according to the testing procedures in Section 3.1 resulted in an average time to completion of $915 \pm 10.3$ seconds, and a success rate of 10%.

This very low success rate arose mostly because, after the first few generations, the population was almost completely composed of copies of the same chromosome. When these copies reproduced, they almost always produced more copies of the same chromosome rather than finding better chromosomes.

There are several problems with this algorithm that reduce its performance. One problem is premature convergence, which is a reduction of genetic diversity to the point where the genetic algorithm is unable to continue exploring the search space for fitter solutions. This occurs when the population contains only the descendents or the copies of one chromosome, so there is no variety of genetic material. This problem arises most often when a locally optimum chromosome is created, and it and its descendents dominate the population and crowd out all other chromosomes. The population has thus converged on a certain chromosome before the genetic algorithm is able to explore much of the search space. In the test above, premature convergence was the primary cause of such a low success rate.

In order to reduce the problem of premature convergence, one technique is incest prevention. Since premature convergence usually occurs when a locally optimum chromosome multiplies out of control by reproducing with near-copies of itself, it can be avoided in many cases by prohibiting such

"incestuous" reproduction. In order to prevent incest, it is necessary to prevent more than one copy of the same board to exist in a population (A.12, page 23, line 78), as well as increasing the mutation rate for reproduction between two boards that are excessively similar (A.1, page 13, line 297). The former will prevent a board from dominating the population with copies of itself, while the latter will introduce genetic diversity through mutation if the population consists of many similar boards.

Another possible solution for premature convergence is weighted random mating. This technique aims to increase genetic diversity by giving a wider fitness range of chromosomes a chance at contributing genetic material to the next generation. Normally, when evolving the next generation by mating the current generation, the boards are paired up according to score—the highest-scoring boards is paired up with the second-highest, the next one is paired up with the next after that, and so on. This has the advantage of ensuring that the boards with the best characteristics get a chance to merge genetic material and produce an even better board, but because boards that have similar scores often came from the same genetic background, it encourages incest and discourages diversity. This may make it more likely for the population to converge prematurely on a local maximum, resulting in a low success rate. Instead, weighted random mating pairs boards up in a more diverse fashion. For every pair, it chooses two random boards from the population, weighted so that the higher a board's score, the greater chance it has of being chosen. This approach should allow lower-scoring boards to pair with higher ones, increasing genetic diversity, while still preferring higher-scoring boards.

Testing the two techniques separately and comparing the results provides a basis for deciding which, if any, technique is more effective at improving performance. Testing the single-population algorithm with incest prevention resulted in an average time of $501 \pm 9.9$ seconds, and a success rate of 70%. Weighted random mating resulted in an average time of $867 \pm 134$ seconds, and a success rate of 14%. Compared to the performance of the simple genetic algorithm, both of these techniques provided improvements, but comparing the success rates of two techniques shows that incest prevention is better for solving the problem of premature convergence, because the greater the success rate, the fewer trials failed early due to premature convergence.

Another problem with the genetic algorithm is the loss of highly fit chromosomes. If, by chance, an especially fit chromosome is produced in one generation, it is usually difficult for the chromosome's children to surpass its fitness in just one generation. However, instead of being given more chances at creating higher-scoring children, the chromosome dies in one generation and only its less-fit children remain. In order to give such a high-scoring chromosome the extra chances at reproduction that it needs, some way of promoting the chromosome to the next generation unmodified.

One way of doing this is elitist selection. This technique involves copying the highest-scoring board from one generation directly into the next generation, as well as allowing it to reproduce.

Testing an implementation of elitist selection, combined with the best premature convergence technique above, incest prevention, resulted in an average time of $368 \pm 9.8$ seconds, and a success rate of 78%. Elitist selection improved the average time by a factor of 1.4, demonstrating that it is an effective solution for the loss of highly fit chromosomes.

Using the best of these techniques, incest prevention and elitist selection, we can proceed from optimization of the genetic algorithm on a single processor to optimization on multiple parallel processors.

### 3.3.2 Coarse-grained distributed genetic algorithm

After creating and optimizing a single-population genetic algorithm, the next step is to take advantage of parallelism by modifying the algorithm to work on multiple computers, called distributed computing. The advantage of using multiple processors instead of one is that evolution can be performed in parallel and therefore takes less time. In addition, it is cheaper to use many commodity computers than just one computer with hundreds of processors [2]. One common way of running the genetic algorithm on multiple computers is called a coarse-grained distributed genetic algorithm (DGA). In this technique, each computer simulates an independent population, called a subpopulation, and chromosomes from each subpopulation migrate periodically to other subpopulations. (The opposite method is the fine-grained DGA, in which each computer is assigned small tasks like scoring a particular board by the master computer. There is only one population in total in this technique, as opposed to one per computer in the coarse-grained DGA.)

The coarse-grained DGA has several advantages over other distributed genetic algorithm variants. It is useful for clusters where network latency is high or bandwidth is low, because it requires relatively little communication between computers (only the overall command signals and the migrant chromosomes need to travel over the network). This means every computer can be working all the time, instead of spending a significant portion of time waiting for another computer to communicate with it over the network [5]. In addition, using multiple subpopulations instead of one large population allows the subpopulations to evolve along different evolutionary paths [6]. This increases diversity by making it less likely that the entire genetic algorithm will be dominated by one gene.

The coarse-grained DGA is implemented using two components—a server and a client. The server is implemented in `Server` (A.13). It handles migration, keeps statistics about the subpopulations, and performs other administrative tasks for all the subpopulations. The client is implemented in `GeneticClient` in the glossary. It evolves an individual subpopulation, communicating with the server to take commands and send and receive migrant chromosomes.

In the unmodified coarse-grained DGA, migration is done every time a subpopulation completes a generation. Each time this happens, the subpopulation submits its most fit chromosome to the server for migration, keeping a copy for itself as well. It also accepts an incoming migrant chromosome from one of the other subpopulations at random.

As with the single-population algorithm, the main problem with this distributed genetic algorithm is premature convergence, not within a subpopulation but rather between subpopulations. This problem causes most or all subpopulations to be dominated by the same genetic materal, losing the advantage of a coarse-grained genetic algorithm—increased diversity because of multiple subpopulations. The most common methods of solving this problem involve modifying the way in which chromosomes migrate between subpopulations.

One solution to this problem is a fitness-based migration hierarchy. Unlike the standard, random method of migration, a fitness-based hierarchy only allows migrant chromosomes to travel to subpopulations with a higher average fitness than their home subpopulations.

This implementation was tested according to the testing procedures in Section 3.1. The implementation used the techniques for single-population genetic algorithms found in the previous section, as well as the fitness-based migration hierarchy. Testing resulted in an average time to completion of $258 \pm 9.9$ seconds, and a success rate of 94%. Therefore, moving to a distributed algorithm and taking advantage of both cores in the same computer, as well as reducing premature convergence, improved the performance of the algorithm by a factor of 1.4.

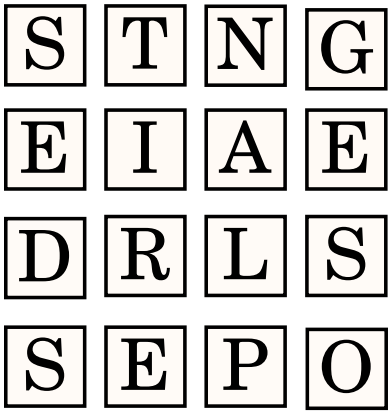


Figure 1: Best Boggle board found. Score: 4410

## 4 Conclusion

By investigating the performance, both in terms of time and success rate, of three global optimization algorithms—hill climbing, genetic algorithms, and coarse-grained DGAs—this paper determined that non-greedy algorithms are generally more successful than greedy ones, and that reimplementing an algorithm to make it distributed provides a significant performance benefit. The specific algorithms and techniques that were found to be the most effective were coarse-grained DGAs with incest prevention, elitist selection, and a fitness-based migration hierarchy. Using this algorithm, a very high-scoring Boggle board was found, given in Figure 1 (one of the longest words on this board is "predestines," worth 11 points in Boggle).

However, these findings are somewhat specific to the problem of optimizing Boggle boards. The testing used to find the highest-performing algorithms attempted to maximize the objective function of the score of a Boggle board, but this objective function is peculiar because evaluating it is a relatively time-consuming operation. Broader testing, using a variety of types of objective functions, is needed to establish the superiority of these algorithms.

In addition, only three types of global optimization algorithms were tested out of the hundreds that exist. While these three are among the most widely used, investigation of many other algorithms is necessary to conclude that any particular global optimization algorithm is superior.

## References

[1] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical report, EECS Department, University of California, Berkeley, December 2006.

[2] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[3] János Pintér. Global optimization. Wolfram MathWorld. <http://mathworld.wolfram.com/GlobalOptimization.html>.

[4] Global optimization. Wikipedia. <http://en.wikipedia.org/w/index.php?title=Global_optimization&oldid=246431368>.

[5] Shisanu Tongchim. Abstract coarse-grained parallel genetic algorithm for solving the timetable problem.

[6] David Patrick, Peter Green, and Trevor York. A distributed genetic algorithm environment for UNIX workstation clusters. In *Second International Conference On Genetic Algorithms in Engineering Systems: Innovations and Applications*, number 446, pages 69–74. IEE, September 1997.

[7] Francisco Herrera and Manuel Lozano. Heterogeneous distributed genetic algorithms based on the crossover operator. In *Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, number 446, pages 203–208. IEE, September 1997.

[8] Ben Paechter, Thomas Bäck, Marc Schoenauer, Michele Sebag, A. E. Eiben, J. J. Merelo, and T. C. Fogarty. A distributed resource evolutionary algorithm machine (DREAM). In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 2, pages 951–958. IEEE, July 2000.

[9] Weilie Yi, Qizhen Liu, and Yongbao He. Dynamic distributed genetic algorithms. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 2, pages 1132–1136. IEEE, July 2000.

[10] Jason Cooper and Chris Hinde. Improving genetic algorithms' efficiency using intelligent fitness functions. In *Proceedings of the 16th international conference on developments in applied artificial intelligence*, pages 636–643. Springer, 2003.

[11] Thomas Weise. Global optimization algorithms: Theory and application. <http://www.it-weise.de>, October 2008.

[12] Justin Andrew Boyan. *Learning Evaluation Functions for Global Optimization*. PhD thesis, Carnegie Mellon University, August 1998.

[13] David Power, Conor Ryan, and R. Muhammed Atif Azad. Promoting diversity using migration strategies in distributed genetic algorithms. In *The 2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1831–1838. IEEE, September 2005.

[14] Carlos Fernandes, Rui Tavares, and Agostinho C. Rosa. niGAVaPS — outbreeding in genetic algorithms. In *Proceedings of the 2000 ACM symposium on applied computing*, pages 477–482. ACM, 2000.

[15] K. C. Tan, M. L. Wang, and W. Peng. A P2P genetic algorithm environment for the Internet. *Communications of the ACM*, 48(4):113–116, April 2005.

[16] Vladimir I. Litvinenko, J. A. Burgher, Alla A. Tkachuk, and Vajcheslav J. Gnatjuk. The application of the distributed genetic algorithm to the decision of the packing in containers problem. In *Proceedings of the 2002 IEEE International Conference on Artificial Intelligence Systems*, pages 386–390. IEEE Computer Society, 2002.

[17] Kojima Kazunori, Matsuo Hiroshi, and Ishigame Maasaki. Asynchronous parallel distributed GA using elite server. In *The 2003 Congress on Evolutionary Computation*, volume 4, pages 2603–2610. IEEE, December 2003.

# A Appendix

## A.1 Board.java

```
1  package com.ankurdave.boggle;
2  import java.util.Arrays;
3  import java.util.HashSet;
4  public class Board implements Comparable<Board> {
5      class Letter {
6          private char data;
7          private boolean hasBeenHit = false;
8          private int X;
9          private int Y;
10         public Letter(char data, int X, int Y) {
11             this.data = data;
12             this.X = X;
13             this.Y = Y;
14         }
15         @Override public Letter clone() {
16             Letter thisClone = new Letter(data, X, Y);
17             thisClone.setHasBeenHit(hasBeenHit);
18             return thisClone;
19         }
20         @Override public boolean equals(Object o) {
21             Letter that = (Letter) o;
22             if (this.getData() == that.getData() && this.getX() == that.getX()
23                     && this.getY() == that.getY()) {
24                 return true;
25             } else {
26                 return false;
27             }
28         }
29         public char getData() {
30             return data;
31         }
32         public boolean getHasBeenHit() {
33             return hasBeenHit;
34         }
35         public int getX() {
36             return X;
37         }
38         public int getY() {
39             return Y;
40         }
41         public void setHasBeenHit(boolean hasBeenHit) {
42             this.hasBeenHit = hasBeenHit;
43         }
44         @Override public String toString() {
45             return "Letter[" + "data=" + getData() + "; " + "X=" + getX()
46                     + "; " + "Y=" + getY() + "; " + "hasBeenHit="
47                     + getHasBeenHit() + "]";
48         }
49         public void traverse(String soFar) {
50             // don't traverse if this has already been used
51             if (hasBeenHit) { return; }
52             soFar += data;
53             // don't traverse deeper if it doesn't begin a word so far
54             if (!dict.beginsWord(soFar.toLowerCase())) { return; }
55             // only add it to the found words if it's longer than 2 chars and is
56             // a word
```

```
57            if (soFar.length() > 2 && dict.isWord(soFar.toLowerCase())) {
58                words.add(soFar);
59            }
60            // mark this Letter as already used so adjacent Letters don't
61            // traverse back onto it
62            hasBeenHit = true;
63            // traverse each Letter around this one recursively
64            // Letter above
65            if (Y - 1 >= 0 && Y - 1 < sideLength) {
66                board[X][Y - 1].traverse(soFar);
67            }
68            // Letter below
69            if (Y + 1 >= 0 && Y + 1 < sideLength) {
70                board[X][Y + 1].traverse(soFar);
71            }
72            // Letter right
73            if (X + 1 >= 0 && X + 1 < sideLength) {
74                board[X + 1][Y].traverse(soFar);
75            }
76            // Letter left
77            if (X - 1 >= 0 && X - 1 < sideLength) {
78                board[X - 1][Y].traverse(soFar);
79            }
80            // Letter up-left
81            if (X - 1 >= 0 && X - 1 < sideLength && Y - 1 >= 0
82                    && Y - 1 < sideLength) {
83                board[X - 1][Y - 1].traverse(soFar);
84            }
85            // Letter up-right
86            if (X + 1 >= 0 && X + 1 < sideLength && Y - 1 >= 0
87                    && Y - 1 < sideLength) {
88                board[X + 1][Y - 1].traverse(soFar);
89            }
90            // Letter down-left
91            if (X - 1 >= 0 && X - 1 < sideLength && Y + 1 >= 0
92                    && Y + 1 < sideLength) {
93                board[X - 1][Y + 1].traverse(soFar);
94            }
95            // Letter down-right
96            if (X + 1 >= 0 && X + 1 < sideLength && Y + 1 >= 0
97                    && Y + 1 < sideLength) {
98                board[X + 1][Y + 1].traverse(soFar);
99            }
100           // now that this word attempt has finished, it's OK for other
101           // letters to traverse onto this one
102           hasBeenHit = false;
103       }
104   }
105   private int age = 1;
106   private Letter[][] board;
107   private Dictionary dict;
108   private char[][] grid;
109   private int score;
110   private int sideLength;
111   private HashSet<String> words = new HashSet<String>();
112   public Board(char[][] grid, Dictionary dict) {
113       assert grid.length == grid[0].length;
114       assert grid.length > 0;
115       assert dict != null;
116       sideLength = grid.length;
117       this.grid = grid;
118       this.dict = dict;
119       // make board from grid
120       board = new Letter[sideLength][sideLength];
121       for (int i = 0; i < sideLength; i++) {
122           for (int j = 0; j < sideLength; j++) {
123               board[i][j] = new Letter(grid[i][j], i, j);
124           }
```

```
125         }
126     }
127     public Board(char[][] grid, String path) {
128         assert grid.length == grid[0].length;
129         assert grid.length > 0;
130         sideLength = grid.length;
131         this.grid = grid;
132         dict = new Dictionary();
133         dict.buildDictionary(path);
134         // make board from grid
135         board = new Letter[sideLength][sideLength];
136         for (int i = 0; i < sideLength; i++) {
137             for (int j = 0; j < sideLength; j++) {
138                 board[i][j] = new Letter(grid[i][j], i, j);
139             }
140         }
141     }
142     public Board(String s, int sideLength, Dictionary dict) {
143         // TODO: error handling
144         String[] parts = s.split(" ", 2);
145         String gridS = parts[0];
146         int score = Integer.parseInt(parts[1]);
147         char[][] grid = new char[sideLength][sideLength];
148         for (int i = 0; i < sideLength; i++) {
149             for (int j = 0; j < sideLength; j++) {
150                 grid[i][j] = gridS.charAt(i * sideLength + j);
151             }
152         }
153         assert grid.length == grid[0].length;
154         assert grid.length > 0;
155         this.score = score;
156         this.grid = grid;
157         this.sideLength = sideLength;
158         this.dict = dict;
159         // make board from grid
160         board = new Letter[sideLength][sideLength];
161         for (int i = 0; i < sideLength; i++) {
162             for (int j = 0; j < sideLength; j++) {
163                 board[i][j] = new Letter(grid[i][j], i, j);
164             }
165         }
166     }
167     @Override public Board clone() {
168         Letter[][] boardClone = new Letter[sideLength][sideLength];
169         for (int i = 0; i < sideLength; i++) {
170             for (int j = 0; j < sideLength; j++) {
171                 boardClone[i][j] = board[i][j].clone();
172             }
173         }
174         Board thisClone = new Board(grid, dict);
175         return thisClone;
176     }
177     public int compareTo(Board that) {
178         if (this.getScore() > that.getScore()) {
179             return 1;
180         } else if (this.getScore() < that.getScore()) {
181             return -1;
182         } else {
183             return 0;
184         }
185     }
186     /**
187      * Traverses the Boggle board, makes a list of words, and finds the score.
188      *
189      */
190     public void generate() {
191         // on each of the letters of the board
```

```
192         // traverse the possible words recursively
193         for (int i = 0; i < sideLength; i++) {
194             for (int j = 0; j < sideLength; j++) {
195                 board[i][j].traverse("");
196             }
197         }
198         int score = 0;
199         Object[] words = getWords().toArray();
200         for (Object word : words) {
201             String wordString = (String) word;
202             int length = wordString.length();
203             // minimum length is 3
204             if (length < 3) {
205                 continue;
206             }
207             // calculate score
208             if (length == 3 || length == 4) {
209                 score += 1;
210             } else if (length == 5) {
211                 score += 2;
212             } else if (length == 6) {
213                 score += 3;
214             } else if (length == 7) {
215                 score += 5;
216             } else if (length >= 8) {
217                 score += 11;
218             }
219         }
220         this.score = score;
221     }
222     public int getAge() {
223         return age;
224     }
225     public Dictionary getDict() {
226         return dict;
227     }
228     // TODO make this automatically call generate() if score is not set. Then make generate private
229     public int getScore() {
230         return score;
231     }
232     public int getSideLength() {
233         return sideLength;
234     }
235     public HashSet<String> getWords() {
236         return words;
237     }
238     public String[] getWordsSorted() {
239         String[] wordsArray = (String[]) words.toArray(new String[words.size()]);
240         Arrays.sort(wordsArray, new ByStringLength());
241         // convert to array
242         return wordsArray;
243     }
244     public String gridToString() {
245         String s = "";
246         for (char c[] : grid) {
247             for (char d : c) {
248                 s += d;
249             }
250         }
251         return s;
252     }
253     public void incrementAge() {
254         age++;
255     }
256     /**
257      * Merges two Boggle boards randomly.<BR>
258      * Calculates the score of each board and on each character in the grid,
259      * chooses randomly between three choices for the child:
```

```
260      * <UL>
261      * <LI>use the character from the higher-scoring grid (weighted 6.6/10, or
262      * 6/10 if incestuous)
263      * <LI>use the character from the lower-scoring grid (weighted 3.3/10, or
264      * 3/10 if incestuous)
265      * <LI>use a random character (weighted 0.1/10, or 1/10 if incestuous)
266      * </UL>
267      * @param that Boggle board to merge with the calling board
268      * @return the child board
269      */
270     public Board merge(Board that) {
271         if (this.sideLength != that.sideLength) { return null; }
272         // init child
273         char[][] childGrid = new char[sideLength][sideLength];
274         // determine which one is higher or lower
275         Board higher;
276         Board lower;
277         // caller is higher
278         if (this.getScore() > that.getScore()) {
279             higher = this;
280             lower = that;
281         }
282         // parameter is higher
283         else if (that.getScore() < this.getScore()) {
284             higher = that;
285             lower = this;
286         }
287         // they are equal; choose randomly
288         else {
289             if ((int) (Math.random() * 2) == 0) {
290                 higher = this;
291                 lower = that;
292             } else {
293                 higher = that;
294                 lower = this;
295             }
296         }
297         // check if the parents are too similar
298         int sameLetters = 0;
299         for (int i = 0; i < sideLength; i++) {
300             for (int j = 0; j < sideLength; j++) {
301                 if (higher.grid[i][j] == lower.grid[i][j]) {
302                     sameLetters++;
303                 }
304             }
305         }
306         // if they are, mark it as incestuous
307         boolean incest = (float) sameLetters / (sideLength * sideLength) >= 0.85;
308         double higherChance = 6.6, lowerChance = 3.3;
309         if (incest) {
310             higherChance = 6;
311             lowerChance = 3;
312         }
313         // construct the child grid
314         double temp;
315         for (int i = 0; i < sideLength; i++) {
316             for (int j = 0; j < sideLength; j++) {
317                 temp = Math.random() * 10; // 0-9.9
318                 // higher
319                 if (temp >= 0 && temp < higherChance) {
320                     childGrid[i][j] = higher.grid[i][j];
321                 } else if (temp >= higherChance
322                         && temp < (higherChance + lowerChance)) {
323                     // 6-9
324                     childGrid[i][j] = lower.grid[i][j];
325                 } else {
326                     // 9.9-10 or 9-10
327                     childGrid[i][j] = randomLetter();
328                 }
```

```
            }
        }
        // make the child board
        Board child = new Board(childGrid, dict);
        return child;
    }
    public Board mutate(int mutationProbability) {
        assert mutationProbability >= 0 && mutationProbability <= 100;
        char[][] gridMutated = new char[sideLength][sideLength];
        for (int i = 0; i < sideLength; i++) {
            for (int j = 0; j < sideLength; j++) {
                if ((int) (Math.random() * 100) < mutationProbability) {
                    gridMutated[i][j] = randomLetter();
                } else {
                    gridMutated[i][j] = grid[i][j];
                }
            }
        }
        Board thisMutated = new Board(gridMutated, dict);
        return thisMutated;
    }
    @Override public String toString() {
        return gridToString() + " " + getScore();
    }
    private char randomLetter() {
        return (char) (Math.random() * (90 - 65 + 1) + 65);
    }
}
```

## A.2 BoardTester.java

```
package com.ankurdave.boggle;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class BoardTester {
    static char[][] grid;
    static String gridImage = "";
    static Scanner in = new Scanner(System.in);
    static int SIDE_LENGTH;
    static Scanner tempIn;
    public static void main(String[] args) {
        // need at least 1 argument
        if (args.length < 1) {
            System.out
                    .println("Usage: java BoggleTester dictionaryPath [sideLength [gridPath]]");
            System.exit(-1);
        }
        // first argument: path of dictionary file
        String path = args[0];
        // second argument (optional): side length
        if (args.length >= 2) {
            SIDE_LENGTH = Integer.parseInt(args[1]);
        } else {
            System.out.print("Length of a side of the Boggle board: ");
            SIDE_LENGTH = in.nextInt();
        }
        // third argument (optional):
        if (args.length >= 3) {
            // can be either a path or a -
            // if it's a -, prompt the user for the board
            // otherwise, read it in from the given path
            if (args[2].equals("-")) {
                tempIn = new Scanner(System.in);
                System.out.println("Enter a " + SIDE_LENGTH + "x" + SIDE_LENGTH
                        + " Boggle board:");
            } else {
```

```
37                  try {
38                      tempIn = new Scanner(new File(args[2]));
39                  }
40                  catch (FileNotFoundException e) {
41                      System.out.println("File " + path + " not found!");
42                      System.exit(-1);
43                  }
44              }
45              String temp;
46              grid = new char[SIDE_LENGTH][SIDE_LENGTH];
47              for (int i = 0; i < SIDE_LENGTH; i++) {
48                  temp = tempIn.nextLine();
49                  for (int j = 0; j < temp.length(); j++) {
50                      grid[i][j] = temp.charAt(j);
51                  }
52              }
53          } else {
54              // make a random grid
55              System.out.println("Randomly generated Boggle board:");
56              grid = new char[SIDE_LENGTH][SIDE_LENGTH];
57              gridImage = "";
58              for (int i = 0; i < SIDE_LENGTH; i++) {
59                  for (int j = 0; j < SIDE_LENGTH; j++) {
60                      grid[i][j] = (char) (Math.random() * (90 - 65 + 1) + 65);
61                      gridImage += grid[i][j] + " ";
62                  }
63                  gridImage += "\n";
64              }
65              // show the user the grid
66              System.out.println(gridImage);
67          }
68          // make the Boggle board from the above information
69          Board board = new Board(grid, path);
70          board.generate();
71
72          String[] words = board.getWordsSorted();
73
74          for (String word : words) {
75              System.out.println(word);
76          }
77
78          System.out.println(board);
79      }
80  }
```

## A.3 ByBoardScore.java

```
1  package com.ankurdave.boggle;
2  import java.util.Comparator;
3  public class ByBoardScore implements Comparator<String> {
4      public int compare(String s1, String s2) {
5          int score1 = Integer.parseInt(s1.split(" ")[2]);
6          int score2 = Integer.parseInt(s2.split(" ")[2]);
7          return score2 - score1;
8      }
9  }
```

## A.4 ByStringLength.java

```
1  package com.ankurdave.boggle;
2  import java.util.Comparator;
3  public class ByStringLength implements Comparator<String> {
4      public int compare(String s1, String s2) {
5          if (s1.length() > s2.length()) {
6              return 1;
7          } else if (s1.length() < s2.length()) {
```

```
            return -1;
        } else {
            return s1.compareTo(s2);
        }
    }
}
```

## A.5 Dictionary.java

```
package com.ankurdave.boggle;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;
public class Dictionary {
    protected ArrayList<Letter> children;
    public Dictionary() {
        this.children = new ArrayList<Letter>();
    }
    public void add(String word) {
        if (word.length() <= 0) { return; }
        for (Letter a : this.children) {
            if (a == null) {
                continue;
            }
            if (a.getData() == word.charAt(0)) {
                a.add(word.substring(1));
                return;
            }
        }
        Letter child = new Letter(word.charAt(0));
        this.children.add(child);
        child.add(word.substring(1));
        Collections.sort(this.children);
    }
    public boolean beginsWord(String word) {
        int index = Collections.binarySearch(this.children, new Letter(word
                .charAt(0)));
        // return false if child matching the first char of word does not exist
        if (index < 0) { return false; }
        // otherwise, check base case
        if (word.length() == 1) { return true; }
        // otherwise, traverse recursively
        return children.get(index).beginsWord(word.substring(1));
    }
    public void buildDictionary(String path) {
        // read dictionary file
        try {
            String temp;
            Scanner file = new Scanner(new File(path));
            while (file.hasNextLine()) {
                temp = file.nextLine().toLowerCase();
                this.add(temp);
            }
        }
        catch (FileNotFoundException e) {
            System.out.println("file " + path + " not found!");
            System.exit(-1);
        }
    }
    public boolean isWord(String word) {
        int index = Collections.binarySearch(this.children, new Letter(word
                .charAt(0)));
        // return false if child matching the first char of word does not exist
        if (index < 0) { return false; }
        // otherwise, check base case
```

```
59          if (word.length() == 1) { return children.get(index).getEndsWord(); }
60          // otherwise, traverse recursively
61          return children.get(index).isWord(word.substring(1));
62      }
63      @Override public String toString() {
64          String s = "Dictionary[]\nchildren=";
65          for (Letter a : this.children) {
66              if (a == null) {
67                  continue;
68              }
69              s += "\n" + a;
70          }
71          return s;
72      }
73  }
74
75  class Letter extends Dictionary implements Comparable<Letter> {
76      private char data;
77      private boolean endsWord = false;
78      public Letter(char data) {
79          super();
80          this.data = Character.toLowerCase(data);
81      }
82      @Override public void add(String word) {
83          if (word.length() == 0) {
84              this.endsWord = true;
85              return;
86          }
87          if (word.length() <= 0) { return; }
88          for (Letter a : this.children) {
89              if (a == null) {
90                  continue;
91              }
92              if (a.getData() == word.charAt(0)) {
93                  a.add(word.substring(1));
94                  return;
95              }
96          }
97          Letter child = new Letter(word.charAt(0));
98          this.children.add(child);
99          child.add(word.substring(1));
100         Collections.sort(this.children);
101     }
102     public int compareTo(Letter that) {
103         if (this.getData() > that.getData()) {
104             return 1;
105         } else if (this.getData() < that.getData()) {
106             return -1;
107         } else {
108             return 0;
109         }
110     }
111     public char getData() {
112         return this.data;
113     }
114     public boolean getEndsWord() {
115         return this.endsWord;
116     }
117     @Override public String toString() {
118         String s = "Letter[data=" + this.data + "; endsWord=" + this.endsWord
119                 + "]\nchildren=";
120         for (Letter a : this.children) {
121             if (a == null) {
122                 continue;
123             }
124             s += "\n" + a;
125         }
```

```
126         return s;
127     }
128 }
```

## A.6 DictionaryTester.java

```
1  package com.ankurdave.boggle;
2  import java.util.Scanner;
3  public class DictionaryTester {
4      public static void main(String[] args) {
5          Dictionary d = new Dictionary();
6          d.buildDictionary("words.txt");
7          Scanner in = new Scanner(System.in);
8          String input = "";
9          while (!(input.equals("QUIT"))) {
10             System.out.print("Enter a word: ");
11             input = in.nextLine();
12             System.out.println("\"" + input + "\""
13                     + (d.isWord(input) ? " is a word." : " is not a word."));
14         }
15     }
16 }
```

## A.7 GenerationEmptyException.java

```
1  package com.ankurdave.boggle;
2  /**
3   * Exception thrown when there are insufficient Boggles in the current
4   * generation to evolve to the next generation.
5   * @author ankur
6   */
7  public class GenerationEmptyException extends Exception {
8      public static final long serialVersionUID = 0;
9      private String message;
10     public GenerationEmptyException(String message) {
11         this.message = message;
12     }
13     @Override public String toString() {
14         return "GenerationEmptyException: " + message;
15     }
16 }
```

## A.8 GeneticClient.java

```
1  package com.ankurdave.boggle;
2  import java.io.BufferedReader;
3  import java.io.IOException;
4  import java.io.InputStreamReader;
5  import java.io.PrintWriter;
6  import java.net.Socket;
7  import java.util.regex.Matcher;
8  import java.util.regex.Pattern;
9  public class GeneticClient {
10     private BufferedReader in;
11     private PrintWriter out;
12     private String serverAddress;
13     private int serverPort;
14     private static final Pattern pair = Pattern
15             .compile("^\\s*([\\w-]+)\\s*:\\s*([\\w -]+)\\s*$");
16     private GeneticClientThread worker;
17     private Board highest;
18     private Board outboundMigrant;
19     private Boolean highestChanged = true, migrantChanged = true;
20     private Socket socket;
21     public GeneticClient(String serverAddress, int serverPort, String dictPath,
22             int sideLength, int startingPopulation, int childrenPerCouple,
23             int popCap) {
24         this.serverAddress = serverAddress;
```

```
25          this.serverPort = serverPort;
26          worker = new GeneticClientThread(dictPath, sideLength,
27                  startingPopulation, childrenPerCouple, popCap, this);
28          connect();
29      }
30      public void connect() {
31          while (true) {
32              try {
33                  socket = new Socket(serverAddress, serverPort);
34                  out = new PrintWriter(socket.getOutputStream(), true);
35                  in = new BufferedReader(new InputStreamReader(socket
36                          .getInputStream()));
37              }
38              catch (IOException e) {
39                  System.err.println("Couldn't connect to server: " + e);
40                  try {
41                      Thread.sleep(2000);
42                  }
43                  catch (InterruptedException ex) {
44                      break;
45                  }
46                  continue; // retry if failure
47              }
48              break; // terminate if success
49          }
50      }
51      public void run() {
52          worker.start();
53          try {
54              while (true) {
55                  // communicate with server
56                  if (highestChanged || migrantChanged) {
57                      giveServerOutput();
58                  }
59                  readServerInput();
60              }
61          }
62          catch (IOException e) {
63              System.err.println(e);
64          }
65          finally {
66              worker.terminate();
67              out.close();
68              try {
69                  in.close();
70              }
71              catch (IOException e) {}
72              try {
73                  socket.close();
74              }
75              catch (IOException e) {}
76          }
77      }
78      public void setHighest(Board b) {
79          if (highest == null || b.getScore() > highest.getScore()) {
80              highest = b;
81          }
82          highestChanged = true;
83      }
84      public void setOutboundMigrant(Board b) {}
85      // TODO send server the score
86      private void giveServerOutput() {
87          if (highestChanged && highest != null) {
88              highestChanged = false;
89              out.println("Highest:" + highest);
90          }
```

```
91          if (migrantChanged && outboundMigrant != null) {
92              migrantChanged = false;
93              out.println("Migrant:" + outboundMigrant);
94          }
95          // end of transmission
96          out.println();
97          out.flush();
98      }
99      private void readServerInput() throws IOException {
100         String line;
101         Matcher m;
102         while (true) {
103             // for each line in the input
104             line = in.readLine();
105             if (line == null) {
106                 throw new IOException("Server closed connection");
107             } else if (line.isEmpty()) {
108                 break;
109             }
110             // try to find data in it
111             m = pair.matcher(line);
112             if (m.matches()) {
113                 storeServerData(m.group(1), m.group(2));
114                 if (m.group(1).equalsIgnoreCase("reset")) {
115                     // throw away the rest of the message
116                     do {
117                         line = in.readLine();
118                     } while (!line.isEmpty());
119                 }
120             }
121         }
122     }
123     private void storeServerData(String name, String value) {
124         if (name.equalsIgnoreCase("migrant")) {
125             Board migrant = new Board(value, worker.getSideLength(), worker
126                     .getDictionary());
127             worker.setInboundMigrant(migrant);
128         } else if (name.equalsIgnoreCase("pop-cap")) {
129             worker.setPopCap(Integer.parseInt(value));
130         } else if (name.equalsIgnoreCase("reset")) {
131             highest = null;
132             worker.reset();
133         }
134     }
135 }
```

## A.9 GeneticClientTester.java

```
1 package com.ankurdave.boggle;
2 public class GeneticClientTester {
3     public static void main(String[] args) {
4         new GeneticClient("192.168.1.123", 4444, "words.txt", 4, 20, 5, 20)
5                 .run();
6     }
7 }
```

## A.10 GeneticClientThread.java

```
1 package com.ankurdave.boggle;
2 public class GeneticClientThread extends Thread {
3     private Population bp;
4     private Dictionary dict;
5     private int sideLength, startingPopulation, startingChildrenPerCouple,
6             startingPopCap;
7     private GeneticClient manager;
8     private Boolean resetRequested = false, terminateRequested = false;
9     private Board inboundMigrant;
```

```
10      public GeneticClientThread(String dictPath, int sideLength,
11              int startingPopulation, int childrenPerCouple, int popCap,
12              GeneticClient manager) {
13          this.sideLength = sideLength;
14          this.startingPopulation = startingPopulation;
15          this.startingChildrenPerCouple = childrenPerCouple;
16          this.startingPopCap = popCap;
17          this.manager = manager;
18          // init dictionary
19          dict = new Dictionary();
20          dict.buildDictionary(dictPath);
21          // init population
22          bp = new Population(sideLength, this.startingPopulation,
23                  startingChildrenPerCouple, startingPopCap, dict);
24      }
25      @Override public void run() {
26          while (true) {
27              try {
28                  if (inboundMigrant != null) {
29                      bp.add(inboundMigrant);
30                      inboundMigrant = null;
31                  }
32                  bp.evolve();
33                  System.out.println(bp);
34                  for (Board b : bp.getCurrentGeneration()) {
35                      System.out.println(b);
36                  }
37                  System.out.println();
38                  if (resetRequested) {
39                      System.out.println("Reset");
40                      bp = new Population(sideLength, this.startingPopulation,
41                              startingChildrenPerCouple, startingPopCap, dict);
42                      inboundMigrant = null;
43                      resetRequested = false;
44                      continue;
45                  }
46                  if (terminateRequested) {
47                      break;
48                  }
49                  // communicate with manager
50                  manager.setHighest(bp.highest());
51                  // TODO analyze migration algorithm
52                  manager.setOutboundMigrant(Util.weightedRandomFromList(bp
53                          .getCurrentGeneration()));
54              }
55              catch (GenerationEmptyException e) {
56                  System.err.println(e);
57                  break;
58              }
59          }
60      }
61      public void terminate() {
62          terminateRequested = true;
63      }
64      public int getSideLength() {
65          return sideLength;
66      }
67      public Dictionary getDictionary() {
68          return dict;
69      }
70      public void setInboundMigrant(Board migrant) {
71          inboundMigrant = migrant;
72      }
73      // TODO analyze variable pop cap
74      public void setPopCap(int popCap) {
75          bp.setPopCap(popCap);
76      }
```

```
77      public void reset() {
78          resetRequested = true;
79      }
80  }
```

## A.11 HillClimber.java

```
1  package com.ankurdave.boggle;
2  public class HillClimber {
3      public static void main(String[] args) {
4          Dictionary dict = new Dictionary();
5          dict.buildDictionary("words.txt");
6          for (int trialNum = 0; trialNum < 100; trialNum++) {
7              // create the starting Boggle
8              char[][] start = Util.randomGrid(4);
9              Board current = new Board(start, dict);
10             current.generate(); // find score of current Boggle
11             Board trial;
12             // start the timer
13             long startTime = System.currentTimeMillis();
14             // begin hill climbing
15             while (current.getScore() < 3500 && (System.currentTimeMillis() - startTime) < 1000000)
                       {
16                 trial = current.mutate(10);
17                 trial.generate();
18                 if (trial.getScore() > current.getScore()) {
19                     current = trial;
20                     System.err.println(current);
21                 }
22             }
23             // stop the timer
24             long stopTime = System.currentTimeMillis();
25             System.out.println(current + " " + (stopTime - startTime));
26         }
27     }
28 }
```

## A.12 Population.java

```
1  package com.ankurdave.boggle;
2  import java.util.ArrayList;
3  import java.util.Collections;
4  import java.util.HashSet;
5  public class Population {
6      private int childrenPerCouple;
7      private ArrayList<Board> currentGeneration;
8      private Dictionary dict;
9      private int generation;
10     private int popCap;
11     private int sideLength;
12     public Population(int sideLength, int startingPopulation,
13             int childrenPerCouple, int popCap, Dictionary dict) {
14         assert sideLength > 0;
15         assert startingPopulation >= 0;
16         assert childrenPerCouple >= 0;
17         assert popCap >= startingPopulation;
18         assert dict != null;
19         // copy params to object fields
20         this.sideLength = sideLength;
21         this.childrenPerCouple = childrenPerCouple;
22         this.popCap = popCap;
23         this.dict = dict;
24         // make the first generation
25         generation = 1;
26         currentGeneration = new ArrayList<Board>();
27         Board temp;
28         for (int i = 0; i < startingPopulation; i++) {
29             temp = new Board(Util.randomGrid(sideLength), dict);
```

```
30            temp.generate();
31            currentGeneration.add(temp);
32        }
33    }
34    public void add(Board boggle) {
35        assert boggle != null;
36        currentGeneration.add(boggle);
37    }
38    public void add(char[][] grid) {
39        assert grid.length == sideLength;
40        currentGeneration.add(new Board(grid, dict));
41    }
42    public int averageScore() throws GenerationEmptyException {
43        if (numBoggles() <= 0) { throw new GenerationEmptyException(
44                "not enough Boggles in current generation to find average"); }
45        int counter = 0;
46        int total = 0;
47        for (Board b : currentGeneration) {
48            counter++;
49            total += b.getScore();
50        }
51        return total / counter;
52    }
53    public void evolve() throws GenerationEmptyException {
54        if (numBoggles() <= 1) { throw new GenerationEmptyException(
55                "not enough Boggles in current generation to evolve"); }
56        // sort the current generation by score
57        Collections.sort(currentGeneration);
58        // make children
59        ArrayList<Board> children = new ArrayList<Board>();
60        Board parent1;
61        Board parent2;
62        Board child;
63        for (int i = 0; i < this.numBoggles() - 1; i += 2) {
64            // get the next two parents
65            parent1 = currentGeneration.get(i);
66            parent2 = currentGeneration.get(i + 1);
67            // mate them childrenPerCouple times
68            for (int j = 0; j < childrenPerCouple; j++) {
69                child = parent1.merge(parent2);
70                children.add(child);
71            }
72        }
73        // do elitist selection
74        // highest() seems to clone the object or something and so age is not
75        // preserved
76        Board highest = currentGeneration.get(currentGeneration.size() - 1);
77        children.add(highest);
78        // make sure there are no duplicates
79        HashSet<String> uniqueGrids = new HashSet<String>();
80        for (int i = 0; i < children.size(); i++) {
81            Board b = children.get(i);
82            if (uniqueGrids.contains(b.gridToString())) {
83                children.remove(b);
84                continue; // skip scoring duplicate boards
85            } else {
86                uniqueGrids.add(b.gridToString());
87            }
88            // score each unique board
89            b.generate();
90        }
91        Collections.sort(children);
92        // make sure number of children <= popCap by removing the worst few
93        Collections.sort(children);
94        while (children.size() > popCap) {
95            children.remove(0);
96        }
```

```
97          // apply changes
98          currentGeneration.clear();
99          currentGeneration.addAll(children);
100         // record generation change
101         generation++;
102     }
103     public ArrayList<Board> getCurrentGeneration() {
104         return currentGeneration;
105     }
106     public int getGeneration() {
107         return generation;
108     }
109     public int getPopCap() {
110         return popCap;
111     }
112     public Board highest() throws GenerationEmptyException {
113         if (numBoggles() <= 0) { throw new GenerationEmptyException(
114                 "not enough Boggles in current generation to find maximum"); }
115         return Collections.max(currentGeneration);
116     }
117     public Board lowest() throws GenerationEmptyException {
118         if (numBoggles() <= 0) { throw new GenerationEmptyException(
119                 "not enough Boggles in current generation to find minimum"); }
120         return Collections.min(currentGeneration);
121     }
122     public int numBoggles() {
123         return currentGeneration.size();
124     }
125     public Board random() {
126         return currentGeneration.get((int) (Math.random() * numBoggles()));
127     }
128     public Board removeHighest() throws GenerationEmptyException {
129         if (numBoggles() <= 0) { throw new GenerationEmptyException(
130                 "not enough Boggles in current generation to find maximum"); }
131         Board highest = Collections.max(currentGeneration);
132         currentGeneration.remove(highest);
133         return highest;
134     }
135     public void setPopCap(int popCap) {
136         this.popCap = popCap;
137     }
138     @Override public String toString() {
139         String s = null;
140         try {
141             s = generation + " " + highest().getScore() + " " + averageScore()
142                     + " " + lowest().getScore();
143         }
144         catch (GenerationEmptyException e) {
145             System.err.println(e);
146         }
147         return s;
148     }
149 }
```

## A.13 Server.java

```
1  package com.ankurdave.boggle;
2  import java.io.IOException;
3  import java.net.ServerSocket;
4  import java.net.Socket;
5  import java.util.ArrayList;
6  import java.util.Collections;
7  /**
8   * Server component of DistBoggle. Manages BoggleClients.
9   * @author ankur
10  */
11 public class Server {
```

```
12      private int trials = 1;
13      private static final int DEFAULT_POP_CAP = 20;
14      private static final int POP_CAP_RANGE = 0;
15      private int curClientID = 0;
16      private Dictionary dict;
17      private Board highest;
18      private ServerSocket socket;
19      private long startTime;
20      private ArrayList<ServerThread> threads = new ArrayList<ServerThread>();
21      public Server(int port) {
22          // create the socket
23          socket = null;
24          try {
25              socket = new ServerSocket(port);
26          }
27          catch (IOException e) {
28              System.err.println("Could not listen on port " + port);
29              System.exit(1);
30          }
31          // create the dictionary
32          dict = new Dictionary();
33          dict.buildDictionary("words.txt");
34      }
35      // TODO analyze migrant allocation algorithm
36      public synchronized void addMigrant(Board migrant, ServerThread caller) {
37          Collections.sort(threads);
38          for (ServerThread c : threads) {
39              if ((c.getMigrant() == null || c.getMigrant().getScore() < migrant
40                      .getScore())
41                      && caller != c) {
42                  c.setMigrant(migrant);
43                  break;
44              }
45          }
46      }
47      public synchronized Dictionary getDictionary() {
48          return dict;
49      }
50      // TODO analyze variable pop cap
51      public synchronized int getPopCapForClient(int clientID) {
52          Collections.sort(threads);
53          if (threads.size() == 1) { return DEFAULT_POP_CAP; }
54          for (int i = 0; i < threads.size(); i++) {
55              if (threads.get(i).getId() == clientID) { return (DEFAULT_POP_CAP + POP_CAP_RANGE / 2)
56                      - i * (POP_CAP_RANGE / (threads.size() - 1)); }
57          }
58          return DEFAULT_POP_CAP;
59      }
60      /**
61       * Starts listening for clients. Starts a new thread for each client. Never
62       * returns.
63       */
64      public void listen() {
65          try {
66              while (true) {
67                  Socket s = socket.accept();
68                  ServerThread serverThread = new ServerThread(this, s,
69                          curClientID++);
70                  threads.add(serverThread);
71                  serverThread.start();
72                  // start the timer
73                  if (startTime == 0) { // if not already started
74                      startTime = System.currentTimeMillis();
75                  }
76              }
77          }
78          catch (IOException e) {
```

```
79              System.err.println("Error while listening: " + e);
80              System.exit(1);
81          }
82      }
83      public synchronized void setHighest(Board b) {
84          if (highest == null || b.getScore() > highest.getScore()) {
85              highest = b;
86              System.err.println(highest);
87              if (highest.getScore() >= 3500) {
88                  reset();
89                  return;
90              }
91          }
92
93          if (System.currentTimeMillis() - startTime >= 1000000) {
94              reset();
95          }
96
97          if (trials >= 100) {
98              System.exit(0);
99          }
100     }
101     public synchronized void removeThread(ServerThread t) {
102         threads.remove(t);
103     }
104     private void reset() {
105         trials++;
106         System.out.println(highest + " "
107                 + Long.toString(System.currentTimeMillis() - startTime));
108         // reset state
109         highest = null;
110         for (ServerThread t : threads) {
111             t.reset();
112         }
113         startTime = System.currentTimeMillis(); // restart timer
114     }
115 }
```

## A.14 ServerTester.java

```
1  package com.ankurdave.boggle;
2  public class ServerTester {
3      public static void main(String[] args) {
4          new Server(4444).listen();
5      }
6  }
```

## A.15 ServerThread.java

```
1  package com.ankurdave.boggle;
2  import java.io.BufferedReader;
3  import java.io.IOException;
4  import java.io.InputStreamReader;
5  import java.io.PrintWriter;
6  import java.net.Socket;
7  import java.util.regex.Matcher;
8  import java.util.regex.Pattern;
9  /**
10  * Thread started by BoggleServer to handle BoggleClients.
11  * @author ankur
12  */
13 public class ServerThread extends Thread implements Comparable<ServerThread> {
14     private static final Pattern pair = Pattern
15             .compile("^\\s*([\\w-]+)\\s*:\\s*([\\w -]+)\\s*$");
16     private int clientID;
17     private BufferedReader in;
18     private Board migrant;
19     private PrintWriter out;
```

```
20      private int score;
21      private Server server;
22      /**
23       * The socket used to communicate with the client.
24       */
25      private Socket socket;
26      public ServerThread(Server server, Socket socket, int clientID) {
27          super("BoggleServerThread");
28          this.server = server;
29          this.socket = socket;
30          this.clientID = clientID;
31      }
32      public int compareTo(ServerThread that) {
33          return that.getScore() - this.getScore(); // descending order by
34          // default
35      }
36      public Board getMigrant() {
37          return migrant;
38      }
39      public int getScore() {
40          return score;
41      }
42      public void reset() {
43          out.println("Reset: yes");
44          // end the transmission
45          out.println();
46          out.flush();
47          score = 0;
48          migrant = null;
49      }
50      @Override public void run() {
51          try {
52              // init the IO facilities for the socket
53              out = new PrintWriter(socket.getOutputStream(), true);
54              in = new BufferedReader(new InputStreamReader(socket
55                      .getInputStream()));
56              while (true) {
57                  readClientInput();
58                  giveClientOutput();
59              }
60          }
61          catch (IOException e) {
62              System.err.println(e);
63          }
64          finally {
65              server.removeThread(this);
66              out.close();
67              try {
68                  in.close();
69              }
70              catch (IOException e) {}
71              try {
72                  socket.close();
73              }
74              catch (IOException e) {}
75          }
76      }
77      public void setMigrant(Board migrant) {
78          this.migrant = migrant;
79      }
80      private void giveClientOutput() {
81          // give the migrant if there is one
82          if (migrant != null) {
83              out.println("Migrant: " + migrant);
84              migrant = null;
85          }
86          // give the new pop cap
```

```
87          out.println("Pop-Cap: " + server.getPopCapForClient(clientID));
88          // end the transmission
89          out.println();
90          out.flush();
91      }
92      private void readClientInput() throws IOException {
93          String line;
94          Matcher m;
95          while (true) {
96              // for each line in the input
97              line = in.readLine();
98              if (line == null) {
99                  throw new IOException("Client closed connection");
100             } else if (line.isEmpty()) {
101                 break;
102             }
103             // try to find data in it
104             m = pair.matcher(line);
105             if (m.matches()) {
106                 storeClientData(m.group(1), m.group(2));
107             }
108         }
109     }
110     private void storeClientData(String name, String value) {
111         if (name.equalsIgnoreCase("score")) {
112             score = Integer.parseInt(value);
113         } else if (name.equalsIgnoreCase("migrant")) {
114             Board migrant = new Board(value, 4, server.getDictionary());
115             server.addMigrant(migrant, this);
116         } else if (name.equalsIgnoreCase("highest")) {
117             Board highest = new Board(value, 4, server.getDictionary());
118             server.setHighest(highest);
119         }
120     }
121 }
```

## A.16 Util.java

```
1  package com.ankurdave.boggle;
2  import java.util.List;
3  public class Util {
4      /**
5       * Creates a character grid filled with random uppercase letters.
6       * @param sideLength length of one side of the random grid
7       * @return the random grid
8       */
9      public static char[][] randomGrid(int sideLength) {
10         char[][] temp = new char[sideLength][sideLength];
11         for (int i = 0; i < sideLength; i++) {
12             for (int j = 0; j < sideLength; j++) {
13                 // rand 65-90
14                 temp[i][j] = (char) (Math.random() * (90 - 65 + 1) + 65);
15             }
16         }
17         return temp;
18     }
19     /**
20      * Taken from http://www.perlmonks.org/?node_id=158482 How it works: on the
21      * first iteration, the if will always be true, establishing the first
22      * Boggle as the random one (unless the first boggle scores 0). On
23      * successive iterations, every other Boggle gets a weighted chance at
24      * replacing the previous Boggle.
25      */
26     public static Board weightedRandomFromList(List<Board> list) {
27         int sum = 0;
28         Board result = null;
29         for (Board b : list) {
30             if (Math.random() * (sum += b.getScore() * b.getScore()) < b
```

```
31                      .getScore()
32                      * b.getScore()) {
33                  result = b;
34              }
35          }
36          assert result != null;
37          return result;
38      }
39  }
```