EVOLUTION OF THE SOUTHERN PINE BEETLE LEGACY SIMULATION MODEL "SPBMODEL" USING GENETIC ALGORITHMS.

Sarah Melissa Satterlee

Thesis submitted to the Faculty of Virginia Polytechnic
Institute and State University in partial fulfillment of the
requirements for the degree of

Master of Science
in
Life Sciences

October 10, 2002
Blacksburg, VA

Keywords: genetic algorithms, southern pine beetle, simulation, model

EVOLUTION OF THE SOUTHERN PINE BEETLE LEGACY SIMULATION MODEL "SPBMODEL" USING GENETIC ALGORITHMS.

Sarah Melissa Satterlee

Abstract

SPBMODEL, a legacy southern pine beetle (SPB) simulation model, was translated into a new Java^{TM} model called Javahog. The Javahog output was verified to be essentially identical to SPBMODEL output by means of standard and paired t-tests. Javahog was placed online and is currently accessible via a servlet.

Genetic algorithms (GAs) were applied to the Javahog model. GAs are a type of optimization heuristic that operate as an analog to evolution. GAs "evolve" a very good solution to a complex problem. In this case, GAs were intended to evolve a very good version of SPBMODEL. GAs were applied in part to improve upon the SPBMODEL design, and in part to demonstrate that GAs are effective tools for recalibrating legacy simulation models. Beyond simply recalibrating model parameters, the GA was used to select optimal functional forms for the development rates of each SPB life stage.

The GA evolved a model that performed better than SPBMODEL at predicting observed field data, according to a balanced fitness function and according to sums of squared errors. However, from a visual comparison of the output of both models versus observed field data, neither model achieved satisfactory performance.

Acknowledgments

I'd like to acknowledge my advisor, Dr. Nick Stone, for all of his help in guiding me through this complex project. I deeply value his advice, encouragement, and patience. I also wish to thank my committee members, Dr. Fred Stephen, Dr. Scott Salom, and Dr. Carlyle Brewster for all of their advice and support.

The members of both Dr. Stone's research group at Virginia Tech and Dr. Stephen's research group at the University of Arkansas were extremely helpful to me. Dr. Ben Cline from Dr. Stone's research group was practically a second advisor to me, and also located the software and maintained the servers used in this project. Jim Venuto shared much in the way of useful programming advice and much-needed humor. From Dr. Stephen's research group, Dr. Vaughn Salisbury went above and beyond the call of duty to get documentation and data to me. Dana Kinney and the rest of Dr. Stephen's group were also extraordinarily helpful in obtaining data. Dr. Marita Lih and Dr. Paul Jeng provided useful information about the design of the SPBMODEL. In general, I'd like to thank the entomology departments at Virginia Tech and at the University of Arkansas for their support.

The funding for this project was provided as a grant from the University of Arkansas, Arkansas Forest Research Center to Virginia Tech, entitled "Modification and Update of the Southern Pine Beetle Model SPBMODEL."

Finally, I'd like to thank my husband, James Maxted, without whom this research would have been impossible. James provided indispensable programming and debugging advice, as well as invaluable emotional support and love.

Table of Contents

List of Figures

List of Tables

Chapter 1 : The Research Problem

1.1 Introduction

The purpose for this research was to enhance the usability and to some extent the accuracy of the simulation model SPBMODEL, making it more accessible to researchers, clearer in its form and documentation, and more consistent with modern computing environments. SPBMODEL was built in 1979 to predict short-term population trends of southern pine beetle (*Dendroctonus frontalis* Zimmermann (Coleoptera: Scolytidae)) and subsequent damage to a forest stand. It was written and revised over many years by researchers working with Dr. Fred Stephen and Dr. Hamdy Taha at the University of Arkansas, becoming more complex with each revision. As is the case with many legacy models, much of the documentation and code for SPBMODEL has been lost over the years, making further modifications and additions difficult. At the same time, SPBMODEL has become widely used, but to remain a viable research tool, it must be updated and its internal structure elucidated or simplified to meet current demands in the southern pine beetle research community.

At least as important as the issue of usability of a model, however, is the issue of accuracy. As more data are collected from a broader environment, errors and limitations become more apparent. As Curry and Feldman (1987) noted,"even a 'good' model must be recalibrated for different situations." SPBMODEL was originally calibrated using data from Arkansas, Louisiana, Mississippi, and Texas, from the years 1976-1985 (Lih and Stephen, 1989). However, more data are currently available, gathered as recently as summer 2001, from Alabama, Georgia, North Carolina, Tennessee, and Virginia. Preliminary testing of SPBMODEL against data from both recent and historical SPB infestation spots indicated that recalibration of the model was needed.

Recalibration of a model is much more than a trivial exercise. Before the work reported here, four previous Ph.D. dissertations have addressed the recalibration of SPBMODEL (Ghosh, 1983; Jeng, 1994; Lih, 1985; Motamedi, 1981). Simulation models in general require this lengthy recalibration process for new site conditions because "models lack the structure to set parameters automatically" (Curry and Feldman, 1987).

An automated method of setting parameters would greatly reduce the amount of time spent in recalibration of SPBMODEL as data from new site conditions become available.

Automated parameter-setting may be performed by any of several computerized optimization techniques, but several researchers have recently had success using genetic algorithms to determine optimal parameters in agricultural and environmental models (e.g., Jacucci et al., 1995; Sequiera et al., 1994; Veith, 2002). Genetic algorithms (GAs) are search procedures based on the mechanism of natural selection (Goldberg, 1989). In comparison to other computerized optimization techniques, such as hill-climbing and simulated annealing, genetic algorithms have been shown to perform consistently better (Koza, 1992).

Genetic algorithms have the power to optimize a simulation model beyond just recalibrating parameter values; they may actually be used to improve the design of the model. This process has been called "genetic programming," in which GA's are used to alter the architecture of a program by adding, changing, or removing program functions (Koza et al., 2000).

Koza's work in evolving programs required vast computing power, so in this research, the question was asked whether much simpler applications of GAs could also be used to suggest structural improvements to a simulation model. In particular, this work addressed whether GAs can be used to improve an insect population model's behavior by selecting the optimal functional forms for relating development rates to temperature. In general, if a simulation model has inherent design flaws, recalibration of parameters will not be enough to improve the model's accuracy. In such cases, GAs may be used to explore alternative simulation model designs.

A GA can only modify a simulation model if the model's structure and code are accessible to the GA. Parameters and functions must be changeable, and the model must be testable with observed data. This is particularly difficult with legacy code, written in languages not suited to dynamic changes and with functions not separated into discrete modules of code.

Both to facilitate the future use of SPBMODEL and to make possible the use of GAs in exploring automated improvements, the first step of this research was to reengineer SPBMODEL into a version with an appropriately modular design, without changing the model's predictive behavior. Making the model accessible for future research purposes included restructuring the model into an object-oriented format, thoroughly documenting the model, and placing the model on the World-Wide Web.

Given the new incarnation of SPBMODEL, an experiment using GAs was then conducted to test whether the model structure and parameters could be usefully changed through an automated process.

1.2 SPBMODEL

1.2.1 Need for SPBMODEL

SPBMODEL was built in 1979 at the University of Arkansas. Its construction was precipitated by severe outbreaks of southern pine beetle in the early 1970's (Payne, 1980). The southern pine beetle had long been noted as a native pest of some importance, and there had been an apparent upward trend of southern pine beetle damage during the decades leading up to the 1970's epidemic (Hicks, 1980; Payne, 1980). The increasing damage may have been due in part to the changing landscape of southeastern forests: many of the southeastern forests in the last hundred years were managed to optimize softwood production, which increased the food source of the southern pine beetle (Hicks, 1980). In a natural, unmanaged state, hardwoods would compose a much larger portion of southeastern forests. High forest stand density, sometimes present in sites with optimized softwood production, is also associated with southern pine beetle outbreaks (Hedden, 1978).

After a few years of recurring southern pine beetle epidemics, the US government was compelled to take serious measures to protect its southern forests (Thatcher, 1980). The federal government began the Expanded Southern Pine Beetle Research and Applications Program (ESPBRAP), a $12 million, 6-year "accelerated research and development effort" (Thatcher, 1980). This program led to extensive research on methods of SPB management, including pheromone isolation, pesticide development, supplementation of natural enemies, and construction of economic and population dynamics models (Thatcher, 1980). SPBMODEL was one of the models designed to help a forester or landowner determine whether an infestation would be worth treating, or whether the infestation would collapse on its own (Lih and Stephen, 1989).

The primary method of treating an SPB infestation has traditionally involved cutting down the infested trees, sometimes followed by removing the logs, or in rare circumstances, burning or spraying the logs (Billings, 1980). Control methods may also involve the creation of a buffer strip by felling a swath of healthy trees around the infestation (*ibid.*). Although both the healthy and infested logs may be sold, salvaging the timber from an infestation is sometimes impractical and uneconomical due to the small, scattered nature of many infestations (Thatcher, 1980). In the event that infestations are large and continuous, as in an epidemic year, another problem can occur: the timber market can be so flooded with infested timber that salvaging yields little profit (Pase, 2002).

Treatment is expensive, but the cost of failing to treat an infestation may be even more expensive to a landowner, because the infestation may spread and cause even greater damage. The expense of treatment can be avoided, however, if it can be determined that the infestation will not spread. Thus, there was, and still is today, a great demand for an accurate predictor of the behavior of a southern pine beetle infestation. Of the southern pine beetle prediction models (see Chapter 2), SPBMODEL is one of the most well-known, validated on a wealth of data collected over a period of nine years (Lih and Stephen, 1989). Many copies have been distributed to the Forest Service for use in the field (*ibid.*).

Another important application of SPBMODEL is its use as a descriptor of SPB ecology. The model may be used to test ecological and management hypotheses (Stephen and Lih, 1985). For example, the Hog Model has been used to study the impacts of predators and parasitoids on the SPB population (Stephen and Lih, 1985), and to examine the influence of food sources for parasitoids on SPB population dynamics (Jeng, 1994).

1.2.2 Southern Pine Beetle Life Cycle

SPBMODEL predicts the extent of southern pine beetle damage to a forest by deterministically simulating the life cycle of the southern pine beetle. The following summary is based on an extensive review article by T.L. Payne (1980) on the life cycle and habits of SPB.

The southern pine beetle (SPB) is a multivoltine pest, undergoing 3 to 9 generations per year according to latitude and temperature. Its primary hosts in the United States are loblolly (*Pinus taeda* L.) and shortleaf pine (*Pinus echinata* Mill.), but it can also attack and kill most southeastern pines. The attack cycle begins as a group of adult beetles converge and attack a tree *en masse*. If too few beetles attack a tree, the resin pressure inside a tree will "pitch out" attacking adult SPB, and the colonization of the tree will be unsuccessful.

The life cycle of SPB (Figure 1.1) begins when adult females attack a tree. Males are attracted to a tree only after the tree has been successfully attacked by the females. The attack process is regulated by a complex of southern pine beetle pheromones acting in conjunction with host odor. Once the tree has been successfully colonized, SPB adult females mate and lay eggs. Mated adults can reemerge from the tree and continue to attack more trees. The eggs are laid in galleries in the cambium layer of the bark. After hatching, the larvae feed in phloem tissue, where they undergo most of their development, totaling four instars. The mature larvae tunnel further outward and pupate the outer bark. Newly matured adults, termed "callow adults," will remain in the outer bark until their cuticle has darkened and until conditions are appropriate for emergence. Emerging SPB will attack other trees.

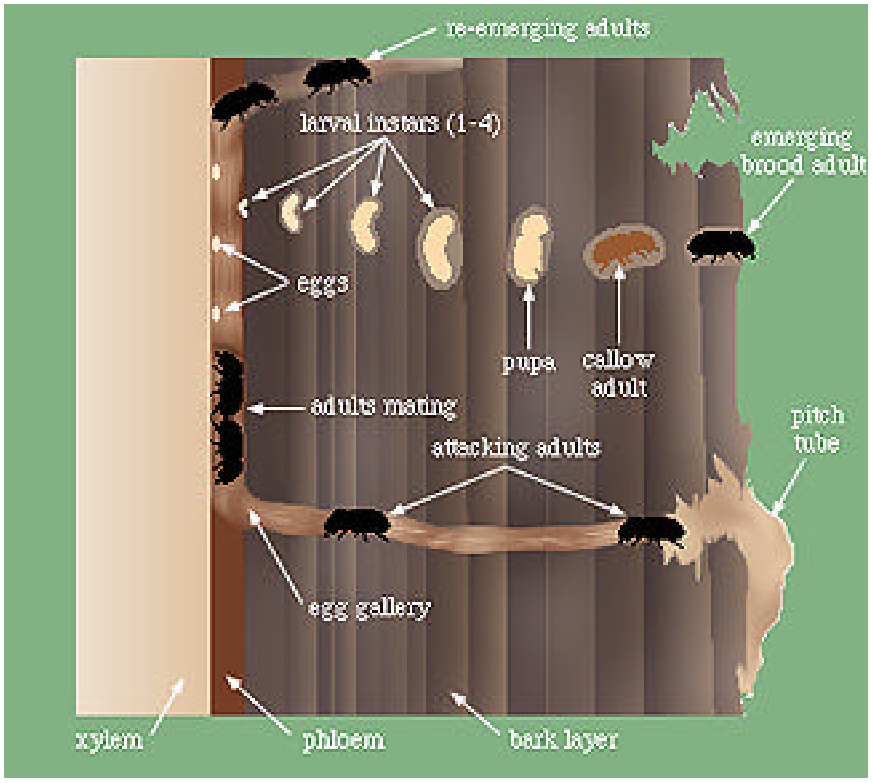


Figure 1.1 The life cycle of the southern pine beetle, as it occurs in the bark of a southern pine tree (from Salom, 1997; used with permission).

1.2.3 SPBMODEL Design

SPBMODEL's structure was based on a simplified version of the SPB life cycle (Figure 1.2). SPBMODEL can estimate the initial population of an SPB infestation from user-provided inputs: primarily the number of initially infested trees, and optionally the numbers of trees infested with each life stage. Given the initial conditions and expressions defining the mortality, fecundity (egg production rate), and development rates, SPBMODEL can estimate the changes in SPB population levels over time, and thus predict SPB damage. For simplicity, the effects of stand conditions are not represented in Figure 1.2, but these effects are key in determining population trends. Stand conditions and date impact mortality rates, and temperature influences fecundity and development rates.

1.2.3.1 Temperature

Because of the sensitivity of the model to temperature, particularly of the insect development rates to temperature, temperature estimation was an important part of SPBMODEL. SPBMODEL was intended for application throughout the southeastern U.S., so an algorithm developed by J. Ballard (1974) was used to calculate hourly temperatures based on daily maximum and minimum ambient temperatures, latitude, longitude, and time of year. The algorithm used a set of cosine functions to represent the hourly patterns, with frequencies and amplitudes determined from daily maximum and minimum temperatures and the time of sunrise. Each day's minimum temperature was assumed to occur one hour after sunrise, and the maximum temperature at three hours after noon (Ghosh, 1983). The time of sunrise was calculated from a Fourier equation of time and an estimate of the declination of the sun based on the day of the year and the approximate latitude and longitude for a particular U.S. state (Ghosh, 1983).

1.2.3.2 Development

For temperature-dependent development, the proportion of development *d* accumulated from time 0 to *t* can be calculated as (Curry and Feldman, 1987):

$$d_t = \int_0^t r(T(x))dx$$

where *r*(*T*) is the instantaneous rate of development for a given temperature, and the function, *T*(*t*), is the temperature at time *t*; *x* is a variable of integration.

In SPBMODEL, the development from one stage to the next was calculated using a rate-summation method, a discrete approximation of the above integration (Hines, 1979). The rate summation method assumes that development rates, as reciprocals of development times, are additive for changing temperatures (Curry and Feldman, 1987). The fraction of development *d* occurring up until time *t* is written as

$$d_t = \frac{1}{n}\sum_{i=1}^{nt} r(T_i)$$

where *n* is the number of time steps in a day, *nt* is the number of time steps in *t* days, and $r(T_i)$ is the development rate function for temperature *T* at time step *i*. The function $r(T_i)$ is calculated as the reciprocal of the mean time required to complete development at temperature T_i (Curry and Feldman, 1987). When d = 1.0, development for a life stage is complete.

The developmental progress of the different SPB life stages in SPBMODEL was recorded in an array of cohorts, each cohort maintaining its own fraction of development. This structure allowed the model to mimic the staggered emergence of SPB individuals in a given instar as a result of staggered oviposition times (Wagner et al., 1985). At the beginning of a simulation, the model was initialized by dividing the SPB populations in each life stage into a set of cohorts, each with a different initial fraction of accumulated development (Hines, 1979). At hourly time steps, the fraction of development for each cohort was increased according to the rate summation method (Hines, 1979). When the fraction of development equaled 1, the cohort developed to the next life stage, and its fraction of development was reset to 0 (Hines, 1979; Wagner et al., 1985).

The development rates used in SPBMODEL were based on measurements of SPB egg developmental times recorded by Gagne (1980) under constant temperatures in laboratory experiments. Brood development times were also recorded at a variety of temperatures (Gagne, 1980). Developmental rates for other life stages, as well as the egg production rate, were estimated as scalar multiples of the egg developmental rate. These estimations were based on observed developmental rates of the other stages and observed egg production rate at just one temperature: the optimal temperature for egg development, 27.5^{o}C (Ghosh, 1983). In other words, the relationship between temperature and developmental rate was assumed to remain constant across all life stages, with only the magnitude of the rate changing (Ghosh, 1983).

The arrival rates of newly emerged adults and of reemerged adults on a new tree were calculated by assuming that one-half of the emerged and reemerged population will have moved to the new tree within 3 hours (Ghosh, 1983). In the model, arrival and rearrival occur only during the day and above the temperature 13.9^{o}C (Ghosh, 1983). Two rates, egg production rate and adult rearrival/arrival rate, were determined empirically because they were considered difficult to measure directly or accurately in the field (Taha and Stephen, 1984), although some rough estimates could be made (Ghosh, 1983).

1.2.3.3 Mortality

Mortality rates in SPBMODEL were calculated as regressions on time of year and stand conditions (Lih, 1985). Stand conditions included pine basal area, hardwood basal area, mean diameter at breast height, and proportion of pines that are loblolly (Table 1.1). Regression equations for egg, larva/pupa, and brood adult mortality were determined through analysis of nine years of intensive within-tree population sampling data (Lih, 1985).

Difficult-to-measure mortality rates were derived from optimizing one rate at a time through trial and error, according to a priority scheme determined by sensitivity analysis (Taha and Stephen, 1984). The attacking adult mortality and emerging/reemerging adult mortality rates were calculated entirely through trial-and-error experimentation (Motamedi, 1981).

1.2.3.4 Tree Colonization

In SPBMODEL, the progression of tree colonization was determined by the relationship between the attacking adult population and two thresholds: the attraction threshold and the resistance threshold (Ghosh, 1983). The behavior of attacking adult SPB in relation to these thresholds was based on observed tree colonization phenomena in the field. The attraction threshold is the minimum number of emerged and reemerged adults required to maintain aggregation (Ghosh, 1983). When the population of flying adults declines below this threshold, pheromone levels drop, dispersal occurs, and population losses result (Ghosh, 1983). If the population is high enough (i.e. above the attraction threshold), a mass attack ensues (Ghosh, 1983).

Attacking adult beetles suffer high mortality until the population has overcome the resistance of the tree (Ghosh, 1983). In the model, this transition was defined as the time at which the attacking population surpassed the resistance threshold. At that point, the tree was considered dead, and the attacking adults proceeded to mate and lay eggs (Ghosh, 1983). SPBMODEL simulated the attack process within a tree until the attacking adult population reached the tree's bole capacity (the maximum number of SPB that can infest a particular tree) (Ghosh, 1983). As the attacking adult population neared the bole capacity, mortality was increased to reflect intraspecific competition (Ghosh, 1983). After the bole capacity was exceeded, SPBMODEL initiated an attack on a new tree (Ghosh, 1983).

1.2.5 SPBMODEL Input and Output

The latest version of SPBMODEL requires several inputs related to stand conditions (Table 1.1), including: the pine basal area and the hardwood basal area of the stand, the percentage of the infested trees constituted by loblolly, the mean diameter at breast height (DBH) of the infested pines, the U.S. state in which the infestation occurred, and the date that the infestation was inspected (Ghosh, 1983). The number of trees previously killed by SPB at the survey date is requested (Ghosh, 1983), but this number is used only in reporting the total number of SPB-killed trees killed, being added to the number of predicted killed trees at the end of the simulation. The number of infested trees at the survey date is a required and crucial input. The life stage distribution may also be entered, including the number of trees infested with each of four infestation stages: under attack, with eggs, with larvae or pupae, and with brood adults (Ghosh, 1983).

The primary outputs of SPBMODEL are the number of dead and infested trees over time (Ghosh, 1983). If the DBH distribution is provided, a prediction of the economic return from a salvage operation may be included as an optional output (Ghosh, 1983).

Table 1.1 Summary of the input variables for SPBMODEL, and of the simulation calculations that require these inputs (Ghosh, 1983). Optional input variables are marked with an asterisk (*).

Input	**Calculations requiring this input**
Date	Hourly temperature
U.S. State	Hourly temperature
Percentage of infested pines that are loblolly	Mortality rates and SPB life stage densities
Pine basal area	Mortality rates and SPB life stage densities
Hardwood basal area	Mortality rates
Diameter at breast height (DBH)	Mortality rates, SPB life stage densities, attraction and resistance thresholds, bole capacity
*DBH distribution of infested trees	Economic predictions
Initially infested trees	Initial population distribution (also dependent on SPB life stage densities)
*SPB life stage distribution of infested trees	Initial population distribution (greater accuracy)
*Previously infested (now dead) trees	Total dead trees

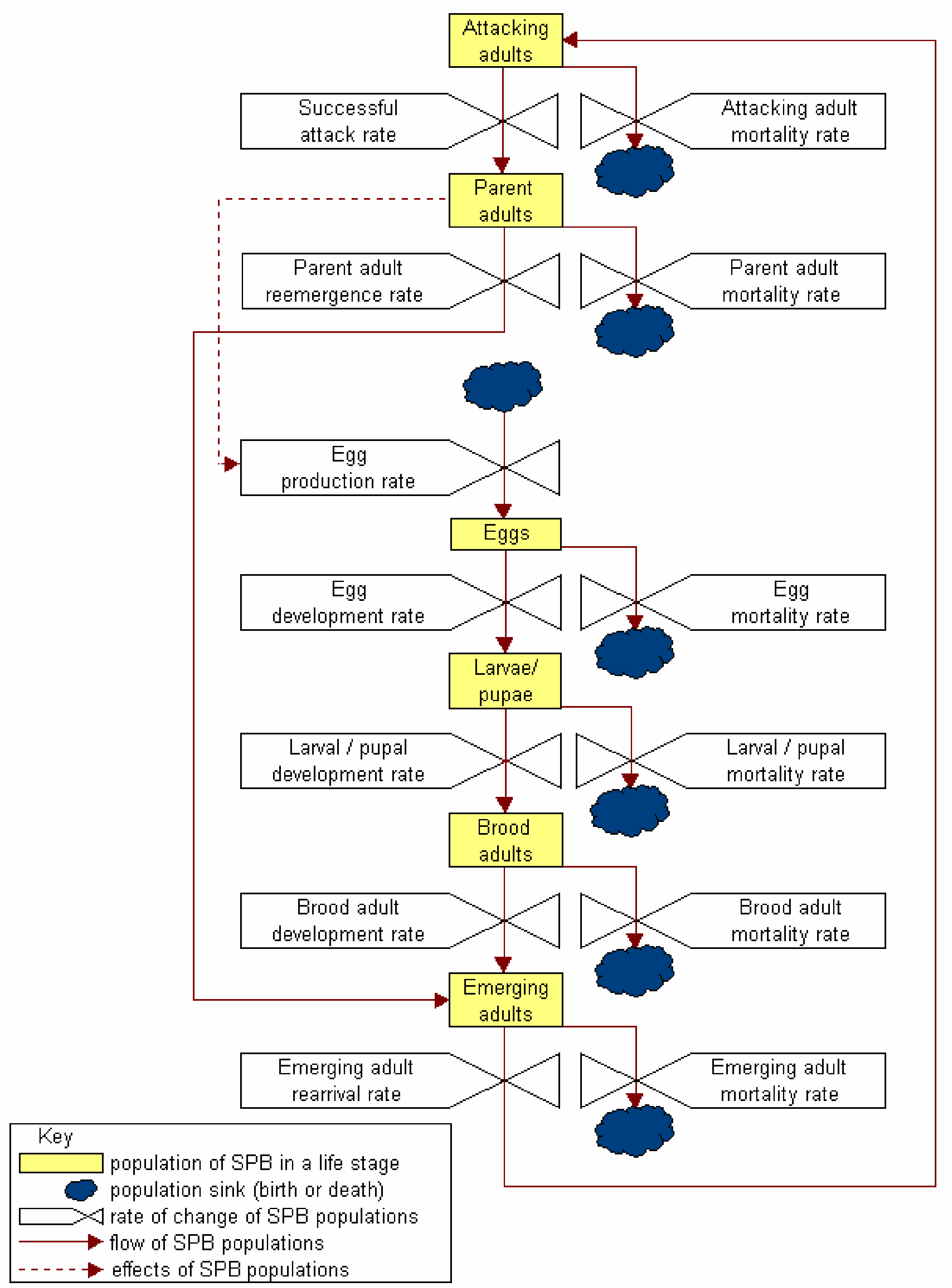

Attacking adults
Successful attack rate
Attacking adult mortality rate
Parent adults
Parent adult reemergence rate
Parent adult mortality rate
Egg production rate
Eggs
Egg development rate
Egg mortality rate
Larvae/ pupae
Larval / pupal development rate
Larval / pupal mortality rate
Brood adults
Brood adult development rate
Brood adult mortality rate
Emerging adults
Emerging adult rearrival rate
Emerging adult mortality rate
Key
population of SPB in a life stage
population sink (birth or death)
rate of change of SPB populations
flow of SPB populations
effects of SPB populations

Figure 1.2 A schematic representation of the flow of SPB populations from one life stage to the next. SPBMODEL simulated the flow of SPB populations according to this diagram. Egg production rate and development rates were temperature dependent; mortality rates were dependent on stand conditions (based on Ghosh [1983] Figure 2.2).

1.3 SPBMODEL Code Status

By the nature of the subject matter, biological models of complex systems will naturally become more complex with time, as more researchers add to or modify portions of the code. SPBMODEL is typical in this regard. Also, because the model has been developed over many years and by many authors (Ghosh, 1983; Hines, 1979; Jeng, 1994; Lih, 1985; Motamedi, 1981), SPBMODEL has a convoluted design and inconsistent coding standards. Such legacy models are common in the realm of software engineering, and they are known to require periodic redesign due either to faults in the model design (Motamedi, 1981; Taha et al., 1980), or to problems inherent in aging software (Pressman, 2001), or both.

1.3.1 Potential problems with model design

Of course, disagreements sometimes occur between biological simulation model predictions and observed data (Motamedi, 1981). Some of this disagreement may be attributed to the need for recalibration of models to new environmental conditions (Curry and Feldman, 1987). However, simulation models may suffer inaccuracy due to a lack of information about the parameters, or a misunderstanding about the key processes underlying the biological system (Motamedi, 1981). Disagreement between predicted and observed data may be attributed to faults in some of the following areas (Motamedi, 1981):

1) Inaccurate assumptions of the biological system
2) Unreliable experimental data
3) Overemphasis of less significant elements
4) Oversimplification of the biological system
5) Inappropriate model
6) Interactions among the model parameters

According to Motamedi (1981), it has been difficult to determine correct parameter values that reflect the natural behavior of SPB. Not surprisingly, disagreements between values predicted by SPBMODEL and actual values have been reported (Motamedi, 1981).

None of these potential faults with simulation models is easy to repair. Continual model refinement and validation are necessary to gradually isolate and eliminate design flaws.

1.3.2 Problems inherent in aging software

Change is inevitable with all software; maintenance in the form of corrections, adaptations, and enhancements is required (Pressman, 2001). Over the course of many years, neglect of good software engineering practices may cause a software program to become very difficult to maintain (*ibid.*). Unfortunately, the problem of unmaintainable software is neither new, rare, nor inexpensive (*ibid.*). A software development organization may spend over 60 percent of its efforts on software maintenance (*ibid.*). The following quotation from Osborne (1990) would probably fit well for many biological systems models being used in research labs around the world today.

> Much of the software we depend on today is on average 10 to 15 years old. Even when these programs were created using the best design and coding techniques known at the time [and most were not], they were created when program size and storage space were principle concerns. They were then migrated to new platforms, adjusted for changes in machine and operating system technology, and enhanced to meet new user needs – all without enough regard to overall architecture. The result is the poorly designed structures, poor coding, poor logic, and poor documentation of the software systems we are now called on to keep running...

Modifications made to software with the intention of improving it can actually cause the failure rate of the software to increase (Figure 1.3). Software reengineering is frequently necessary in order to bring a software program back to a maintainable state (Pressman, 2001).

Because documentation may be lacking, software reengineering often involves reverse engineering, or the process of rediscovering the design of a program through examination of source code (*ibid.*). Forward engineering may then be applied to create an improved, more maintainable program (*ibid.*).

Modularity is key to creating an easily maintainable program, because it increases the readability and modifiability of the code (Carrano, 1995). One way to achieve modularity is to use object-oriented programming (*ibid.*). Object-oriented programming has been shown to enhance the maintainability of simulation model programs (Yeh, 1997).

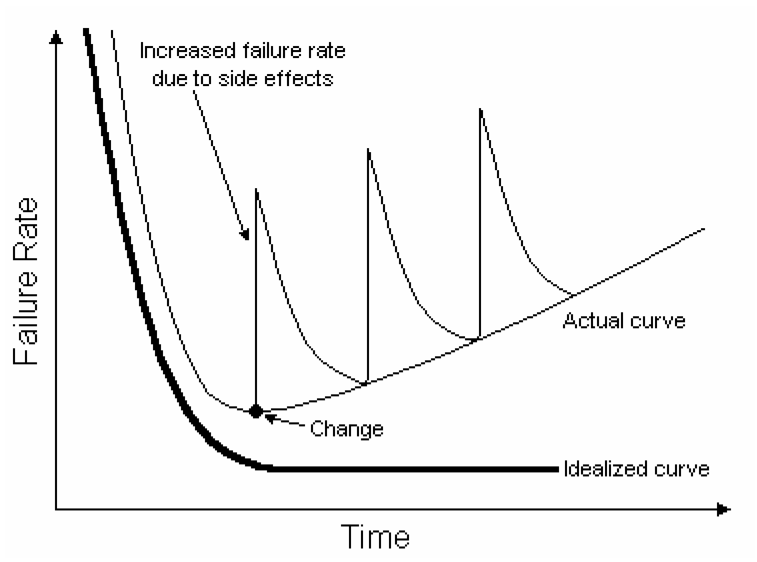


Figure 1.3 Idealized and actual failure rates for software. Modifications intended to improve a program often cause increased failure of the program due to side effects. Redrawn from Pressman 2001.

1.4 Genetic algorithms

Genetic algorithms are automated optimization heuristics designed to resolve a solution to a problem. GA's were invented by John Holland in 1975, and pioneered by Holland and his colleagues (Goldberg, 1989). Since their conception, GA's have been adapted for use in a variety of scientific and engineering application areas, including optimization, automatic programming, machine learning, economics, immune systems, ecology, population genetics, evolution and learning, and social systems (Mitchell, 1999). They have even been adapted to parameterize insect models (Sequiera et al., 1994).

Genetic algorithms are so named because their fundamental process is a simplified analog of biological evolution: a population of individuals evolves by random variation (mutation, recombination, and other operations), then the population is subjected to natural selection, in which the fittest individuals are more likely to survive to pass their genetic material on to the next generation (Mitchell, 1999). In genetic algorithms, the individuals are not organisms, but are rather potential solutions to a certain complex mathematical problem. For example, an individual could be a solution (including methods and parameters) to a simulation model. The criterion by which potential solutions are considered fit is called a fitness function (Mitchell, 1999). A simple fitness function could be the absolute difference between a predicted value and an observed value. Solutions with smaller differences between predicted and observed values would be considered more fit.

The genetic material used in GA's is represented as objects called chromosomes. GA chromosomes are a code for all of the characteristics of the potential solutions for the GA, just as biological chromosomes are a code for all of the characteristics for a biological organism. GA chromosomes are composed of genes, each gene coding for a particular characteristic of the potential solution. For example, if the chromosomes coded for solutions to a simulation model, the genes in each simulation model chromosome would represent the parameter values and other characteristics of the model. GA genes are made up of nucleotides. Instead of biological bases, the GA nucleotides are 1's and 0's. As in biological chromosomes, several kinds of genetic operations can occur to GA chromosomes during replication. The simplest type of GA includes mutation and crossover operations (Mitchell, 1999).

The basic genetic algorithm follows these steps (Mitchell, 1999):

1. Begin with a randomly generated population of *n* chromosomes, each representing a potential solution to a problem.
2. Calculate the fitness of each chromosome in the population, according to the fitness function.
3. Repeat until *n* offspring have been created:

   a. Choose a pair of chromosomes from the population to become parents.

   b. Allow crossover at a randomly chosen point between the parent chromosomes, with probability p_c (the crossover rate). Form two offspring that are exact copies of the parent chromosomes, replicating the results of the crossover operation if it occurred.

   c. Allow mutation of the offspring with probability p_m, the mutation rate.
4. Replace the current population with the new population.

5. Go to step 2.

Over several generations (repetitions of steps 2 to 5), the population will converge on a chromosome that is fit according to the fitness function (Mitchell, 1999). If the GA has worked correctly, this chromosome will code for a good solution to the complex problem under study.

In this research, each GA chromosome represented an SPB simulation model based on SPBMODEL. The genes contained parameter values for development and mortality rates, as well as a value to determine the functional form to be used in calculating development rates. The fitness function was a measure of how well the potential simulation models predicted actual numbers of dead and infested trees observed in the field, according to initial stand conditions provided as input.

1.5 Objectives

Objective 1.

Reengineer SPBMODEL code into an easily editable, object-oriented format so that it produces output indistinguishable from SPBMODEL.

Objective 2.

Demonstrate that genetic algorithms (GA's) are capable of improving upon SPBMODEL through improvement in parameterization and in choice of functional forms for specific functions.

The scope of this work allowed only for a proof-of-concept approach to Objective 2. I hoped to show that an evolutionary approach could be of use in identifying potential areas for model improvement and perhaps in automating that process.

Chapter 2 : Literature Review

2.1 Southern Pine Beetle Simulation Models

Most SPB models emerged in the 1970's as part of the federally funded Expanded Southern Pine Beetle Research and Applications Program (ESPBRAP) (Saunders et al., 1985). As of 1985, there were more than 35 mathematical models developed for SPB (Saunders et al., 1985), which can be grouped into five general categories (Turnbow et al., 1983):

1. Stand Hazard Models – to predict the likelihood of SPB outbreak in a stand.
2. Stand Risk Models – to predict the likelihood of an existing infestation to spread.
3. Spot Growth Models – to predict the development and spread of an existing infestation over time. SPBMODEL is of this type.
4. Stand Growth and Yield Models – to predict timber yield at rotation age, accounting for expected SPB damage.
5. Economics Models – to predict economic gains or losses from salvage operations or other treatment.

There are two types of spot growth models, regression models and mechanistic models (Table 2.1). Regression models rely on correlation, rather than underlying models of cause and effect. For example, Hedden and Billings used two metrics to predict spot growth:

- Spot Growth Index (SGI) = $\dfrac{\text{BA of new trees killed/day}}{\text{BA of active trees at initial visit}}$

- Active Tree Index (ATI) = $\dfrac{\text{BA of active trees at day 30}}{\text{BA of active trees at initial visit}}$

Likewise, Moore used the ratio of attacking holes to emergence holes on bark to develop a scheme for predicting the numbers of trees killed over a given period. Because they tend to be based on correlation of observed factors, regression models are usually limited to application within the range of the original data on which they were developed (Hain, 1980).

Mechanistic spot growth models require a deeper understanding of the southern pine beetle ecology (Hain, 1980). If the assumptions behind a mechanistic spot growth model are correct, the model should be applicable beyond the range of the original data (Hain, 1980). They are also frequently used to test ecological and management hypotheses (Stephen and Lih, 1985). Examples of mechanistic spot growth models include TAMBEETLE and SPBMODEL (Table 2.1).

TAMBEETLE, developed at Texas A&M University by Coulson et al. (1989), simulated several processes, including host selection by adults, pheromone drift, gallery construction, oviposition, reemergence, and temperature-dependent development. Development was determined according to Sharpe and DeMichele's (1977) poikilotherm model (Coulson et al., 1989). TAMBEETLE is not used in forest applications today, because of the extensive data input required to initialize the model (Coulson et al., 1989).

SPBMODEL (see Chapter 1 for overview) was first published in 1979 as part of Gail Hines' Ph.D. dissertation at the University of Arkansas. It included temperature-dependent SPB development but modeled the attack and dispersal processes without as much detail as TAMBEETLE (Ghosh, 1983).

Whereas the designers of TAMBEETLE attempted to obtain a field or lab estimation for all developmental rates (Coulson et al., 1989), the SPBMODEL designers incorporated some functions for which parameters could not easily be measured in the field, namely: attacking, emerged, and reemerged adult mortality rates, as well as the egg production rate and adult re-arrival rate (Taha and Stephen, 1984). These parameters were derived empirically, in an order chosen based on sensitivity analysis (Taha and Stephen, 1984).

Researchers at the University of Arkansas, most notably Dr. Fred Stephen and Dr. Hamdy Taha, were involved in much of the development and validation of this model. Five Ph.D. students have worked on the development and improvement of SPBMODEL over the years (Table 2.2).

SPBMODEL was originally written in GASP-PL/I, a high-level simulation language (Hines, 1979). It was later translated into standard FORTRAN "to address the need for wider dissemination" (Ghosh, 1983). In 1994, the FORTRAN code was translated by an automated process into C++ so that a graphical user interface could be added for running the model on a personal computer (Jeng, 1994). A natural enemy component was also added to SPBMODEL at that time (Jeng, 1994), and further elaboration of this component is being considered (Stephen, personal communication). SPBMODEL is currently available to foresters and other users on disk or over the Internet via FTP (file transfer protocol) (Lih and Stephen, 1989; Stephen, personal communication).

Table 2.1 SPB spot growth models.

Author(s)	Model	Year	Model Type	Spot Growth Determination	Area of Application
Hedden and Billings	N/A	1979	Regression	Attack to emergence ratio	Louisiana
Moore	N/A	1978	Regression	Spot growth index and active tree index	Georgia
Coulson, Feldman, Sharpe, Pulley, Wagner, Payne	TAMBEETLE	1979	Mechanistic	Simulation of emergence, oviposition, reemergence, host selection, pheromone emission and dispersal, tree dying; development based on Sharpe and DeMichele poikilotherm model	potentially broad
Hines, Taha, and Stephen	SPBMODEL	1980	Mechanistic	Simulation of attack/dispersion, and brood production based on a rate summation methodology	potentially broad

Table 2.2 The authors of SPBMODEL and their code modifications.

Author	Year	Language	Modifications
Hines	1979	GASP-PL/I	Original version completed.
Motamedi	1981	GASP-PL/I	Changes to the tree colonization procedure. Simplification of model according to the results of sensitivity analysis. Sensitivity analysis also provided the means for later calibration of unknown parameter values.
Ghosh	1983	FORTRAN	Some modularity introduced to the model. Multiple-cohort/rate-summation method implemented for SPB development.
Lih	1985	FORTRAN	User interface developed. Refinement of SPB density and mortality rates using regression equations.
Jeng	1994	C++	Graphic interface developed. Foundation laid to allow for predator/parasitoid interactions. Larval mortality rate distributed per time step. Infested tree calculation altered because: the larval mortality was changed, and the emerged adults were no longer considered part of the population infesting a tree.

2.2 Genetic Algorithms

Genetic algorithms have been used in a variety of fields, to improve and parameterize models from economics, immunology, machine learning, and other areas (Mitchell, 1999). Genetic algorithms have had differing effectiveness according to the fitness function selected and on the internal parameter settings (Mitchell, 1999). Several researchers have tried to find optimal combinations of parameter settings and optimal fitness functions (De Jong, 1975; Grefenstette, 1986; Schaffer et al.,1989; Sequiera et al., 1994).

De Jong (1975) conducted one of the first experiments to optimize the GA running parameters for a particular set of functions. In his experiment, he discovered that a population of 50-100 chromosomes was ideal, with a 0.6 crossover rate, and a mutation rate of 0.001. De Jong's GA settings performed very well in "off-line" performance, meaning the best fitness value averaged up to and including the current evaluation step. (Mitchell, 1999). De Jong's settings also performed well in "on-line" performance, referring instead to the best fitness value at each evaluation step (Mitchell, 1999). An experiment by Grefenstette (1986) on the same problem domain obtained GA settings that slightly outperformed De Jong's in on-line performance, namely: population size 30, crossover 0.95, and mutation rate 0.001. Grefenstette's experiment allowed one GA to optimize the parameters of another GA. Schaffer et al. (1989), after an extensive study, found a similar set of optimal GA parameters to Grefenstette: population size 20-30 and crossover 0.75-0.95, although with a higher mutation rate of 0.005-0.01.

In some studies, it has been shown that an "elite rate," or rate at which chromosomes are preserved from one generation to the next without mutation or crossover, improves the convergence rate of the GA (De Jong, 1975).

The different settings determined by these classic studies can serve to guide other research. However, the settings are expected to be somewhat different for each case of GA-aided problem-solving, because the best combination of settings depends on the structure and parameters of the fitness function, and on the specifics of the programming for each problem (Mitchell, 1999).

Because the structure of the fitness function affects the efficiency of the GA (Mitchell, 1999), some research has been directed towards the optimization of the fitness function. Sequiera et al. (1994) explored alternate fitness functions to optimize a GA used in the re-parameterization of a generalized poikilotherm model (Sharpe and DeMichele, 1977) for observed *Ips calligraphus* (Germar) data (Wagner et al., 1987).

Each potential fitness function that Sequiera et al. (1994) tested compared modeled data points to observed data points. The fitness functions included:

1. Least squares minimization (also called the sums of squares method):

$$\text{minimize} \sqrt{\sum_{i=1}^{m} (\text{observed}_i - \text{modeled}_i)^2}$$

where observed$_i$ = observation at data point i, modeled$_i$ = model output at data point i, m = total number of data points (Kennedy, 1985).

2. Coefficient of determination (R^2) (particular version of R^2 from Sequiera et al. (1994)):

$$R^2 = \frac{\sum_i \left(\hat{y}_i - \bar{\hat{y}}\right)^2}{\sum_i \left(y_i - \bar{y}\right)^2}$$

where y_i = observation, and $\hat{y}_i$ = model estimate.

3. The absolute difference:

$$\text{minimize} \sum_{i=1}^{m} | r_i |$$

where r$_i$ = residual at data point i, and m = total number of data points.

4. The quotient method:

$$\text{minimize} \sqrt{\sum_{i=1}^{m} \left(1 - \frac{\text{observed}_i}{\text{modeled}_i}\right)^2}$$

where observed$_i$ = observation at data point i, modeled$_i$ = model output at data point i, *m* = total number of data points.

Sequiera et al. (1994) discovered that a balanced index like the absolute difference index evolved a good fit for the Sharpe and DeMichele poikilotherm model, as well as for two photosynthesis models. The sums-of-squares method was an unbalanced index, as it weighted "deviations of larger magnitude more and this resulted in models that demonstrated those effects" (Sequiera et al., 1994).

Sequiera et al. (1994) compared the form of the model and regression parameters selected by the statistical methods of Wagner et al. (1987) to the form of the model and regression parameters selected by genetic algorithms. Wagner et al. (1987) had initially parameterized the Sharpe and DeMichele poikilotherm model using field data, and by using a SAS program to parameterize and select the form of the regression model (Wagner et al., 1987). Using GA's, Sequiera et al. (1994) obtained a model that fit the independent data as well as, if not better than, the Wagner model (1987).

2.3 Summary

There have been several southern pine beetle models built over the years. Of these, SPBMODEL may have the greatest potential for continued application in the field. Its mechanistic nature allows for potential application across a broad geographic area (Hain, 1980), and its simple set of required inputs allows for day-to-day utilization by forest pest managers (Lih and Stephen, 1989).

Effective use of GAs requires tuning of the internal parameters of the GA: the parameters that define the population size, mutation rate, cross-over rate, and elite rate. Experience of other researchers, particularly those working in similar problem domains, can be helpful when addressing a new problem. For this research, the work by Sequiera et al. (1994) on insect developmental models provided a good basis, in particular his suggestion that the fitness function be balanced, instead of biased towards data sets with large deviations (Sequiera et al., 1994). The optimal GA parameters must be obtained experimentally, but they are expected to include a low mutation rate, a large crossover rate, and a sizable population of chromosomes (De Jong, 1975; Grefenstette, 1986; Schaffer et al., 1989).

Chapter 3 : Reengineering SPBMODEL: Javahog, a web-based, object-oriented simulation model

3.1 Introduction

One of the primary reasons for reengineering SPBMODEL, officially called SPBMODEL, was to facilitate its continued use as a research tool. As mentioned, SPBMODEL in its current form is nearly impossible to update or maintain due to lack of consistency between code, documentation, and the published literature, but also due to poor program structure. The SPBMODEL version in standard FORTRAN made frequent use of the then necessary but presently shunned GOTO command (Ghosh, 1983). The use of the GOTO command disrupts the orderly flow of a program, and thus distracts from good program structure (Carrano, 1995). Poorly structured programs are very difficult to maintain or modify (*ibid*.). Due to the difficulty involved, direct modification of SPBMODEL FORTRAN code has been avoided in the past. In order to make updates in 1994, researchers used an automated translation program, called F2C, developed by AT&T Bell Laboratories and Bellcore (Jeng, 1994). F2C translated SPBMODEL from FORTRAN into C++, so that additional model code could be written in C++ (Jeng, 1994). The automated translation using F2C worked for the purposes then required, but the resulting code still looked very much like FORTRAN, with the GOTO commands unfortunately preserved. In its new format in C++, the code was, if anything, more inscrutable than it had been. The goal of the work reported here, therefore, was to produce a new, modifiable, maintainable version of SPBMODEL by reengineering the model.

The newly reengineered model was called Javahog, a name derived from the programming language Java™ and from a nickname of SPBMODEL, "The Hog Model," referring to the University of Arkansas' razorback hog mascot. There were three main attributes desired of Javahog: accuracy, accessibility, and ease of modification. In this context, *accuracy* refers to the ability of Javahog to mimic the predictions of SPBMODEL, *accessibility* refers to the ability of users to obtain and use Javahog, and *ease of modification* refers to the ability of other programmers to make changes to Javahog when needed.

3.1.1. Accuracy

In reengineering a model, errors may be made that result in discrepancies between the old and new model predictions. However, some discrepancies may occur even if no mistakes are made. Computer languages and compilers generally differ in the way they allocate memory for and perform operations on floating point numbers (real numbers) (Dowd and Severance, 1998). Differences in treatment of floating point values can be magnified over many calculations, so that the output of two programs written in two languages may be very different. To ensure that no errors are made and that the effect of any differing treatments of floating point values is small, it was necessary to statistically compare the predictions of Javahog and SPBMODEL.

3.1.2 Accessibility

One of a land-grant university's goals is to make information accessible to the public (Dooley, 1998). However, distribution of information can be very costly and time-consuming. In the past, SPBMODEL has been somewhat accessible by a variety of means. The model had been placed on the USDA Forest Service's Data General computing system, providing access to USDA Forest Service employees (Lih and Stephen, 1987, 1989). To distribute the model to other users, however, it was necessary to invest money in sending diskettes and user's manuals upon request, or to invest time in running the model via an interactive phone connection (Lih and Stephen, 1987).

In 1989, it cost approximately $30 US to prepare and ship to one user the floppy disks and documentation for a university-developed scientific software program (Logan, 1989). The cost of software distribution can quickly become unreasonable (Logan, 1989), especially if there are many users and several updates of a program. Today, a program like SPBMODEL, along with its documentation, can be obtained by a user free of charge and very quickly over the Internet via FTP (file transfer protocol) (Stephen, personal communication). However, as updates and corrections are made to the model, it is difficult to ensure that only the most recent model version is in use. Several versions of the model are likely to be in circulation at any one time, some of which may contain known errors. Providing simultaneous technical support for many versions of a program can be problematic.

Maintaining and providing technical support for just one copy of a program would be much easier. The World-Wide Web (or simply the "web") has made it possible for a diverse community of users anywhere on the Internet to access a single copy of a program (Pressman, 2001). Such a program, available on the Internet, is easily kept current through immediate updates, and users can be kept aware of changes through documentation posted in association with the program (Pressman, 2001). Beyond the initial programming cost incurred to permit users to run a program remotely through the web, the model's owner or author can continue to maintain the model at very little cost.

3.1.3 Ease of modification

3.1.3.1 Program structure

Good program structure and documentation are essential in order for code to be efficiently modified when needed (Carrano, 1995). A good program should have a highly modular structure, so that a change in the requirements for a program should require only small changes in the code. (*ibid.*). Modifications of non-modular programs can often have far-reaching side-effects, requiring changes throughout the program's code, whereas modifications of modular programs typically require changes to only a few modules (*ibid.*). One way to achieve a modular program design is to use object-oriented programming (OOP). OOP "encapsulates" functions and data together into discrete objects. Properly designed objects are very independent, such that they can be easily manipulated and reused within a program, or even among different programs (*ibid.*). An object can "inherit" one or more functions or data members from another object, so that code need not be rewritten for similar objects (*ibid.*). OOP provides for well-structured code that is relatively easy and fast to debug and edit (*ibid.*).

3.1.3.2 Documentation

Good documentation of a program is extremely important for readability, usability, and ease of modification (Carrano, 1995). Documentation styles differ, but there is agreement about several essential features to a program's documentation (*ibid.*), each of which centers on the inclusion of comments (non-executed statements in the code). The comments should include: the purpose, key data, assumptions, and possible errors or exceptions involved in both the program itself, and the individual functions (*ibid.*). The author and date should also be included in the comments (*ibid.*).

If it is properly done, documentation should provide enough details so that someone unfamiliar with the program can understand the code, and modify the code if necessary (Lewis and Loftus, 2001).

3.2 Objective

The goal of this reengineering was to develop a new object-oriented SPB simulation model based on SPBMODEL and to deliver it via the Web with appropriate documentation. Success was dependent on the comparison of the outputs of the new and old models; the two models were expected to have essentially identical outputs.

3.3 Selecting a programming language

Most modern languages would have been suitable for creating a new version of SPBMODEL, but a language providing for exceptional accessibility and ease of modification was preferred. Ultimately, the Java language was selected because of its object-oriented structure, its embedded documentation facility, the ability to run Java programs on all major computer platforms without modification[1], and the ability of Java programs to be easily run on the Web in a variety of ways.

[1] After the project was initiated, Microsoft announced that it would no longer support Java, but the issue has not been resolved. Java compilers for Microsoft Windows continue to be available.

3.3.1 Determining a Distribution Strategy

Given that the new model would be written in Java, there were still a number of different ways it could be implemented as an interactive web application (Hunter, 2001). One major decision was whether the model would run as client-side application or as a server-side application. Client-side programs or "applets" are downloaded to the client's browser before running, just as an image or movie clip is downloaded before being viewed. The server is then free to do other things, and the user may run the program multiple times without any further load on the server. Furthermore, the applet can include a custom graphical user interface.

Server-side programs, on the other hand, are run on the server, and are interfaced through the user's web browser. Downloading is not an issue with server-side applications, but difficulties are encountered when many users want to run the program at the same time. Each run causes the same load on the server, and the system is liable to get bogged down.

Client-side Java applications are more susceptible to compatibility problems because there are many versions of the Java virtual machine (JVM) found embedded in different web browsers and in different operating systems. Server-side Java applications, however, use standard Web interfaces to communicate with the user, so compatibility is much less of an issue. Java Applets, therefore, require testing on all possible client platforms (Hunter, 2001) and should generally be small applications to minimize download time from the server (Horton, 1997).

A server-side solution was chosen because the interface needs of the model could be handled with standard HTML (Hypertext Markup Language), the model's size and complexity would make an applet unwieldy, and compatibility problems would be minimized. The two most widely used tools for building interactive server-side applications are Java servlets and CGI (Common Gateway Interface) (Hunter, 2001). The latter was the original method for creating dynamic web-based applications and is quite powerful (Hunter, 2001); however, Java servlets, being a more recent innovation, offer several advantages including multithreading (reducing the burden on the server), better portability from one server to another, and seamless integration with Java programs (Hunter, 2001).

3.3.2 Documentation

Java has a standardized and partially automated documentation tool called javadocs (Horton, 1997). Javadocs allows other programmers to view the code documentation in an easy to navigate HTML format (Horton, 1997). Javadocs provides standards for and encourages good documentation practices.

3.4 Building the new model

The new model was written in Java using an Integrated Development Environment (IDE) called Forte for Java, Community Edition v. 1.0 (Build 842), using the JDK1.3 development kit on a PC computer running Windows 98.

3.4.1 Selection of preexisting code

Because SPBMODEL existed in several different versions (Table 3.1), selecting a basis for the reengineering process was necessary. Virtually complete code files of SPBMODEL written in both FORTRAN and C++ were available. Initially, however, no complete code was available in any language that would compile on the Windows/Intel or Unix computers in our laboratory. Several C++ files were missing, the FORTRAN code generated multiple errors when compiled using available compilers, and the documentation did not specify the compiler originally used (Table 3.1). Fixing the FORTRAN code so that it would compile was deemed the best approach. Based on the observation that the file input and output code caused most of the compiler errors, a version of the code that compiled in the F77 compiler under Unix was obtained by removing or modifying the lines of code containing file paths and hard-coding the path to a user-generated input file. This broke the user-interface that generated the input file for the model, however.

Consequently, each input file was painstakingly generated by hand to match the required format. This was tedious, but it did allow the use of a symbolic debugger to step through the code, a critical process in the reengineering of the model.

Table 3.1 The authors of SPBMODEL, and their code status and code use in this research.

Author, Year	**Language**	**Code status**	**Compilers**	**Executable**	**Use**
Hines, 1979	GASP-PL/I	Text only	No	No	Background information
Motamedi, 1981	GASP-PL/I	Text only	No	No	Background information
Ghosh, 1983	FORTRAN	Text only	No	No	Background information
Lih, 1985	FORTRAN	Complete code and supporting files, but available compiler version limits input capability	DOS, Windows, Unix	Yes	Limited input capability; Symbolic debugging code, initial code testing, and background information
Jeng, 1984	C++	Incomplete (missing files); will not compile	Windows, Unix	Yes	Results, comparisons; Flexible input capabilities

3.4.2 Reverse engineering

There was much information about SPBMODEL in the literature, which included user's manuals, five Ph.D. theses (Ghosh, 1983; Hines, 1979; Jeng, 1994; Lih, 1985; Motamedi, 1981), and several publications. To gain a better idea of SPBMODEL process, I began by building a simple model in Stella® (High Performance Systems, 1997), based an early description of the structure and processes of the model (Ghosh, 1983). Although Stella® has some limitations, it was a useful instructional tool. For instance, experimentation with Stella® made clear that SPBMODEL diagram used to demonstrate the flow of SPB populations in the literature was not technically correct. In that diagram (Figure 3.1), the parent adult population is shown to flow into both emerging adult populations and egg populations. In reality, of course, parent adults do not become eggs. The corrected diagram is shown in Figure 1.2.

Based on the experience developing a simple model from the literature, a set of key objects and their relationships was developed (Fig. 3.2). These included a ***Spot*** object representing the infested spot itself, including its environmental conditions; an object class, ***Tree***, to represent infested and dead trees; and an object class, ***Stage***, to represent the SPB life stages, with subclasses representing each of the specific life stages in the model. The ***Simulator*** object was designed to control the simulation, so it handled the hourly simulation loop, and ***AttackPool*** recorded the progress of tree colonization.

As much as possible, the algorithms from SPBMODEL were replicated in the new model, though the organization of the procedures was changed to fit the new object model. A complete description of the model is available on-line (Satterlee, 2002), including descriptions of each object class; however, a few points are worth mentioning here. According to the fractional development scheme used in SPBMODEL and re-implemented in Javahog, populations are divided into cohorts. In Javahog, ***SPBCohort*** objects keep track of how many SPB of a given life stage exist at each stage of development as well as their mortality rates. These cohorts are stored in a dynamic array of unlimited size. This is one improvement on SPBMODEL, in which the cohort information was stored in a static array. When the size of the array was exceeded, such as when a very large SPB population was present, SPBMODEL crashed.

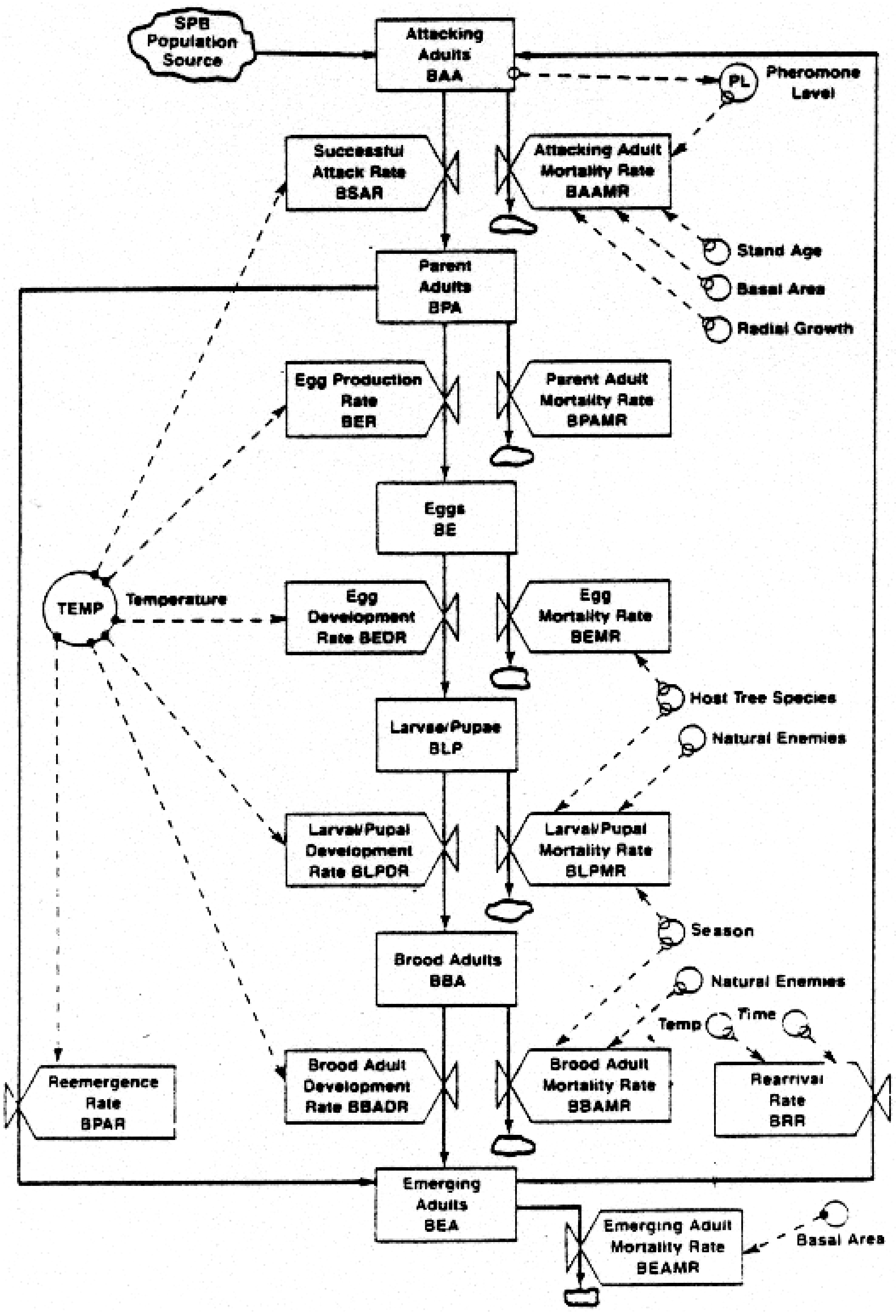


Figure 3.1 The original diagram showing SPB population flows, as used in building SPBMODEL (Ghosh, 1983).

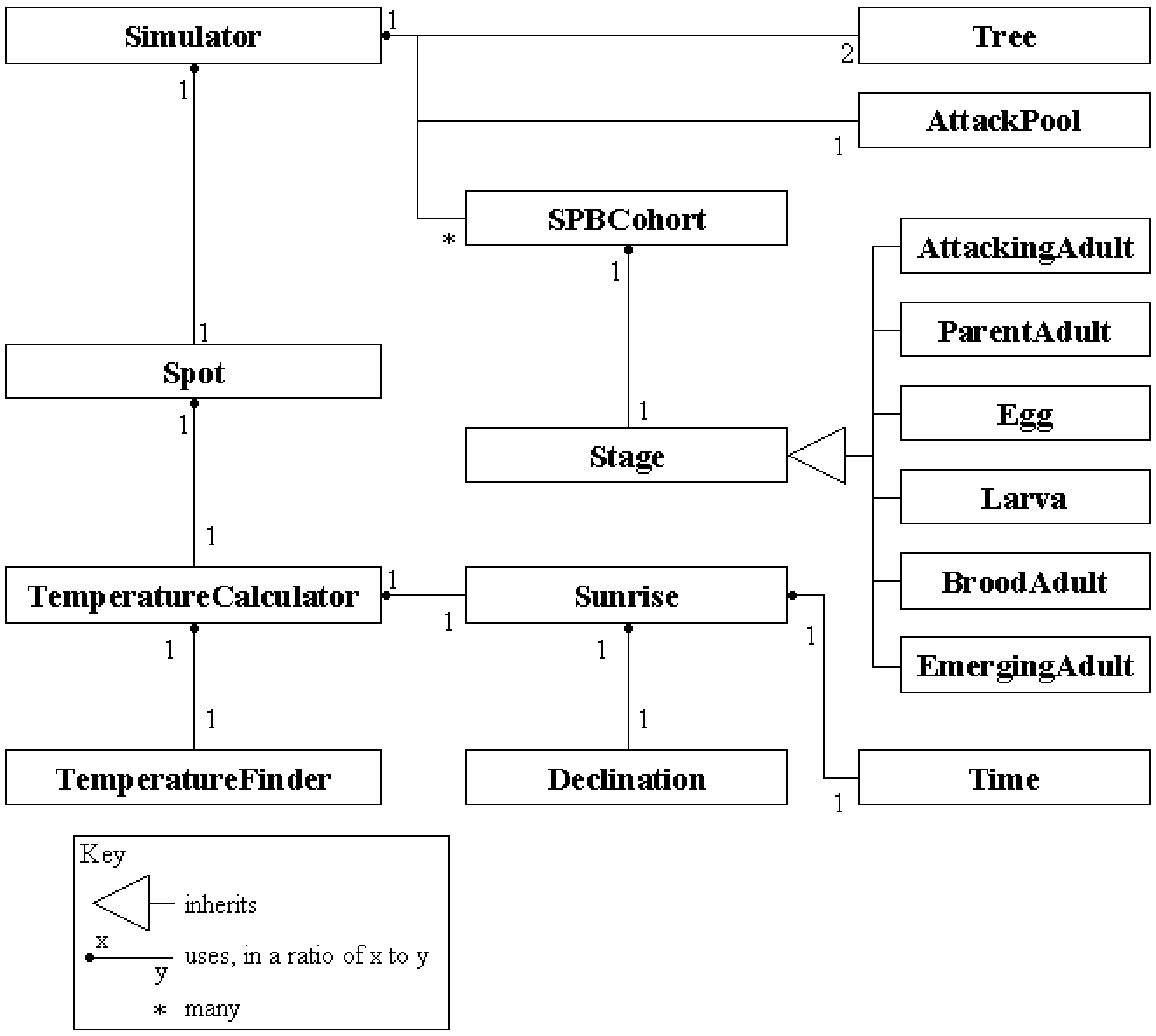


Figure 3.2 A class diagram showing the relationships between the classes that make up the Javahog simulation model.

3.4.3 The servlet

A World-Wide Web interface to the simulation model was implemented by a Java servlet. The servlet interface was designed to obtain user input about a stand, run the model, and output dead and infested trees over time in graphical and numerical formats.

The servlet was placed on a UNIX machine running JServ, a server that allows for the operation of Java servlets (The Apache Software Foundation, 2002). JServ has a default graphics package, but an additional program called KavaChart, by Visual Engineering, Inc., was needed to handle dynamically drawn graphs (Visual Engineering, Inc., 2002). A third program, called GifEncoder, was used to handle the conversion of the graph into a .gif format (Pokanzer, 2002).

A special main class was required to handle input and output in a web interface. This main class was called ***Javahog*** (Figure 3.3). A package called ServletHelper was added to include classes for helping to check the data and produce the output. Class ***PrintForm*** was included to set up the HTML interface for the user to enter information about the SPB spot. Class ***TestBounds*** was included for error-checking of input values. Classes ***PrintPops***, ***PrintTrees***, ***PrintTable***, and ***Delimited*** were included to output the SPB population trends to a graph, output the infested and dead tree trends to a graph, output all trends to an HTML table, and output all trends to a comma-delimited text file, respectively.

A few additional HTML files were required to organize the overall appearance of the Javahog website. These were written with HTML in a simple text editor. Graphics on the website were edited in Microsoft Paint for Windows 98 and Adobe PhotoShop 4.0 for Macintosh.

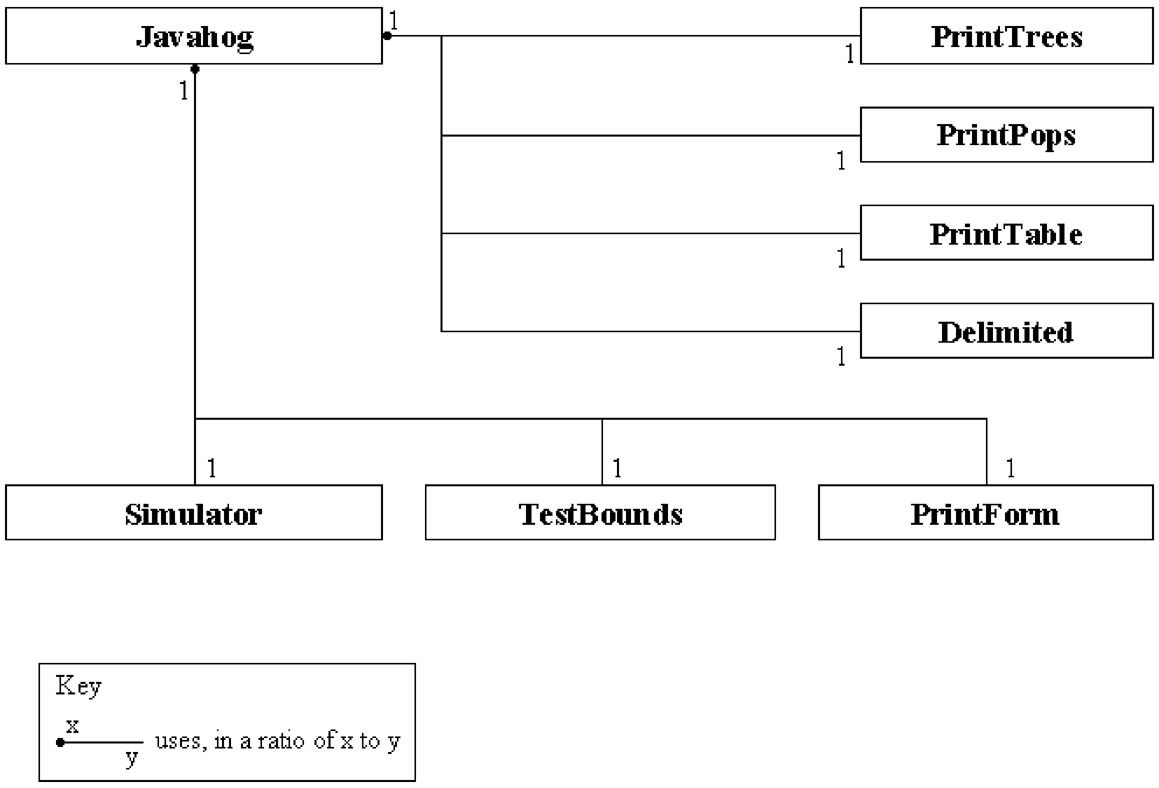


Figure 3.3 A class diagram relating the classes involved in the servlet interface for the Javahog simulation model.

3.5 Testing the new model

3.5.1 Selection of test data

The outputs of the old and new models were compared by running both models against a set of input conditions taken from field data on infested sites. There were approximately 100 field data sets available from a broad area: Alabama, Arkansas, Georgia, North Carolina, Louisiana, Mississippi, Virginia, Tennessee, and Texas. The dates ranged from 1976 to 2001. Because several different groups of researchers had collected the data over those 25 years, there was some inconsistency among the data sets, and some were so incomplete that they could not be used. Data sets were discarded if they were missing the following critical input data:

- Date
- U.S. state
- Number of initially infested trees

In addition, the number of infested or dead trees for at least one date in addition to the initial date was needed for use in subsequent validation (see Chapter 4).

A total of 61 data sets met these criteria, but most of these data were still somewhat incomplete. The following adjustments were made to the data sets to standardize and complete them:

- Years greater than 1999 were changed to 1999 because SPBMODEL was not Y2K compliant.
- If a required input was missing for percent loblolly composition, diameter at breast height (DBH), pine basal area, and/or hardwood basal area, the average value from all other spots was substituted. In the case of four spots, replacement values were entered for 3 required inputs. All other spots required 2 or fewer replacements.
- Input data were converted to metric units.
- Non-standard input data was recoded. SPBMODEL allows the user to input the number of trees under attack and containing eggs, larvae or pupae, and/or brood adults. In some cases, the data showed a number of trees listed as

harboring an infestation stage of "EO," a code occasionally in use in the field but not allowed as input in SPBMODEL. In such cases, values for the inputs "Trees under attack" and "Trees with eggs" were estimated based on the following scheme (Anonymous, 1996, unpublished):

Trees under attack = trees under attack + 0.25*EO.

Trees with eggs = trees with eggs + 0.75*EO.

- If the number of trees in an infestation stage was unknown, -1 was entered to indicate a missing value, in accordance with SPBMODEL input interface.
- Although the mean DBH was considered an important input, the distribution of individual trees into different DBH classes was always ignored. The DBH distribution impacts only the economic estimation of loss, a calculation separate from the predictions of dead and infested trees. Only the predictions of dead and infested trees were of interest in comparing the Javahog model to SPBMODEL. Thus, a -1 was always entered for the number of trees in each DBH class.
- If the number of initially dead trees was unknown, the number of initially infested trees was entered for that value. Unlike the initial number of infested trees, the initial number of dead trees has no bearing on the growth rate of the spot, but only on the final calculation of total dead trees.
- In some cases, only newly infested trees were listed at each survey date, instead of total infested trees. Assuming that each tree infested by SPB dies (Motamedi, 1981), the number of dead trees can be calculated at each date by the following equation:

  number of dead trees = number of newly infested trees + all newly infested trees from previous dates + initial number of dead trees

3.5.2 Statistical methods

The predicted numbers of dead and infested trees output by SPBMODEL and the Javahog model were compared at 10-day increments, from 10 to 90 days, using standard and paired t-tests. There were 61 replications, each a separate infestation spot, equating to 60 degrees of freedom for use in the t-tests. Because the objective of this experiment was to see whether the two models differed and because the null hypothesis was that the results were the same, a high α level (0.2) was set for the standard t-test to make it more likely to reject the null hypothesis, and thus to provide a more compelling result if the null hypothesis was not rejected.

The paired t-test looked for non-random differences between the output of the two models. In this case, because the models are unlikely to be exactly the same and because they are both deterministic, one should expect to see non-random difference increasing with increasing simulation time, even if the overall differences are very small. The alpha level for the paired t-test was again set at 0.2, to make the results more compelling if the null hypothesis was not rejected.

The outputs of the models were also visually compared by plotting the predictions made by Javahog at 10-day increments from 10 to 90 days, versus corresponding predictions of SPBMODEL for both dead and infested trees.

3.6 Results

3.6.1 Accuracy

Results showed that Javahog and SPBMODEL produced statistically indistinguishable predictions according to pooled means and variances (Table 3.2).

According to a paired t-test, there were some differences between the predictions of Javahog and SPBMODEL. In general, there were no differences at the beginning of the simulation, and differences increased over time (Table 3.3), as expected. When the predictions were not statistically identical, the mean of their differences ranged from 0.45 to 3.23 trees, or from 0.1% to 3.6% error.

Graphical representations of the comparisons between the Javahog model and SPBMODEL can be seen in Figures 3.4 and 3.5. If the comparisons were perfect, all data points would lie perfectly on the line y=x; in fact, all points are very close to the line.

Table 3.2 Pooled t-test comparing the results of the Javahog model and SPBMODEL, when simulating the same 61 spots.

Day of Simulation	Infested Trees ± Std. Dev. (n = 61)		p-value	Dead Trees ± Std. Dev. (n = 61)		p-value
	SPBMODEL	Javahog		SPBMODEL	Javahog	
10	91.82 ± 107.58	91.80 ± 107.42	>0.99	253.07 ± 296.55	253.10 ± 296.54	>0.99
20	89.34 ± 105.86	89.34 ± 105.70	>0.99	277.92 ± 326.89	277.93 ± 326.96	>0.99
30	89.66 ± 110.78	86.43 ± 106.21	0.87	301.98 ± 355.44	301.93 ± 355.44	>0.99
40	85.43 ± 115.63	85.66 ± 115.95	>0.99	354.15 ± 416.09	353.87 ± 415.78	>0.99
50	89.62 ± 129.71	89.52 ± 130.06	>0.99	386.82 ± 461.21	386.36 ± 460.85	>0.99
60	95.15 ± 146.09	93.75 ± 144.57	0.96	414.75 ± 502.75	414.15 ± 502.25	>0.99
70	104.80 ± 176.80	103.87 ± 175.40	0.98	459.07 ± 567.89	457.85 ± 566.68	>0.99
80	120.18 ± 226.62	118.74 ± 225.99	0.97	511.67 ± 665.18	509.87 ± 663.46	0.99
90	136.03 ± 274.06	134.21 ± 272.81	0.97	556.61 ± 760.43	554.34 ± 758.33	0.99

Table 3.3 Paired t-test comparing the results of the Javahog model and SPBMODEL, when simulating the same 61 spots. Significant differences are marked with an asterisk (*).

Day of Simulation	Mean Difference in Infested Trees, SPBMODEL – Javahog ± Std. Dev. (n = 61)	Mean Difference as Percent Error	p-value	Mean Difference in Dead Trees, SPBMODEL – Javahog ± Std. Dev. (n = 61)	Mean Difference as Percent Error	p-value
10	0.02 ± 1.26	0.0%	0.919	-0.03 ± 0.26	0.0%	0.321
20	0.00 ± 0.98	0.0%	>0.999	-0.02 ± 0.43	0.0%	0.766
30	3.23 ± 6.29	3.6%	0.000*	0.05 ± 0.56	0.0%	0.496
40	-0.23 ± 1.66	0.3%	0.284	0.28 ± 0.86	0.1%	0.014*
50	0.10 ± 1.58	0.1%	0.628	0.46 ± 1.30	0.1%	0.008*
60	1.39 ± 2.66	1.5%	0.000*	0.61 ± 1.68	0.1%	0.006*
70	0.93 ± 2.51	0.9%	0.005*	1.21 ± 2.58	0.3%	0.001*
80	1.44 ± 3.38	1.2%	0.002*	1.80 ± 3.49	0.4%	0.000*
90	1.82 ± 3.55	1.3%	0.000*	2.26 ± 4.73	0.4%	0.000*

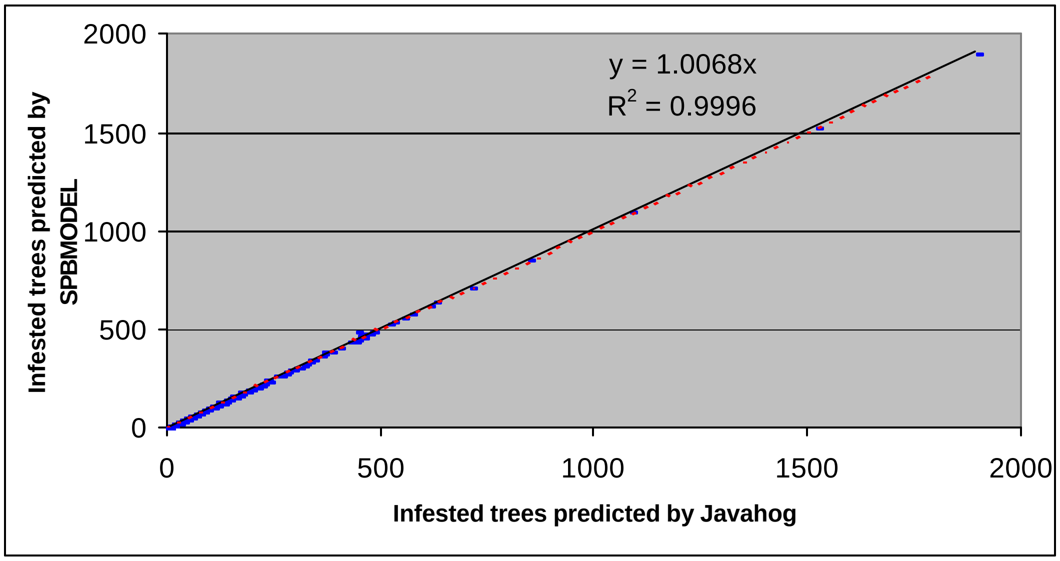


Figure 3.4 Comparison of the numbers of infested trees predicted by Javahog and by SPBMODEL. This graph contains data points in 10-day increments from 10 to 90 days of the 90-day simulations. Data points from the simulations of all 61 input data sets are included. The equation represents the black regression line. The dotted red line indicates the true y=x line.

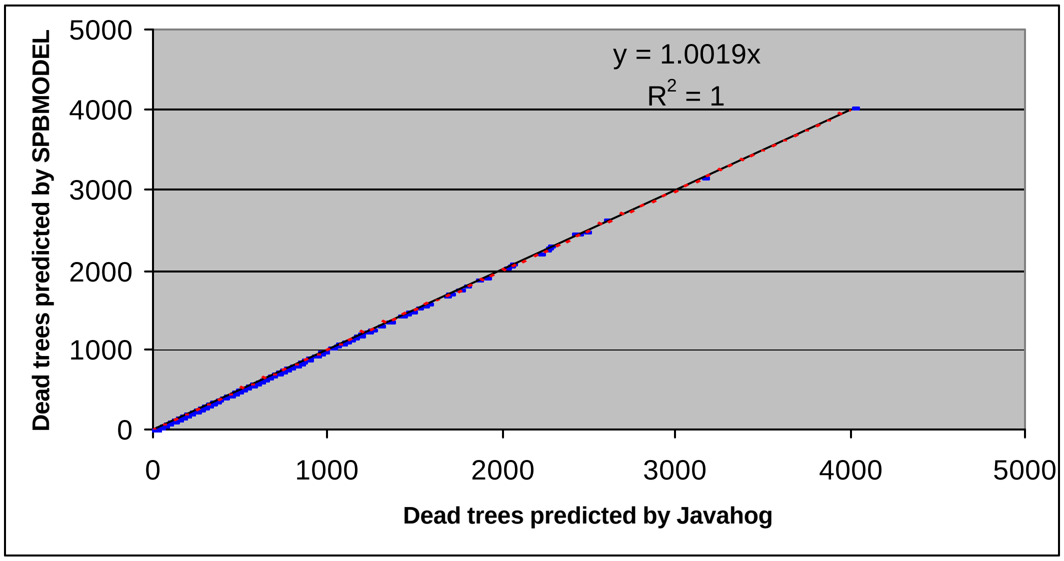


Figure 3.5 Comparison of the numbers of dead trees predicted by Javahog and by SPBMODEL. This graph contains data points in 10-day increments from 10 to 90 days of the 90-day simulations. Data points from the simulations of all 61 input data setsare included. The equation represents the black regression line. The dotted red line indicates the true y=x line.

3.6.2 Ease of Modification

The new Javahog model is highly modular and thoroughly documented, characteristics that in general provide for ease of modification (Carrano, 1995). Because Java compilers are available free of charge, future programmers should be able to obtain an appropriate compiler to modify Javahog.

3.6.3 Accessibility

The finalized version of the Javahog model can be found at http://ultra.isis.vt.edu/~sarah/javahog.html. The model can be executed entirely online, with no downloads required. Source code and Javadocs documentation, as well as help pages and background information are all available on this site. There is also a link to the Javahog model from the SPBICC website, the Southern Pine Beetle Internet Control Center, at http://www.spbicc.vt.edu.

3.7 Discussion

Output of the Javahog model was statistically identical to the output of SPBMODEL based on the standard t-tests and from visual inspection of the output. There were some small discernible differences between the outputs of the two models, as would be expected from two models with a different programming structure and potentially different floating point formats. These differences are most evident from the infested tree comparison at 30 days (Table 3.3). However, even at thirty days, when the differences are most evident, they are still small, less than 4% error. Some of this difference may be attributed to the slightly different treatment of infested trees in the FORTRAN version of SPBMODEL, upon which Javahog's algorithms were based, and in the C++ version, which was used in these tests (Table 3.1).

Because the Javahog model has been implemented through the Internet, it is much more accessible to the public than SPBMODEL. It should also be much easier to maintain and document versions of the code as changes are made, because the software itself need not be redistributed. Full documentation of the Javahog model is provided on the website, so future programmers should find it easy to change and adapt the Javahog model to their own purposes.

The functionality of the current Javahog model is similar to SPBMODEL, in that it dynamically produces either numerical or graphical output of the simulation. There are a couple of drawbacks of the Javahog model, however. There is no current capability for saving the user input, and also the simulation takes longer to run than SPBMODEL simulation. The online Javahog model typically requires between 20 and 30 seconds for processing, whereas SPBMODEL takes about 5 seconds on a modern PC computer. However, this has more to do the Web implementation than the new design. When run on a modern PC, Javahog runs a simulation in under one-half a second.

Another drawback to the Javahog model is that there is no current economic analysis of loss due to the predicted spread of the SPB infestation. However, the economic analysis calculations are independent of the modeling process, based only on the final output of the model, the stand DBH distribution, and current stumpage prices (Ghosh, 1983). An economic calculator could be added later either as part of the Javahog model or separately.

Chapter 4 : Application of a Genetic Algorithm

4.1 Introduction

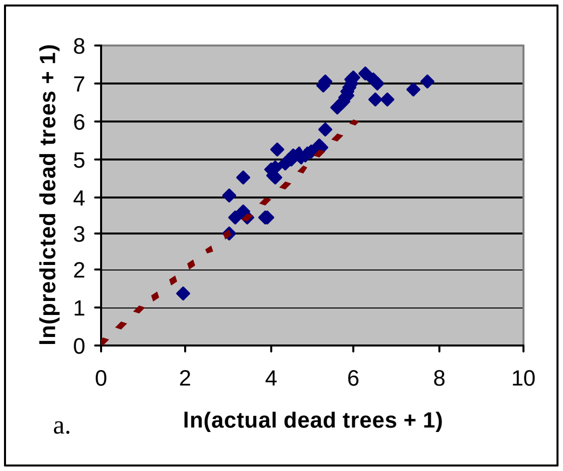
Given a new and more easily manipulated SPB model, *Javahog*, that essentially behaved identically to SPBMODEL, the question of whether the new model's performance could be improved through an automated, evolutionary approach was explored. In this chapter, the Javahog model was used for all tests of SPBMODEL against data, and for all GA runs modifying SPBMODEL.

As mentioned in Chapter 1, preliminary tests with SPBMODEL had shown that it did not perform well against current data. To quantify the performance as a benchmark for comparison with GA-generated models, SPBMODEL was used to simulate all 61 SPB spots described in Chapter 3, here randomly divided into two sets ("A" and "B") for use later in the optimization analysis (see below). Visual inspection indicated that the model performed poorly (Figures 4.1 - 4.2). Performance against data set B improved somewhat by eliminating one spot from Texas (Fig. 4.3). This spot showed a high potential for growth and was tracked for over 200 days. SPBMODEL was originally validated for runs up to 92 days (Lih and Stephen, 1989), so it is understandable that it would not predict data very well on the 200th day.

Looking at the population of dead and infested trees up to the 92nd day of simulation for 61 spots, SPBMODEL was off by an average of 76.1% and 39.2% respectively. In an earlier validation, percent error rates were found to be much lower, 2.8% and 8.6% respectively (Lih and Stephen, 1989). The percent error in both modern and earlier validations was calculated by:

$$100\% * \left(\sum_{i}^{n} (\text{observed}_i - \text{modeled}_i) / \text{observed}_i \right) / n$$

where observed_i is the observed number of dead or infested trees, modeled_i is the number of dead or infested trees predicted by the model, and n is the total number of observations available for comparison, including one or more observations from each infestation spot.



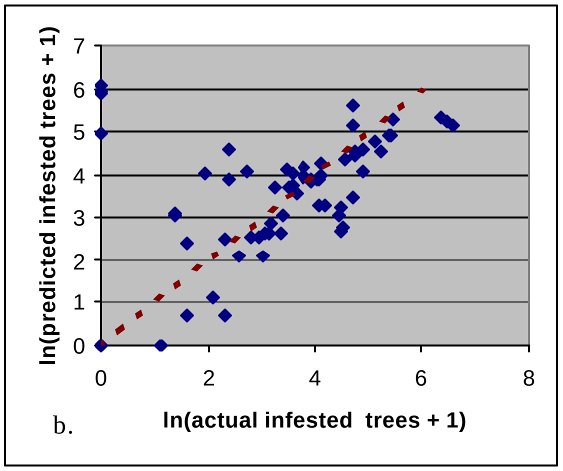
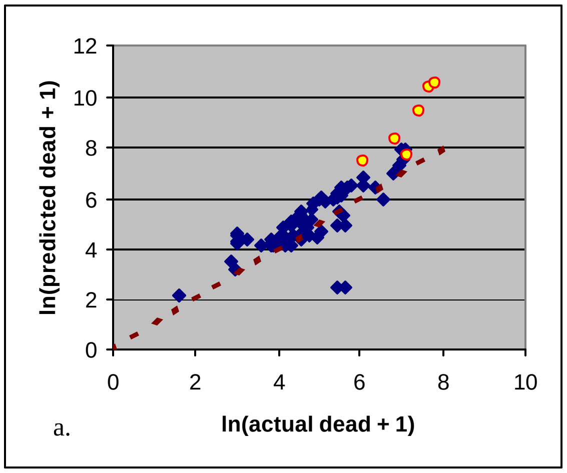


Figure 4.1(a) SPBMODEL predictions of dead trees, and (b) SPBMODEL predictions of infested trees for field data set 'A,' consisting of 30 spots. The dotted red lines represent the line 'y=x.'



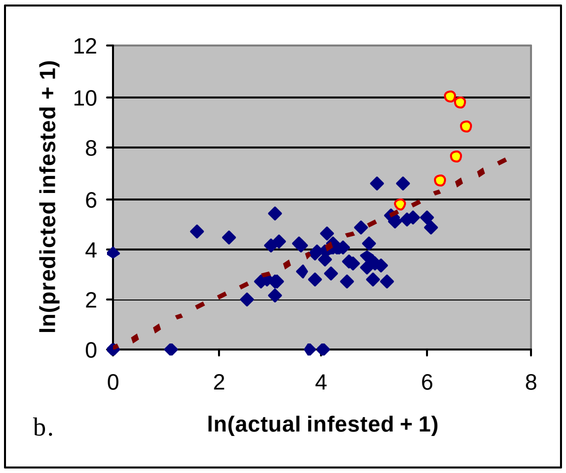
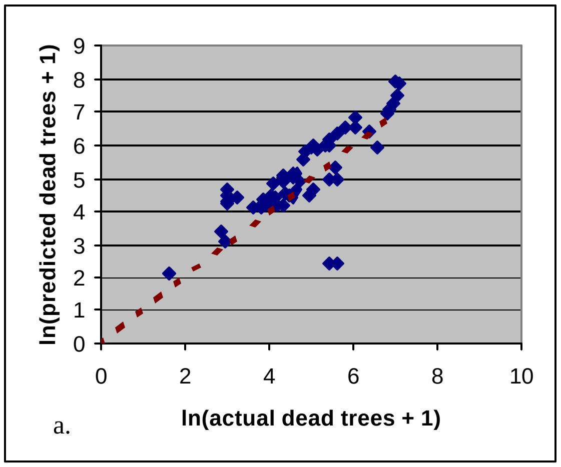


Figure 4.2 (a) SPBMODEL predictions of dead trees, and (b) SPBMODEL predictions of dead trees for field set 'B,' consisting of 31 spots. The dotted red lines represent the line 'y=x.' Data points from an outlier infestation are highlighted in yellow.



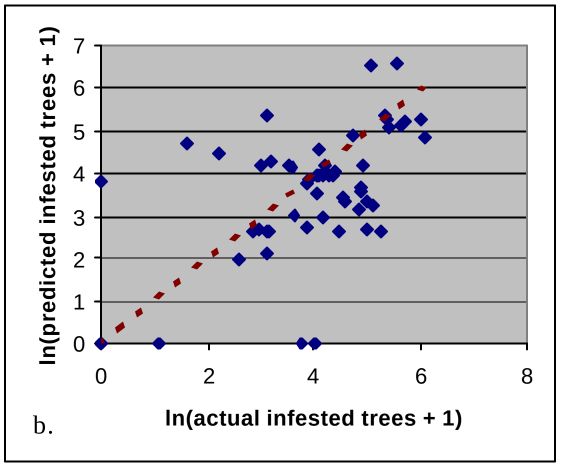


Figure 4.3 (a) SPBMODEL predictions of dead trees, and (b) SPBMODEL predictions of infested trees for field set 'B,' disregarding an outlier. The dotted red line represents the line 'y=x.'

4.2 Objective

The objective of the work described here was not to modify and re-validate SPBMODEL, but to demonstrate that an automated approach can help in that process. In particular, the goal was to show that a search heuristic, a genetic algorithm (GA), is capable of improving SPBMODEL's performance through automated parameterization and choice of alternative functional forms for modeling specific processes.

4.3 Methods

4.3.1 Selection of model parameters to optimize

The first step in applying the GA was to select components (parameters and functions) of SPBMODEL that would be manipulated by the heuristic. According to previous sensitivity analyses (Motamedi, 1981; Taha et al., 1980), two of the most important variables in SPBMODEL were the mortality rates for emerging and attacking adults. These mortality rates were also among four rates that were originally established empirically based on overall model behavior, rather than from direct measurement (Motamedi, 1981). Therefore, these two mortality rates were chosen as variables that the GA should optimize.

In SPBMODEL, both the emerging and attacking mortality rates were calculated from the pine basal area (BA) using a step function. The model chose from three constants depending on whether BA was low (< 21 m^2/hectare), medium (21 m^2/hectare - 30 m^2/hectare), or high (> 30 m^2/hectare). To allow the GA to modify these mortality rates, the step functions were replaced by linear functions fit through the three points, and the slope and intercept for these linear functions were varied by the GA. Upper and lower ranges for the intercepts and slopes for each mortality rate (Table 4.1) were set slightly higher and lower than the values for the intercept and slope required to generate the corresponding constant mortality in SPBMODEL and the total mortality was constrained to lie in the range of 0.0 to 1.0. These ranges were also enforced after mutation and cross-over events at every generation of the GA.

Developmental rates were also selected for parameterization by the GA. As mentioned (§1.2.3.2), development rates for each SPB life stage, except for brood adults and emerging adults, were calculated as scalar multiples of the SPB egg development rate (Ghosh, 1983). The egg development rate, in turn, was modeled using linear interpolation between observed development rates at discrete temperatures, rather than as a continuous function fit to observed data. Since the entomological literature is rich with alternative approaches to simulating the relationship between temperature and developmental rate (e.g., Logan, 1988; Sharpe and DeMichele, 1977; Stinner, 1975), this functional component of SPBMODEL was selected for inclusion in the GA. The GA was allowed to select functional forms to describe the development rates of each life stage, and the Javahog model was modified to accommodate alternative functional forms (see below).

Two alternative GA versions were envisioned, one in which the GA would select the optimal functional form for each life stage of SPB and a second in which the GA would select one functional form for use, with different parameters, for all life stages. The latter had a certain appeal from a biological point of view, and it would require the GA to estimate fewer parameters and use fewer genes. Since assigning a potentially different functional form to each life stage was the general case, the GA implementation was designed with that capability.

4.3.2 Selection of functional forms

The GA was set up to allow chromosomes to code for the selection of SPB developmental rate functions from eight different options. One was simply to apply the existing SPBMODEL development calculations. Of the other seven, six were developmental rate functions taken from Logan (1988); the last was a Gaussian function from Hochberg et al. (1986). These were chosen to include simple (e.g., linear) and more complex models. Each model expresses the instantaneous rate of development, r, as a function of temperature, T:

1. Linear

$r(T) = r(T - T_b)$,

where the r is the slope, and T_b is the x intercept or minimum temperature required for development.

2. Exponential

$$r(T) = y e^{rT}$$

where y and r are fitted constants.

3. Exponential with Base Temperature (Exponential Tb)

$$r(T) = y e^{r(T - T_b)} - 1.0$$

where y and r are constants, and T_b is the minimum temperature required for development.

4. Stinner (Stinner, 1975)

$$r(T) = \begin{cases} T_O / (1.0 + e^{k_1 + k_2 T}) \text{ for } T \le T_O \\ T_O / (1.0 + e^{k_1 + k_2 (2T_O - T)}) \text{ for } T > T_O \end{cases}$$

where T_O is the optimum temperature for development, and k_1 and k_2 are constants.

5. Logan 1 (Logan et al., 1976)

$$r(T) = y\,(e^{-rT} - e^{-rt})$$

where y is a constant; t is $(T_{\max} - (T - T_b))/\Delta T$; $T_{\max}$ is the maximum temperature at which development can occur; T_b is an arbitrary base temperature; and ΔT is $T_{\max} - T_O$.

6. Type III (Hilbert and Logan, 1983)

$$r(T) = r_{\max}\left[\frac{T^2}{T^2 + D^2}\right] - e^{-t}$$

where $r_{\max}$ is the maximum development rate; D is a constant; and t is as defined in Logan 1.

7. Gaussian (Hochberg et al., 1986)

$$r(T) = r_{\max}\, e^{[-(T_O - T)^2 / 2s^2]}$$

where $r_{\max}$ is the maximum development rate; T_O is the optimum temperature for development; and σ is a measure of the spread of the curve.

As with the mortality parameters, valid ranges were established using Logan (1988) and Hochberg et al. (1986) as guides for all fitted constants in these developmental rate functions (Table 4.1).

4.3.3 The GA program

The GA was implemented in Java using a public-domain GA package, Gajit, written by Matthew Faupel (2002). A new main class, ***GaUser***, was written to link Gajit with Javahog (Figure 4.4). This involved setting GA parameters (e.g., mutation and crossover rates) and the significant tasks of translating the chromosome into a model, running the model, and evaluating the model fitness (Table 4.2). Classes ***FileHandler*** and ***Analyzer*** were also added to implement calculation of the fitness function. The former interpreted initial stand conditions for multiple infestations, and the latter analyzed multiple simulations for each potential solution (chromosome).

In addition, the Javahog model was extended by the addition of three classes (Figure 4.5). ***Mortality*** and ***Development*** were used to represent the mortality rates and the developmental rate processes used by the SPB life stages. ***GaSimulator***, a subclass of ***Simulator***, was needed to perform the integration of the ***Mortality*** and ***Development*** objects with the rest of the code. Creation of these classes allowed dynamic substitution of one rate or developmental model for another based on the values of genes found in the chromosomes of the GA. With these new objects, it was possible for the GA to create a unique version of the Javahog model from each chromosome in the GA at each generation based on the numerical values decoded from each gene (Table 4.1). Gajit provided the utilities for determining the value of genes within certain ranges. Based on the values for the development parameters, ***GaSimulator*** instantiated (created a new instance of) the ***Development*** class for each life stage in the model. This object implemented the functional form of temperature-dependent development selected by the GA for the particular life stage.

Similarly, based on the mortality slope and intercept values represented in the genes of each chromosome, a new ***Mortality*** object was instantiated for two life stages, *Emerging Adults* and *Attacking Adults.*

Table 4.1 Parameters of Javahog modified by the GA, including parameters of the functional forms used in calculating development rates. The initial ranges of the parameters are listed.

Functional Form Parameter	**Lower Range**	**Upper Range**	**Relevant Functional Form(s)**
Mortality Slope	0.1	0.9	N/A
Mortality Intercept	0.001	0.01	N/A
Functional Form Type	0	7	N/A
r	0.0	1.0	Linear, Exponential, Exponential Tb, Logan 1
y	0.0	1.0	Exponential, Exponential Tb, Logan 1
s	30.0	90.0	Gaussian
k_1	0.0	1.0	Stinner
k_2	0.0	30.0	Stinner
D	50.0	100.0	Type III
T_b (^{o}F)	40.0	60.0	Linear, Exponential Tb, Logan 1, Type III
T_{max} (^{o}F)	80.0	100.0	Logan 1, Type III
T_O (^{o}F)	70.0	90.0	Stinner, Gaussian, Logan 1, Type III
r_{max}	0.0	0.1	Type III, Gaussian

Table 4.2 The steps involved in the selection of a good version of the model using a GA. The public-domain Gajit package provided for most of the functionality of the GA.

Step	GA Function	Function provider	Required specifications for Gajit
1	Establish a chromosome in the GA to hold information describing the potential new versions of the Javahog model.	Gajit	Gene length, number of genes
2	Allow genetic operations to occur to the chromosomes at each time step.	Gajit	Mutation and crossover rates
3	Translate the chromosome into a model.	GaUser	N/A
4	Run the model with a set of observed data.	GaUser	N/A
5	Evaluate the fitness of the model chromosome against the data set.	GaUser	N/A
6	Select from the model chromosomes according to their fitness.	Gajit	Survival rate, fitness score
7	Create a new generation of chromosomes.	Gajit	Elite rate
8	Repeat steps 2 through 7 until the GA has finished.	Gajit	Termination criterion

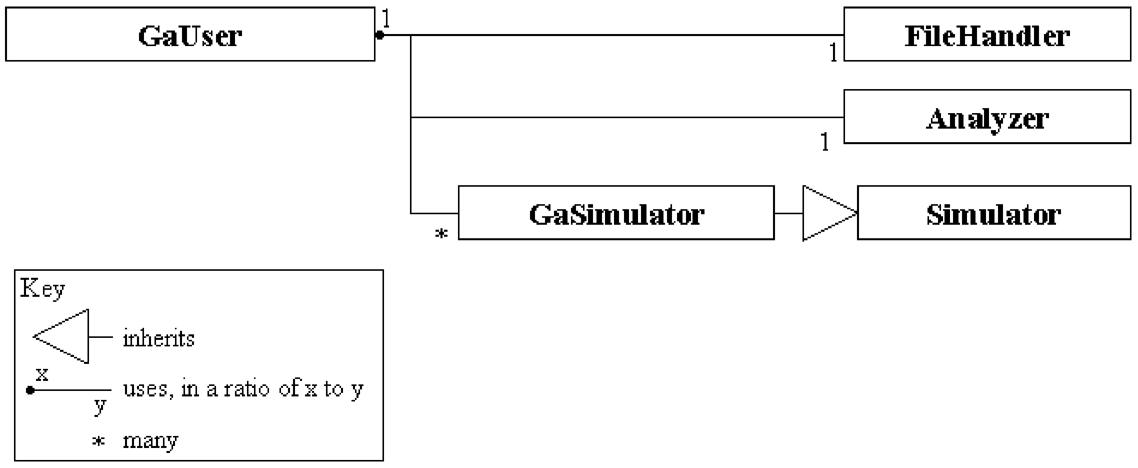


Figure 4.4 Class diagram of the classes manipulated by GaUser in conducting the GA.

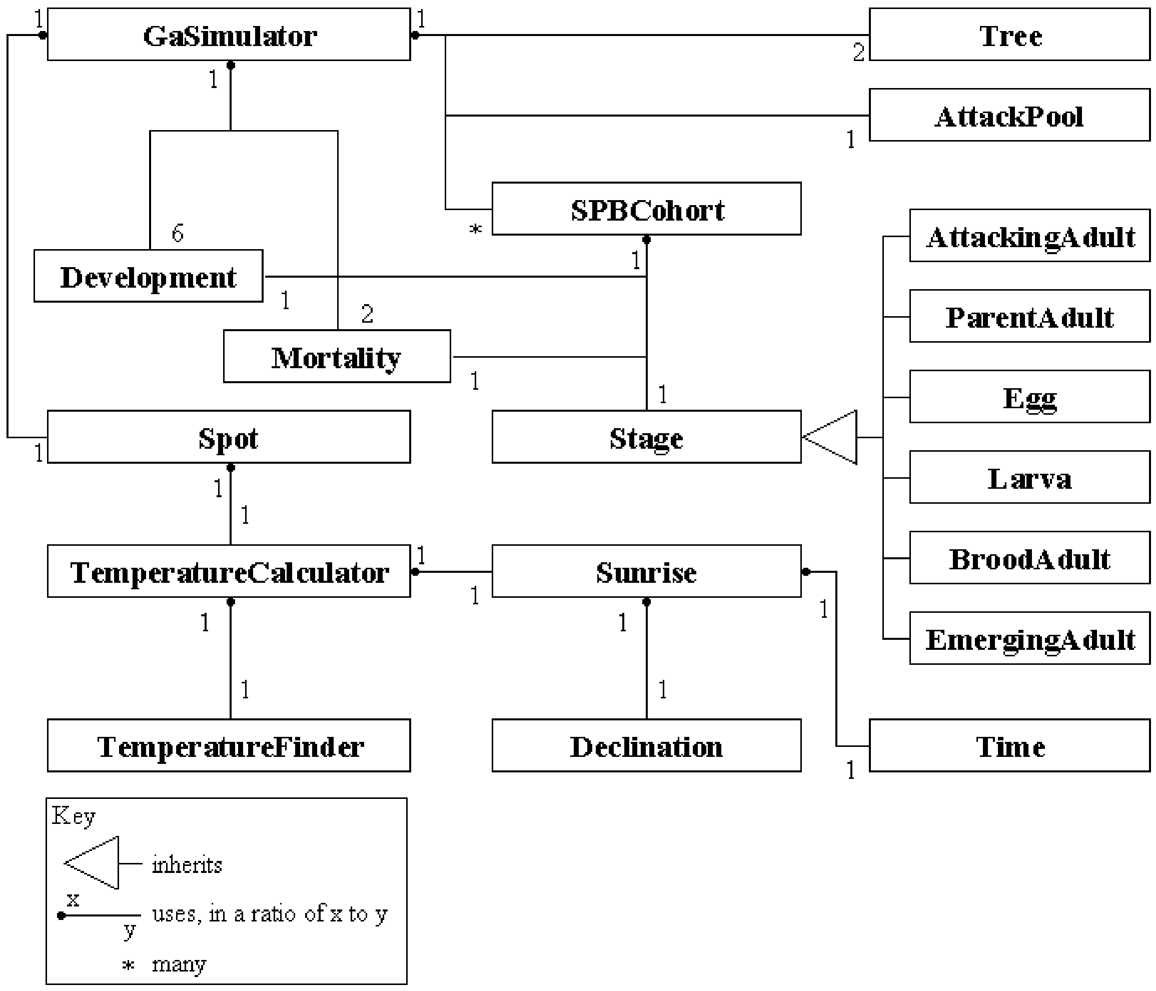


Figure 4.5 Diagram representing the classes used by GaSimulator.

4.3.4 The fitness function

The fitness function for this research measured how well the model represented by each chromosome predicted observed field data. Four error terms contributed to the fitness function, $\varepsilon_{\text{mean,infested}}$, $\varepsilon_{\text{mean,dead}}$, $\varepsilon_{\text{final,infested}}$, and $\varepsilon_{\text{final,dead}}$. These represented the disparity between the model and the field data in terms of: the mean numbers of *infested* trees throughout the simulation, the mean numbers of *dead* trees throughout the simulation, the numbers of *infested* trees at the end of the simulation, and the numbers of *dead* trees at the end of the simulation, respectively. To create a more balanced error index, as was found beneficial in Sequiera et al. (1994), the error terms were based on log-transformed data. The two error terms $\varepsilon_{\text{mean,infested}}$ and $\varepsilon_{\text{mean,dead}}$ were calculated from the mean difference between the log-transformed numbers of dead and infested trees, averaged across all infestation spots:

$$\varepsilon_{\text{mean}} = \tfrac{1}{n}\sum_{i=1}^{n}\tfrac{1}{t}\sum_{j=1}^{t}\left(\ln(\text{observed}_{ij}+1)-\ln(\text{modeled}_{ij}+1)\right)$$

where n is the number of infestation spots; t is the number of time steps in the simulation; and observed$_{ij}$ and modeled$_{ij}$ are observed and modeled numbers of dead or infested trees over time for each infestation spot. Using the same notation, the two terms $\varepsilon_{\text{final,infested}}$ and $\varepsilon_{\text{final,dead}}$ were calculated by the following similar formula, comparing only the final numbers of dead and infested trees.

$$\varepsilon_{\text{final}} = \tfrac{1}{n}\sum_{i=1}^{n}\left(\ln(\text{observed}_{it}+1)-\ln(\text{modeled}_{it}+1)\right)$$

where observed$_{it}$ and modeled$_{it}$ are observed and modeled numbers of dead or infested trees at the end of the simulation for each infestation spot.

To ensure that a high fitness score represented a low error, the fitness was set equal to the reciprocal of the summed, weighted error terms above, and constrained to have a maximum value of 10,000:

fitness = 10.0/(maximum(0.001, $\varepsilon_{\text{mean,infested}} + \varepsilon_{\text{mean,dead}} + 2 * \varepsilon_{\text{final,infested}} + 3 * \varepsilon_{\text{final,dead}}$))

The $\varepsilon_{\text{final}}$ error terms were weighted more heavily than the average error terms, because the numbers of dead and infested trees near the end of a simulation are more difficult to predict (Stephen and Lih, 1987). Therefore, a model capable of predicting these later values should be considered more fit. The $\varepsilon_{final,dead}$ term is weighted more heavily than the $\varepsilon_{final,infested}$ term, because being a cumulative measure, the number of dead trees at the end of the simulation is a clearer indicator of the magnitude of the infestation than the number of currently infested trees.

Fitness was also adjusted by subtracting 1 each time the number of iterations through the tree colonization procedure during one time step exceeded 100,000. This fitness penalty accumulated across all infestation spots for the model chromosome. This adjustment was made to avoid models that ran too long. Based on experience with the GA, it was determined that poor models sometimes allowed the tree colonization procedure to enter a near-infinite loop, processing millions of calculations and requiring up to several days to run to completion on a personal computer. This adjustment effectively eliminated such models before the looping became excessive and run-time increased to prohibitive levels.

4.3.5 Field data

Observed data were required for two purposes: to evaluate the fitness of chromosomes (based on the performance of their corresponding models) within the GA, and as the basis for independent testing of the resulting model. This required two data sets, so (as mentioned above) the 61 data sets described in Chapter 3 were divided into two sets, A and B, consisting of 30 and 31 spots, respectively; set A was used to evaluate fitness in the GA and B was used for independent evaluation of the resulting models.

Each spot recorded the number of infested and/or dead trees in the spot over time. If the number of dead trees was not recorded over time, only the number of infested trees over time was used to test the model, and vice versa.

The spots were sorted into set A or B randomly. Spots were arbitrarily assigned a unique integer index from 1 to 61. A non-repeating sequence of the integers from 1 to 61 was then generated using the random number generator service available at http://www.random.org (Haahr, 2002). The data sets with indices corresponding to the first 30 numbers in the random sequence were assigned to set A, and the rest were assigned to set B.

4.3.7 GA Analysis

4.3.7.1 Finding an optimal model quickly

Convergence occurs when further evolutionary computations with the GA no longer improve upon the fitness of the chromosomes. At convergence, the population of chromosomes is generally very similar, with a high fitness and low variance in fitness. Highly efficient running parameters provide for rapid convergence.

A low-efficiency run, with a large number of chromosomes (200), high mutation (0.01), low crossover (0.5), and conservative termination criteria (identical mean integer fitness for 10 generations) was conducted to determine an approximate maximum fitness. This run allowed the GA to select different developmental rate functions for each life-stage, and it took several days to complete on a personal computer with a 1.4GHz AMD Athlon Thunderbird processor. Runs with other combinations of GA parameters were then conducted to determine whether approximately the same fitness could be achieved faster (Table 4.3). The termination criterion of these other runs was to end the run when the integer value of the average fitness of the population had remained unchanged for 5 generations.

Table 4.3 The running parameters for the GA runs. All combinations of 30 and 40 chromosomes, 0.001 and 0.005 mutation rate, and 0.5 and 0.9 crossover rates were run. One run with 200 chromosomes was run under conditions designed for postponing convergence. A survival rate of 0.75 and an elite rate of 0.1 were used in all runs.

GA Run ID	Number of Chromosomes	Mutation Rate	Crossover Rate	Number of Similar Generations Before Termination
200_01_5d	200	0.01	0.5	10
30_001_5d	30	0.001	0.5	5
30_001_9d	30	0.001	0.9	5
30_005_5d	30	0.005	0.5	5
30_005_9d	30	0.005	0.9	5
40_001_5d	40	0.001	0.5	5
40_001_9d	40	0.001	0.9	5
40_005_5d	40	0.005	0.5	5
40_005_9d	40	0.005	0.9	5

4.3.7.2 Testing uniform functional forms

Based on the best set of GA parameters discovered, a second experiment was conducted to find an improved model by requiring the GA to select the same developmental rate functional form for all life stages.

4.3.7.3 Testing the optimal model

Once a model with a high fitness was found, this model was compared against SPBMODEL to test for the improvement power of the GA. The GA-generated model and SPBMODEL were tested against data used as input into the GA fitness function (*A* data sets), as well as against independent data (*B* data sets). Testing the GA models against the 'A' data set provided an estimate of how well the model described the dependent data; tests against the 'B' set measured the same model's predictive power against independent data.

Four metrics were used for comparison:

1) The GA fitness function

2) Standard deviation from the observed values:

$$\text{Std. dev.} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(\text{observed}_i - \text{modeled}_i\right)^2}$$

3) Number of times the general trend of growth or decline of an infestation was predicted correctly.

4) Visual comparison, by plotting SPBMODEL predictions vs. GA-generated model predictions.

4.4 Results

Based on the general trends observed during the GA runs, the GA seemed to be performing properly. Starting from a randomly generated set of chromosomes, which should have poor fitness and wide variability, the GA should select for chromosomes with higher fitness, and as the GA converges, the variability should decline. Even with a very high mutation rate, the 200-chromosome run, showed these expected patterns (Figure 4.6).

4.4.1 Finding an optimal model

The maximum fitness achieved by all models from the GA run over 120 generations with a population of 200 chromosomes and relatively high mutation rate was 106 (Table 4.4).

The runs to determine optimal GA parameter settings converged before reaching this fitness, but some were close (Figures 4.7 - 4.10). Possibly, the termination criterion allowed for premature convergence, but just as likely, the combination of population size and recombination / mutation rates simply made it unlikely to achieve an optimal solution. One combination of parameters achieved fitness levels of above 99 percent of the 200-chromosome run in much less time, i.e., the run 40_005_5d (Figure 4.8) which achieved a maximum fitness of 105 with a population of 40 chromosomes, a mutation rate of 0.5% and a 50% cross-over rate. The best solution found using a population of 30 chromosomes had a fitness of 100. Each of these results was an improvement over SPBMODEL, which generated a fitness score of 92 against data set *A* (Table 4.4).

Parameters from the chromosome with the highest fitness overall are listed in Table 4.5. Note that some temperature parameters in this table are biologically unreasonable, though allowed by the constraints imposed on the GA.

Table 4.4 Comparison of the fit of SPBMODEL and the model resulting from the GA run, 200_01_5d. Both models were compared to dependent data set 'A,' independent data set 'B,' and data set 'B' without an outlier infestation. The best runs with 30 and 40 chromosomes are also shown.

Model	Fitness Score		Std. Deviation			
	Data Set 'A'	Data Set 'B'	Data Set 'A'		Data Set 'B'	
			Dead	Infested	Dead	Infested
SPBMODEL	92	87	418	136	5273	3272
SPBMODEL without outlier		89			342	125
200_01_5d	106	99	327	120	252	201
200_01_5d without outlier		99			178	93
40_005_5d	105	96	333	99	290	170
30_005_9d	100	100	328	112	278	144

Table 4.5 The values of the parameters optimized by the GA, for the best chromosome from the run 200_01_5d. The initialized ranges and the GA -determined values for each applicable SPB development stage are listed.

Functional Form Parameter	**Initial Range**	**Attacking Adult**	**Parent Adult**	**Egg**	**Larva / Pupa**	**Brood Adult**	**Emerging Adult**
Mortality Slope	0.1 - 0.9	0.3477					0.1205
Mortality Intercept	0.001 - 0.01	0.0016					0.0035
Functional Form Type	0 - 7	1 Exponential	1 Exponential	5 Logan 1	5 Logan 1	2 Exponential Tb	5 Logan 1
r	0.0 - 1.0	0.005	0.480	0.983	0.914	0.499	0.460
y	0.0 - 1.0	0.527	0.059	0.284	0.028	0.442	0.141
T_b (^{o}F)	40.0 - 60.0			61.3	84.6	83.1	81.8
T_{max} (^{o}F)	80.0 - 100.0			153.9	123.5		163.0
T_O (^{o}F)	70.0 - 90.0			74.39	105.7		95.2

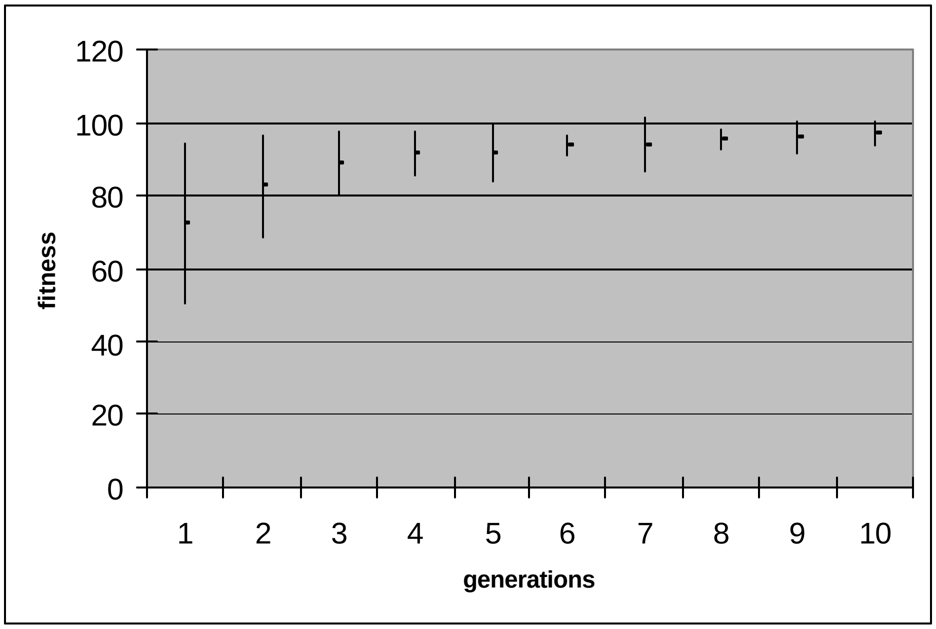


Figure 4.6 The first 10 generations of the 200 chromosome run (200_01_5d). The variances decreased as the population of chromosomes converged to a constant mean fitness.

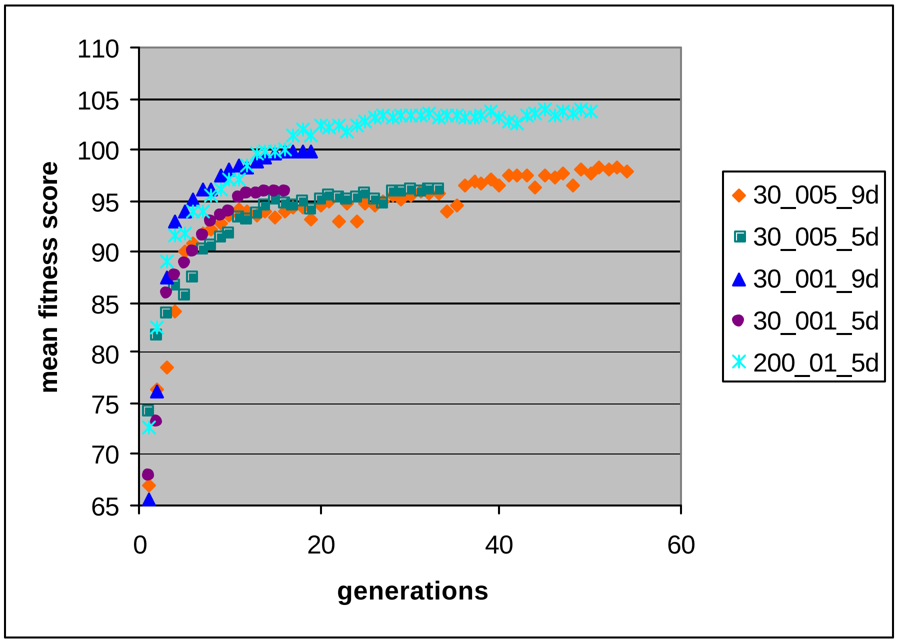


Figure 4.7 Convergence of GA runs with 30 chromosomes, allowing different functional forms according to the SPB development stage. The first 50 generations of the GA run with 200 chromosomes is shown for comparison. See Table 4.3 for more detailed descriptions of the runs.

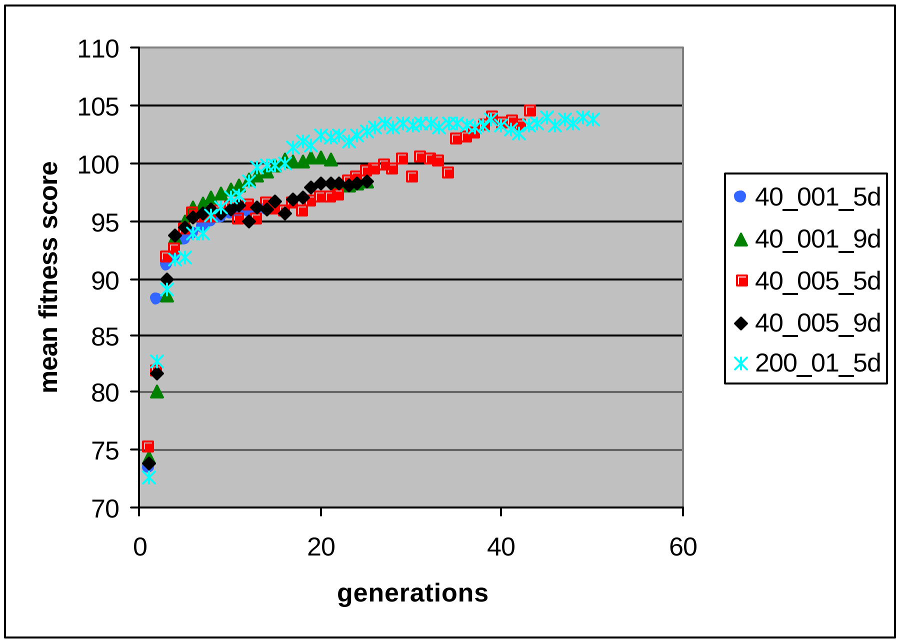


Figure 4.8 Convergence of GA runs with 40 chromosomes, allowing different functional forms according to the SPB development stage. The first 50 generations of the GA run with 200 chromosomes is shown for comparison. See Table 4.3 for more detailed descriptions of the runs.

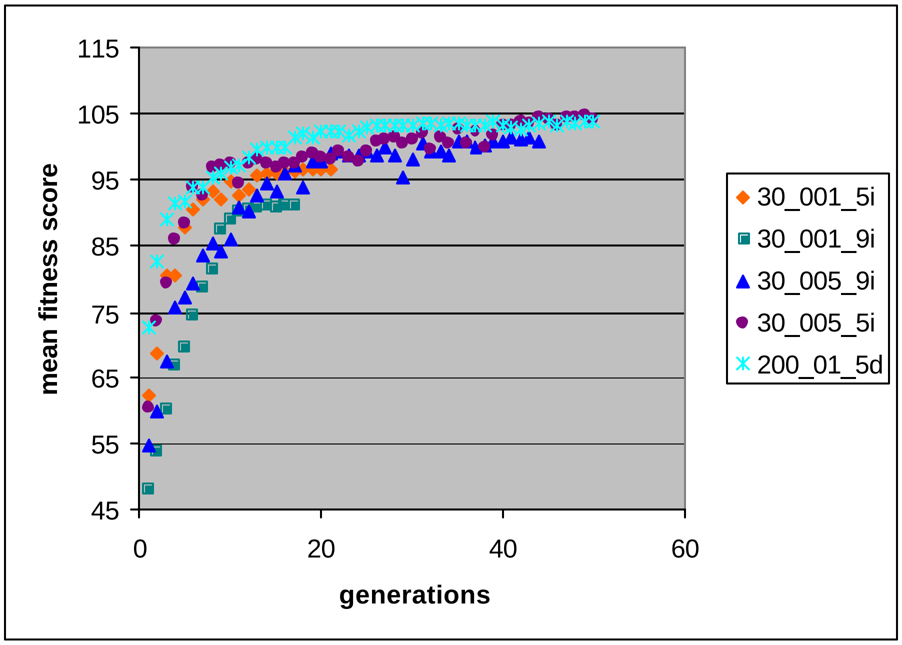


Figure 4.9 Convergence of GA runs with 30 chromosomes, forcing the SPB development stages to choose the same functional form. The first 50 generations of the GA run with 200 chromosomes is shown for comparison. See Table 4.3 for more detailed descriptions of the runs.

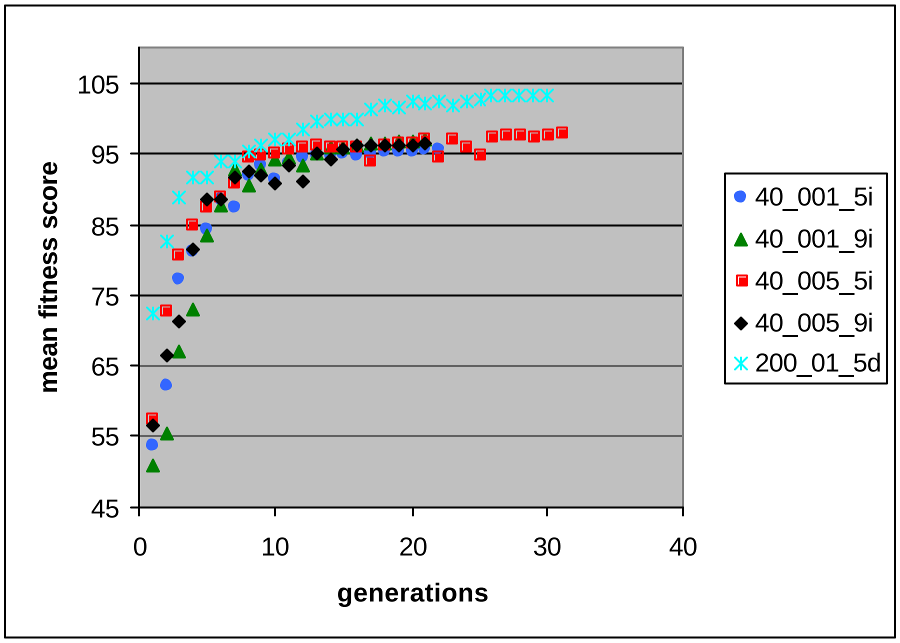


Figure 4.10 Convergence of GA runs with 40 chromosomes, forcing the SPB development stages to choose the same functional form. The first 50 generations of the GA run with 200 chromosomes is shown for comparison. See Table 4.3 for more detailed descriptions of the runs.

4.4.2 Testing uniform functional forms

Using the best GA parameters found above (40 chromosomes, 0.005 mutation rate, and 0.5 crossover rate), the best model requiring all life stages to use the same developmental functional form achieved a fitness of 98.4 against data set *A* and 113.0 against the independent data. However, further testing with other GA parameters showed that there were better models, at least in terms of descriptive performance against data set *A* (Table 4.6). The best model found based on fitness against Data Set *A* overall had a fitness of 105.8, but interestingly, its performance against the independent data was much worse.

4.4.3 Prediction of general trends

The model generated from the GA using the parameters: 200 chromosomes, 0.01 mutation rate, and 0.5 crossover rate, was compared to SPBMODEL in terms of ability to predict the infestations' general trends of growth or decline. "Growth" was defined to be at least a 20% increase over time; anything less was considered "decline". The GA-generated model and SPBMODEL were similar in their predictions of dead tree trends, both overestimating the growth of declining infestations (Tables 4.9, 4.10). The GA-generated model predicted infested tree trends slightly better than SPBMODEL, but all of the GA-generated model errors were overestimations, whereas SPBMODEL errors were more balanced (Tables 4.11, 4.12).

4.4.4 Visual Inspection of Model Performance

Although the best GA-generated model performed measurably better than SPBMODEL against both the dependent and independent data, it still does not perform as well as desired. That is, using a simple graphical comparison, it was hard to see any obvious improvement in the way the GA-generated model matched the observed data (Figures 4.11-4.12).

Table 4.6 Fitness according to the functional form selected by GA runs where functional forms were uniform across life stages. Data Set 'A' is dependent data, Data Set 'B' is independent data. The running parameters of 40_005_5 were expected to achieve high fitness for Data Set 'A.' Compare with Table 4.8. See Table 4.7 for details of the running parameters for these runs.

GA Run ID	Data Set 'A' Fitness	Data Set 'B' Fitness	Fitness Difference Between the Data Sets ('B' - 'A')	Functional Form
30_005_9i	103.7	100.0	-4	Type III
30_005_5i	105.8	87.3	-19	Type III
30_001_9i	92.9	107.6	15	Linear
30_001_5i	97.0	106.8	10	Type III
40_005_9i	98.2	101.1	3	Type III
40_005_5i	98.4	113.0	15	Logan 1
40_001_9i	97.5	98.3	1	Type III
40_001_5i	96.3	114.6	18	Logan 1

Table 4.7 Descriptions of the parameters for GA runs with uniform functional forms across life stages.

GA Run ID	Chromosomes	Mutation Rate	Crossover Rate	Number of Similar Generations Before Termination
30_001_5i	30	0.001	0.5	5
30_001_9i	30	0.001	0.9	5
30_005_5i	30	0.005	0.5	5
30_005_9i	30	0.005	0.9	5
40_005_9i	40	0.005	0.9	5
40_005_5i	40	0.005	0.5	5
40_001_9i	40	0.001	0.9	5
40_001_5i	40	0.001	0.5	5

Table 4.8 Fitness according to the functional form selected by GA runs where functional forms were allowed to vary across life stages. Data Set 'A' is dependent data, Data Set 'B' is independent data. See Table 4.3 for details of the running parameters for these runs.

GA Run ID	Data Set 'A' Fitness	Data Set 'B' Fitness	Fitness Difference Between the Data Sets ('B' - 'A')
30_001_9d	99.8	99.1	-1
30_001_5d	99.0	112.6	14
30_005_9d	100.2	100.0	0
30_005_5d	99.5	101.4	2
40_001_9d	98.4	91.8	-7
40_001_5d	96.5	106.4	10
40_005_9d	100.2	111.9	12
40_005_5d	105.2	96.0	-9

Table 4.9 The numbers of infestations predicted by SPBMODEL to decline in terms of dead trees, vs. the number observed to decline. Decline is defined as less than 20% growth.

	SPBMODEL, Predicted Dead Trees			
Observed Dead Trees		Declining Infestations	Growing Infestations	Total Infestations
	Declining Infestations	1	5	6
	Growing Infestations	1	54	55
	Total Infestations	2	59	61

Table 4.10 The numbers of infestations predicted by the GA -generated model to decline in terms of dead trees, vs. the number observed to decline. Decline is defined as less than 20% growth.

	GA-generated Model, Predicted Dead Trees			
Observed Dead Trees		Declining Infestations	Growing Infestations	Total Infestations
	Declining Infestations	2	4	6
	Growing Infestations	1	54	55
	Total Infestations	3	58	61

Table 4.11 The numbers of infestations predicted by SPBMODEL to decline in terms of infested trees, vs. the number observed to decline. Decline is defined as less than 20% growth.

	SPBMODEL, Predicted Infested Trees			
Observed Infested Trees		Declining Infestations	Growing Infestations	Total Infestations
	Declining Infestations	17	12	29
	Growing Infestations	9	23	32
	Total Infestations	26	35	61

Table 4.12 The numbers of infestations predicted by the GA-generated model to decline in terms of infested trees, vs. the number observed to decline. Decline is defined as less than 20% growth.

	GA-generated Model, Predicted Infested Trees			
Observed Infested Trees		Declining Infestations	Growing Infestations	Total Infestations
	Declining Infestations	29	0	29
	Growing Infestations	15	17	32
	Total Infestations	44	17	61

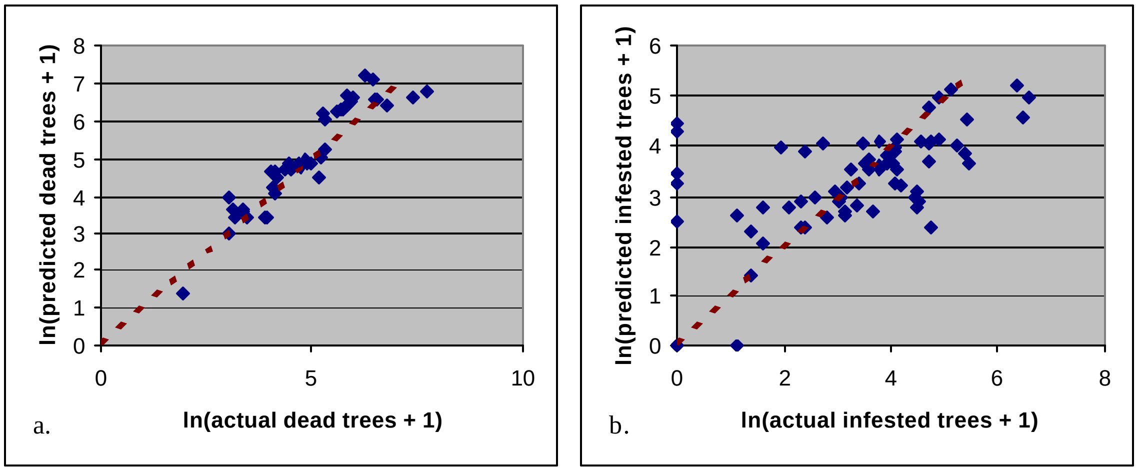


Figure 4.11(a) The GA -generated model prediction of dead trees and (b) The GA -generated model prediction of infested trees compared to the actual number of dead and infested trees in field data set 'A.' The dotted red line represents the line 'y=x.'

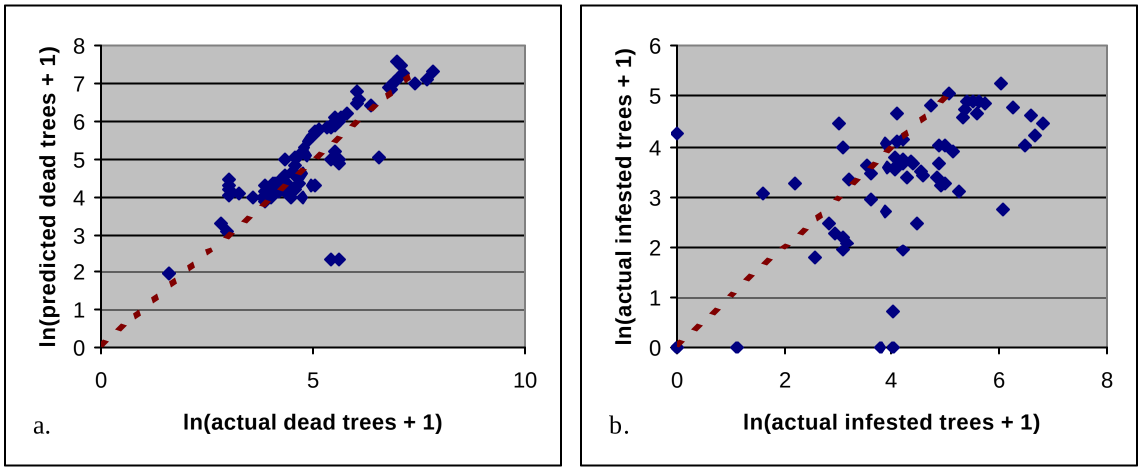


Figure 4.12 (a) The GA -generated model prediction of dead trees and (b) The GA -generated model prediction of infested trees compared to the actual number of dead and infested trees in field data set 'B.' The dotted red line represents the line 'y=x.'

4.5 Discussion

Overall, the GA was able to evolve a modified version of the Javahog model that was superior to SPBMODEL, at least according to the criteria used to judge. However, the model created also included some unrealistic parameters, pointing out the danger of using a purely automated method, or the need for further refinement of the method.

Though this analysis shows clearly the potential for using GAs to help build robust models, it was not extensive enough to answer some specific questions about improvements to SPBMODEL. Additional and more complete data sets would help. Also, it is not clear whether new functional forms should be considered for use in SPBMODEL or whether further broader re-parameterization alone could improve its performance against new data.

Certainly, other submodels or components of SPBMODEL would lend themselves well to this type of analysis. For instance, the functional response of the number of SPB attacks in relation to host density may have a variety of forms and parameters. Similarly, there are a variety of ways to simulate the effects of natural enemies, perhaps involving alternative submodels for competition and the effects of alternate food sources and hosts. Details from other models like TAMBEETLE could be incorporated as options within SPBMODEL, to be selected by a GA. Also, penalties could be included for GAs selecting biologically unrealistic parameters (Table 4.5).

Chapter 5 : Summary and Conclusions

Summary

A new Java version of SPBMODEL, Javahog, was built and its code and extensive documentation were made available globally through the Web. Javahog was shown to produce output essentially identical to the output of the old model written in FORTRAN and translated to C++.

Javahog was extended to allow some of its components, parameters and functions, to be modified by a GA, and the use of GAs to improve the performance of the model has been successfully demonstrated. The value of the reengineering process includes not only the availability of modern and documented code and the accessibility and ease of maintenance afforded by the use of the Web, but also the ability to perform optimizations on the model itself.

Ideas for the Future

GAs may be used to adapt SPBMODEL to new data as it becomes available. SPBMODEL was validated for spots in Alabama, Arkansas, Louisiana, Mississippi, and Texas, but not yet for other states. The southern pine beetle may behave differently according to geography, so SPBMODEL may require recalibration for other southeastern states. GAs would provide a relatively quick way to perform this recalibration. GAs may also help SPBMODEL predict the behavior of winter infestations, which have been historically difficult to predict (Lih and Stephen, 1989).

References

Anonymous. 1996. SPB Field Tally Sheet. Montgomery County, Arkansas, Womble Township. Plot No. 96-33. Visit No. 2. Date Surveyed 7-16-1996.

The Apache Software Foundation. 2002. Java Apache Project. http://java.apache.org. Accessed 6 September 2002.

Ballard, J.L. 1974. A population model for the *Heliothis zea* (Boddie). Ph.D. dissertation, University of Arkansas.

Billings, R.F. 1980. Direct Control. In: The Southern Pine Beetle. Thatcher, R.C., J.L. Searcy, J.E. Coster, G.D. Hertel (eds.). US Department of Agriculture. Forest Service. Science and Education Administration Technical Bulletin 1631.

Boillot, M. 1984. Understanding FORTRAN 77 with Structured Program Solving. West Publishing Company, NY.

Carrano, F.M. 1995. Data Abstraction and Problem Solving with C++: Walls and Mirrors. The Benjamin/Cummings Publishing Company, Inc. Redwood City, CA.

Coulson, R.N., R.M. Feldman, P.J.H. Sharpe, P.E. Pulley, T.L. Wagner, T.L. Payne. 1989. An overview of the TAMBEETLE model of Dendroctonus frontalis population dynamics. Holarctic Ecology 12(4): 445-450.

Curry, G.L. and R.M. Feldman. 1987. Mathematical Foundations of Population Dynamics. Texas A&M University Press, College Station, TX.

Dale, N.B., C. Weems, and M. Headington. 2002. Programming and Problem Solving with C++. 3rd ed. Jones and Bartlett Publishers, Inc. Sudbary, MA.

De Jong, K. A. 1975. An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Ph.D. thesis, University of Michigan, Ann Arbor. As Cited in Mitchell (1999).

Dooley, J.E. 1998. Defining the mission of Virginia Cooperative Extension: An interpretative analysis from a historical perspective. Ph.D. dissertation. Virginia Polytechnic Institute and State University.

Dowd, K. and C.R. Severance. 1998. High Performance Computing, 2nd ed. O'Reilly & Associates, Inc. Sebastopol, CA.

Faupel, Matthew. 2002. Gajit – A simple Java genetic algorithms package. http://www.angelfire.com/ca/Amnesiac/gajit.html. Accessed 6 September 2002.

Gagne, J.A. 1980. The effects of temperature on population processes of the southern pine beetle, *Dendroctonus frontalis* Zimmermann. Ph.D. dissertation. Texas A&M University. As cited in Ghosh (1983).

Ghosh, J. 1983. Mathematical and simulation modeling of southern pine beetle infestations. Ph.D. dissertation, University of Arkansas.

Goldberg, D.E. 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA. As cited in Sequiera et al. (1994).

Grefenstette, J. J. 1986. Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics 16(3/4): 507-522. As cited in Mitchell (1999).

Haahr, Mads. 2002. True random number service. http://www.random.org/. Accessed 3 July 2002.

Hain, F.P. 1980. Sampling and predicting populaiton trends. In: The Southern Pine Beetle. Thatcher, R.C., J.L. Searcy, J.E. Coster, G.D. Hertel (eds.). US Department of Agriculture. Forest Service. Science and Education Administration Technical Bulletin 1631.

Hedden, R.L. 1978. The need for intensive forest management to reduce southern pine beetle activity in east Texas. Southern Journal of Applied Forestry 2:19-22.

Hedden, R.L. and R. F. Billings. 1979. Southern pine beetle: factors influencing the growth and decline of summer infestations in east Texas. Forest Science 70: 547-566.

Hicks, R.R. Jr. 1980. Climatic, site, and stand factors. In: The Southern Pine Beetle. Thatcher, R.C., J.L. Searcy, J.E. Coster, G.D. Hertel (eds.). US Department of Agriculture. Forest Service. Science and Education Administration Technical Bulletin 1631.

High Performance Systems. 1997. Stella II Version 5.0 for Macintosh. 46 Centerra Parkway, Suite 200, Lebanon, NH 03766-1487. Phone 603-643-9636 / Fax 603-643-9502

Hines, G.S. 1979. A simulation model for investigating the population dynamics of *Dendroctonus frontalis* Zimm. Ph.D. dissertation, University of Arkansas.

Horton, I. 1997. Beginning Java. Wrox Press Ltd. Birmingham, UK.

Hunter, J. 2001. Java Servlet Programming. O'Reilly & Associates, Inc. Sebastopol, CA.

Jacucci, G., M. Foy, and C. Uhrik. 1996. Developing transportable agricultural decision support systems: Part 2: an example. Computers and Electronics in Agriculture 14: 301-315.

Jeng, K.J. 1994. Development of animation modules for continuous and discrete systems. Ph.D. dissertation, University of Arkansas.

Koza, 1992. Genetic Programming. MIT Press, Cambridge, MA.

Koza, J.R., M.A. Keane, J. Yu, F.H. Bennett III, and W. Mydlowec. 2000. Automatic Creation of Human-Competitive Programs and Controllers by Means of Genetic Programming. Genetic Programming and Evolvable Machines 1: 121-164.

Koza, J. R., F.H. Bennett, J. Lohn, F. Dunlap, M.A. Keane, D. Andre. 1997. Automated synthesis of computational circuits using genetic programming. Proceedings of the IEEE international conference on evolutionary computation: 447-452.

Lih, M.P. 1985. Refinement and Use of a Simulation Model for the Prediction of Southern Pine Beetle Population Growth and Tree Mortality. Ph.D. dissertation, University of Arkansas.

Lih, M.P. and F.M. Stephen. 1987. Arkansas SPBMODEL – A computer simulation model. Southern Pine Beetle Fact Sheet Number 42. Forest Pest Management Protection Report R8-PR 5. USDA Forest Service.

Lih, M.P. and F.M. Stephen. 1989. Modeling southern pine beetle (Coleoptera: Scolytidae) population dynamics: methods, results and impending challenges. In: McDonald, L., B. Manly, J. Lockwood, J. Logan (eds.) Estimation and Analysis of Insect Populations. In: Berger, J., S. Fienberg, J. Gani, K. Krickeberg, and B. Singer (eds.) Lecture Notes in Statistics. Springer-Verlag, New York.

Lewis, J. and Loftus, W. 2001. Java Software Solutions: Foundations of Program Design. 2nd ed. Addison-Wesley Longman, Inc. NY.

Logan, J.A. 1989. Toward an expert system for pest simulation models: lessons learned from application. BCPC MONO. No. 43 Progress and Prospects in Insect Control. 233-243.

Logan, J.A. 1998. Toward an expert system for development of pest simulation models. Environmental Entomology 17(2): 359-376.

Mitchell, M. 1999. An Introduction to Genetic Algorithms. The MIT Press, Cambridge, Massachusetts.

Moore, G.E. 1978. Factors for determining population trends in southern pine beetle spots. Environmental Entomology 7: 335-342. As cited in Hain (1980).

Motamedi, M. 1981. Sensitivity analysis applied to the simulation model of *Dendroctonus frontalis* Zimm. Ph.D. dissertation, University of Arkansas.

Osborne, W.M. and E.J. Chikofsky. 1990. Fitting pieces to the maintenance puzzle. IEEE Software, Janurary 1990: 10-11. As cited in Pressman (2001).

Pase, H.A. III, Chair, Southern Forest Insect Work Conference. 2002. Letter to Dale Bosworth, Chief, Forest Service, U.S. Department of Agriculture.

Payne, T.L. 1980. Life History and Habitats. In: The Southern Pine Beetle. Thatcher, R.C., J.L. Searcy, J.E. Coster, G.D. Hertel (eds.). US Department of Agriculture. Forest Service. Science and Education Administration Technical Bulletin 1631.

Pidd, M. 1989. Developments in Discrete Simulation. In: Computer Modelling for Discrete Simulation. M. Pidd, ed. John Wile & Sons Ltd. NY.

Pokanzer, Jef. 2002. ACME Laboratories. http://www.acme.com. Accessed 6 September 2002.

Pressman, R.S. 2001. Software Engineering: A Practitioner's Approach. Fifth edition. McGraw-Hill, NY.

Salom, S. M. 1997. Southern Pine Beetle Factsheet. Entomology Publication 444-243. Entomology Department, Virginia Tech. http://everest.ento.vt.edu/~salom/SPBbiology/soupibee.html. Accessed 24 September 2002.

Satterlee, S.M. 2002. Javahog. http://ultra.isis.vt.edu/~sarah/javahog.html. Accessed 24 September 2002.

Saunders, M.C., D.K. Loh, E.J. Rykiel, R.N. Coulson, T.L. Payne, P.E. Pulley, P.J.H. Sharpe, and L. Hu. 1985. The southern pine beetle decision support system. In: Software solutions: proceedings of a computer symposium, software fair and second annual meeting of Forest Resources System Institute, April 21-24, 1985, Clarksville, Indiana. Massey, J.G., B.J. Greber, and T.M. Cooney (eds.).

Sequiera, R.A., R.L. Olson, J.L. Willers, and J.M. McKinion. 1994. Automating the parameterization of mathematical models using genetic algorithms. Computers and Electronics in Agriculture 11: 265-290.

Sharpe and DeMichele. 1977. Reaction kinetics of poikilotherm development. Journal of Theoretical Biology 64: 649-670. Cited in Wagner (1987), Coulson et al. (1989), and Sequiera (1994).

Stephen, F.M. and M.P. Lih. 1985. A Dendroctonus frontalis infestation growth model: organization, refinement, and utilization. Integrated Pest Management Research

Symposium: the Proceedings. General Technical Report, Southern Forest Experimental Station: 186-194.

Sun Microsystems, Inc. 2002. The Source for Java^{TM} technology. http://www.java.sun.com. Accessed 30 September 2002.

Taha, H.A., F.M. Stephen, and M. Motamedi. 1980. Sensitivity analysis and uncertainty in estimation of rates for a southern pine beetle model. USDA, Forest Service Technical Bulletin No. 1630. Modeling Southern Pine Beetle Populations Symposium Proceedings. Asheville, NC. February 1980: 13-19.

Taha, H.A. and F.M. Stephen. 1984. Modeling with imperfect data: A case study simulating a biological system. Simulation 42(3): 109-115.

Thatcher, R.C. 1980. Introduction. In: The Southern Pine Beetle. Thatcher, R.C., J.L. Searcy, J.E. Coster, G.D. Hertel (eds.). US Department of Agriculture. Forest Service. Science and Education Administration Technical Bulletin 1631.

Turnbow, R.H., L.C. Hu, E.J. Rykiel, T.N. Coulson, and D. Loh. 1983. Procedural guide for FERRET, the question analysis system for the decision support system for southern pine beetle management. Texas Agricultural Experiment Station Miscellaneous Publication 1523. As cited in Saunders et al. (1985).

Visual Engineering, Inc. 2002. KavaChart. http://www.ve.com/index.html. Accessed 30 May 2002

Veith, T.L. 2002. Agricultural BMP Placement for Cost-Effective Pollution Control at the Watershed Level. PhD. dissertation. Virginia Polytechnic Institute and State University.

Wagner, T.L, W.S. Fargo, R.O. Flamm, R.N. Coulson, and P.E. Pulley. 1987. Development and Mortality of *Ips calligraphus* (Coleoptera: Scolytidae) at Constant Temperatures. Environmental Entomology 16(2): 484-496.

Wagner, T.L., H-I. Wu, R.M. Feldman, P.J.H. Sharpe, and R.N. Coulson. 1985. Multiple-cohort approach for simulating development of insect populations under variable temperatures. Annals of the Entomological Society of America 78(6): 691-704.

Yeh, G.K, K.K. Bagchi, J.B. Burr, and A.M. Peterson. 1997. Object-oriented simulation and OPERAS. In: Object-Oriented Simulation: Reusability, Adaptability, and Maintainability. Zobrist, G.W. and J.V. Leonard (eds.) IEEE PRESS. NY.

Appendix A: Input and Output Data

Table A.0.1 Key for Table A.2 and Table A.3.

Symbol	**Definition**
⬜	Optional input data
▨	Data cannot be used as input into SPBMODEL, but can be used in calculation of other optional inputs
■	Input datum is missing in field data, so a mean value is substituted
-1	Output datum is missing, so no comparisons of model predictions are made with this datum

Table A.0.2 Data used as input for SPBMODEL and for use with the genetic algorithm. See Table A.1.

State	Date	Spot ID	dbh (cm)	% lob-lolly	pba (m)	hba (m)	trees under attack	trees w/ EO or parents	trees with eggs	trees w/ larvae or pupae	trees w/ brood adults	infest-ed trees	dead trees
AL	6/20/99	9021	23.1	100.0	23.3	6.8	14		0	0	1	15	
AL	6/22/99	9026	18.5	100.0	23.3	6.8	1		1	16	5	23	
AL	7/1/99	9045	28.4	100.0	10.3	13.8	10		4	44	4	62	
AL	7/23/99	9076	41.4	80.0	18.4	9.2	12		9	2	2	25	
AL	7/27/99	9084	35.1	65.3	12.6	6.9	10		10	13	1	34	
AL	8/6/99	9093	34.3	99.0	23.0	3.4	5		14	25	13	57	
AL	6/1/00	0017	21.1	22.0	25.3	13.8	13		13	24	24	74	
AL	6/16/00	0068	21.1	100.0	23.3	6.8	21		28	104	22	175	
AL	7/7/00	0097	17.5	100.0	23.3	6.8	8		9	12	2	31	
AL	7/27/00	0117	23.4	100.0	18.4	18.4	0		2	11	6	19	
AL	8/20/97	AL-4/97	27.5	90.0	20.7	6.9	15		9	12	0	36	
AL	7/12/00	BH#0638	23.4	20.0	23.0	13.8	3		37	11	2	53	
AL	6/20/00	BH#25	29.0	20.0	16.1	11.5	5		15	19	6	45	
AR	6/13/78	AR1	28.5	73.4	40.0	0.0						3	3
AR	6/15/78	AR3	24.4	73.4	39.2	4.2						164	832
AR	6/8/76	AR4	37.2	73.4	26.4	5.7						39	39
AR	6/8/76	AR6	32.7	73.4	23.3	6.8						19	19
AR	7/12/76	AR8	30.7	73.4	28.7	9.9						22	24
AR	7/12/76	AR9	55.3	73.4	16.1	11.5						1	2
AR	7/12/76	AR10	37.2	73.4	20.7	13.8						46	46
AR	6/8/76	AR7+3	35.1	73.4	19.2	9.5						215	285
AR	7/12/76	T13	18.8	73.4	44.8	0.0						6	138
AR	8/16/76	T32	38.2	73.4	23.3	6.8						3	6
AR	7/1/96	womble33	28.4	0.0	13.0	15.0	8		9	22	4	43	6
GA	8/9/79	4197	27.5	73.4	18.8	7.3	1	0	0	0	3	4	49
GA	8/6/79	4220	27.5	73.4	19.3	11.5	32	4	1	10	1	48	71
GA	8/8/79	4888	27.5	73.4	9.6	14.5	26	9	0	96	55	186	332
GA	8/14/79	5060	27.5	73.4	14.5	4.6	78	3	12	61	27	181	321
GA	8/7/79	5066	27.5	73.4	18.6	1.6	2	0	0	1	0	3	18
GA	8/16/79	4570#1	27.5	73.4	18.8	7.8	77	0	15	102	105	299	425
GA	8/15/79	4570#2	27.5	73.4	18.8	7.8	35	3	14	83	38	173	347
GA	7/19/79	5147-48#2	27.5	73.4	20.7	2.3	0	0	0	0	10	10	16
GA	7/19/79	5147-48#5	27.5	73.4	23.2	3.9	208	1	14	160	33	416	492
GA	7/21/79	5174-75	27.5	73.4	25.7	2.3	56	17	13	57	24	167	218
GA	7/22/79	5209#1	27.5	73.4	17.0	3.9	60	40	51	165	33	349	412
GA	7/20/79	5209#5	27.5	73.4	23.7	3.4	57	89	45	220	57	468	598
GA	7/20/79	5211#2	27.5	73.4	32.4	1.1	11	7	2	54	11	85	166
GA	7/20/79	5211#4	27.5	73.4	17.9	0.7	43	20	9	71	35	178	237
GA	7/12/79	5259#2	27.5	73.4	14.9	11.5						5	15
GA	7/22/79	5259#5	27.5	73.4	30.1	3.9	62	5	5	25	1	98	120
GA	8/29/95	GA-10/95	16.3	100.0	27.6	1.1	4		10	0	4	18	
GA	6/23/94	GA-4/94	27.5	100.0	32.2	1.1	25		121	95	13	254	
GA	7/20/97	GA-7/97	13.2	100.0	27.6	2.3	7		8	0	0	15	
GA	7/4/95	GA-8/95	29.2	100.0	29.9	1.1	10		17	0	0	27	
GA	7/20/97	GA-8/97	16.1	100.0	32.2	2.3	23		16	0	0	39	
GA	7/28/97	GA-9/97	26.9	100.0	23.3	6.8	10		24	16	0	50	
LA	7/19/83	LA9	30.5	100.0	22.3	16.1	11		0	45	12	68	
MS	7/21/82	MS2	45.3	95.5	23.3	6.8	7	6	2	5	2	22	36
NC	8/17/95	NC-12/95	27.4	73.4	26.4	6.9	1		14	59	14	88	
NC	8/16/95	NC-13/95	31.2	73.4	12.2	11.5	5		11	13	11	17	
NC	7/15/97	NC-5/97	19.3	0.0	23.3	6.8	11		25	20	9	65	
NC	6/28/95	NC-6/95	33.0	60.0	16.8	9.9	33		25	8	5	71	
NC	7/15/97	NC-6/97	19.1	73.4	23.3	6.8	3		10	13	5	31	
NC	6/28/95	NC-7/95	27.5	100.0	23.3	6.8						158	
TN	8/15/00	Bowaters	17.8	73.4	41.3	0.0	4		11	23	10	48	
TN	8/21/01	TN-5/01	20.3	73.4	33.3	1.1	55		81	35	26	197	
VA	7/5/01	RAAP3	27.5	0.0	20.7	0.0	4		14	3	3	24	
VA	6/20/01	RAAP4	25.4	0.0	32.2	0.0	7		6	12	6	31	
TX	5/22/90	TX19	21.1	77.0	21.0	11.0	44		64	53	0	161	45
TX	5/13/92	TX1	24.6	98.0	18.9	5.5	61		59	72	29	221	321
TX	5/20/91	TX3	28.4	74.0	30.6	14.2	45		40	67	15	167	178

Table A.0.3 Dead and infested trees observed over time, as used for comparison with model predictions, and as input into the genetic algorithm. See Table A.1.

State	Spot ID	Day	Dead Trees	Infested Trees	Group
AL	9021		15	15	b
		8	-1	16	
		16	-1	18	
		22	-1	21	
		29	-1	22	
		36	-1	21	
		43	-1	12	
AL	9026		23	23	a
		9	-1	20	
		17	-1	19	
		23	-1	7	
		30	-1	2	
		37	-1	0	
AL	9045		62	62	b
		8	-1	68	
		14	-1	60	
		21	-1	47	
		28	-1	133	
		35	-1	150	
		42	-1	168	
AL	9076		25	25	a
		8	-1	58	
		14	-1	67	
		22	-1	90	
		29	-1	87	
		36	-1	94	
		42	-1	90	
AL	9084		34	34	b
		8	-1	57	
		16	-1	93	
		23	-1	98	
		30	-1	128	
		37	-1	148	
		44	-1	191	
AL	9093		57	57	a
		7	-1	60	
		14	-1	60	
		21	-1	50	
		28	-1	34	
		34	-1	32	
		41	-1	24	
AL	0017		74	74	a
		7	-1	62	
		14	-1	44	
		21	-1	31	
		28	-1	14	
		35	-1	6	
		42	-1	10	
AL	0068		175	175	a
		19	-1	168	
		44	-1	135	
		63	-1	113	
AL	0097		31	31	a
		7	-1	29	
		13	-1	23	
		21	-1	18	
		28	-1	12	
		35	-1	9	
		42	-1	4	
AL	0117		19	19	a
		7	-1	27	
		14	-1	22	
		21	-1	22	
AL	0117	26	-1	15	
	(continued)	35	-1	9	
		41	-1	4	
AL	AL-4/97		36	36	b
		7	45	-1	
		14	62	-1	
AL	BH#0638		53	53	b
		6	-1	57	
		13	-1	66	
		20	-1	77	
		27	-1	82	
		34	-1	68	
		38	-1	33	
AL	BH#25		45	45	a
		7	-1	50	
		14	-1	56	
		21	-1	51	
		28	-1	43	
		35	-1	43	
		42	-1	34	
AR	AR1		3	3	b
		36	4	2	
		56	4	0	
AR	AR3		832	164	b
		6	901	223	
		14	964	280	
		32	1040	309	
		54	1156	214	
		74	1253	206	
AR	AR4		39	39	b
		34	113	73	
		69	273	137	
		111	718	441	
AR	AR6		19	19	a
		34	22	3	
		69	27	3	
AR	AR8		24	22	a
		35	61	37	
		77	179	118	
AR	AR9		2	1	a
		77	6	2	
AR	AR10		46	46	a
		35	113	61	
AR	AR7+3		285	215	a
		34	870	572	
		69	1613	722	
		111	2288	651	
AR	T13		138	6	b
		35	235	56	
		77	284	43	
AR	T32		6	3	b
		42	16	4	
AR	womble33		6	43	b
		15	19	50	
		24	24	58	
		45	59	35	
		58	72	23	
		73	88	8	
		86	92	4	
GA	4197		49	4	a
		56	19	0	
GA	4220		71	48	a
		92	210	10	

Table A.3 (continued)

State	Spot ID	Day	Dead Trees	Infested Trees	Group
GA	4888		332	186	a
		71	345	0	
GA	5060		321	181	a
		76	658	137	
GA	5066		18	3	b
		71	18	0	
GA	4570#1		425	299	a
		81	523	0	
GA	4570#2		347	173	b
		82	432	21	
GA	5147-48#2		16	10	b
		92	16	0	
GA	5147-48#5		492	416	b
		105	1198	263	
GA	5174-75		218	167	a
		101	700	110	
GA	5209#1		412	349	a
		110	640	228	
GA	5209#5		598	468	b
		103	1106	159	
GA	5211#2		166	85	a
		97	199	0	
GA	5211#4		237	178	b
		103	600	132	
GA	5259#2		15	5	a
		105	19	0	
GA	5259#5		120	98	a
		102	206	0	
GA	GA-10/95		18	18	a
		8	27	-1	
GA	GA-4/94		254	254	a
		7	274	-1	
		13	294	-1	
		19	314	-1	
		26	334	-1	
		33	343	-1	
		42	359	-1	
		48	373	-1	
		57	383	-1	
		64	403	-1	
GA	GA-7/97		15	15	a
		25	23	-1	
		32	30	-1	
		39	46	-1	
		47	49	-1	
GA	GA-8/95		27	27	b
		15	35	-1	
		23	44	-1	
		31	54	-1	
		45	87	-1	
		58	116	-1	
GA	GA-8/97		39	39	b
		25	55	-1	
		32	78	-1	
		39	103	-1	
GA	GA-9/97		50	50	a
		10	55	-1	
		17	61	-1	
		24	77	-1	
		31	87	-1	
		38	90	-1	
		44	107	-1	
		52	145	-1	
LA	LA9		68	68	a
		9	129	95	
		17	187	118	
		30	-1	189	
		49	-1	221	
		66	-1	112	
		72	-1	238	
MS	MS2		36	22	b
		20	56	36	
		41	93	48	
		62	145	89	
		90	161	66	
NC	NC-12/95		88	88	b
		20	122	-1	
		33	131	-1	
		48	158	-1	
NC	NC-13/95		17	40	b
		21	19	-1	
		34	19	-1	
		49	19	-1	
NC	NC-5/97		65	65	a
		17	85	-1	
		31	106	-1	
		44	134	-1	
NC	NC-6/95		71	71	b
		8	73	-1	
		15	93	-1	
		20	101	-1	
		27	123	-1	
		49	250	-1	
NC	NC-6/97		31	31	b
		17	47	-1	
		31	64	-1	
		44	74	-1	
NC	NC-7/95		158	158	b
		8	178	-1	
		15	212	-1	
		20	230	-1	
		27	258	-1	
TN	Bowaters		48	48	b
		27	74	-1	
		48	93	-1	
TN	TN-5/01		197	197	b
		35	336	-1	
TX	TX19		45	161	b
		34	152	114	
		63	233	60	
		91	278	19	
		119	297	0	
TX	TX1		221	321	b
		29	433	413	
TX	TX3		167	178	b
		21	253	249	
		57	441	527	
		93	958	718	
		133	1669	895	
		176	2170	787	
		200	2565	646	
VA	RAAP3		24	24	b
		43	45	-1	
VA	RAAP4		31	31	a
		16	56	-1	
		58	62	-1	

Appendix B: Program Code

B.1 Main Classes and Helper Classes

B.1.1 User.java - Runs the simulation locally

```
/**User.java is a main program for running the Javahog simulation locally
*(predicts population trends in the southern pine beetle, and subsequent
*changes in numbers of dead and infested trees in a stand).  Allows for
*input and output into the simulation.
*/

import JavaSPB1.*;
import java.io.*;
import java.util.*;
import java.util.StringTokenizer;
import Analyzer;
import FileHandler;

/**allows for input and output into Simulator, the SPB population simulator
*@see JavaSPB1
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,        6 June 2002
*/
public class User{

   public Vector _deadPercentErrors;
   public Vector _infestedPercentErrors;
   public Vector _deadSumsOfSquares;
   public Vector _infestedSumsOfSquares;
   public Vector _deadErrors;
   public Vector _infestedErrors;
   public Vector _analyzedFiles;
   public Vector _spotFiles;
   public Vector _checkFiles;
   public double _meanInfestedSumsOfSquares;
   public double _meanDeadSumsOfSquares;
   public int _end;

   /**default constructor*/
   public User(){
      _deadPercentErrors = new Vector();
      _infestedPercentErrors = new Vector();
      _deadSumsOfSquares = new Vector();
      _infestedSumsOfSquares = new Vector();
      _deadErrors = new Vector();
      _infestedErrors = new Vector();
      _analyzedFiles = new Vector();
      _spotFiles = new Vector();
      _checkFiles = new Vector();
   }
```

```
/**gets data from user, runs the simulation, and produces output
*@param args     any information to be sent to this program (not used)
*/
static public void main(String[] args) {
    /*get data from the user*/
    User user = new User();
    String standDataDirectoryName = user.getDirectoryName();
    String checkDataDirectoryName = standDataDirectoryName + "check";
    String outDataDirectoryName = standDataDirectoryName + "out";
    FileHandler fileHandler = new FileHandler();
    Vector allSpots;
    Vector allCheckArrays;
    Vector allCheckFiles;
    /*the number of lines to ignore in a file*/
    int numberOfJunkLines = 86;

    /*whether there are minimum and maximum numbers for the numbers of dead
        and infested trees, true for .SPB files*/
    boolean mins_maxes = false;
    int[][] arrayToCompare;
    int[][] simulationResults;
    double infestedPercentError;
    double deadPercentError;
    int lastDay;
    int position;
    String comparisonFile;
    String spotFile;

    allCheckArrays = new Vector();
    allCheckArrays =
        fileHandler.getFilesInDirectory(checkDataDirectoryName,
        numberOfJunkLines, mins_maxes);
    allCheckFiles = new Vector();
    allCheckFiles = fileHandler.getAllCheckFiles();
    int[] lastDays = fileHandler.getLastDays();
    allSpots = new Vector();
    allSpots = user.getStandFilesInDirectory(standDataDirectoryName);
    int i = 0;
    int j;
    boolean found = false;
    int treeIndex;
    /*allCheckArrays and allSpots have alternating elements of filenames
        and information gathered from those filenames*/
    for(j = 0; j < allCheckArrays.size(); j++){
        comparisonFile = (String)allCheckFiles.elementAt(j);
        for(position = 0; position < allSpots.size(); position ++){
            spotFile = (String)user._spotFiles.elementAt(position);
            if(spotFile.equals(comparisonFile)){
                arrayToCompare = (int[][])allCheckArrays.elementAt(j);
                lastDay = lastDays[i];
                i ++;
                Spot spot = new Spot();
                spot = (Spot)allSpots.elementAt(position);
                user._end = (int)(spot.start() + (lastDay * 24f));
                Simulator simulator = new Simulator(spot, user._end);
                simulator.event();
                simulationResults = simulator.infestedAndDead();
                user.results(outDataDirectoryName + "/" + comparisonFile,
                    spot, simulationResults, arrayToCompare);
                found = true;
            }//end if
        }//end for
    }//end for
    if(!found){
        System.out.println("Error finding matching files.");
    }
}//end main

/**gets the data from a user-specified file; the file is a 2-line
*   tab-delimited file containing only numeric data, contains
*   integer values: _start (in Julian hours), _end (in Julian hours),
*   _dead, _infested, _underAttack, _withImm, _withBrood, on the first line
*   and float values: _loblollyRatio (a decimal proportion, range 0-1),
*   _pineBA (sq m/hectare), _hardwoodBA (sq m/hectare), _standAvgDBH (cm),
*   _longitude (degrees), and _latitude (degrees) on the second line,
*   in that order (Julian hours are hours since 12AM January 1)
*@param inputFileName   the name of the file from which to read the input data
*@return    the name of an output file name, based on the input file name
*/
public Spot standData(String inputFileName){
    int startDate = 0;
    int daysDuration = 0;
    int dead = 0;
    int infested = 0;
    int underAttack = 0;
    int withEggs = 0;
    int withImm = 0;
    int withBrood = 0;
    int start = 0;
    int end = 0;
    float loblollyRatio = 0;
    float pineBA = 0;
    float hardwoodBA = 0;
    float standAvgDBH = 0;
    float longitude = 0;
    float latitude = 0;
    Spot spot;

    StringTokenizer tokenizer, tokenizer2;
    String line, line2;
    //System.out.println("Processing file " + inputFileName);

    try{
        FileReader fr = new FileReader(inputFileName);
        BufferedReader inFile = new BufferedReader(fr);

        line = inFile.readLine();
        tokenizer = new StringTokenizer(line);
        if(line != null){
            try{
                startDate = Integer.parseInt(tokenizer.nextToken());
                daysDuration = Integer.parseInt(tokenizer.nextToken());
```

```
          dead = Integer.parseInt(tokenizer.nextToken());
          infested = Integer.parseInt(tokenizer.nextToken());
          underAttack = Integer.parseInt(tokenizer.nextToken());
          withEggs = Integer.parseInt(tokenizer.nextToken());
          withImm = Integer.parseInt(tokenizer.nextToken());
          withBrood = Integer.parseInt(tokenizer.nextToken());

          start = convertToJulianHours(startDate);
          end = (int)(start + daysDuration * 24f);

      }   catch(NumberFormatException exception){
          System.out.println("Error in input. Line ignored:");
          System.out.println(line);
      }//end catch
  }   else{
      System.out.println("Error: file empty.");
  }//end else

  line2 = inFile.readLine();
  tokenizer2 = new StringTokenizer(line2);
  if(line2 != null){
      try{
          loblollyRatio = Float.parseFloat(tokenizer2.nextToken());
          pineBA = Float.parseFloat(tokenizer2.nextToken());
          hardwoodBA = Float.parseFloat(tokenizer2.nextToken());
          standAvgDBH = Float.parseFloat(tokenizer2.nextToken());
          longitude = Float.parseFloat(tokenizer2.nextToken());
          latitude = Float.parseFloat(tokenizer2.nextToken());
      }   catch(NumberFormatException exception){
          System.out.println("Error in input. Line ignored:");
          System.out.println(line2);
      }//end catch
  }   else{
      System.out.println("Error: file incomplete.");
  }//end else

  /*assume the user did not know the numbers of trees infested with
  individual SPB development stages, unless there were no -1's
  entered for any of those numbers of trees*/
  boolean develDistributionKnown = false;
  if(underAttack != -1 && withEggs != -1 && withImm != -1 &&
      withBrood != -1){

      /*ensure that the number of infested trees equals the numbers of
          trees infested with the individual life stages*/
      if((underAttack + withEggs + withImm + withBrood) != infested){
          System.out.println("Error: trees infested with " +
              "individual life stages of SPB do not sum " +
              "to the total infested trees.");
      }   else{
          develDistributionKnown = true;
      }//end else
  }//end if
  spot = new Spot(start, end, dead, infested, develDistributionKnown,
      underAttack,  withEggs, withImm, withBrood, loblollyRatio,
      pineBA, hardwoodBA, standAvgDBH, longitude, latitude);
```

```
        inFile.close();
     }   catch(FileNotFoundException exception){
        System.out.println("The file " + inputFileName + " was not found.");
        spot = new Spot();
     }   catch (IOException exception){
        System.out.println(exception);
        spot = new Spot();
     }//end catch
     return spot;
  }//end standData

  /**gets the name of the input file, as entered by the user on a keyboard
  *@return    the name of the input file
  */
  public String getFileName(){
      BufferedReader kb =
        new BufferedReader(new InputStreamReader(System.in));
     try{
        System.out.println("\nEnter input file name: ");
        String inputFileName = kb.readLine().trim();
        if(inputFileName.length() == 0){
           return null;
        }
        return inputFileName;
     }   catch(Exception e){
        System.out.println(e);
        return null;
     }//end catch
  }//end getFileName

  /**asks the user for a directory where the input files are
  *@return    the directory name where the input files are
  */
  public String getDirectoryName(){
      BufferedReader kb =
        new BufferedReader(new InputStreamReader(System.in));
     try{
        System.out.println("\nEnter input directory name: ");
        String inputDirectoryName = kb.readLine().trim();
        if(inputDirectoryName.length() == 0){
           return null;
        }
        return inputDirectoryName;
     }   catch(Exception e){
        System.out.println(e);
        return null;
     }//end catch
  }//end getDirectoryName

  /**obtains all of the input files, each containing information
  *   about a specific infestation spot, in a specified directory
  *@param directoryName   the name of the directory where the input files are
  *@return    a Vector of all of the infestation spots that were
  *           described in the input files in the specified directory
  */
  public Vector getStandFilesInDirectory(String directoryName){
    String[] inputFiles;
    Vector allSpots = new Vector();
    File directory = new File(directoryName);
    Spot spot = new Spot();
    if(directory.exists() && directory.isDirectory()){
        inputFiles = directory.list();
        if(inputFiles != null){
            for(int i = 0; i < inputFiles.length; i ++){
                spot = standData(directory + "/" + inputFiles[i]);
                _spotFiles.addElement(inputFiles[i]);
                allSpots.addElement(spot);
            }//end for
        }//end if
    }//end if
    return allSpots;
}//end getStandFilesInDirectory

/**converts the date from mmddyy to Julian date, in hours
*@param date    the date in mmddyy format
*@return        the date in Julian hours
*/
public int convertToJulianHours(int date){
    int month;
    int year;
    int day;
    int jDay;
    int jHour;
    month = (int)(date / 10000f);
    day = (int)((date - month * 10000f) / 100f);
    year = (int)(date - month * 10000f - day * 100f);

    jDay = (int)((month - 1f) * 28f + day + (int)(month / 2f) * 3f +
        (int)(month / 9f) * (float)(month % 2) +
        (int)((month - 2.5f) / 2f) * 2f);
    if(year % 4 == 0 && month > 2){ //leap year
        jDay ++;
    }
    jHour = (int)(24f * jDay);
    return jHour;
}//end convertToJulianHours

/**print the simulation output versus the observed field data to a file
*@param file    the name of the output file
*@param spot    the infestation spot
*@param simulationResults   the results of the simulation
*@param arrayToCompare      the observed field data
*/
public void results(String file, Spot spot,
    int[][] simulationResults, int[][] arrayToCompare){

    int time;
    int day;
    int days;
    int j;
    float population;
    float inflow;
float mort;
float devel;
Analyzer analyzer;
double infestedSumsOfSquares;
double deadSumsOfSquares;
double infestedError;
double deadError;
try{
   FileWriter fw = new FileWriter(file);
   BufferedWriter bw = new BufferedWriter(fw);
   PrintWriter outFile = new PrintWriter(bw);

   outFile.println();
   outFile.println("input variables");
   outFile.println("start: " + spot.start() + " end: " + _end +
      " dead: " + spot.dead() + " infested: " + spot.infested() +
      " underAttack: " + spot.initialUnderAttack() + " withEggs: " +
      spot.initialWithEggs() + " withImm: " + spot.initialWithImm() +
      " withBrood: " + spot.initialWithBrood());
   outFile.println("loblolly: " + spot.loblollyRatio() +
      " pineBA: " + spot.pineBA() + " hardwood: " + spot.hardwoodBA() +
      " DBH: " +  spot.standAvgDBH() + " Long: " + spot.longitude() +
      " Lat: " + spot.latitude());
   outFile.println();
   outFile.println("infested trees");
   days = (int)((_end - spot.start()) / 24f);

   //Hog model file output starts at day 1
   for(day = 1; day <= days; day++){
      outFile.println(day + "  " + simulationResults[0][day] +
         "       " + simulationResults[1][day] + "  " +
         arrayToCompare[0][day] + "  " + arrayToCompare[1][day]);
      outFile.println();
   }//end for
   analyzer = new Analyzer(days + 1);
   analyzer.compareInfestations(arrayToCompare, simulationResults);
   infestedSumsOfSquares = analyzer.sumsOfSquares(0);
   infestedError = analyzer.meanError(0);
   deadSumsOfSquares = analyzer.sumsOfSquares(1);
   deadError = analyzer.meanError(1);
   outFile.println("Infested sums of squares: " + infestedSumsOfSquares);
   outFile.println("Dead sums of squares: " + deadSumsOfSquares);
   outFile.close();
   System.out.println("Output file has been created: " + file);

   Double infestedSumsOfSquaresWrapper = new Double(infestedSumsOfSquares);
   Double deadSumsOfSquaresWrapper = new Double(deadSumsOfSquares);
   _infestedSumsOfSquares.addElement(infestedSumsOfSquaresWrapper);
   _deadSumsOfSquares.addElement(deadSumsOfSquaresWrapper);
   _analyzedFiles.addElement(file);

   Double infestedErrorWrapper = new Double(infestedError);
   Double deadErrorWrapper = new Double(deadError);
   _infestedErrors.addElement(infestedErrorWrapper);
   _deadErrors.addElement(deadErrorWrapper);
}   catch (IOException exception){
        System.out.println(exception);
      }//end catch
  }//end results

  /**prints the percent error difference between the simulation output
  *   and the observed field data to a file
  *@param outFileName     the name of the output file for the percent errors
  */
  public void writePercentErrors(String outFileName){
      Double infestedWrapper = new Double(0d);
      Double deadWrapper = new Double(0d);
      String analyzedFile;
      double sumInfestedErrors = 0d;
      double sumDeadErrors = 0d;
      try{
          FileWriter fw = new FileWriter(outFileName);
          BufferedWriter bw = new BufferedWriter(fw);
          PrintWriter outFile = new PrintWriter(bw);
          for(int i = 0; i < _analyzedFiles.size(); i ++){
              analyzedFile = (String)_analyzedFiles.elementAt(i);
              infestedWrapper = (Double)(_infestedPercentErrors.elementAt(i));
              deadWrapper = (Double)(_deadPercentErrors.elementAt(i));
              outFile.println(analyzedFile + "   " +
                  infestedWrapper.doubleValue() + " " +
                  deadWrapper.doubleValue());
              sumInfestedErrors += infestedWrapper.doubleValue();
              sumDeadErrors += deadWrapper.doubleValue();
          }//end for
          outFile.println("overall infested % error: " +
              sumInfestedErrors/_infestedPercentErrors.size());
          outFile.println("overall dead % error: " +
              sumDeadErrors/_deadPercentErrors.size());
          outFile.close();
      }   catch(IOException e){
          System.out.println(
              "Error printing infested and dead percent errors.");
      }
  }//end writePercentErrors

  /**prints the sum of squared error difference between the simulation output
  *   and the observed field data to a file
  *@param outFileName     the name of the output file for the sum of squared errors
  */
  public void writeSumsOfSquares(String outFileName){
      Double infestedWrapper = new Double(0d);
      Double deadWrapper = new Double(0d);
      String analyzedFile;
      double sumInfestedSumsOfSquares = 0d;
      double sumDeadSumsOfSquares = 0d;
      try{
          FileWriter fw = new FileWriter(outFileName);
          BufferedWriter bw = new BufferedWriter(fw);
          PrintWriter outFile = new PrintWriter(bw);
          for(int i = 0; i < _analyzedFiles.size(); i ++){
              analyzedFile = (String)_analyzedFiles.elementAt(i);
              infestedWrapper = (Double)(_infestedSumsOfSquares.elementAt(i));
```

```
            deadWrapper = (Double)(_deadSumsOfSquares.elementAt(i));
            outFile.println(analyzedFile + "   " +
               infestedWrapper.doubleValue() + " " +
               deadWrapper.doubleValue());
            sumInfestedSumsOfSquares += infestedWrapper.doubleValue();
            sumDeadSumsOfSquares += deadWrapper.doubleValue();
         }//end for
         _meanInfestedSumsOfSquares = sumInfestedSumsOfSquares /
            _infestedSumsOfSquares.size();
         _meanDeadSumsOfSquares = sumDeadSumsOfSquares /
            _deadSumsOfSquares.size();
         outFile.println("mean infested sums of squares: " +
            _meanInfestedSumsOfSquares);
         outFile.println("mean dead sums of squares: " +
            _meanDeadSumsOfSquares);
         outFile.close();
      }   catch(IOException e){
         System.out.println(
            "Error printing infested and dead sums of squares.");
      }
   }//end writeSumsOfSquares

   /**returns the mean infested sums of squared error of infested trees
   *   between the simulation output and the observed field data to a file
   *@return       the mean sums of squared error between the numbers of
   *              infested trees predicted by the model and those observed
   *              in the field
   */
   public double meanInfestedSumsOfSquares(){
      return _meanInfestedSumsOfSquares;
   }

   /**returns the mean dead sums of squared error of infested trees
   *   between the simulation output and the observed field data to a file
   *@return       the mean sums of squared error between the numbers of
   *              dead trees predicted by the model and those observed
   *              in the field
   */
   public double meanDeadSumsOfSquares(){
      return _meanDeadSumsOfSquares;
   }
}//end User
```

B.1.2 javahog.java - Runs the Javahog simulation via a servlet

```
/**Javahog.java is the servlet for the Javahog program.  It obtains
*    information about SPB-infested forest stands via a web-based
*    user interface.  The servlet then makes sure that all the
*    information is of the right type and within the set bounds.
*    Then the servlet runs the Javahog model (based on SPBMODEL, as
*    documented in Gail Hines' 1979 PhD dissertation from Arkansas
*    University, "A simulation model for investigating the population
*    dynamics of Dendroctonus frontalis Zimm.").  Finally, the servlet
*    outputs the numbers of trees infested by SPB and killed by SPB
*    for each day in the simulation.
*Conditions: the server on which javahog is running is set up such
*    that it knows where javahog is, and knows javahog is a servlet.
*    The server must also be correctly configured for importing
*    java.awt.*
*/

import javaSPB1.*;
import servletHelper.*;
import java.io.*;
import java.awt.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import Acme.JPM.Encoders.GifEncoder;
import javachart.chart.*;

/**runs a simulation of southern pine beetle populations over time,
*    depending on initial forest stand conditions and date
*@see <a href = http://java.apache.org/jserv/index.html> Apache JServ</a>
*@see <a href =
http://www.acme.com/java/software/Package-Acme.JPM.Encoders.html>
GifEncoder</a>
*@see <a href = http://www.ve.com>KavaChart</a>
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  8 July 2002
*/
public class javahog extends HttpServlet{
   /**prints out the title of the javahog webpage, and calls handleData()
   *@param request          provides user request information for the servlet
   *@param response         the response to the user via HTTP
   *@exception ServletException   the servlet may have an error processing data
   *@exception IOException         an error may occur while reading from or
   *                               printing to an HTML page
   */
   public void doGet (HttpServletRequest request,
       HttpServletResponse response) throws ServletException, IOException{
          handleData(request,response);
   }//end doGet

   /**gets & parses all of the information about the SPB spot
       from a form and reprints the form and any errors.  It also
       calls the actual javahog program (runs the simulation)
```

```
    and outputs the simulation results.
  *@param request          provides user request information for HTTP servlets
  *@param response        sends a response to the user via HTTP
  */
  protected void handleData(HttpServletRequest request,
    HttpServletResponse response){

    TestBounds test = new TestBounds(1);
    PrintForm form1 = new PrintForm(1);
    PrintTrees trees1 = new PrintTrees(1);
    PrintPops pops1 = new PrintPops(1);
    PrintTable table1 = new PrintTable(1);
    Delimited delimit1 = new Delimited(1);

    /*initialize all variables to hold input from the form, &
       to send information to the simulation*/
    /*the state of the form -- when spotData=0, the form has not
      been submitted, when =1, it has been submitted */
    int spotData = 0;

    boolean treeChart = false;
    boolean table = false;
    boolean delimit = false;
    String spotID = "";      //the name of the SPB spot
    String dbhUnits = ""; //metric,English units (cm or in)
    String pBAunits = "";         //sq.ft/acre or sq.m/hectare
    String hBAunits = "";         //sq.ft/acre or sq.m/hectare
    char state = ' ';                          //abbreviation for the U.S. state
    int start = 0;        //the starting day of the simulation
    int end =0;                   //the ending day of the simulation
    int dead = 0;        //# of initially SPB-killed trees
    int rawDead = 0;
    int underAttack = 0;   //# trees initially under attack
    int rawUnderAttack = 0;           //holds user input for underAttack
    /*initially assume user does not know SPB brood distribution*/
    boolean develDistributionKnown = false;
    int withEggs =0;         //# of trees initially with eggs
    int rawWithEggs= 0;        //holds user input for withEggs
    int withImm = 0;        //# trees initially with larvae/pupae
    int rawWithImm = 0;        //holds user input for withImm
    int withBrood =0;       //# trees initially with brood adults
    int rawWithBrood = 0;             //holds user input for withBrood
    int yesOldAttack = 0;
    int yesAttack = 0;
    int yesParent = 0;
    int yesEgg = 0;
    int yesImm = 0;
    int yesBrood = 0;
    int yesEmerge = 0;
    boolean yesDead = false;
    boolean yesInfested = false;
    int infested =0;           //# of initially SPB-infested trees
    int duration =0;           //the duration of the simulation
    int rawDuration = 0;
    int month =0;        //the month the simulation starts
    int day =0;                   //day of month the simulation starts
```

```
int year =0;   //the year for the simulation
float  loblollyRatio = (float) 0;       //decimal proportion of loblolly in stand,i.e. 0.75
float pineBA = (float)0;            //the pine basal area
float rawPineBA = (float)0;      //holds user input for pineBA
float hardwoodBA = (float)0;   //the hardwood basal area
float rawHardwoodBA = (float)0;          //holds user input for hardwoodBA
float standAvgDBH = (float)0; //the average diameter at breast height for all trees in the stand
float rawStandAvgDBH = (float)0;        //holds user input for standAvgDBH
//longitude and latitude are just estimates for the state
double  longitude = 0;              //the longitude of the spot
double  latitude =0;    //the latitude of the spot

//define an errorString to append all errors concerning
//    entries in the form
StringBuffer errorString = new StringBuffer("");

//get the parameters from the query string
Enumeration enum = request.getParameterNames();

/*Progress through enum so that each of the input parameter
   values can be assigned to the appropriate servlet
   variable.  The query string seems to list parameters
   in no particular order, so this while loop does not
   require certain parameters to be parsed before others. */
while(enum.hasMoreElements()){
   //define the input parameter name
   String name = (String)enum.nextElement();

   //get the input parameter value from the query string
   String value = request.getParameter(name);

   //spotData is a hidden field
   if(name.equals("spotData")){
      if(!value.equals("")){
         try{
            //spotData is set to 1 when the
            //   form is submitted
            spotData = Integer.parseInt(value);
         }
         catch(NumberFormatException nfe){
            errorString.append( "Error: the form was not " +
               "named with an integer value. <br>");
         }
      }//end if
      else{
         errorString.append("Error accessing spot data. <br>");
      }
   } else if(name .equals("spotID")){
      if(!value.equals("")){
         spotID = value;
      } else {
         errorString.append("Error: a name for the spot must be " +
               "entered. <br>");
         }
   } else if(name.equals("state")){
      if(!value.equals(""))
```

```
        state = value.charAt(0);
      } else if(name.equals("month")){
        try{
            month = Integer.parseInt(value);
        } catch(NumberFormatException nfe){
            errorString.append("Error parsing the value for month." +
                "<br>");
        }
    } else if(name.equals("day")){
        try {
            day = Integer.parseInt(value);
        } catch(NumberFormatException nfe){
            errorString.append("Error parsing the value for the day." +
                    "<br>");
        }
    } else if(name.equals("year")){
        if(!value.equals("")){
            try{
                year = Integer.parseInt(value);
                year = test.yearBounds(year, errorString);
            } catch(NumberFormatException nfe){
            errorString.append( "Error: the value for the year must " +
                    "be an integer. <br>");
            }
          }  else{
            errorString.append("Error: a value must be entered for " +
                    "the year. <br>");
        }
    } else if(name.equals("dead")){
        if(!value.equals("")){
            try{
                dead = Integer.parseInt(value);
                rawDead = dead;
                dead = test.treeBounds(dead, errorString);
            } catch(NumberFormatException nfe){
                errorString.append( "Error: the value for previously" +
                    " infested (SPB-killed) trees must be an " +
                    "integer. <br>");
            }
        } else {
            errorString.append("Error: a value must be entered for " +
                    "the number of previously infested (SPB-killed) " +
                    "trees. <br>");
        }
    } else if(name.equals("infested")){
        if(!value.equals("")){
            try{
                infested = Integer.parseInt(value);
                infested = test.treeBounds(infested, errorString);
            } catch(NumberFormatException nfe){
                errorString.append( "Error: the value for infested " +
                    "trees must be an integer. <br>");
            }
        } else{
            errorString.append("Error: a value must be  entered for " +
                    "the number of infested trees. <br>");
    }
  } else if(name.equals("attack")){
    /*"attack" is optional input -- if the user does not enter a
    value for the number of trees currently under SPB infestation,
    then the servlet will divide the number of infested trees among
    all the stages of SPB infestation (eggs, larvae, etc.)*/

    if(!value.equals("")){
      try{
        underAttack = Integer.parseInt(value);
        rawUnderAttack = underAttack;
        underAttack = test.treeBounds(underAttack, errorString);
      } catch(NumberFormatException nfe){
        errorString.append("Error: the value for trees under " +
          "attack must be an integer. <br>");
      }
    } else {      //input is optional, so don't output error
      underAttack = 0;
    }
  } else if(name.equals("eggs")){
    /*"eggs" is optional input -- if the user does not
    enter a value for number of trees currently with SPB eggs, then
    the servlet will divide the number of infested trees among all
    the stages of SPB infestation (eggs, larvae, etc.)*/

    if(!value.equals("")){
      try{
        withEggs = Integer.parseInt(value);
        rawWithEggs = withEggs;
        withEggs = test.treeBounds(withEggs, errorString);
      } catch(NumberFormatException nfe){
        errorString.append( "Error: the value for trees with " +
          "eggs must be an integer. <br>");
      }
    } else {      //input is optional; don't output error
      withEggs = 0;
    }
  } else if(name.equals("immatures")){
    /*"immatures" is optional input--if user does not
    enter a value for number of trees currently with larvae/pupae,
    then servlet will divide the number of infested trees among all
    the stages of SPB infestation(eggs, larvae, etc.) */

    if(!value.equals("")){
      try{
          withImm = Integer.parseInt(value);
          rawWithImm = withImm;
          withImm = test.treeBounds(withImm, errorString);
      } catch(NumberFormatException nfe){
          errorString.append( "Error: the value for trees " +
          "with larvae or pupae must be an integer." +
           "<br>");
      }//end catch
    } else {      //input is optional; don't output error
      withImm = 0;
    }
```

```
} else if(name.equals("brood")){ /*"brood" is optional input --
   if the user does not enter a value for number of trees
   currently with SPB brood adults, servlet will divide the
   number of infested trees among all the stages of SPB
   infestation(eggs, larvae, etc.)*/

   if(!value.equals("")){
      try{
         withBrood = Integer.parseInt(value);
         rawWithBrood = withBrood;
         withBrood = test.treeBounds(withBrood, errorString);
      } catch(NumberFormatException nfe){
         errorString.append( "Error: the value for trees " +
            "with brood adults must be an integer. <br>");
      }//end catch
   }else{       //input is optional; don't output error
      withBrood = 0;
   }
} else if(name.equals("loblolly")){
   if(!value.equals("")){
      try{
         loblollyRatio = Float.parseFloat(value);
         loblollyRatio = test.ratioBounds(loblollyRatio,
            errorString);
      } catch(NumberFormatException nfe){
         errorString.append( "Error: the value for " +
            "loblolly pine proportion must be numeric." +
            "<br>");
      }//end catch
   }else
      errorString.append("Error: a value must be entered " +
         "for the decimal proportion of loblolly in the " +
         "stand. <br>");
} else if(name.equals("DBH")){
   if(!value.equals("")){
      try{
         standAvgDBH = Float.parseFloat(value);
         rawStandAvgDBH = standAvgDBH;
      } catch(NumberFormatException nfe){
         errorString.append( "Error: the value for the " +
            "stand average DBH must be numeric. <br>");
      }//end catch
   } else{
      errorString.append("Error: a value must be " +
            "entered for the stand average DBH. <br>");
   }
} else if(name.equals("DBHunits")){
   if(!value.equals("")){
      dbhUnits = value;
   }
} else if(name.equals("hardwood")){
   if(!value.equals("")){
      try{
         hardwoodBA = Float.parseFloat(value);
         rawHardwoodBA = hardwoodBA;
      } catch(NumberFormatException nfe){
            errorString.append("Error: the value for " +
              "hardwood BA must be numeric. <br>");
          }
        } else{
          errorString.append("Error: a value must be " +
            "entered for the hardwood BA. <br>");
        }
      } else if(name.equals("hBAunits")){
        if(!value.equals("")){
          hBAunits = value;
        }
      } else if(name.equals("pine")){
        if(!value.equals("")){
          try{
            pineBA = Float.parseFloat(value);
            rawPineBA = pineBA;
          } catch(NumberFormatException nfe){
            errorString.append( "Error: the value for pine " +
              "BA must be numeric. <br>");
          }
        } else{
          errorString.append("Error: a value must be entered " +
            "for the pine BA. <br>");
        }
      } else if(name.equals("pBAunits")){
        if(!value.equals("")){
          pBAunits = value;
        }
      } else if(name.equals("duration")){
        if(!value.equals("")){
          try{
            duration = Integer.parseInt(value);
            rawDuration = duration;
            duration = test.durationBounds(duration, errorString);
            /*duration + 1 to account for initial values*/
            duration += 1;
          } catch(NumberFormatException nfe){
            errorString.append( "Error: the number of days " +
              "for simulation must be an integer value. <br>");
          }
        } else{
          errorString.append("Error: a value must be entered for " +
            "the number of days for the simulation. <br>");
          }
      } else if(name.equals("oldAttacking")){
        if(value.equals("yes")){
          yesOldAttack = 1;
        }
      } else if(name.equals("attacking")){
        if(value.equals("yes")){
          yesAttack = 1;
        }
      } else if(name.equals("parent")){
        if(value.equals("yes"))       {
          yesParent = 1;
        }
        } else if(name.equals("egg")){
            if(value.equals("yes")){
                yesEgg = 1;
            }
        } else if(name.equals("imm")){
            if(value.equals("yes")){
                yesImm = 1;
            }
        } else if(name.equals("broodAdult")){
            if(value.equals("yes")){
                yesBrood = 1;
            }
        } else if(name.equals("emerge")){
            if(value.equals("yes")){
                yesEmerge = 1;
            }
        } else if(name.equals("deadChecked")){
            if(value.equals("yes")){
                yesDead = true;
            }
        } else if(name.equals("infestedChecked")){
            if(value.equals("yes")){
                yesInfested = true;
            }
        } else if(name.equals("theAction")){
            /*"theAction" is a radio button, where "run" is the
            value for the trees choice and "pops" is the
            value for the beetle populations choice */

            if(value.equals("run")){
                treeChart = true;
            } else if(value.equals("pops")){
                treeChart = false;
            }
        } else if(name.equals("outFormat")){
            /*outFormat is a select list, where graphical means
            graphical output is requested, comma means comma-
            delimited output is requested, and table means an
            HTML table is requested*/
            if(value.equals("graphical")){
                table = false;
            } else if(value.equals("comma")){
                table = true;
                delimit = true;
            } else {
                table = true;
                delimit = false;
            }
        }//end else if
    }//end while(enum.hasMoreElements)

    /*Convert rough input data to simulation parameters, 1st by
        determining the Julian days for start and end of the simulation*/
    start = test.convertToJulian(month, day, year, errorString);
    end = (int)(start + duration);
```

```
if(underAttack==0 && withEggs==0 && withImm==0 && withBrood==0){
   develDistributionKnown = false;
}

if(((yesOldAttack + yesAttack + yesParent + yesEgg + yesImm +
   yesBrood + yesEmerge) == 0) && !treeChart && !table){

    errorString.append("Error: please select at least one " +
      "population to simu late. <br>");
}
if(treeChart && !yesDead && !yesInfested && !table){
   errorString.append("Error: please select either dead or " +
      "infested trees to simulate. <br>");
}

/*Ensure that the numbers of trees in each stage of SPB
   infestation sum to the number of infested trees*/
infested = test.infested(underAttack, withEggs, withImm,
   withBrood, infested, errorString);
if((underAttack != 0) && (withEggs != 0) &&
         (withImm != 0) && (withBrood != 0)){

   develDistributionKnown = true;
}

//see if DBH is out of range, according to its units
standAvgDBH = test.dBHbounds(standAvgDBH, dbhUnits, errorString);

//see if pine BA is out of range, according to its units
pineBA = test.pBAbounds(pineBA, pBAunits, errorString);

//see if hardwood BA is out of range, according to units
hardwoodBA = test.hBAbounds(hardwoodBA, hBAunits, errorString);

//get a latitude and longitude for the state
double[] location = test.estimateLatLong(state, errorString);
latitude = location[0];
longitude = location[1];

/*convert the error StringBuffer to a String so that it's
   easier to check if it's empty*/
String errors = errorString.toString();

/*If there were no errors AND spotData has been changed (the
   form has been submitted), then run the simulation.*/
if(errors.equals("") && spotData!=0){
   //initialize the spot
   Spot spot = new Spot((int)start*24, (int)end*24, dead, infested,
      develDistributionKnown, underAttack, withEggs,
      withImm, withBrood, loblollyRatio, pineBA, hardwoodBA,
      standAvgDBH, (float)longitude, (float)latitude);
   Simulator sim1 = new Simulator(spot, (int)end*24);

   //run the simulation
   sim1.event();
   try{
        if(table){
            response.setContentType("text/html");
            ServletOutputStream out = response.getOutputStream();
            if(delimit){
                delimit1.delimited(request, out, spotID, month, day,
                    year, state, rawDead, infested, rawUnderAttack,
                    rawWithEggs, rawWithImm, rawWithBrood,
                    loblollyRatio, rawStandAvgDBH, dbhUnits, rawPineBA,
                    pBAunits, rawHardwoodBA, hBAunits, rawDuration,
                    sim1);
            } else{
                table1.printTable(request, out, spotID, month, day,
                    year, state, rawDead, infested, rawUnderAttack,
                    rawWithEggs, rawWithImm, rawWithBrood,
                    loblollyRatio, rawStandAvgDBH, dbhUnits, rawPineBA,
                    pBAunits, rawHardwoodBA, hBAunits, rawDuration,
                    sim1);
            }
            out.close();
        } else{
            response.setContentType("image/gif");
            ServletOutputStream out = response.getOutputStream();
            //print out the simulation results
            if(treeChart){
                trees1.printTrees(sim1, duration, out, yesDead,
                      yesInfested, spotID);
            } else{
                pops1.printPops(sim1, duration, out, yesOldAttack,
                    yesAttack, yesParent, yesEgg, yesImm, yesBrood,
                    yesEmerge, spotID);
            }
            out.close();
        }//end else
    } catch(IOException e){
    }
} else{
    //if this is 1st call to server, print out a default form
    try{
        response.setContentType("text/html");
        ServletOutputStream out = response.getOutputStream();
        if(spotData == 0){
        form1.printForm(out, 1, "my_spot", 6, 1, 2001, 'A', 0, 25, 0,
            0, 0, 0, 0.95f, 16f, "in", 100f, "ft", 0f, "ft", 25,
            "http://ultra.isis.vt.edu/servlet/javahog",
            "http://ultra.isis.vt.edu/~sarah/images/goButton.gif",
            "http://ultra.isis.vt.edu/~sarah/images/helpButton.gif",
            "http://ultra.isis.vt.edu/~sarah/help.html",
            "http://whizlab.isis.vt.edu/servlet/sf/spbicc/" +
            "topic.html?topic=toolbox",
            "http://whizlab.isis.vt.edu/servlet/sf/spbicc/");
        } else{
            /*If the form was submitted, but there were errors,
                then print out the errors */

            try{
                out.println("<HTML><BODY BGCOLOR=\"#FFFFFF\"><br><UL>");
            out.print(errors);
            out.println("</UL></BODY></HTML>");
          } catch(IOException e){
          }
        }//end else
          out.close();
      } catch(IOException e){
      }
    }//end else
  }//end handleData
}//end javahog
```

B.1.3 GaUser.java - Runs the genetic algorithm

```
/**GaUser.java runs a genetic algorithm for finding an optimized version
*   of the Javahog simulation model (which predicts population trends in the
*   southern pine beetle, and subsequent changes in numbers of dead and
*   infested trees in a stand).
*@author    Sarah Satterlee
*@version 1.01   7 July 2002
*/

import JavaSPB1.*;
import User;
import UK.na.faupel.mc.gajit.*;
import UK.na.faupel.mc.util.Iterator;
import UK.na.faupel.mc.util.ListIterator;
import UK.na.faupel.mc.util.SortedList;
import java.util.*;
import java.io.*;

/**Runs a genetic algorithm for finding an optimized version of the Javahog
*   simulation model.  Creates a chromosome representation of each version of the model,
*   and allows crossover and mutation to occur with each generation of new
*   models.  Mutation allows for new models to be created during the course of the program.
*   Selects the best chromosomes (the ones that predict observed data
*   best) for replication at each generation.  The end result should be a very good version
*   of the model.
*@see JavaSPB1
*/
public class GaUser extends User
{
   /**the number of genes in the GA chromosome*/
   static final int NUMGENES = 65;

   /**the size of each gene (a gene is a binary string)*/
   static final int SIZEGENES = 15;

   /**the maximum number of generations for the GA to run*/
   static final int MAXGENERATIONS = 100;

   /**the minimum number of generations for the GA to run*/
   static final int MINGENERATIONS = 5;

   /**the minimum number of identical generations before the GA terminates*/
   static final int MINGENERATIONS_NOCHANGE = 5;

   /**the number of GA chromosomes*/
   static final int NUMCHROMS = 30;

   Vector _finalInfestedErrors;
   Vector _finalDeadErrors;

   /**constructor*/
   public GaUser(){
      super();
      _finalInfestedErrors = new Vector();
```

```
    _finalDeadErrors = new Vector();
  }

  /**gets data from user, runs the genetic algorithm, and produces output
  *@param args     any information to be sent to this program (not used)
  */
  static public void main(String[] args) {
    /*get data from the user*/

    GaUser gaUser = new GaUser();
    String standDataDirectoryName = gaUser.getDirectoryName();
    String checkDataDirectoryName = standDataDirectoryName + "check";
    String outDataDirectoryName = standDataDirectoryName + "out";
    FileHandler fileHandler = new FileHandler();
    Vector allSpots;
    Vector allCheckArrays;
    Vector allCheckFiles;

    /*the number of lines to ignore in a file, 87 for SPB files*/
    int numberOfJunkLines = 87;

    /*whether there are minimum and maximum numbers for the numbers of dead
       and infested trees, true for .SPB files*/
    boolean mins_maxes = false;
    int[][] arrayToCompare;
    int[][] simulationResults;
    double[][] fitnesses = new double[NUMCHROMS + 1][MAXGENERATIONS + 1];
    int[] averageFitnesses = new int[MAXGENERATIONS + 1];
    double sumFitness;
    double infestedPercentError;
    double deadPercentError;
    int lastDay;
    int position;
    String comparisonFile;
    String spotFile;
    allCheckArrays = new Vector();
    allCheckArrays =
       fileHandler.getFilesInDirectory(checkDataDirectoryName,
       numberOfJunkLines, mins_maxes);
    allCheckFiles = new Vector();
    allCheckFiles = fileHandler.getAllCheckFiles();
    int[] lastDays = fileHandler.getLastDays();
    allSpots = new Vector();
    allSpots = gaUser.getStandFilesInDirectory(standDataDirectoryName);
    int k;
    int j;
    int stage = 0;
    int params = 0;
    boolean found = false;
    double mutationRate = 0.005;
    double cullRate = 0.25;
    double crossOverRate = 0.5;
    double eliteRate = 0.1;
    XOverGeneOp crossOver = new XOverGeneOp(SIZEGENES);
    Population people = new Population(NUMCHROMS, SIZEGENES, NUMGENES,
       NUMGENES, mutationRate);
```

```
people.addOp(crossOver, crossOverRate);
int stages = 6;
int[] develTypes = new int[stages + 1];
double[] rhos = new double[stages + 1];
double[] psis = new double[stages + 1];
double[] sigmas = new double[stages + 1];
double[] k1s = new double[stages + 1];
double[] k2s = new double[stages + 1];
double[] ds = new double[stages + 1];
double[] minTemps = new double[stages + 1];
double[] maxTemps = new double[stages + 1];
double[] optimalTemps = new double[stages + 1];
double[] maxRates = new double[stages + 1];
double maxEggDevelRate;
double minEggDevelRate;
int gen;
int unmatchedAverages = 0;
double tmpTemp;
double fitness;
int geneIndex = 0;
int numberOfDevelGenes = 10;
double[][] develParameters;
int chromosome;
int summedChromosomes = 0;
int problemIterations = 0;
int develType = 0;
int theDevelType = 0;
double attackingMortSlope = 0.001;
double attackingMortIntercept = 0.5;
double emergingMortSlope = 0.001;
double emergingMortIntercept = 0.5;

View viewMortSlope = new FixView(SIZEGENES, 0.1, 0.9);
View viewMortIntercept = new FixView(SIZEGENES, 0.001, 0.01);
View viewDevelType = new FixView(SIZEGENES, -0.4, 7.4);
View viewRho = new FixView(SIZEGENES, 0.0, 1.0);
View viewPsi = new FixView(SIZEGENES, 0.0, 1.0);
View viewSigma = new FixView(SIZEGENES, 30.0, 90.0);
View viewK1 = new FixView(SIZEGENES, 0.0, 1.0);
View viewK2 = new FixView(SIZEGENES, 0.0, 30.0);
View viewD = new FixView(SIZEGENES, 50.0, 100.0);
View viewMinTemp = new FixView(SIZEGENES, 40.0, 60.0);
View viewMaxTemp = new FixView(SIZEGENES, 80.0, 100.0);
View viewOptimalTemp = new FixView(SIZEGENES, 70.0, 90.0);
View viewMaxRate = new FixView(SIZEGENES, 0.0, 0.1);
int chromNumber = 0;
people.setCullRate(cullRate);
people.setEliteRate(eliteRate);
try{
   FileWriter fw = new FileWriter("ParameterValues.dat");
   BufferedWriter bw = new BufferedWriter(fw);
   PrintWriter outFile = new PrintWriter(bw);
   for (Iterator censo = people.iterator(); censo.hasNext(); ){
      outFile.println(censo.next());
   }
   int i = 0;
```

```
boolean converged = false;
while(!converged && i < MAXGENERATIONS){
   Iterator censo;

   if (i % 10 == 0){
      censo = people.iteratorAll();
      System.out.println("Generation : " + i +
         ", Resetting fitness values." );
   } else{
      censo = people.iteratorNewOnly();
      System.out.println("generation " + i);
   }

   chromNumber = 0;
   while (censo.hasNext()){
      fitness = 0d;
      ChromItem tmp = (ChromItem) censo.next();
      Chrom chrom = tmp.getChrom();
      geneIndex = 0;
      numberOfDevelGenes = 10;
      //System.out.println("Getting gene values");
      develType = (int)(Math.abs(
         ((Double)viewDevelType.getGene(chrom, geneIndex)).doubleValue()));
      for(stage = 0; stage < stages; stage++){
         rhos[stage] = Math.abs(
            ((Double)viewRho.getGene(chrom, geneIndex + 1)).doubleValue());
         psis[stage] = Math.abs(
            ((Double)viewPsi.getGene(chrom, geneIndex + 2)).doubleValue());
         sigmas[stage] = Math.abs(
            ((Double)viewSigma.getGene(chrom, geneIndex + 3)).doubleValue());
         k1s[stage] = 0d - Math.abs(
            ((Double)viewK1.getGene(chrom, geneIndex + 4)).doubleValue());
         k2s[stage] = Math.abs(
            ((Double)viewK2.getGene(chrom, geneIndex + 5)).doubleValue());
         ds[stage] = Math.abs(
            ((Double)viewD.getGene(chrom, geneIndex + 6)).doubleValue());
         minTemps[stage] = Math.abs(
            ((Double)viewMinTemp.getGene(chrom, geneIndex + 7)).doubleValue());
         maxTemps[stage] = Math.abs(
            ((Double)viewMaxTemp.getGene(chrom, geneIndex + 8)).doubleValue());
         optimalTemps[stage] = Math.abs(
            ((Double)viewOptimalTemp.getGene(chrom, geneIndex + 9)).doubleValue());
         maxRates[stage] = Math.abs(
            ((Double)viewMaxRate.getGene(chrom, geneIndex + 10)).doubleValue());
         geneIndex += 10;

         while((optimalTemps[stage] > maxTemps[stage]) ||
            (minTemps[stage] > optimalTemps[stage])){

            if(optimalTemps[stage] > maxTemps[stage]){
               /*if optimal temp is greater than max temp, switch their
                     values*/
               tmpTemp = maxTemps[stage];
               maxTemps[stage] = optimalTemps[stage];
               optimalTemps[stage] = tmpTemp;
            }
```

```
        if(minTemps[stage] > optimalTemps[stage]){
            tmpTemp = minTemps[stage];
            minTemps[stage] = optimalTemps[stage];
            optimalTemps[stage] = tmpTemp;
        }//end if
    }//end while
}//end for stage


attackingMortSlope = Math.abs(
    ((Double)viewMortSlope.getGene(chrom, 67)).doubleValue());
attackingMortIntercept = Math.abs(
    ((Double)viewMortIntercept.getGene(chrom, 68)).doubleValue());
emergingMortSlope = Math.abs(
    ((Double)viewMortSlope.getGene(chrom, 69)).doubleValue());
emergingMortIntercept = Math.abs(
   ((Double)viewMortIntercept.getGene(chrom, 70)).doubleValue());

Mortality attackingMortality = new Mortality(attackingMortSlope,
    attackingMortIntercept);
Mortality emergingMortality = new Mortality(emergingMortSlope,
    emergingMortIntercept);
develParameters = new double[stages + 1][numberOfDevelGenes + 1];
for(stage = 0; stage < stages; stage ++){
    develParameters[stage][0] = rhos[stage];
    develParameters[stage][1] = psis[stage];
    develParameters[stage][2] = sigmas[stage];
    develParameters[stage][3] = k1s[stage];
    develParameters[stage][4] = k2s[stage];
    develParameters[stage][5] = ds[stage];
    develParameters[stage][6] = minTemps[stage];
    develParameters[stage][7] = maxTemps[stage];
    develParameters[stage][8] = optimalTemps[stage];
    develParameters[stage][9] = maxRates[stage];
    develTypes[stage] = develType;
}//end for

outFile.println(i + " <-generation ");

for(int stageIndex = 0; stageIndex < 6; stageIndex ++){
    outFile.print(stageIndex + " <-stage \n");
    outFile.print(develTypes[stageIndex] + " <-develType ");
    outFile.print(rhos[stageIndex] + " <-rho ");
    outFile.print(psis[stageIndex] + " <-psi ");
    outFile.print(sigmas[stageIndex] + " <-sigma \n");
    outFile.print(k1s[stageIndex] + " <-k1 ");
    outFile.print(k2s[stageIndex] + " <-k2 ");
    outFile.print(ds[stageIndex] + " <-d " + "\n");
    outFile.print(minTemps[stageIndex] + " <-minTemp ");
    outFile.print(maxTemps[stageIndex] + " <-maxTemp ");
    outFile.print(maxRates[stageIndex] + " <-maxRate ");
    outFile.print(optimalTemps[stageIndex] + " <-optimalTemp ");
}//end for

outFile.println(attackingMortSlope + " <-attackingMortSlope ");
outFile.print(attackingMortIntercept + " <-attackingMortIntercept ");
outFile.print(emergingMortSlope + " <-emergingMortSlope ");
outFile.print(emergingMortIntercept + " <-emergingMortIntercept ");
outFile.println(fitness + " <-fitness ");

k = 0;
problemIterations = 0;
for(j = 0; j < allCheckArrays.size(); j ++){
   comparisonFile = (String)allCheckFiles.elementAt(j);
   for(position = 0; position < allSpots.size(); position ++){
      spotFile = (String)gaUser._spotFiles.elementAt(position);
      if(spotFile.equals(comparisonFile)){
         arrayToCompare = new int[2][gaUser._end + 1];
         simulationResults = new int[2][gaUser._end + 1];
         arrayToCompare = (int[][])allCheckArrays.elementAt(j);
         lastDay = lastDays[k];
         k ++;
         Spot spot = new Spot();
         spot = (Spot)allSpots.elementAt(position);
         gaUser._end = (int)(spot.start() + (lastDay * 24f));

         GASimulator simulator = new GASimulator(spot,
            gaUser._end, develParameters, develTypes,
            attackingMortality, emergingMortality);
         simulator.event();
         simulationResults = simulator.infestedAndDead();
         problemIterations += simulator.problemIterations();
         gaUser.results(outDataDirectoryName + "/" +
            comparisonFile, spot,
            simulationResults, arrayToCompare);
         found = true;
      }//end if
   }//end for
}//end for
if(!found){
   outFile.println("Error finding matching files.");
}
if(problemIterations > 0){
   System.out.println("problemIterations " + problemIterations);
}
fitness = gaUser.fitness() - problemIterations;
problemIterations = 0;
tmp.setFitness(fitness);
fitnesses[chromNumber][i] = fitness;
System.out.println(fitness);
if(i > MINGENERATIONS){
   sumFitness = 0d;
   //System.out.println("Calculating average fitness");
   summedChromosomes = 0;
   for(chromosome = 0; chromosome < NUMCHROMS; chromosome ++){
      if(fitnesses[chromosome][i] > 0){
         sumFitness += fitnesses[chromosome][i];
         summedChromosomes ++;
      }
   }//end for
   averageFitnesses[i] = (int)(sumFitness/(double)summedChromosomes);
   gen = i;
```

```
                unmatchedAverages = 0;
                while(gen > (i - MINGENERATIONS_NOCHANGE)){
                    if(averageFitnesses[gen] != averageFitnesses[gen - 1]){
                        unmatchedAverages++;
                    }
                    gen --;
                }
                if(unmatchedAverages == 0){
                    converged = true;
                }
            }//end if(i> MINGENERATIONS)

            gaUser._analyzedFiles.removeAllElements();
            gaUser._deadErrors.removeAllElements();
            gaUser._infestedErrors.removeAllElements();
            gaUser._finalInfestedErrors.removeAllElements();
            gaUser._finalDeadErrors.removeAllElements();
            chromNumber ++;

        }//end while
        if (i < MAXGENERATIONS && !converged){
            people.newGeneration();
        }//end if
        i++;
    }//end while

    outFile.println();
    for(int genNum = 0; genNum < i; genNum ++){
        outFile.print(genNum + " ");
        for(int chromNum = 0; chromNum < NUMCHROMS; chromNum ++){
            outFile.print(fitnesses[chromNum][genNum] + " ");
        }
        outFile.println();
    }
    outFile.close();
} catch(IOException exception){
}//end catch

SortedList results = new SortedList( people );
ListIterator iter = results.listIterator( results.size() );

try{
    FileWriter fileW = new FileWriter("Results.dat");
    BufferedWriter bufferedW = new BufferedWriter(fileW);
    PrintWriter outFile2 = new PrintWriter(bufferedW);
    outFile2.println(" Population size: " + NUMCHROMS);
    outFile2.println(" Mutation rate: " + mutationRate);
    outFile2.println(" Cull rate: " +  cullRate);
    outFile2.println(" Crossover rate: " + crossOverRate);

    while (iter.hasPrevious()){
        //System.out.println("Printing results");
        ChromItem item = (ChromItem) iter.previous();
        Chrom chrom = item.getChrom();
        //print out the last values of all of the chromosomes
        theDevelType = (int)((Double)viewDevelType.getGene(chrom, 0)).doubleValue();
```

```
            for(params = 0; params < 65; params += 10){
              outFile2.print( theDevelType + " ");
                      outFile2.print( viewRho.getGene(chrom, params + 1) + " " );
              outFile2.print( viewPsi.getGene(chrom, params + 2) + " " );
              outFile2.print( viewSigma.getGene(chrom, params + 3) + " ");
              outFile2.print( -((Double)viewK1.getGene(
                chrom, params + 4)).doubleValue() + " ");
              outFile2.print( viewK2.getGene(chrom, params + 5) + " ");
              outFile2.print( viewD.getGene(chrom, params + 6) + " ");
              outFile2.print( viewMinTemp.getGene(chrom, params + 7) + " ");
              outFile2.print( viewMaxTemp.getGene(chrom, params + 8) + " ");
              outFile2.print( viewOptimalTemp.getGene(
                chrom, params + 9) + " ");
              outFile2.print( viewMaxRate.getGene(chrom, params + 10) + " ");
              outFile2.println();
            }//end for
            outFile2.print(viewMortSlope.getGene(chrom, 67) + " ");
            outFile2.print(viewMortIntercept.getGene(chrom, 68) + " ");
            outFile2.print(viewMortSlope.getGene(chrom, 69) + " ");
            outFile2.print(viewMortIntercept.getGene(chrom, 70) + " ");
            outFile2.println( " = " + item.getDoubleFitness() );
          }//end while
          outFile2.close();
        } catch(IOException e){
        }//end catch
      }//end main

      /**print the GA-produced simulator output data versus the observed field data to a file
      *@param file    the name of the output file
      *@param spot    the infestation spot
      *@param simulationResults   the results of the simulation (from a GA-produced simulator)
      *@param arrayToCompare      the observed field data
      */
      public void results(String file, Spot spot,
        int[][] simulationResults, int[][] arrayToCompare){

        int days;
        Analyzer analyzer;
        double infestedError;
        double deadError;
        double finalInfestedError;
        double finalDeadError;
        days = (int)((_end - spot.start()) / 24f);
        analyzer = new Analyzer(days + 1);
        analyzer.compareInfestations(arrayToCompare, simulationResults);
        infestedError = analyzer.meanError(0);
        deadError = analyzer.meanError(1);
        finalInfestedError = analyzer.meanFinalError(0);
        finalDeadError = analyzer.meanFinalError(1);
        _analyzedFiles.addElement(file);
        Double infestedErrorWrapper = new Double(infestedError);
        Double deadErrorWrapper = new Double(deadError);
        _infestedErrors.addElement(infestedErrorWrapper);
        _deadErrors.addElement(deadErrorWrapper);
        Double finalInfestedErrorWrapper = new Double(finalInfestedError);
        Double finalDeadErrorWrapper = new Double(finalDeadError);
        _finalInfestedErrors.addElement(finalInfestedErrorWrapper);
        _finalDeadErrors.addElement(finalDeadErrorWrapper);
    }//end results

    /**calculates the fitness of the GA chromosome, as determined by the average
    *   error of all of the dead trees and all of the infested trees predicted for
    *   all of the infestation spots, versus the true numbers of dead and infested
    *   trees for all of the available days from all of those spots
    *@return          the fitness of the GA chromosome
    */
    public double fitness(){
        Double infestedWrapper = new Double(0d);
        Double deadWrapper = new Double(0d);
        Double finalInfestedWrapper = new Double(0d);
        Double finalDeadWrapper = new Double(0d);
        String analyzedFile;
        double grandMeanInfestedError = 0d;
        double grandMeanDeadError = 0d;
        double sumMeanInfestedError = 0d;
        double sumMeanDeadError = 0d;
        double sumMeanFinalInfestedError = 0d;
        double sumMeanFinalDeadError = 0d;
        double meanFinalInfestedError = 0;
        double meanFinalDeadError = 0d;
        double fitness = 0d;

        for(int i = 0; i < _analyzedFiles.size(); i ++){
            analyzedFile = (String)_analyzedFiles.elementAt(i);
            infestedWrapper = (Double)(_infestedErrors.elementAt(i));
            deadWrapper = (Double)(_deadErrors.elementAt(i));
            finalInfestedWrapper = (Double)(_finalInfestedErrors.elementAt(i));
            finalDeadWrapper = (Double)(_finalDeadErrors.elementAt(i));
            sumMeanInfestedError += infestedWrapper.doubleValue();
            sumMeanDeadError += deadWrapper.doubleValue();
            sumMeanFinalInfestedError += finalInfestedWrapper.doubleValue();
            sumMeanFinalDeadError += finalDeadWrapper.doubleValue();
        }//end for
        grandMeanInfestedError = sumMeanInfestedError / _infestedErrors.size();
        grandMeanDeadError = sumMeanDeadError / _deadErrors.size();
        meanFinalInfestedError = sumMeanFinalInfestedError/_finalInfestedErrors.size();
        meanFinalDeadError = sumMeanFinalDeadError/_finalDeadErrors.size();
        fitness = 10d / (Math.max(0.001, grandMeanInfestedError +
            grandMeanDeadError + 2 * meanFinalInfestedError + 3 * meanFinalDeadError));
        return fitness;
    }//end fitness
}//end GaUser
```

B.1.4 FileHandler.java

```
/**FileHandler.java reads in files containing true observations for numbers of dead and
*   infested trees over time, and stores the observations in arrays and Vectors.
*/

import java.util.*;
import java.io.*;

/**Reads in files containing true observations for numbers of dead and
*   infested trees over time, and stores the observations in arrays and Vectors.
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  8 July 2002
*/
public class FileHandler {

   int _lastDay;
   int[] _lastDays;

   /**the names of the files containing true infestation data*/
   protected Vector _allCheckFiles;

   /** Creates new FileHandler */
   public FileHandler() {
      _allCheckFiles = new Vector();
   }

   /**retrieves a Vector of arrays containing "true" observations of dead and infested trees over time,
   *   where each infestation is stored in a separate file in a specified directory
   *@param directoryName   the name of the directory where the "true" infestations are
   *@param junkLines       the number of lines of the file to ignore before reading in the
   *                       observations of dead and infested trees, =87 in an SPBMODEL output file
   *@param mins_maxes      if false, there is just one value for dead trees, and one for infested
   *                       trees; if true, there are minimum and maximum values denoting the
   *                       possible range for the numbers of dead and infested trees, as in
   *                       an output file from SPBMODEL
   *@return                a Vector of arrays containing true observations of dead and infested trees
   *                       over time, where each array holds observations from one infestation
   */
   public Vector getFilesInDirectory(String directoryName, int junkLines, boolean mins_maxes){
      Vector allCheckArrays = new Vector();
      File directory = new File(directoryName);
      if(directory.exists() && directory.isDirectory()){
         String[] checkFiles = directory.list();
         _lastDays = new int[checkFiles.length];
         if(checkFiles != null){
            for(int i = 0; i < checkFiles.length; i ++){
               int[][] currentArray = fileToArray(directory + "/" + checkFiles[i], junkLines, mins_maxes);
               _lastDays[i] = _lastDay;
               _allCheckFiles.addElement(checkFiles[i]);
               allCheckArrays.addElement(currentArray);
            }//end for
         }//end if
      }   else{
         System.out.println("Error: directory was incorrectly specified.");
      }//end else
      return allCheckArrays;
   }//end getFilesInDirectory

   /**returns the names of the files containing true infestation data
   *@return    the names of the files containing true infestation data
   */
   public Vector getAllCheckFiles(){
      return _allCheckFiles;
   }

   /**converts a file containing "true" observations of dead and infested trees over time
   *   into an array of those observations, where the first dimension of the array =2, where
   *   0 represents infested trees, and 1 represents dead trees, and where the second dimension
   *   of the array =the total number of days elapsed during the true infestation
   *@param filename        the name of the file containing the "true" infestation
   *@param junkLines       the number of lines of the file to ignore before reading in the
   *                       observations of dead and infested trees, =87 in an SPBMODEL output file
   *@param mins_maxes      if false, there is just one value for dead trees, and one for infested
   *                       trees; if true, there are minimum and maximum values denoting the
   *                       possible range for the numbers of dead and infested trees, as in
   *                       an output file from SPBMODEL
   *@return                an array containing true observations of dead and infested trees
   *                       over time, containing a -1 where an observation is missing for that day
   */
   public int[][] fileToArray(String filename, int junkLines,
      boolean mins_maxes){

      int day = 0;
      int dayIndex;
      int infested;
      int dead;
      int minInfested = 0;
      int maxInfested;
      int minDead;
      int maxDead = 0;

      int[][] fullArray = new int[3][300];
      int[][] partialArray = new int[2][300];
      StringTokenizer tokenizer, tokenizer2;
      String line;
      boolean EOF = false;    //end of file flag
      boolean lookingForData = true;
      boolean firstPass; //indicates first pass through while loop

      _lastDay = 0;
      if(mins_maxes){
         try{
            FileReader fr = new FileReader(filename);
            BufferedReader inFile = new BufferedReader(fr);
            int numberOfLines = 0;
            while(!EOF && numberOfLines < junkLines){
               line = inFile.readLine();
               numberOfLines ++;
            }//end while
        line = inFile.readLine();

        while(lookingForData && !EOF && line!= null){
            tokenizer = new StringTokenizer(line);
            try{
                day = Integer.parseInt(tokenizer.nextToken());
                if(day == 1){
                    lookingForData = false;
                    firstPass = true;
                    while(line!=null && !lookingForData){
                        if(!firstPass){ //already found day for the first pass through while
                            day = Integer.parseInt(tokenizer.nextToken());
                        }

                        minInfested = Integer.parseInt(tokenizer.nextToken());
                        maxInfested = Integer.parseInt(tokenizer.nextToken());
                        minDead = Integer.parseInt(tokenizer.nextToken());
                        maxDead = Integer.parseInt(tokenizer.nextToken());

                        try{
                            fullArray[0][day] = day;
                            fullArray[1][day] =
                                (int)Math.max((int)(minInfested + maxInfested) / 2f, 0);
                            fullArray[2][day] =
                                (int)Math.max((int)(minDead + maxDead) / 2f, 0);
                        }   catch(ArrayIndexOutOfBoundsException e){
                            System.out.println("Error: SPB input file " +
                                "contains a value for a day that is out " +
                                "of bounds");
                        }//end catch

                        line = inFile.readLine();
                        if(line != null){
                            tokenizer = new StringTokenizer(line);
                        }
                        firstPass = false;
                    }//end while
                }   else if(day > 1){
                    System.out.println(
                        "Error: too many lines skipped in file.");
                }
            }   catch(NumberFormatException e){
                lookingForData = true;
            }   catch(NoSuchElementException e){
                lookingForData = true;
            }   catch(IOException exception){
                System.out.println(exception);
            }
            line = inFile.readLine();
        }//end while
        _lastDay = day;

    }   catch (EOFException e){ //end of file reached
        EOF = true;
    }   catch (FileNotFoundException exception){
        System.out.println("The file " + filename + " was not found.");
    }   catch (IOException exception){
        System.out.println(exception);
    }//end catch
} else if(!mins_maxes){
    try{
        FileReader fr = new FileReader(filename);
        BufferedReader inFile = new BufferedReader(fr);

        line = inFile.readLine();

        do {
            tokenizer = new StringTokenizer(line);
            try{
                day = Integer.parseInt(tokenizer.nextToken());
                lookingForData = false;
                firstPass = true;
                while(line!=null && !lookingForData){
                    if(!firstPass){ //already found day for the first pass through while
                        day = Integer.parseInt(tokenizer.nextToken());
                    }

                    dead = Integer.parseInt(tokenizer.nextToken());
                    infested = Integer.parseInt(tokenizer.nextToken());
                    try{
                        fullArray[0][day] = day;
                        fullArray[1][day] = infested;
                        fullArray[2][day] = dead;
                    }   catch(ArrayIndexOutOfBoundsException e){
                        System.out.println("Error: SPB input file " +
                            "contains a value for a day that is out " +
                            "of bounds");
                    }//end catch

                    line = inFile.readLine();
                    if(line != null){
                        tokenizer = new StringTokenizer(line);
                    }
                    firstPass = false;
                }//end while
            }   catch(NumberFormatException e){
                lookingForData = true;
            }   catch(NoSuchElementException e){
                lookingForData = true;
            }   catch(IOException exception){
                System.out.println(exception);
            }
            line = inFile.readLine();
        } while(lookingForData && !EOF && line!= null);
        _lastDay = day;
    }   catch (EOFException e){ //end of file reached
        EOF = true;
    }   catch (FileNotFoundException exception){
        System.out.println("The file " + filename + " was not found.");
    }   catch (IOException exception){
        System.out.println(exception);
        }//end catch
    }//end else

    //mark unknown values with a -1
    for(int j = 1; j <= _lastDay; j ++){
        if(fullArray[0][j] != j){ //if the day was not entered
            fullArray[1][j] = -1;
            fullArray[2][j] = -1;
        }//end if
        partialArray[0][j] = fullArray[1][j];
        partialArray[1][j] = fullArray[2][j];
    }//end for
    return partialArray;
}//end fileToArray

/**retrieves the last day for which there was an observation for dead and/or infested trees
*   from the last infestation read in
*@return    the last day of observed data for the last infestation file read
*/
public int getLastDay(){
    return _lastDay;
}

/**retrieves the last days for which there were observations for dead and/or infested trees
*   from all infestations
*@return    the last days of observed data for all infestation files
*/
public int[] getLastDays(){
    return _lastDays;
}
}//end FileHandler
```

B.1.5 Analyzer.java

```
/**Analyzer.java compares the simulator output about the spread of an SPB
*   infestation over time to a set of true data of the infestation spread over time
*/

import java.util.*;

/**tests model predictions against observed data
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  8 July 2002
*/
public class Analyzer {
   double[][] _percentErrors;
   double[][] _squareDifferences;
   int _simulationDuration;
   double _meanErrorInfested;
   double _meanErrorDead;
   double _meanFinalErrorInfested;
   double _meanFinalErrorDead;

   /** Creates new Analyzer
   *@param arrayLength     duration of the simulation, in hours
   */
   public Analyzer(int arrayLength) {
      _percentErrors = new double[2][arrayLength];
      _squareDifferences = new double[2][arrayLength];
      _simulationDuration = arrayLength;
      _meanErrorInfested = 0;
      _meanErrorDead = 0;
      _meanFinalErrorDead = 0;
      _meanFinalErrorInfested = 0;
   }

   /**compares two sets of data of the SPB infestation spread
   *@param trueInfestation     the infestation considered to have true values
   *@param testInfestation     the simulator output of the infestation over
   *                           time
   */
   public void compareInfestations(int[][] trueInfestation, int[][] testInfestation){
      int i = 0;
      int j = 1;
      double differenceInfested = 0;
      double differenceDead = 0;
      double sumFinalDifferenceInfested = 0;
      double sumFinalDifferenceDead = 0;
      double sumInfested = 0;
      double sumDead = 0;

      try{
         for(j = 1; j < _simulationDuration; j ++){
            i = 0;
            differenceInfested = compareTransformedValues(trueInfestation[i][j], testInfestation[i][j], i, j);
            sumInfested += differenceInfested;
```

```
                if(j == (_simulationDuration - 1)){
                    sumFinalDifferenceInfested += differenceInfested;
                }
                i = 1;
                differenceDead = compareTransformedValues(trueInfestation[i][j], testInfestation[i][j], i, j);
                sumDead += differenceDead;
                if(j == (_simulationDuration - 1)){
                    sumFinalDifferenceDead += differenceDead;
                }
            }
            _meanErrorInfested = sumInfested/_simulationDuration;
            _meanErrorDead = sumDead/_simulationDuration;
            _meanFinalErrorInfested = sumFinalDifferenceInfested/_simulationDuration;
            _meanFinalErrorDead = sumFinalDifferenceDead/_simulationDuration;

        }   catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Error: an attempt was made to access a value out " +
                "of the bounds of the infestation arrays; value at: " + i + " " + j);
        }
    }//end compareInfestations

    /**calculates the R-square statistic, a measure of the difference between
    *   the simulator-predicted values and the "true values," observed in the field or otherwise
    *@param trueInfestations    all of the true infested or dead tree observations
    *@param testInfestations    all of the simulator-predicted observations
    *@param treeType            specifies dead or infested trees, treeType=0 for infested, =1 for dead
    *@return                    the R-square value
    */
    public double calculateR2(Vector trueInfestations, Vector testInfestations, int treeType){
        int i = 0;
        int j = 1;
        double sumTrueTrees = 0;
        double sumTestTrees = 0;
        int[][] testInfestation;
        int[][] trueInfestation;
        double meanTrueTrees = 0;
        double meanTestTrees = 0;
        int vectorIndex = 0;
        double rSquare = 0;
        //testInfestation = x, trueInfestation = y
        double sxy = 0;
        double syy = 0;
        double sxx = 0;
        int checkedDays = 0;
        if(!(treeType == 0 || treeType == 1)){
            System.out.println("Error: treeType not infested or dead.");
            return 0;
        }
        try{
            for(vectorIndex = 0; vectorIndex < testInfestations.size(); vectorIndex ++){
                testInfestation = (int[][])testInfestations.elementAt(vectorIndex);
                trueInfestation = (int[][])trueInfestations.elementAt(vectorIndex);
                for(j = 1; j < testInfestation[0].length; j++){
                    if((trueInfestation[treeType][j] != -1) && (testInfestation[treeType][j] != -1)){
                        sumTrueTrees += trueInfestation[treeType][j];
                        sumTestTrees += testInfestation[treeType][j];
                        checkedDays ++;
                    }
                }//end for
            }//end for vectorIndex


            meanTrueTrees = sumTrueTrees / checkedDays;
            meanTestTrees = sumTestTrees / checkedDays;

            for(vectorIndex = 0; vectorIndex < testInfestations.size(); vectorIndex ++){
                testInfestation = (int[][])testInfestations.elementAt(vectorIndex);
                trueInfestation = (int[][])trueInfestations.elementAt(vectorIndex);
                int missingValues = 0;
                for(j = 1; j < testInfestation[0].length; j++){
                    if((trueInfestation[treeType][j] != -1) && (testInfestation[treeType][j] != -1)){
                        sxy += (trueInfestation[treeType][j] - meanTrueTrees) *
                            (testInfestation[treeType][j] - meanTestTrees);
                        sxx += Math.pow((testInfestation[treeType][j] - meanTestTrees), 2);
                        syy += Math.pow((trueInfestation[treeType][j] - meanTrueTrees), 2);
                    } else {
                        missingValues ++;
                    }
                }//end for j
            }//end for vectorIndex

            if(syy > 0 && sxx > 0){
                rSquare = Math.min((sxx / syy), (syy / sxx));
            } else {
                rSquare = 0;
            }//end else

        } catch(ArrayIndexOutOfBoundsException aioobe){
            System.out.println("Error: an attempt was made to access a value out " +
            "of the bounds of the infestation arrays; value at: " + i + " " + j);
        }
        return rSquare;
    }//end calculateR2

    /**retrieves the mean error between the observations of true and predicted
    *   observations of dead or infested trees over the course of the simulation
    *@param treeType          specifies dead or infested trees, treeType=0 for infested, =1 for dead
    *@return    the mean error between true and predicted dead or infested trees
    */
    public double meanError(int treeType){
        if(treeType == 0){
            return _meanErrorInfested;
        } else if(treeType == 1){
            return _meanErrorDead;
        } else {
            System.out.println("Error: neither dead nor infested trees were specified.");
            return 9999;
        }
    }//end meanError

    /**retrieves the mean error of just the final true and final predicted
    *   observations of dead or infested trees
```

```
*@param treeType          specifies dead or infested trees, treeType=0 for infested, =1 for dead
*@return    the mean error between final observations of true and predicted dead or infested trees
*/
public double meanFinalError(int treeType){
    if(treeType == 0){
       return _meanFinalErrorInfested;
    } else if(treeType == 1){
       return _meanFinalErrorDead;
    } else {
       System.out.println("Error: neither dead nor infested trees was specified.");
       return 9999;
    }
}//end meanFinalError

/**compares a true and a predicted observation of dead or infested trees at one time step
*@param trueValue   the true observation of dead or infested trees
*@param testValue   the predicted observation of dead or infested trees
*@param treeType    specifies dead or infested trees, treeType=0 for infested, =1 for dead
*@param j       the time step
*@return        the percent error difference between the true and predicted observations
*/
public double compareValues(int trueValue, int testValue, int treeType, int j){
    double percentError = 0;
    if((trueValue != -1) && (testValue != -1)){ //data point is available
       double absoluteDifference = Math.abs(trueValue - testValue);
       double squareDifference = absoluteDifference * absoluteDifference;
       _squareDifferences[treeType][j] = squareDifference;
       if(trueValue != 0){ // avoid dividing by zero
          percentError = absoluteDifference / trueValue;
          _percentErrors[treeType][j] = percentError;
            } else if(absoluteDifference == 0){ /*both data points are zero in this case*/
          _percentErrors[treeType][j] = 0d;
       } else{
          _percentErrors[treeType][j] = -1; //cannot calculate a value
       }
    }   else{ //if a data point is unavailable
       _percentErrors[treeType][j] = -1;
    }
    return percentError;
}//end compareValues

/**compares log-transformed values of a true and a predicted observation
*   of dead or infested trees at one time step, using absolute and squared differences
*@param trueValue   the true observation of dead or infested trees
*@param testValue   the predicted observation of dead or infested trees
*@param treeType    specifies dead or infested trees, treeType=0 for infested, =1 for dead
*@param j       the time step
*@return        the absolute difference between the transformed true and predicted observations
*/
public double compareTransformedValues(int trueValue, int testValue, int treeType, int j){
    /*calculate the natural logs of the values passed in */
    double transformedTrueValue = Math.log(trueValue + 1);
    double transformedTestValue = Math.log(testValue + 1);
    double absoluteDifference = 0;
    double squareDifference = 0;
    if((trueValue != -1) && (testValue != -1)){ //data point is available
```

```
        absoluteDifference = Math.abs(transformedTrueValue - transformedTestValue);
        squareDifference = absoluteDifference * absoluteDifference;
        _squareDifferences[treeType][j] = squareDifference;
    }
    return absoluteDifference;
}//end compareValues

/**retrieves the percent error between a true and predicted observation of dead or infested trees
*@param treeType    specifies dead or infested trees, treeType=0 for infested, =1 for dead
*@param j       the time step
*@return        the percent error between the true and predicted observations
*/
public double percentError(int treeType, int j){
    return _percentErrors[treeType][j];
}//end percentError

/**retrieves the mean percent error of the true and predicted
*   observations of dead or infested trees over the simulation duration
*@param treeType            specifies dead or infested trees, treeType=0 for infested, =1 for dead
*@return    the mean percent error between observations of true and predicted dead or infested trees
*/
public double meanPercentError(int treeType){ /*if treeType == 0, then infested,
                                              if treeType==1, then dead*/
    double meanPercentError = 0;
    try{
        double sumPercentErrors = 0d;
        int numberOfPoints = 0;
        for(int j = 0; j < _simulationDuration; j++){
            if(_percentErrors[treeType][j] != -1){ //if data point available
                sumPercentErrors += _percentErrors[treeType][j];
                            numberOfPoints ++;
            }
        }//end for
        if(numberOfPoints > 0){
            meanPercentError = sumPercentErrors / numberOfPoints;
        }   else{
            meanPercentError = -1; //no data points available
        }
    }   catch(ArrayIndexOutOfBoundsException e){
        System.out.println("Error: an attempt was made to access an " +
            "out-of-bounds array field in array _percentErrors.");
    }//end catch
    return meanPercentError;
}//end meanPercentErrorInfested

/**retrieves the sum of squared error of the true and predicted
*   observations of dead or infested trees over the simulation duration
*@param treeType            specifies dead or infested trees, treeType=0 for infested, =1 for dead
*@return    the sum of squared error between observations of true and predicted dead or infested trees
*/
public double sumsOfSquares(int treeType){
    double sumsOfSquares = 0;
    try{
        for(int j = 0; j < _simulationDuration; j++){
            if(_squareDifferences[treeType][j] != -1){
                sumsOfSquares += _squareDifferences[treeType][j];
                }
            }//end for
        } catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Error: an attempt was made to access an " +
                "out-of-bounds array field in array _squareDifferences.");
        }//end catch
        if(sumsOfSquares < 0){
            System.out.println("Error: error sums of squares is negative.");
        }
        return sumsOfSquares;
    }//end sumsOfSquares
}//end Analyzer
```

B.2 Package javaSPB1 - Runs the simulation.

B.2.1 Simulator.java

```
/**Simulator.java simulates the growth of the SPB infestation over the
*specified period. Simulator accepts input data (containing infested stand
*conditions) from the calling function and passes this information on to Spot.
*Simulator also has public functions to make the output data accessible.
*/

package JavaSPB1;

import java.util.*;
import java.io.*;

/**Simulates the growth of the SPB infestation by a fractional development
*scheme.  Calculates populations of SPB in each life stage at each time step,
*and calculates the numbers of dead and infested trees each day.
*@see AttackingAdult
*@see ParentAdult
*@see Egg
*@see Immature
*@see BroodAdult
*@see EmergingAdult
*@see Stage
*@see SPBCohort
*@see Spot
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class Simulator
{
   /**the SPB infestation, holds stand characteristics*/
   protected Spot _spot;

   /**the Julian hour of the start of the simulation*/
   protected int _start;

   /**the Julian hour of the end of the simulation*/
   protected int _end;

   /**the attacking adult life stage of SPB*/
   protected AttackingAdult _attacking;

   /**the parent adult life stage of SPB*/
   protected ParentAdult _parent;

   /**the egg life stage of sPB*/
   protected Egg _egg;

   /**the immature life stage of SPB, including larvae and pupae*/
   protected Immature _larva;

   /**the brood adult life stage of SPB*/
```

```
protected BroodAdult _brood;

/**the emerging adult life stage of SPB*/
protected EmergingAdult _emerging;

/**holds SPB cohorts*/
protected Vector _initialAttackingCohorts = new Vector();

/**holds just the attacking adult SPB cohorts*/
protected Vector _attackingCohorts = new Vector();

/**holds just the parent adult SPB cohorts*/
protected Vector _parentCohorts = new Vector();

/**holds just the egg SPB cohorts*/
protected Vector _eggCohorts = new Vector();

/**holds just the larval/pupal SPB cohorts*/
protected Vector _larvaCohorts = new Vector();

/**holds just the brood SPB cohorts*/
protected Vector _broodCohorts = new Vector();

/**generic cohort for looping through SPB life stages*/
protected SPBCohort _spbCohort;

/**a cohort of emerging SPB adults*/
protected SPBCohort _emergingCohort;

/**the number of attacking adult SPB entering the parent adult life stage*/
protected float _parentInflow;

/**the number of new SPB eggs oviposited*/
protected float _eggInflow;

/**the number of SPB eggs entering the larva/pupa life stage*/
protected float _immatureInflow;

/**the number of SPB larvae/pupae entering the brood adult life stage*/
protected float _broodInflow;

/**the number of SPB parent adults leaving the parent adult life stage*/
protected float _parentOutflow;

/**the number of SPB emerging adults leaving the emerging adult life stage*/
protected float _emergingOutflow;

/**the number of SPB attacking adults leaving the attacking adult life stage*/
protected float _attackingOutflow;

/**the number of brood adult SPB entering the emerging adult life stage*/
protected float _emergingInflow;

/**the number of parent adult SPB entering the attacking adult life stage*/
protected float _attackingInflow;
```

```
/**the object that keeps track of SPB-killed trees*/
protected Tree _dead;

/**the attack pool; important in the attacking routine, determines successful
*or unsuccessful attacks of trees, and subsequent growth or mortality
*of attacking adult SPB
*/
protected AttackPool _attackPool;

/**the number of SPB (attacking adults) in the attack pool*/
protected float _pool;

/**holds the SPB populations of each life stage at each time step,
   [lifeStage][timeStep]*/
protected float[][] _beetles;

/**holds numbers of infested trees at each [day]*/
protected int[] _infestedTrees;

/**holds numbers of dead trees at each [day]*/
protected int[] _deadTrees;

/**holds temperatures at each [timeStep]*/
protected float[] _temperatures;

/**holds population flow into each life stage at each time step,
*[lifeStage][timeStep]
*/
protected float[][] _inflows;

/**holds fractional development rates for each life stage at each time step,
*[lifeStage][timeStep]
*/
protected float[][] _develRates;

/**holds mortality rates for each life stage at each time step,
*[stage][timeStep]
*/
protected float[][] _mortRates;

/**number of simulation iterations where attack pool is too high*/
protected int _problemIterations;

/**constructor
*@param spot    the SPB infestation spot
*@param end     the Julian date (in hours) specified for
*               the simulation to terminate
*/
public Simulator(Spot spot, int end){
   _spot = new Spot();
   _spot = spot;
   //initialize start and end of the simulation
   _start = spot.start();
   _end = (int)(end + 24); //simulate through the final day

   //initialize SPB stages
```

```
_attacking = new AttackingAdult(_spot);
_parent = new ParentAdult(_spot);
_egg = new Egg(_spot);
_larva = new Immature(_spot);
_brood = new BroodAdult(_spot);
_emerging = new EmergingAdult(_spot);
/*divide the total populations of each life stage into cohorts;
   cohorts will develop at slightly different times into the next life
   stage*/

/*set up 5 cohorts for initial attacking adults and for parent adults*/
int totalCohorts = 5;
int cohortNumber;
for (cohortNumber = 0; cohortNumber < totalCohorts; cohortNumber++){
   _spbCohort = new SPBCohort(_attacking, cohortNumber, totalCohorts);
   _initialAttackingCohorts.addElement(_spbCohort);
}
for (cohortNumber = 0; cohortNumber < totalCohorts; cohortNumber++){
   _spbCohort = new SPBCohort(_parent, cohortNumber, totalCohorts);
   _parentCohorts.addElement(_spbCohort);
}
/*set up 20 cohorts for SPB eggs*/
totalCohorts = 20;
for(cohortNumber = 0; cohortNumber < totalCohorts; cohortNumber++){
   _spbCohort = new SPBCohort(_egg, cohortNumber, totalCohorts);
   _eggCohorts.addElement(_spbCohort);
}
/*set up 100 cohorts for SPB larvae and pupae*/
totalCohorts = 100;
for(cohortNumber = 0; cohortNumber < totalCohorts; cohortNumber++){
   _spbCohort = new SPBCohort(_larva, cohortNumber, totalCohorts);
   _larvaCohorts.addElement(_spbCohort);
}
/*set up 20 cohorts for SPB brood adults*/
totalCohorts = 20;
for(cohortNumber = 0; cohortNumber < totalCohorts; cohortNumber++){
   _spbCohort = new SPBCohort(_brood, cohortNumber, totalCohorts);
   _broodCohorts.addElement(_spbCohort);
}

/*there is only one emerging cohort*/
_emergingCohort = new SPBCohort(_emerging, _spot.emerging());

/*initialize dead tree object*/
_dead = new Tree(_spot);

/*initialize attack pool*/
_attackPool = new AttackPool(_spot, _dead);

/*initialize arrays for holding numbers of dead and infested trees;
   there will be (1 + the number of the days in the simulation)
   fields in the array, to account for the initial numbers of dead and
   infested trees; there are 2 extra hours because the simulation
   goes from 2AM to 2AM (the first 2 fields of _beetles[][]
   will be empty*/
int hours = (int)(_end - _start);    //total hours in the simulation
```

```
        _infestedTrees = new int[(int)(hours / 24f) + 1];
        _deadTrees = new int[(int)(hours / 24f) + 1];
        _temperatures = new float[hours];
        _mortRates = new float[7][hours];
        _develRates = new float[7][hours];
        /*set initial inflows to 0*/
        _parentInflow =0;
        _eggInflow = 0;
        _immatureInflow = 0;
        _broodInflow = 0;
        _emergingInflow = 0;
        _emergingOutflow = 0;
        _attackingInflow = 0;
        _parentOutflow = 0;

        /*initialize array for holding the SPB populations*/
        _beetles = new float[7][hours];
        _infestedTrees[0] = _spot.infested();
        _deadTrees[0] = _spot.dead();
        for(int i = 0; i < 1; i++){
            _beetles[0][i] = _spot.attacking();
            _beetles[1][i] = 0;
            _beetles[2][i] = _spot.parent();
            _beetles[3][i] = _spot.egg();
            _beetles[4][i] = _spot.immature();
            _beetles[5][i] = _spot.brood();
            _beetles[6][i] = _spot.emerging();
        }//end for
        _problemIterations = 0;
    }//end constructor

    public Simulator(){
    }

    /**Runs the simulation; develops the SPB individuals from one life stage to
    *another, accounting for mortality and fecundity, and temperature-dependent
    *development rates.  Uses a fractional development scheme.  Determines the
    *populations of SPB of each life stage at each time step, and determines
    *the numbers of dead and infested trees each day of the simulation.
    */
    public void event(){
        int jHour;        //time in Julian hours (hours since 12AM January 1)
        int timeIndex = 1;  //time in hours from the beginning of the simulation
        float temperature;     //the current temperature (calculated each hour)
        int cohortNumber;   //counter for traversing through cohort vectors
        _pool = 0f;     //the attack pool is initially zero
        int startingHour = 1;
        int updateHour = startingHour;

        /*the current day of the simulation (number of days since _start)*/
        int today;
        float mortRate = 0;
        float parentPop = 0;
        float eggProductionRate = 0;

        /*set all densities initially to zero*/
```

```
float eggDensity = _spot.eggDensity();
float eggMortality = _spot.eggInitMort();
float larvaDensity = _spot.immatureDensity();
float larvaMortality = _spot.immatureInitMort();
float broodDensity = _spot.broodDensity();
float broodMortality = _spot.broodInitMort();
int initialAttackingLevel = 0;
int attackingLevel = 0;
int parentLevel = 0;
int eggLevel = 0;
int larvaLevel = 0;
int broodLevel= 0;
int emergingLevel = 0;
_spot.setTemperatures(_start);

/*begin the simulation loop*/
for(jHour = _start + startingHour; jHour < _end; jHour ++){
   //timeIndex starts at 1, because the simulation starts at 2AM
   timeIndex = (int)(jHour - _start);
   today = (int)((timeIndex + 1) / 24f);

   /*update the set of temperatures for each new day*/
   if((timeIndex + 1) % 24 == updateHour){
      _spot.setTemperatures(jHour);
   }

   /*retrieve the temperature for this hour*/
   temperature = _spot.getTemperature(jHour);
   _temperatures[timeIndex] = temperature;

   /*loop backwards through life stages, to avoid aging cohorts twice,
      as would happen when looping forwards if the cohort develops
      to the following stage*/

   /*develop the emerging adult cohort*/
   _emergingCohort.developEmerging(jHour, temperature);
   /*if emerging adults have developed, add a new attacking adult cohort*/
   _emergingOutflow = (float)(_emergingCohort.pop() * _emergingCohort.develRate());

   /*develop the initial attacking cohorts*/
   cohortNumber = 0;
   mortRate = 0;
   while(cohortNumber < _initialAttackingCohorts.size()){
      _spbCohort = (SPBCohort)_initialAttackingCohorts.elementAt(cohortNumber);
      _spbCohort.develop(jHour, temperature);

      if(_spbCohort.totalDevelopment() >= 1){ //development complete
         _spbCohort.takeMortality();
         /*this old attacking adult cohort will become parents*/
         _parentInflow += _spbCohort.pop();
         /*remove the old attacking adult cohort*/
         _initialAttackingCohorts.removeElementAt(cohortNumber);
      }  else{   //else continue through the cohort vector
         _initialAttackingCohorts.setElementAt(_spbCohort, cohortNumber);
         cohortNumber ++;
      }//end else
```

```
}//end while

_attackingInflow = (float)(_emergingOutflow * (1f - _emergingCohort.mortRate()));

_attackingOutflow = 0;
 _pool = 0;
/*develop the attacking adult cohorts*/
cohortNumber = 0;
mortRate = 0;
while(cohortNumber < _attackingCohorts.size()){
   _spbCohort =
       (SPBCohort)_attackingCohorts.elementAt(cohortNumber);
   _spbCohort.develop(jHour, temperature);
   if(_spbCohort.totalDevelopment() >= 1){ //development complete
       _spbCohort.takeMortality();
      /*add the attacking adult population to the attack pool*/
       _pool += _spbCohort.pop();

       /*remove the attacking adult cohort*/
       _attackingCohorts.removeElementAt(cohortNumber);
   }   else{   //else continue through the cohort vector
       _attackingCohorts.setElementAt(_spbCohort, cohortNumber);
       cohortNumber ++;
   }//end else
}//end while

_attackingOutflow = _attackPool.colonizeTrees(jHour, _pool);
if(_attackingOutflow < 0){
   _attackingOutflow = 0;
   _problemIterations ++;
}

/*this attack pool will become a new parent
   cohort, along with population from the developed old
   attacking adult cohorts*/

_parentInflow += _attackingOutflow;

/*develop the parent adult cohorts*/
cohortNumber = 0;
_eggInflow = 0;
mortRate = 0;
parentPop = 0;
eggProductionRate = 0;
_parentOutflow = 0;
while(cohortNumber < _parentCohorts.size()){
   _spbCohort = (SPBCohort)_parentCohorts.elementAt(cohortNumber);
   eggProductionRate = _spbCohort.eggProductionRate();
   _spbCohort.develop(jHour, temperature);
   /*allow for fecundity; new egg cohorts are laid*/
   parentPop += _spbCohort.pop();
   if(_spbCohort.totalDevelopment() >= 1){ //development complete
       _spbCohort.takeMortality();
       /*this parent cohort will develop into emerging adults*/
       _parentOutflow += _spbCohort.pop();
       /*remove the old parent cohort*/
```

```
            _parentCohorts.removeElementAt(cohortNumber);
        }   else{   //else continue through the cohort vector
            _parentCohorts.setElementAt(_spbCohort, cohortNumber);
            cohortNumber ++;
        }//end else
    }//end while
    if(timeIndex - 1 >= 0){
        _eggInflow = _beetles[2][timeIndex - 1] * eggProductionRate;
    } else {
        _eggInflow = 0;
    }

    /*develop the egg cohorts*/
    cohortNumber = 0;
    _immatureInflow = 0;
    mortRate = 0;
    while(cohortNumber < _eggCohorts.size()){
        _spbCohort = (SPBCohort)_eggCohorts.elementAt(cohortNumber);
        _spbCohort.develop(jHour, temperature);
        if(_spbCohort.totalDevelopment() >= 1){ //development complete
            _spbCohort.takeMortality();
            /*this egg cohort will develop into a new larva cohort*/
            _immatureInflow += _spbCohort.pop();
            /*remove the old egg cohort*/
            _eggCohorts.removeElementAt(cohortNumber);
        }   else{   //else continue through the cohort vector
            _eggCohorts.setElementAt(_spbCohort, cohortNumber);
            cohortNumber ++;
        }//end else
    }//end while

    /*develop the larva/pupa cohorts*/
    cohortNumber = 0;
    _broodInflow = 0;
    mortRate = 0;
    while(cohortNumber < _larvaCohorts.size()){
        _spbCohort = (SPBCohort)_larvaCohorts.elementAt(cohortNumber);
        _spbCohort.develop(jHour, temperature);
        if(_spbCohort.totalDevelopment() >= 1){ //development complete
            _spbCohort.takeMortality();
            _broodInflow += _spbCohort.pop();
            /*remove the old larva cohort*/
            _larvaCohorts.removeElementAt(cohortNumber);
        }   else{   //else continue through the cohort vector
            _larvaCohorts.setElementAt(_spbCohort, cohortNumber);
            cohortNumber ++;
        }//end else
    }//end while

    /*develop the brood adult cohorts*/
    cohortNumber=0;
    mortRate = 0;
    while(cohortNumber < _broodCohorts.size()){
        _spbCohort = (SPBCohort)_broodCohorts.elementAt(cohortNumber);
        _spbCohort.develop(jHour, temperature);
        if(_spbCohort.totalDevelopment() >= 1){ //development complete
```

```
          _spbCohort.takeMortality();
          /*these brood adults develop into emerging adults*/
          _emergingInflow += _spbCohort.pop();
          /*remove the old brood adult cohort*/
          _broodCohorts.removeElementAt(cohortNumber);
       }   else{   //else continue through the cohort vector
          _broodCohorts.setElementAt(_spbCohort, cohortNumber);
          cohortNumber ++;
       }//end else
    }//end while
    addItem(_attackingInflow, _attackingCohorts, _attacking);
    addItem(_parentInflow, _parentCohorts, _parent);
    addItem(_eggInflow, _eggCohorts, _egg);
    addItem(_immatureInflow, _larvaCohorts, _larva);
    addItem(_broodInflow, _broodCohorts, _brood);
    _emergingInflow += _parentOutflow;
    if(_emergingInflow < 0.01){
       _emergingInflow = 0;
    }
    _emergingCohort.setPop((float)(_emergingCohort.pop() +
       _emergingInflow - _emergingOutflow));

    /*to obtain the current population of intialAttackingAdults, sum the
       number of SPB in each initialAttackingAdult cohort
       (store the population in _beetles[][] during each time step)*/
    for(cohortNumber = 0; cohortNumber < _initialAttackingCohorts.size(); cohortNumber ++){

       _spbCohort = (SPBCohort)_initialAttackingCohorts.elementAt(cohortNumber);
       _beetles[0][timeIndex] += _spbCohort.pop();
    }//end for

    /*to obtain the current population of attackingAdults, sum the
       number of SPB in each attackingAdult cohort
       (store the population in _beetles[][] during each time step)*/
    for(cohortNumber = 0; cohortNumber < _attackingCohorts.size(); cohortNumber ++){

       _spbCohort = (SPBCohort)_attackingCohorts.elementAt(cohortNumber);
       _beetles[1][timeIndex] += _spbCohort.pop();
    }//end for

    /*to obtain the current population of attackingAdults, sum the
       number of SPB in each attackingAdult cohort
       (store the population in _beetles[][] during each time step)*/
    for(cohortNumber = 0; cohortNumber < _parentCohorts.size(); cohortNumber ++){
       _spbCohort = (SPBCohort)_parentCohorts.elementAt(cohortNumber);
       _beetles[2][timeIndex] += _spbCohort.pop();
    }//end for

    /*to obtain the current population of eggs, sum the number
       of SPB in each egg cohort
       (store the population in _beetles[][] during each time step)*/
    for(cohortNumber = 0; cohortNumber < _eggCohorts.size(); cohortNumber ++){

       _spbCohort = (SPBCohort)_eggCohorts.elementAt(cohortNumber);
       _beetles[3][timeIndex] += _spbCohort.pop();
    }//end for
```

```
/*to obtain the current population of immatures, sum the number
   of SPB in each immature cohort
   (store the population in _beetles[][] during each time step)*/
for(cohortNumber = 0; cohortNumber < _larvaCohorts.size(); cohortNumber ++){
   _spbCohort = (SPBCohort)_larvaCohorts.elementAt(cohortNumber);
   _beetles[4][timeIndex] += _spbCohort.pop();
}//end for

/*to obtain the current population of broodAdults, sum the number
   of SPB in each broodAdult cohort
   (store the population in _beetles[][] during each time step)*/
for(cohortNumber = 0; cohortNumber < _broodCohorts.size();
   cohortNumber ++){
   _spbCohort = (SPBCohort)_broodCohorts.elementAt(cohortNumber);
   _beetles[5][timeIndex] += _spbCohort.pop();
}//end for

/*store the population of the current emerging cohort*/
_beetles[6][timeIndex] = _emergingCohort.pop();

if(((timeIndex + 1) % 24 == updateHour) && (timeIndex != startingHour)){

   /*if there is at least one parentCohort, and if a day has
   passed, update the number of infested trees that is occupied by
   these cohorts*/
   if(_initialAttackingCohorts.size() >= 1 ||
      _attackingCohorts.size() >= 1){
      if(_initialAttackingCohorts.size() >= 1){
         _spbCohort = (SPBCohort)_initialAttackingCohorts.elementAt(0);
      } else {
         _spbCohort = (SPBCohort)_attackingCohorts.elementAt(0);
      }
      attackingLevel = (int)(((_beetles[0][timeIndex] +
         _beetles[1][timeIndex]) * (1f - _spbCohort.mortRate())
         + _attackPool.attackPool()) / (_spbCohort.density() * _spot.avgInfBarkArea()));
   }
   if(_parentCohorts.size() >= 1){
      _spbCohort = (SPBCohort)_parentCohorts.elementAt(0);
      parentLevel = (int)(_beetles[2][timeIndex] / (_spbCohort.density() * _spot.avgInfBarkArea()));
   }//end if
   if(_eggCohorts.size() >= 1){
      _spbCohort = (SPBCohort)_eggCohorts.elementAt(0);
      eggMortality = _spbCohort.mortRate();
      eggDensity = _spbCohort.density();
      eggLevel = (int)(_beetles[3][timeIndex] / (eggDensity * _spot.avgInfBarkArea()));
   }
   if(_larvaCohorts.size() >= 1){
      _spbCohort = (SPBCohort)_larvaCohorts.elementAt(0);
      larvaDensity = ((1f - eggMortality) * eggDensity);
      larvaMortality = _spbCohort.mortRate();
      larvaLevel = (int)(_beetles[4][timeIndex] / (larvaDensity * _spot.avgInfBarkArea()));
   }
   if(_broodCohorts.size() >= 1){
      _spbCohort = (SPBCohort)_broodCohorts.elementAt(0);
      broodDensity = (float)((1f - larvaMortality) * larvaDensity);
```

```
            broodMortality = _spbCohort.mortRate();
            broodLevel = (int)(_beetles[5][timeIndex] / (broodDensity * _spot.avgInfBarkArea()));
          }
          _emergingCohort.setDensity((1f - broodMortality) * broodDensity);
          emergingLevel = (int)(_beetles[6][timeIndex] /
            (_emergingCohort.density() * _spot.avgInfBarkArea()));
          _infestedTrees[today] = attackingLevel + parentLevel +
            eggLevel + larvaLevel + broodLevel + emergingLevel;
          _deadTrees[today] = _dead.num();
        }//end if

        _emergingInflow = 0; //reset _emergingInflow
        _attackingInflow = 0;
        _parentInflow = 0;
    }//end for(jHour)
  }//end event

  /**adds a new cohort item to a cohort Vector
  *@param population  the initial population of the cohort
  *@param spbCohorts  the cohort Vector, holds cohorts of a certain stage
  *@param stage   the life stage of the SPB cohorts in the Vector
  */
  public void addItem(float population, Vector spbCohorts, Stage stage){
    if(population > 0.01){
        SPBCohort spbCohort = new SPBCohort(stage, population);
        spbCohorts.addElement(spbCohort);
    }//end if
  }//end addItem

  /**returns the number of SPB-infested trees on a certain day
  *@param day     the day for which the number of infested trees is sought
  *              the number of days since the beginning of the simulation
  *@return    the number of infested trees on a certain day
  */
  public int infestedElement(int day){
    int infestedTrees = 0;
    try{
        infestedTrees = (int)_infestedTrees[day];
    }   catch(ArrayIndexOutOfBoundsException e){
        System.out.println("Error: the number of infested trees was" +
          " requested for a day outside of the simulation");
    }//end catch
    return infestedTrees;
  }//end infestedElement

  public int[] infestedArray(){
    return _infestedTrees;
  }

  /**returns the number of SPB-killed trees on a certain day of simulation
  *@param day     the day for which the number of dead trees is sought;
  *              the number of days since the beginning of the simulation
  *@return    the number of dead trees on a certain day
  */
  public int deadElement(int day){
    int deadTrees = 0;
```

```
        try{
            deadTrees = (int)_deadTrees[day];
        }   catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Error: the number of dead trees was requested" +
                " for a day outside of the simulation.");
        }//end catch
        return deadTrees;
    }//end deadElement

    public int[] deadArray(){
        return _deadTrees;
    }

    public int[][] infestedAndDead(){
        int hours = _end - _start;
        int days = (int)(hours / 24f);
        int[][] infestedAndDead = new int[2][days];
        for(int i = 0; i < days; i ++){
            infestedAndDead[0][i] = _infestedTrees[i];
            infestedAndDead[1][i] = _deadTrees[i];
        }//end for
        return infestedAndDead;
    }
    /**returns the number of SPB individuals in a life stage during a certain
    *hour of simulation
    *@param lifeStage   the life stage of SPB for which the population is
    *                   sought; can be in range 0-6, where 0 is old attacking
    *                   adults and 6 is emerging adults
    *@param hour    the hour for which the number of SPB individuals is sought;
    *               the number of hours since the beginning of the simulation
    *@return    the number of SPB individuals in a life stage in a certain hour
    */
    public int beetleElement(int lifeStage, int hour){
        int beetles = 0;
        try{
            beetles = (int)_beetles[lifeStage][hour];
        }   catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Error: the SPB population was requested for" +
                " an incorrect life stage or for an hour outside of the" +
                " simulation");
        }//end catch
        return beetles;
    }//end beetleElement

    /**returns the number of simulation iterations where
    *   the attack pool became much higher than reasonable
    *@return    the number of iterations where the attack pool
    *           became too high
    */
    public int problemIterations(){
        return _problemIterations;
    }
}//end Simulator
```

B.2.2 GASimulator.java

```
/**GASimulator.java provides for overriding development and mortality
*   rates for the SPB life stages, as is desired when running the
*   genetic algorithm (GA)
*/

package JavaSPB1;
import java.util.*;
import java.io.*;

/**provides for overriding development and mortality rates for the
*   SPB life stages, as is desired when running the genetic algorithm (GA)
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  8 July 2002
*/
public class GASimulator extends Simulator{

    Development _attackingDevelopment;
    Development _parentDevelopment;
    Development _eggDevelopment;
    Development _larvaDevelopment;
    Development _broodDevelopment;
    Development _emergingDevelopment;
    Mortality _attackingMortality;
    Mortality _emergingMortality;

    /**default constructor*/
    public GASimulator(){
        _attackingDevelopment = new Development();
        _parentDevelopment = new Development();
        _eggDevelopment = new Development();
        _larvaDevelopment = new Development();
        _broodDevelopment = new Development();
        _emergingDevelopment = new Development();
        _attackingMortality = new Mortality();
        _emergingMortality = new Mortality();
    }

    /**constructor when all development rates and two mortality rates
    *   are to be overridden
    *@param spot       the infestation spot
    *@param end        the Julian date, in hours, of the simulation termination
    *@param develParameters     the parameters for the development rate
    *                     functional forms
    *@param develTypes  the types of functional forms for the development rates
    *@param attackingMortality  the object for determining the mortality rate
    *                     for attacking adult SPB
    *@param emergingMortality   the object for determining the mortality rate
    *                     for emerging adult SPB
    */
    public GASimulator(Spot spot, int end, double[][] develParameters,
        int[] develTypes, Mortality attackingMortality, Mortality
        emergingMortality){
```

```
_attackingMortality = attackingMortality;
_emergingMortality = emergingMortality;
int numberOfParams = develParameters[0].length;
double[] develParameters_1 = new double[numberOfParams];
double[] develParameters_2 = new double[numberOfParams];
double[] develParameters_3 = new double[numberOfParams];
double[] develParameters_4 = new double[numberOfParams];
double[] develParameters_5 = new double[numberOfParams];
double[] develParameters_6 = new double[numberOfParams];
int stage = 0;
for(int params = 0; params < numberOfParams; params ++){
    develParameters_1[params] = develParameters[0][params];
    develParameters_2[params] = develParameters[1][params];
    develParameters_3[params] = develParameters[2][params];
    develParameters_4[params] = develParameters[3][params];
    develParameters_5[params] = develParameters[4][params];
    develParameters_6[params] = develParameters[5][params];
}
_attackingDevelopment = new Development(develTypes[0], develParameters_1);
_parentDevelopment = new Development(develTypes[1], develParameters_2);
_eggDevelopment = new Development( develTypes[2], develParameters_3);
_larvaDevelopment = new Development(develTypes[3], develParameters_4);
_broodDevelopment = new Development(develTypes[4], develParameters_5);
_emergingDevelopment = new Development(develTypes[5], develParameters_6);

_spot = new Spot();
_spot = spot;
//initialize start and end of the simulation
_start = spot.start();
_end = (int)(end + 24); //simulate through the final day

//initialize SPB stages
_attacking = new AttackingAdult(_spot);
_parent = new ParentAdult(_spot);
_egg = new Egg(_spot);
_larva = new Immature(_spot);
_brood = new BroodAdult(_spot);
_emerging = new EmergingAdult(_spot);

/*divide the total populations of each life stage into cohorts;
   cohorts will develop at slightly different times into the next life stage*/

/*set up 5 cohorts for initial attacking adults and for parent adults*/
int totalCohorts = 5;
int cohortNumber;
for (cohortNumber = 0; cohortNumber < totalCohorts; cohortNumber++){
   _spbCohort = new SPBCohort(_attacking, cohortNumber, totalCohorts);
   _spbCohort.setMortality(_attackingMortality);
   _spbCohort.setDevelopment(_attackingDevelopment);
   _initialAttackingCohorts.addElement(_spbCohort);
}
for (cohortNumber = 0; cohortNumber < totalCohorts; cohortNumber++){
   _spbCohort = new SPBCohort(_parent, cohortNumber, totalCohorts);
   _spbCohort.setDevelopment(_parentDevelopment);
   _parentCohorts.addElement(_spbCohort);
}
```

```
/*set up 20 cohorts for SPB eggs and for brood adults*/
totalCohorts = 20;
for(cohortNumber = 0; cohortNumber < totalCohorts; cohortNumber++){
   _spbCohort = new SPBCohort(_egg, cohortNumber, totalCohorts);
   _spbCohort.setDevelopment(_eggDevelopment);
   _eggCohorts.addElement(_spbCohort);
   _spbCohort = new SPBCohort(_brood, cohortNumber, totalCohorts);
   _spbCohort.setDevelopment(_broodDevelopment);
   _broodCohorts.addElement(_spbCohort);
}

/*set up 100 cohorts for SPB larvae and pupae*/
totalCohorts = 100;
for(cohortNumber = 0; cohortNumber < totalCohorts; cohortNumber++){
   _spbCohort = new SPBCohort(_larva, cohortNumber, totalCohorts);
   _spbCohort.setDevelopment(_larvaDevelopment);
   _larvaCohorts.addElement(_spbCohort);
}

/*there is only one emerging cohort*/
_emergingCohort = new SPBCohort(_emerging, _spot.emerging());
_emergingCohort.setDevelopment(_emergingDevelopment);
_emergingCohort.setMortality(_emergingMortality);

/*initialize dead tree object*/
_dead = new Tree(_spot);

/*initialize attack pool*/
_attackPool = new AttackPool(_spot, _dead);

/*initialize arrays for holding numbers of dead and infested trees;
there will be (1 + the number of the days in the simulation)
fields in the array, to account for the initial numbers of dead and
infested trees; there are 2 extra hours because the simulation
goes from 2AM to 2AM (the first 2 fields of _beetles[][] will be empty*/
int hours = (int)(_end - _start);    //total hours in the simulation
_infestedTrees = new int[(int)(hours / 24f) + 1];
_deadTrees = new int[(int)(hours / 24f) + 1];
_temperatures = new float[hours];
_mortRates = new float[7][hours];
_develRates = new float[7][hours];
/*set initial inflows to 0*/
_parentInflow =0;
_eggInflow = 0;
_immatureInflow = 0;
_broodInflow = 0;
_emergingInflow = 0;
_emergingOutflow = 0;
_attackingInflow = 0;
_parentOutflow = 0;

/*initialize array for holding the SPB populations*/
_beetles = new float[7][hours];
_infestedTrees[0] = _spot.infested();
_deadTrees[0] = _spot.dead();
for(int i = 0; i < 1; i++){
```

```
            _beetles[0][i] = _spot.attacking();
            _beetles[1][i] = 0;
            _beetles[2][i] = _spot.parent();
            _beetles[3][i] = _spot.egg();
            _beetles[4][i] = _spot.immature();
            _beetles[5][i] = _spot.brood();
            _beetles[6][i] = _spot.emerging();
        }//end for
    }//end constructor

    /**adds a new cohort item to a cohort Vector
    *@param population  the initial population of the cohort
    *@param spbCohorts  the cohort Vector, holds cohorts of a certain stage
    *@param stage   the life stage of the SPB cohorts in the Vector
    */
    public void addItem(float population, Vector spbCohorts, Stage stage){
        if(population > 0.01){
            SPBCohort spbCohort = new SPBCohort(stage, population);
            spbCohort.setDevelopment(getDevelopment(stage));
            if(stage.name().equals("attacking")){
                spbCohort.setMortality(_attackingMortality);
            } //emerging cohorts never add items, so don't set emerging mortality here
            spbCohorts.addElement(spbCohort);
        }//end if
    }//end addItem

    /**finds the Development object for an SPB life stage
    *@param stage   the SPB life stage
    *@return        the Development object
    */
    public Development getDevelopment(Stage stage){
        Development devel = new Development();
        if(stage.name().equals("attacking")){
            devel = _attackingDevelopment;
        }else if(stage.name().equals("parent")){
            devel = _parentDevelopment;
        }else if(stage.name().equals("egg")){
            devel = _eggDevelopment;
        }else if(stage.name().equals("immature")){
            devel = _larvaDevelopment;
        }else if(stage.name().equals("brood")){
            devel = _broodDevelopment;
        } else if(stage.name().equals("emerging")){
            devel =  _emergingDevelopment;
        } else{
            System.out.println("Error: development stage specified incorrectly.");
        }
        return devel;
    }//end getDevelopment
}//end GASimulator
```

B.2.3 Development.java

```
/**Development.java holds parameters and functions to calculate
*   the development rates for the SPB life stages when it is
*   desired to override the default development rates.
*/

package JavaSPB1;

import java.io.*;

/**Allows for calculation of development rates, when it is desired
*   to override the default development rates for the SPB life stages
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  8 July 2002
*/
public class Development{

   /**the type of functional form used to calculate development rate*/
   protected int _develType;

   /**the development rate*/
   protected float _develRate;

   /**the current temperature*/
   protected float _temperature;

   /**the minimum temperature at which development can occur*/
   protected double _minTemperature;

   /**the maximum temperature at which development can occur*/
   protected double _maxTemperature;

   /**the optimal temperature for development to occur*/
   protected double _optimalTemperature;

   /**the maximum rate at which development can occur (occurs at
       the optimal temperature for development)*/
   protected double _maxRate;

   /**a small positive number, approx range 0-1*/
   protected double _rho;

   /**a small positive number, approx range 0-1*/
   protected double _psi;

   /**the spread of the Gaussian curve, approx range 30-90*/
   protected double _sigma;

   /**small positive number, approx range 0-1*/
   protected double _k1;

   /**positive number, approx range 0-30*/
   protected double _k2;
```

```
/**positive number, approx range 50-100*/
protected double _d;

/**default constructor*/
public Development(){
}

/**constructor
*@param develType      the type of functional form used to
*                      calculate development rate
*@param develParameters the parameters for the functional form
*/
public Development(int develType, double[] develParameters){
   _develType = develType;

   try{
      _rho = develParameters[0];
      _psi = develParameters[1];
      _sigma = develParameters[2];
      _k1 = develParameters[3];
      _k2 = develParameters[4];
      _d = develParameters[5];
      _minTemperature = develParameters[6];
      _maxTemperature = develParameters[7];
      _optimalTemperature = develParameters[8];
      _maxRate = develParameters[9];
      } catch(ArrayIndexOutOfBoundsException e){
         System.out.println("Error accessing rate parameter array");
      }
}//end constructor

/**returns the development rate for a certain temperature,
*   according to the functional form of the development rate
*   (in other words, the function relating the development rate
*   to temperature)
*@param temperature     the current temperature
*@return                the development rate at a given temperature
*/
public float develRate(float temperature){
   temperature = (float)Math.max(temperature, _minTemperature);
   temperature = (float)Math.min(temperature, _maxTemperature);
   float develRate = 0;
   switch(_develType){
      case 0:   develRate = linear(temperature);              break;
      case 1:   develRate = exponential(temperature);         break;
      case 2:   develRate = exponentialTb(temperature);       break;
      case 3:   develRate = gaussian(temperature);            break;
      case 4:   develRate = stinner(temp erature);            break;
      case 5:   develRate = logan1(temperature);              break;
      case 6:   develRate = type3(temperature);               break;
      default:  develRate = 0;
   }//end switch
   develRate = (float)(Math.min(1.0, Math.max(0.0, develRate)));
   return develRate;
}
```

```
/**obtains the development rate as a linear function of temperature
*@param temperature     the current temperature
*@return               the development rate at the given temperature
*/
private float linear(float temperature){
    float develRate = 0;
    develRate = (float)((1f / 24f) * _rho * (temperature - _minTemperature));
    return develRate;
}

/**obtains the development rate as an exponential function of temperature
*@param temperature     the current temperature
*@return               the development rate at the given temperature
*/
private float exponential(float temperature){
    float develRate = 0;
    develRate = (float)((1f / 24f) * _psi * Math.exp(_rho * temperature));
    return develRate;
}

/**obtains the development rate as an exponential function of temperature,
*   using a base temperature
*@param temperature     the current temperature
*@return               the development rate at the given temperature
*/
private float exponentialTb(float temperature){
    float develRate = 0;
    develRate = (float)((1f / 24f) * Math.exp(_rho * (temperature - _minTemperature) - 1.0f));
    return develRate;
}

/**obtains the development rate as a Stinner function of temperature
*@param temperature     the current temperature
*@return               the development rate at the given temperature
*/
private float stinner(float temperature){
    float develRate = 0;
        if(temperature <= _optimalTemperature){
            develRate = (float)((1f / 24f) * _optimalTemperature / (1.0 +
                Math.exp(_k1 + _k2 * temperature)));
        } else{
            develRate = (float)((1f / 24f) * _optimalTemperature / (1.0 +
                Math.exp(_k1 + _k2 * (2 * _optimalTemperature - temperature))));
        }
    return develRate;
}

/**obtains the development rate as a Logan 1 function of temperature
*@param temperature     the current temperature
*@return               the development rate at the given temperature
*/
private float logan1(float temperature){
    float develRate = 0;
    double tau = (_optimalTemperature + _minTemperature - temperature) /
        (_maxTemperature - _optimalTemperature);
        develRate = (float)((1f / 24f) * _psi * (Math.exp(_rho *
              (temperature - _minTemperature)) - Math.exp (_rho * tau)));
        return develRate;
    }

    /**obtains the development rate as a Type III function of temperature
    *@param temperature     the current temperature
    *@return                the development rate at the given temperature
    */
    private float type3(float temperature){
        float develRate = 0;
        double tau = (_optimalTemperature + _minTemperature - temperature) /
              (_maxTemperature - _optimalTemperature);
        develRate = (float)(_psi * (temperature * temperature /
             (temperature * temperature + _d * _d)) - Math.exp(-tau));
        return develRate;
    }

    /**obtains the development rate as a gaussian function of temperature
    *@param temperature     the current temperature
    *@return                the development rate at the given temperature
    */
    private float gaussian(float temperature){
        float develRate = 0;
        develRate = (float)((1f/24f) * _maxRate * Math.exp(-(_maxTemperature - temperature) *
            (_maxTemperature - temperature) / 2f * _sigma * _sigma));
        return develRate;
    }

    /**returns the type of functional form used in calculating the
    *   development rate according to temperature
    *@return        the functional form type, currently in range 0-6
    */
    public int develType(){
        return _develType;
    }
}//end Development
```

B.2.4 Mortality.java

```
/**Mortality.java holds parameters and functions to calculate
*   the mortality rates for the SPB life stages when it is
*   desired to override the default mortality rates.
*/

package JavaSPB1;

/**Allows for calculation of mortality rates, when it is desired
*   to override the default mortality rates for the SPB life stages
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  8 July 2002
*/
public class Mortality{

   /**the slope of the linear equation for calculating mortality*/
   protected double _mortSlope;

   /**the intercept of the linear equation for calculating mortality*/
   protected double _mortIntercept;

   /**the SPB infestation spot*/
   protected Spot _spot;

   /**the SPB life stage*/
   protected Stage _stage;

   /**default constructor*/
   public Mortality(){
   }

   /**constructor
   *@param mortSlope       the slope of the linear function of
   *                       mortality in terms of the pine basal area
   *@param mortIntercept   the intercept of the linear function of
   *                       mortality in terms of the pine basal area
   */
   public Mortality(double mortSlope, double mortIntercept){
      _stage = new Stage();
      _spot = new Spot();
      _mortSlope = mortSlope;
      _mortIntercept = mortIntercept;
   }

   /**calculates the mortality rate for an SPB life stage
   *@param stage   the SPB life stage
   *@return        the mortality rate for an SPB life stage
   */
   public float mortRate(Stage stage){
      float mortality = 0f;
      _stage = stage;
      _spot = stage.spot();
      if (_stage.name().equals("attacking")){
            mortality = attackingMort();
        } else if(_stage.name().equals("emerging")){
            mortality = emergingMort();
        } else{
            System.out.println("Error: mortality specified for an incorrect stage.");
        }
        return mortality;
    }

    /**calculates the mortality for attacking adult SPB as a linear function of
    *   the pine basal area
    *@return    the attacking adult mortality rate
    */
    public float attackingMort(){
        float attackingMort;
        attackingMort = (float)(-_mortSlope * _spot.pineBA() + _mortIntercept);
        attackingMort = (float)Math.max(0.001, Math.min(0.999, attackingMort));
        return attackingMort;
    }//end attackingMort()


    /**calculates the mortality for emerging adult SPB as a linear function of
    *   the pine basal area
    *@return    the emerging adult mortality rate
    */
    public float emergingMort(){
        float emergingMortality;
        emergingMortality = (float)(-_mortSlope * _spot.pineBA() + _mortIntercept);

        //check hardwood BA conditions
        if(_spot.hardwoodBA() > 20){
            emergingMortality = (float)(emergingMortality * 0.9f + 0.1f);
        }
        emergingMortality = (float)Math.max(0.001, Math.min(0.999, emergingMortality));
        return emergingMortality;
    }//end emergingMort
}//end Mortality
```

B.2.5 Spot.java

```
/**Spot.java holds all of the initial conditions of the SPB infestation spot--
*location, temperature, initial dead trees, etc.  Spot also contains the current
*day and temperature.
*/

package JavaSPB1;
import java.io.*;
import java.util.*;

/**contains all initial conditions of the SPB infestation, as well as the
*current day, hour, and temperature; performs some preliminary calculations
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class Spot{
   /**the object to calculate the set of temperatures for the day*/
   protected TemperatureCalculator _tempCalc;

   /**start date, in Julian hours*/
   protected int _start;

   /**end date, in Julian hours*/
   protected int _end;

   /**number of initially dead trees*/
   protected int _dead;

   /**number of initially infested trees*/
   protected int _infested;

   private int _initialUnderAttack;
   private int _initialWithEggs;
   private int _initialWithImm;
   private int _initialWithBrood;
   /**the number of trees initially under SPB attack*/
   protected int _treesUnderAttack;

   /**the number of trees initially containing SPB parent adults*/
   protected int _treesWithParents;

   /**the number of trees initially containing SPB eggs*/
   protected int _treesWithEggs;

   /**the number of trees initially containing larvae and pupae (immatures)*/
   protected int _treesWithImm;

   /**the number of trees initially containing brood adults */
   protected int _treesWithBrood;

   /**the decimal proportion of loblolly in the infested stand*/
   protected float _loblollyRatio;
```

```
/**the pine basal area, in square meters per hectare*/
protected float _pineBA;

/**the hardwood basal area, in square meters per hectare*/
protected float _hardwoodBA;

/**mean diameter at breast height for the stand, in centimeters*/
protected float _standAvgDBH;

/*_latitude and _longitude values may be exact or estimates*/
/**latitude of the infestation location, in degrees*/
protected float _latitude;
/**longitude of the infestation location, in degrees*/
protected float _longitude;

protected float _initialAttackPool;
protected float _attractionThreshold;
protected float _boleCapacity;
protected int _failedHours;
protected boolean _enoughToAttack;
protected boolean _enoughToKill;

/**constructor that sets the values of the stand conditions, as from a user
*input file
*@param start   the Julian date when the spot was first surveyed
*@param end     the Julian date specified for the simulation to terminate
*@param dead    the number of trees initially killed by SPB
*@param infested    the number of trees initially infested with SPB
*@param develDistributionKnown  whether the values for the development
*                stages of SPB were entered by the user; if not, Spot
*                will calculate the distribution from the number of
*                infested trees; if so, Spot must calculate a value for
*                the number of trees infested with SPB parents
*@param underAttack the number of trees initially under SPB attack
*@param withEggs    the number of trees initially infested with SPB eggs
*@param withImm the number of trees initially infested with SPB larvae/pupae
*@param withBrood   the number of trees initially infested with SPB brood
*                adults
*@param loblollyRatio   the decimal fraction of loblolly trees in the SPB-
*                   infested stand
*@param pineBA      the pine basal area (square meters/hectare) of the stand
*@param hardwoodBA  the hardwood basal area (sq/m/hectare) of the stand
*@param standAvgDBH the average diameter at breast height of the trees in
*                the infested stand
*@param longitude   the longitude (often approximate) of the infested stand
*@param latitude    the latitude (often approximate) of the infested stand
*/
public Spot(int start, int end, int dead, int infested,
   boolean develDistributionKnown, int underAttack, int withEggs,
   int withImm, int withBrood, float loblollyRatio, float pineBA,
   float hardwoodBA, float standAvgDBH, float longitude, float latitude){

   /*set global variable values to those values passed in*/
   _start = start;
   _end = end;
   _infested = infested;
```

```
    /*all initially infested trees are assumed to die, so they are
        included in the model's initially dead trees*/
    _dead = infested + dead;

    _initialUnderAttack = underAttack;
    _initialWithEggs = withEggs;
    _initialWithImm = withImm;
    _initialWithBrood = withBrood;

    /*if the SPB development stage distribution is known, find a value for
        initial trees with parents*/
    if(develDistributionKnown){
        _treesUnderAttack = treesUnderAttack(underAttack);
        _treesWithEggs = treesWithEggs(withEggs);
        _treesWithParents = treesWithParents(underAttack, withEggs);
        _treesWithImm = withImm;
        _treesWithBrood = withBrood;
    }   else{ /*distribute the infested trees among the development stages*/
        distributeInfested(infested);
    }

    _loblollyRatio = loblollyRatio;
    _pineBA = pineBA;
    _hardwoodBA = hardwoodBA;
    _standAvgDBH = standAvgDBH;
    _longitude = longitude;
    _latitude = latitude;

    _tempCalc = new TemperatureCalculator();
    //initialize the temperature calculator with the location
    _tempCalc.setLatAndLong(_longitude, _latitude);
    setTemperatures(_start);
    initializeAttackPool();
}//end constructor

/**default constructor*/
public Spot(){
}

/**calls the temperature calculation, and sets up an array in a
*TemperatureCalculator of all the temperatures for the current day
*@param jHour    the current Julian hour (hours since 12AM January 1)
*/
public void setTemperatures(int jHour){
    int jDay = jDay(jHour);
    _tempCalc.calculateTemperatures(jDay);
}//end setTemperatures

/**obtains the temperature for the specific hour of the day from the
*temperature array in TemperatureCalculator
*@param jHour    the current Julian hour (hours since 12AM January 1)
*@return         the current temperature
*/
public float getTemperature(int jHour){
    /*the temperature array in the Temperature Calculator holds one
        temperature for each hour of one day*/
    float temperature;
    int hour = jHour % 24;
    temperature = _tempCalc.temperatureAtTimeT(jHour % 24);
    temperature = Math.max(0f, Math.min(130f, temperature));
    return temperature;
}//end getTemperature

/**converts Julian hours into Julian days
*@param jHour    current Julian hour (hours since 12AM January 1)
*@return    the current Julian day (days since January 1)
*/
public int jDay(int jHour){
         return (int)(jHour / 24f);
}//end jDay

/**squares the Julian days
*@param jHour    current Julian hour (hours since 12AM January 1)
*@return    current Julian day squared
*/
public int jDaySquared(int jHour){
         return (int)(jDay(jHour) * jDay(jHour));
}//end jDaySquared

/**calculates the average infested bark area in square dm
*@return    the average infested bark area in square decimeters
*/
public float avgInfBarkArea(){
    /*use Math.max to ensure avgInfBarkArea is at least 50 square dm*/
    float avgInfBarkArea = (float)Math.max(50f, (float)(-650.60f + (float)(55.23f * _standAvgDBH)));
    return avgInfBarkArea;
}

/**calculates the number of pines per hectare
*@return    the number of pines per hectare
*/
public float pinePerHectare(){
    float pinePerHectare = (float)(_pineBA / (_standAvgDBH * _standAvgDBH * 0.00007853982f));
    return pinePerHectare;
}

/**calculates the number of trees containing SPB parents if this value
*was not specified (this value cannot be empirically measured, so was either
*calculated ouside of Spot, or will be calculated now)
*@param initTreesUnderAttack  the number of trees initially under SPB attack
*@param initTreesWithEggs  the number of trees initially containing SPB eggs
*@return        the number of trees containing SPB parents
*/
protected int treesWithParents(int initTreesUnderAttack,
    int initTreesWithEggs){

    int treesWithParents = (int)(initTreesWithEggs + initTreesUnderAttack -
        (float)(treesWithEggs(initTreesWithEggs) + treesUnderAttack(initTreesUnderAttack)));
    return treesWithParents;
}

/**adjusts trees under attack for unmeasured attacking adults; uses no
```

```
*global variables, so it is safe to use anywhere in the constructor
*@param initTreesUnderAttack    the initial number of trees under SPB attack
*@return    the adjusted number of trees under SPB attack
*/
protected int treesUnderAttack(int initTreesUnderAttack){
     return (int)(0.9123f * initTreesUnderAttack + 0.5f);
}

/**adjusts trees with eggs for unmeasured presence of eggs; uses no global
*variables, so it is safe to use anywhere in the constructor
*@param initTreesWithEggs   the initial number of trees containing SPB eggs
*@return the adjusted number of trees containing SPB eggs
*/
protected int treesWithEggs(int initTreesWithEggs){
    return (int)(0.7826f * initTreesWithEggs + 0.5f);
}

/**adjusts trees with eggs for unmeasured presence of eggs; uses global
*variables, so _treesWithEggs must be set to some value before calling
*this method
*@return the adjusted number of trees containing SPB eggs
*/
protected int treesWithEggs(){
    return _treesWithEggs;
}

/**distributes the number of infested trees among the different life
*stages of SPB, if this distribution was initially unknown
*@param infested    the total number of SPB-infested trees
*/
protected void distributeInfested(int infested){
    _treesUnderAttack = (int)(0.25f * infested + 0.5f);
    _treesWithParents = (int)(0.05f * infested + 0.5f);
    _treesWithEggs = (int)(0.09f * infested + 0.5f);
    _treesWithBrood = (int)(0.11f * infested + 0.5f);
    _treesWithImm = (int)(infested - _treesUnderAttack -
    _treesWithParents - _treesWithEggs - _treesWithBrood);
    if(_infested == 1){
        _treesWithImm = infested;
    }//end if
}//end distributeInfested

/**calculates the egg mortality at _start
*@return    the initial egg mortality
*/
public float eggInitMort(){
    float eggInitMort;
    int day = jDay(_start);
    eggInitMort = (float)(Math.min(0.1474f,
    Math.exp(-3.37044786f - 0.03688004f * _pineBA -
        0.04255650f * _standAvgDBH + 0.00864398f * day)));
    if(eggInitMort <= 0){
       eggInitMort = 0.001f;
    }
    return eggInitMort;
}//end eggInitMort
```

```
/**calculates the immature mortality rate at _start
*@return    the initial immature mortality rate per hour
*/
public float immatureInitMort(){
    float immatureMortRate = (float)Math.min(0.9193f, Math.max(0.4183f,
        (0.4174929f - 0.01014249f * (_hardwoodBA + _pineBA) +
        0.00361542f * _standAvgDBH + 0.00554016f * jDay(_start) -
        0.00001458f * jDaySquared(_start) + 0.12780644f * _loblollyRatio)));
    return immatureMortRate;
}//end immatureInitMort

/**calculates the brood mortality at _start
*@return    the initial brood mortality per hour
*/
public float broodInitMort(){
    float broodAdultMortRate = (float)Math.min(0.8344f, Math.max(0.1394f,
        (-.38424717f - 0.01768258f * _pineBA + 0.01049436f * _standAvgDBH +
        0.00122082f * pinePerHectare() + 0.00508942f * jDay(_start) -
        0.00001107f * jDaySquared(_start))));
    return broodAdultMortRate;
}//end broodInitMort

/**calculates the attacking SPB mortality, as used by class AttackPool
*@see AttackPool
*@return    the attacking SPB mortality
*/
public float attackingMort(){
    float mort = 0.0910f;
    float attackingMort;
    if(_pineBA < 21){
        attackingMort = (float)(0.97f * mort + 0.03f);
    }   else if(_pineBA > 30){
        attackingMort = (float)(1.03f * mort - 0.03f);
    }   else{
        attackingMort = mort;
    }//end else
    if(attackingMort <= 0){
        attackingMort = 0.001f;
    } else if(attackingMort >= 1){
        attackingMort = 0.999f;
    }
    return attackingMort;
}//end attackingMort()

/**adjusts the attacking mortality "to account for increased mortality in
*high density populations" as in the original Hog Model (Gail Hines 1979,
*PhD Thesis from the University of Arkansas)
*@return    the adjusted attacking mortality at the beginning of the
*           simulation
*/
public float attackingInitMort(){
    float attackingInitMort = (float)(0.001f + 0.999f * attackingMort());
    return attackingInitMort;
}//end attackingInitMort
```

```
/**calculates the density of  SPB eggs per sq dm
*@return    the density of SPB eggs per square decimeter
*/
public float eggDensity(){
    float density = (float)Math.min(268.5f, Math.max(58.7f, (164.4660239f +
        49.3082156f * _loblollyRatio - 0.1304835f * pinePerHectare())));
    return density;
}//end eggDensity

/**calculates the density of attacking adult SPB (units unspecified)
*@return    the density of attacking adult SPB
*/
public float attackingDensity(){
    float density =(float)Math.min(19.17f, Math.max(2.0f, 2 * (6.5567561f -
        0.0074305f * pinePerHectare() + 1.0535825f * _loblollyRatio)));
    return density;
}//end attackingDensity

/**calculates the density of larval and pupal SPB per sq dm
*@return    the density of larval and pupal SPB per square decimeter
*/
public float immatureDensity(){
    float density = (float)(eggDensity() * (1f - eggInitMort()));
    return density;
}//end immatureDensity

/**calculates the density of brood adult SPB per sq dm
*@return    the density of brood adult SPB per square decimeter
*/
public float broodDensity(){
    float density = (float)(immatureDensity() * (1f - immatureInitMort()));
    return density;
}//end broodDensity

/**calculates the density of emerging adult SPB (units unspecified)
*@return    density of emerging adult SPB
*/
public float emergingDensity(){
    float density = (float)(broodDensity() * (1f - broodInitMort()));
    return density;
}

/**calculates the density of parent adult SPB per square dm
*(equivalent to the density of attacking adult SPB)
*@return    the density of parent adult SPB per square dm
*/
public float parentDensity(){
    return attackingDensity();
}

/**calculates initial population of attacking adult SPB from number of
*trees under attack and other stand variables
*@return    initial population of attacking adult SPB
*/
public float attacking()
{
    float initialPop;
    float attackingInitMort = attackingInitMort();
    if(attackingInitMort < 1 && attackingInitMort >= 0){
        initialPop = (float)((_treesUnderAttack * 0.75f + 0.2f * _treesWithParents) * attackingDensity() *
            avgInfBarkArea() / (1f - attackingInitMort)) - (_initialAttackPool / (1f - attackingInitMort));
    }   else{
        initialPop = 0f;
    }//end else
    return initialPop;
}//end attacking

/**calculates initial population of parent adult SPB from number of trees
*initially under attack, number of initial trees with eggs, and other
*initial stand characteristics
*@return    the initial SPB parent population
*/
public float parent(){
    float initialPop = (float)((_treesUnderAttack * 0.2f +
        0.6f * _treesWithParents + 0.1f * _treesWithEggs) * parentDensity() * avgInfBarkArea());
    return initialPop;
}//end parent()

/**calculates initial number of SPB eggs from the number of trees initially
*under attack, number of trees with eggs, and other stand characteristics
*@return    initial population of SPB eggs
*/
public float egg(){
    float initialPop = (float)((_treesUnderAttack * 0.05f +
        0.2f * _treesWithParents + 0.6f * _treesWithEggs) * eggDensity() * avgInfBarkArea());
    return initialPop;
}//end egg

/**calculates initial number of SPB larvae/pupae from numbers of trees
*with eggs, with larvae/pupae, and with brood adults
*@return    the initial population of SPB larvae and pupae
*/
public float immature(){
    float initialPop;
    initialPop = (float)((0.3f * _treesWithEggs + 0.95f * _treesWithImm + 0.1f * _treesWithBrood) *
        immatureDensity() * avgInfBarkArea());
    return initialPop;
}//end immature

/**calculates initial number of SPB brood adults from numbers of trees
*with larvae/pupae and with brood adults, and other stand characteristics
*@return    initial population of SPB brood adults
*/
public float brood(){
    float initialPop = (float)((0.05f * _treesWithImm +
        0.6f * _treesWithBrood) * broodDensity() * avgInfBarkArea());
    return initialPop;
}//end brood

/**calculates initial number of SPB emerging adults from numbers of
*trees with brood adults, and other stand characteristics
*@return    initial population of SPB emerging adults
```

```
*/
public float emerging(){
    float initialPop = (float)(0.3f * _treesWithBrood *
        emergingDensity() * avgInfBarkArea());
    return initialPop;
}//end emerging

/**finds initial number of SPB in attack pool, and also determines which
*thresholds the initial attack pool has passed
*/
public void initializeAttackPool(){
    _initialAttackPool = 0;
    _attractionThreshold = initialAttractionThreshold();
    _boleCapacity = initialBoleCapacity();
    _failedHours = 0;
    _enoughToAttack = false;
    _enoughToKill = false;

    int intR;
    float quarterBoles;
    /*determine the value for the constant intR; values can be 0,1,2,3*/
    intR = (int)(3f * _treesUnderAttack -
        4f * (int)(3f * _treesUnderAttack / 4f));

    /*find the number of quarter boles under attack; values can be
        0, 0.25, 0.5, 0.75*/
    quarterBoles = (float)((float)(intR )/ 4f);

    /*check to see if any trees are initially under attack*/
    if(_treesUnderAttack == 0){
          /*if no trees under attack, assume that pheromone cloud is lost,
              and dispersion is occurring, so attraction threshold will be
              more difficult to attain*/
        _attractionThreshold =  (float)(7.5f * initialAttractionThreshold());

          //the attack failed for morning when the spot was ground checked
        _failedHours = 7;
    }   else if(intR != 0){  //the bole is partially filled
        _enoughToAttack = true;     //the tree is under attack

    //if initial attack pool < resistance threshold multiplier
    if((quarterBoles) * attackingDensity() < 2f){

        _initialAttackPool = quarterBoles * initialBoleCapacity();
    }   else{ //if attack pool > resistance threshold
        _enoughToKill = true;
        _boleCapacity = ((float)(initialBoleCapacity() *
                ((float)(4 - intR)) / 4f));
        }//end else
    }//end else if
}//end initializeAttackPool

/**calculates the initial resistance threshold
*@return    the initial resistance threshold, the population that the SPB
*           must exceed in order to successfully kill a tree
*/
```

```
public float resistanceThreshold(){
   float resistanceThreshold = (float)(2f * avgInfBarkArea());
   return resistanceThreshold;
}

/**calculates the initial bole capacity
*@return    the initial bole capacity, the total number of SPB that may
*           fit within one tree
*/
public float initialBoleCapacity(){
   float boleCapacity = (float)(attackingDensity() * avgInfBarkArea());
   return boleCapacity;
}

public float initialAttackPool(){
   return _initialAttackPool;
}
public float attractionThreshold(){
   return _attractionThreshold;
}
public float bole Capacity(){
   return _boleCapacity;
}
public int failedHours(){
   return _failedHours;
}
public boolean enoughToAttack(){
   return _enoughToAttack;
}
public boolean enoughToKill(){
   return _enoughToKill;
}

/**calculates the initial attraction threshold
*@return    the initial attraction threshold, the number of SPB required
*           to begin a successful attack of a tree; this is related to
*           aggregation pheromones
*/
public float initialAttractionThreshold(){
   float initialAttractionThreshold = (float)(0.26f * avgInfBarkArea());
   return initialAttractionThreshold;
}

/*make initial stand characteristics publicly available*/

/**returns the Julian hour when the simulation began
*@return    the Julian hour when the simulation began, on the date of the
*           first SPB infestation survey
*/
public int start(){
   return _start;
}

/**returns the Julian hour when the simulation is to terminate
*@return    the Julian hour when the simulation will terminate
*/
```

```
  public int end(){
    return _end;
  }

  /**returns the number of initially infested trees
  *@return    the number of trees initially infested with SPB
  */
  public int infested(){
    return _infested;
  }

  /**returns the number of initially dead trees
  *@return    the number of trees intially killed by SPB
  */
  public int dead(){
    return _dead;
  }

  /**returns the initial number of trees under SPB attack
  *@return    the adjusted number of trees under SPB attack
  */
  public int treesUnderAttack(){
    return _treesUnderAttack;
  }

  /**returns the decimal portion of loblolly in the stand (less than 1)
  *@return    the decimal portion of loblolly in the infested stand
  */
  public float loblollyRatio(){
    return _loblollyRatio;
  }

  /**returns the pine basal area of the infested stand, in square m/hectare
  *@return    pine basal area, in square meters per hectare
  */
  public float pineBA(){
    return _pineBA;
  }

  /**returns the hardwood basal area of the infested stand, in square m/ha
  *@return    hardwood basal area, in square meters per hectare
  */
  public float hardwoodBA(){
    return _hardwoodBA;
  }

  /**returns the mean diameter at breast height for the infested stand, in cm
  *@return    mean diameter at breast height, in centimeters
  */
  public float standAvgDBH(){
    return _standAvgDBH;
  }
  public int initialUnderAttack(){
    return _initialUnderAttack;
  }
  public int initialWithEggs(){
        return _initialWithEggs;
    }
    public int initialWithImm(){
        return _initialWithImm;
    }
    public int initialWithBrood(){
        return _initialWithBrood;
    }
    public float longitude(){
        return _longitude;
    }
    public float latitude(){
        return _latitude;
    }
}//end class Spot
```

B.2.6 Tree.java

```
/**Tree.java holds information about the dead trees in the SPB infestation.
*Infested trees are calculated from the portions of trees each SPB life stage
*infests.  The calculation for the current number of infested trees, and the
*values for the numbers of infested trees over time are found in Simulator.
*Tree.java is currently only for calculations and storage of numbers of dead
*trees. The number of dead trees is calculated by incrementing the number of
*dead trees each time the SPB attack pool exceeds the bole capacity of a tree.
*Dead trees include the number of infested trees, as this model assumes any
*infested tree dies.
*/

package JavaSPB1;

/**holds information about SPB-killed trees
*@see Spot
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class Tree{
   /**the SPB infestation, contains all initial stand conditions*/
   protected Spot _spot;

   /**the number of SPB-killed trees*/
   protected int _numTrees;

   /**the amount of space left in the tree bole for the SPB to infest*/
   protected float _boleCapacity;

   /**constructor for a tree in a specific infestation spot
   *@param spot    the SPB infestation spot
   */
   public Tree(Spot spot){
      /*set the SPB spot equal to the one passed in*/
      _spot = new Spot();
      _spot = spot;
      /*initialize the number of trees in this stage of infestation*/
      _numTrees = initialTrees();
      /*initialize the bole capacity*/
      _boleCapacity = _spot.boleCapacity();
   }//end constructor

   /**default constructor*/
   public Tree(){
   }

   /**returns the number of initial dead trees
   *@return    number of initial dead trees
   */
   public int initialTrees(){
      return _spot.dead();
   }
```

```
    /**returns the number of currently dead trees
    *@return    number of dead trees
    */
    public int num(){
        return _numTrees;
    }

    /**Sets the current number of dead trees.
    *@param trees   the new number of dead trees
    */
    public void set(int trees){
        _numTrees = trees;
    }

    /**Adds a certain number of trees to the current number of dead trees.
    *@param newTrees    the number of trees to be added to the number of
    *                   currently dead trees
    */
    public void addTrees(int newTrees){
        _numTrees += (int)newTrees;
    }

    /**Sets a smaller bole capacity, after some bole capacity has been occupied
    *by SPB, or resets to the initial bole capacity after a tree has been
    *consumed and the SPB move on to a new tree.
    *@param newBoleCapacity     the new value for the bole capacity
    */
    public void setBoleCapacity(float newBoleCapacity){
        _boleCapacity = newBoleCapacity;
    }//end setBoleCapacity

    /**returns the bole capacity of the tree being currently attacked
    *@return    the current bole capacity
    */
    public float boleCapacity(){
        return _boleCapacity;
    }//end boleCapacity
}//end class Tree
```

B.2.7 AttackPool.java

```
/**AttackPool.java determines the success of an attack on each tree, increments
*numbers of dead trees, and adjusts the population of attacking adults that
*survive to become parent adults.
*/

package JavaSPB1;
import java.io.*;

/**determines the success of an attack on each tree, increments the numbers of
*dead trees if the attack is successful, and allows for attacking adult SPB
*mortality if the attack is unsuccessful
*@see Spot
*@see AttackingAdult
*@see Tree
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/

public class AttackPool{
   /**the SPB infestation spot; contains information about the infestation*/
   protected Spot _spot;

   /**the object to hold information about the trees under SPB attack*/
   protected Tree _tree;

   /**the number of SPB in the attack pool*/
   protected float _attackPool;

   /**the number of SPB that will be passed on to parent adult cohorts*/
   protected float _augment;

   /**number of daytime hours when there haven't been enough SPB to
      establish an attack on a tree*/
   protected int _failedHours;

   /**the number of SPB required to establish an attack on a tree;
      related to aggregation pheromone concentrations*/
   protected float _attractionThreshold;

   /*declare booleans to describe whether thresholds have been met*/

   /**whether there are enough SPB to establish an attack on a tree*/
   protected boolean _enoughToAttack;

   /**whether there are enough SPB to kill a tree*/
   protected boolean _enoughToKill;

   /**whether there are enough SPB to fill a tree,
      in other words to exceed the bole capacity*/
   protected boolean _enoughToFill;

   /**boolean to describe whether attack pool is being distributed; if true,
    the attack pool is being distributed among trees, and more trees are
    being attacked, killed, and filled*/
protected boolean _filling;

/**constructor
*@param spot    the SPB infestation spot
*@param tree    the tree object to hold information about the trees under
*               attack
*/
public AttackPool(Spot spot, Tree tree){
    float boleCapacity;
    /*initialize the global variables whose new values were passed in*/
    _spot = new Spot();
    _spot = spot;
    _tree = new Tree(_spot);
    _tree = tree;

    /*initialize the rest of the global variables*/
    _failedHours = _spot.failedHours();
    _attractionThreshold = _spot.attractionThreshold();

    _enoughToAttack = _spot.enoughToAttack();
    _enoughToKill = _spot.enoughToKill();
    _filling = false;   //not yet distributing the attack pool

    /*initialize the number of SPB in the attack pool*/
    _attackPool = _spot.initialAttackPool();
}//end constructor

/**determines the success of an attack on each tree, increments numbers of
*dead trees, and adjusts population of attacking adults that survive to
*become parent adults
*@param jHour    the Julian hour (hours since 12AM January 1)
*@param pool     the new number of SPB attacking adults that are joining the
*               attack pool
*@return        the number of SPB attacking adults remaining after attacking
*               a tree or trees
*/
public float colonizeTrees(int jHour, float pool){
    /*suplement the current attack pool with the new population passed in*/
    _attackPool += pool;
    int counter = 0;
    _augment = 0;           //initialize _augment for the time step

    /*this is a do{ }while(filling) loop
          It is important that this loop be called only once during a time
       step unless filling == true, which only occurs when the attack pool
       exceeds the attraction and resistance thresholds, as well as the
       bole capacity.  In this case ONLY will the attack pool be allowed to
       attack new trees (one at a time). The loop will continue until the
          attack pool has been exhausted and sinks below the bole capacity.
       There are 3 booleans: _enoughToKill, _filling, and _enoughToAttack,
       which control which groups of statements are reached*/
    do{
        if(counter > 100000){
            try{
```

```
            FileWriter fw = new FileWriter("EscapeAttack.dat");
            BufferedWriter bw = new BufferedWriter(fw);
            PrintWriter outFile = new PrintWriter(bw);
            outFile.println("Too many iterations through attack pool."
               + _attackPool);
            outFile.close();
        } catch (IOException e){
        }
        _attackPool = 0; //attack pool is way too big!
        return -1;
    }//end if

    /*the following if statement is true if it is not yet known if the
       attraction threshold, resistance threshold, or bole capacity
       have been reached*/
    if(!_enoughToKill && !_enoughToAttack){

       /*if the attackPool falls below the attraction threshold, the
          pheromone cloud will disperse, and there will not be enough
          SPB to attack a tree */
       if(_attackPool < _attractionThreshold){
          dispersal(jHour);
          _enoughToAttack = false;
          _filling = false;
       }   else{             //else, there will be enough SPB to attack
          aggregation();
          _enoughToAttack = true;
       }//end else
    }//end if

    /*the following statement is true if the attraction threshold has
       been reached, but it is not yet known if the resistance
       threshold or bole capacity have been reached this time step*/
    if(_enoughToAttack && !_enoughToKill){

       /*if the attack pool falls below the resistance threshold, the
          pine will be able to pitch out the SPB and the attack will
          fail. There will not be enough SPB to kill the tree.*/
       if(_attackPool < _spot.resistanceThreshold()){
          attackFailing();
          _enoughToKill = false;
          _filling = false;
       }   else{   //there are enough SPB to successfully kill the tree
          _tree.addTrees(1);
          _enoughToKill = true;
       }//end else
    }//end if

    /*the following statement is true if the attraction threshold and
       resistance threshold have been reached, but it is not yet known
       if the bole capacity has been reached this time step*/
    if(_enoughToKill){

       /*if attack pool is less than bole capacity, there will not be
          enough SPB to continue attacking new trees*/
       if(_attackPool < _tree.boleCapacity()){
```

```
                partlyFullTree();
                _filling = false; //can't fill the tree
            }   else{   /*there will be too many SPB to fit in one tree,
                            and they will attack new trees*/
                fullTree();
                _filling = true; //still filling trees
                _enoughToKill = false;
                _enoughToAttack = false;
            }//end else
        }//end if(_enoughToKill)
        counter ++;
    }while(_filling);       //while more than enough SPB for 1 tree

    /*return surviving attack pool to be added to parent adult cohorts*/
    return _augment;
}//end colonizeTrees

/**handles case in which attackPool falls below  attraction threshold,
*and pheromone cloud will disperse
*@param jHour    the Julian hour (hours since 12AM January 1)
*/
protected void dispersal(int jHour){
    /*if the hour is between 8AM and 8PM (daylight hours)*/
    if((jHour + 1) % 24 > 8 && (jHour + 1) % 24 <= 20 &&
        jHour != _spot.start()){

        _failedHours ++;    //increment failed daytime hours
        if(_failedHours == 6){
            //the attraction threshold becomes harder to attain
            _attractionThreshold = (float)(7.5f * _attractionThreshold);
        }//end if
    }//end if
}//end dispersal

/**handles case in which the attackPool has reached the attraction
*threshold (pheromone cloud is strong enough for an attack to establish)
*/
protected void aggregation(){
    /*if there had been 6 hours before aggregation was achieved, decrease
        the attack pool*/
    if(_failedHours >= 6){
        _attackPool = (float)(_attackPool - _attractionThreshold + _spot.initialAttractionThreshold());
        //reset the attraction threshold to its initial value
        _attractionThreshold = _spot.initialAttractionThreshold();
    }//end if
    _failedHours = 0; //reset the number of failed days
}//end aggregation

/**handles case where there aren't enough SPB to kill a tree
*/
protected void attackFailing(){
    //attack pool suffers mortality if tree's defenses haven't been overcome
    _attackPool *= (float)(1f-_spot.attackingMort());
}//end attackFailing

/**handles case where there are not enough SPB to completely fill a
```

```
  *successfully attacked tree bole
  */
  protected void partlyFullTree(){
    /*send the attack pool on to become parents*/
    _augment += (float)_attackPool;
    /*tree bole now contains the number in the attack pool less than its
       initial bole capacity*/
    _tree.setBoleCapacity((float)(_tree.boleCapacity() - _attackPool));
    _attackPool = 0;    //reset attack pool
  }//end partlyFullTree

  /**handles case where there are more than enough SPB to fill 1 tree
  */
  protected void fullTree(){

    /*all of the attack pool (equivalent to the bole capacity, because the
       tree is full) in the tree will become parents*/
    _augment += (float)(_tree.boleCapacity());

    /*the attack pool is decreased by those in the attack pool that will
       become parents*/
    _attackPool -= (float)(_tree.boleCapacity());

    /*a new tree to attack will be completely empty*/
    _tree.setBoleCapacity(_spot.initialBole Capacity());
  }//end fullTree

  /**returns the number of SPB in the attack pool
  *@return    the number of attacking adult SPB in the attack pool
  */
  public float attackPool(){
    return _attackPool;
  }
}//end attackPool()
```

B.2.8 SPBCohort.java

```
/**SPBCohort.java keeps track of the SPB populations within each cohort of each
*developmental stage.  It also has access to some of the attributes of the
*cohort's developmental stage.
*/
package JavaSPB1;

/**Keeps track of the current population and other attributes of each cohort of
*each SPB developmental stage.
*@see Stage
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class SPBCohort{
   /**the cohort's current population*/
   protected float _population;

   /**the cohort's current density per sq dm of bark*/
   protected float _density;

   /**the cohort's developmental stage*/
   protected Stage _stage;

   /**the amount of development the cohort has achieved, range 0-1*/
   protected float _totalDevelopment;

   /**whether the default development rate for the SPB life stage has been
      overridden for this cohort*/
   protected boolean _develRateOverridden;

   /**whether the default mortality rate for the SPB life stage has been
      overridden for this cohort*/
   protected boolean _mortRateOverridden;

   /**a development object which may override the default development rate
      for the SPB life stage for this cohort*/
   protected Development _development;

   /**a development object which may override the default mortality rate
      for the SPB life stage for this cohort*/
   protected Mortality _mortality;

   /**the current temperature*/
   protected float _temperature;

   /**constructor for all initial cohorts
   *@param stage   the development stage of the SPB cohort
   *@param cohortNumber    the number of initial cohorts up to the current
   *                     one in the current development stage
   *@param totalCohorts    the total number of initial cohorts in the current development stage
   */
   public SPBCohort(Stage stage, int cohortNumber, int totalCohorts){
      /*make this cohort's devel stage equal to the one passed in*/
```

```
        _stage = new Stage();
        _stage = stage;
        /*initialize the population in the cohort*/
        _population = (float)(_stage.initialPop() / (float)totalCohorts);
        /*initialize the total development of the cohort*/
        _totalDevelopment = (float)(1f - (cohortNumber + 1) /(float)(totalCohorts));
        _development = new Development();
        _mortality = new Mortality();
        /*initialize the density*/
        _density = _stage.density();
        _develRateOverridden = false;
        _mortRateOverridden = false;
    }//end constructor

    /**constructor for creating a new cohort during a simulation
    *@param stage   the development stage of the cohort
    *@param population  the number of SPB in the cohort
    */
    public SPBCohort(Stage stage, float population){
        _stage = new Stage();
        _stage = stage;
        /*the total development of a new cohort is zero*/
        _totalDevelopment = 0;
        _population = population;
        _density = _stage.density();
        _development = new Development();
        _mortality = new Mortality();
        _develRateOverridden = false;
        _mortRateOverridden = false;
        _temperature = 0f;
    }//end constructor

    /**default constructor*/
    public SPBCohort(){
    }

    /**sets a new current population for the cohort
    *@param population  the new population of the cohort
    */
    public void setPop(float population){
        _population = population;
    }//end setPop

    /**returns the current population of the cohort
    *@return    the current population of the cohort
    */
    public float pop(){
        return _population;
    }//end pop

    /**returns the cohort's current density per square decimeter of bark
    *@return    the current density of the cohort
    */
    public float density(){
        return _density;
    }//end density
```

```
/**set the cohort's current density per square decimeter of bark
*@param density     the new density of the cohort
*/
public void setDensity(float density){
    _density = density;
}//end setDensity

/**returns the development stage object for the cohort
*@return    the development stage object for the cohort
*/
public Stage stage(){
    return _stage;
}//end stage

/**returns the name of the development stage of the current cohort
*@return    the name of the development stage of the current cohort
*/
public String nameOfStage(){
    return _stage.name();
}//end nameOfStage

/**updates the temperature and time for the current time step, so that
*temperature-dependent development rates will be accurate for the cohort;
*also adds to the total fractional development for the cohort
*@param jHour           the Julian hour (hour since 12AM January 1)
*@param temperature     the current temperature
*/
public void develop(int jHour, float temperature){
    _stage.setTimeAndTemperature(jHour, temperature);
    _temperature = temperature;
    _totalDevelopment += develRate();
}//end develop

/**updates the temperature and time for the current time step, so that
*temperature-dependent development rates will be accurate for the cohort;
*this is used only for developing the emerging adult cohort
*@param jHour           the Julian hour (hour since 12AM January 1)
*@param temperature     the current temperature
*/
public void developEmerging(int jHour, float temperature){
    _stage.setTimeAndTemperature(jHour, temperature);
    _temperature = temperature;
}//end developEmerging

/**returns the fractional development rate for the cohort of the current
*development stage per time step at the current temperature
*@return    the fractional development rate for the cohort
*/
public float develRate(){
    float develRate;
    if(_develRateOverridden){
        develRate = _development.develRate(_temperature);
    } else{
        develRate = _stage.develRate();
    }//end else
```

```
        return develRate;
    }//end develRate

    /**returns the total fractional development of the cohort so far;
    *when total development = 1, the cohort is fully developed
    *@return    the total fractional development the cohort has achieved
    */
    public float totalDevelopment(){
        return _totalDevelopment;
    }

    /**returns the mortality rate for the cohort of the current development
    *stage per time step for the current day
    *@return    the mortality rate for the cohort per time step
    */
    public float mortRate(){
        float mortRate;
        if(_mortRateOverridden){
            mortRate = _mortality.mortRate(_stage);
        } else{
            mortRate = _stage.mortRate();
        }
        return mortRate;
    }//end mortRate

    /**decreases the population of the cohort to allow for mortality
    */
    public void takeMortality(){
        _population = (float)(_population * (1f - mortRate()));
    }//end takeMortality

    /**returns the egg production rate (only applicable for parent adults) at this time step
    *@return    the egg production rate (for parent adult cohorts)
    */
    public float eggProductionRate(){
        return _stage.eggProductionRate();
    }//end eggProductionRate

    /**sets the Development object to override the default development rate for this cohort
    *@param development     the Development object defining this cohort's development rate
    */
    public void setDevelopment(Development development){
        _development = development;
        if(development.develType() != 7){
            _develRateOverridden = true;
        }
    }

    /**sets the Mortality object to override the default mortality rate for this cohort
    *@param mortality     the Mortality object defining this cohort's mortality rate
    */
    public void setMortality(Mortality mortality){
        _mortality = mortality;
        _mortRateOverridden = true;
    }
}//end SPBCohort
```

B.2.9 Stage.java

```
/**Stage.java is the base class for all of the developmental stages of SPB,
*including AttackingAdult, ParentAdult, Egg, Immature, BroodAdult, and
*EmergingAdult
*/

package JavaSPB1;

/**Stage is the base class for all life stages of Southern Pine Beetle
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class Stage{
   /**the Julian hour of the current time step*/
    protected int _jHour;
   /**the temperature at the current time step*/
   protected float _temperature;

    /**the SPB infestation, including all initial stand conditions*/
   protected Spot _spot;

   /**the constructor for a SPB life stage in a specific infestation spot
   *@param spot    the infestation spot
   */
   public Stage(Spot spot){
      //set the SPB spot equal to the one passed in
      _spot = new Spot();
      _spot = spot;
      _jHour = _spot.start() + 1;
       _temperature = _spot.getTemperature(_jHour);
    }

   /**the default constructor
   */
   public Stage(){
    }

   /**sets the time and temperature for this time step; the variables _jHour and
   *    _temp will be used by the Stage subclasses in their rate calculations
    *@param jHour          the Julian hour of the current time step
   *@param temperature     the temperature of the current time step
   */
   public void setTimeAndTemperature(int jHour, float temperature){
      _jHour = jHour;
      _temperature = temperature;
    }

   /**calculates the fraction of egg development per day; also used in the
   *calculation of AttackingAdult, ParentAdult, and Immature development per
   *day, as well as the fecundity per day
   *@return    the fraction of egg development per day
   */
   public float eggDevelPerDay(){
    float eggDevelPerDay = 0f; //the fraction of egg development per day
    double[] temperatures = {0d, 49.9, 50d, 59d, 68d, 77d, 81.5, 86d,
        90.5, 92.5, 92.6, 130d};
    double[] rates = {0d, 0d, 0.0309, 0.0852, 0.1438, 0.2207, 0.2599,
        0.2871, 0.2485, 0.1961, 0d, 0d};
    eggDevelPerDay = interpolate(temperatures, rates, _temperature);
    return eggDevelPerDay;
  }//end eggDevelPerDay()

  /**interpolates the rate for a given temperature from two arrays--
  *one with temperatures, and the other with corresponding
  *development rates for those temperatures
  *@param temperatures    an array of temperatures
  *@param rates   an array of known fractional development rates at each of
  *              the temperatures in the temperature array
  *@param temperature     the temperature for which a fractional development rate
  *                    is sought
  *@return the interpolated rate of development at a certain temperature
  */
  public float interpolate(double[] temperatures, double[] rates,
    float temperature){

    int i = 1;        //index to loop through temperature array
    float rate = 0;              //the development rate at the passed-in temperature

    /*whether a temperature in the array has been found that is higher than
      the passed-in temperature*/
    boolean found = false;

    while((i <= temperatures.length) && (found == false)){
      if(temperature <= temperatures[i]){
        if(temperatures[i] != temperatures[i - 1]){
          rate = (float)(rates[i - 1] + (rates[i] - rates[i - 1]) *
            (temperature - temperatures[i - 1]) /
            (temperatures[i] - temperatures[i - 1]));
        }  else{
          rate = (float)rates[i - 1];
        }//end else
        found = true;
      }//end if
      i ++;
    }//end while
    return rate;
  }//end interpolate

  /**returns the fractional development rate of a SPB life stage at the jHour
  *and temperature held in Stage's global variables _jHour and _temperature;
  *to be overridden by Stage's subclasses
  *@return    the fractional development rate for a SPB life stage for the
  *          current time step
  */
  public float develRate(){
    return 0;
  }

  /**returns the mortality rate for a SPB life stage at the jHour held in
  *Stage's global variable _jHour;
  *to be overridden by Stage's subclasses
  *@return    the mortality rate for the current time step
  */
  public float mortRate(){
     return 0;
  }

  /**returns the current density of the SPB life stage;
  *to be overridden by Stage's subclasses
  *@return    the current density of the SPB life stage
  */
  public float density(){
     return 0;
  }

  /**returns the initial population of an SPB life stage;
  *to be overridden by the subclasses of Stage
  *@return    the initial population of an SPB life stage
  */
  public float initialPop(){
     return 0;
  }

  /**returns the initial mortality rate of an SPB life stage;
  *to be overridden by Stage's subclasses
  *@return    the initial mortality rate of an SPB life stage
  */
  public float initMort(){
     return 0;
  }

  /**returns the SPB fractional egg production rate for the current time step;
  *to be overridden by Stage's subclass ParentAdult only
  *@return    the SPB fractional egg production rate for the current time step
  */
  public float eggProductionRate(){
     return 0;
  }

  /**returns the name of the SPB life stage;
  *to be overridden by Stage's subclasses
  *@return    the name of the SPB life stage
  */
  public String name(){
     return "stage";
  }

  /**returns the infestation spot object
  *@return    the infestation spot
  */
  public Spot spot(){
     return _spot;
  }
}//end class Stage
```

B.2.10 AttackingAdult.java

```
/**AttackingAdult.java is a subclass of Stage.  It is one of the SPB
*developmental stages.  This class provides information about SPB attacking
*and initial attacking adults.
*/

package JavaSPB1;

/**provides methods and attributes for the attacking adult life stage of SPB
*@see Spot
*@see Stage
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class AttackingAdult extends Stage{

   /**constructor
   *@param spot    the SPB infestation spot
   */
   public AttackingAdult(Spot spot){
       //call the base class Stage
       super(spot);
   }//end constructor

   /**returns the initial population of attacking adults
   *@return    the initial population of SPB attacking adults
   */
   public float initialPop(){
       return (float)_spot.attacking();
   }//end initialPop

   /**returns the density of attacking adults per square decimeter
   *@return    density of SPB attacking adults per square decimeter of bark
   */
   public float density(){
       return  (float)_spot.attackingDensity();
   }//end float density

   /**returns the fractional development rate (in this case, equivalent to the
   *successful attack rate) of attacking adults per time step
   *@return    the fractional development rate of SPB attacking adults per time
   *          step at the current temperature
   */
   public float develRate(){
       float successfulAttackRate = (float)(attackRatePerDay() / 24f);
       return successfulAttackRate;
   }//end float develRate

   /**returns the rate at which developing attacking adults will not survive
   *to become parent adults at each time step
   *@return    the mortality rate of SPB attacking adults per time step for the
   *          current day
   */
```

```
    public float mortRate(){
        return _spot.attackingInitMort();
    }//end float mortRate

    /**returns the development rate (attack rate) of attacking adults per day
    *@return    the development rate of attacking adult SPB per day for the
    *           current temperature
    */
    protected float attackRatePerDay(){
        return (float)(eggDevelPerDay() * 4f);
    }//end attackRatePerDay

    /**returns the name of this SPB life stage
    *@return    the name of the attacking adult SPB life stage, 'attacking'
    */
    public String name(){
        return "attacking";
    }//end name
}//end class AttackingAdult
```

B.2.11 ParentAdult.java

```
/**ParentAdult.java is a subclass of Stage.  It is one of the SPB developmental
*stages.  This class provides information about SPB parent adults.
*/

package JavaSPB1;

/**contains methods and attributes of the parent adult SPB life stage
*@see Stage
*@see Spot
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class ParentAdult extends Stage{

   /**constructor
   *@param spot    the SPB infestation spot
   */
   public ParentAdult(Spot spot){
       //call the constructor of the base class, Stage
       super(spot);
   }//end constructor

   /**returns the initial population of parent adults
   *@return     the initial population of SPB parent adults
   */
   public float initialPop(){
       return (float)_spot.parent();
   }//end initialPop

   /**returns the density of parent adults per square decimeter
   *@return    the density of SPB parent adults per square decimeter of bark
   */
   public float density(){
       return (float)_spot.parentDensity();
   }//end density

   /**returns the fractional development rate, which in this case is the
   *reemergence rate (the rate at which parent adults become emerging adults)
   *per time step
   *@return    the parent adult development rate per time step for the current
   *           temperature
   */
   public float develRate(){
       float reemergenceRate = (float)(reemergencePerDay() / 24f);
       return reemergenceRate;
   }//end develRate

   /**returns the rate at which developing parent adults will not survive to
   *become emerging adults at each time step
   *@return    the parent adult mortality rate per time step
   */
   public float mortRate(){
       float mortRate = 0.35f;
       return mortRate;
   }//end mortRate

   /**returns the reemergence rate per day for parent adults
   *@return    the reemergence rate of SPB parent adults per day
   */
   protected float reemergencePerDay(){
       return (float)(eggDevelPerDay() * 4f);
   }//end reemergencePerDay

   /**returns the rate at which parent adults produce eggs per time step
   *@return    the egg production rate per time step at the current temperature
   */
   public float eggProductionRate(){
       float eggProductionRate = (float)(eggProductionPerDay() / 24f);
       return eggProductionRate;
   }//end eggProductionRate

   /**returns the rate at which parent adults lay eggs per day
   *@return    the egg production rate per day at the current temperature
   */
   protected float eggProductionPerDay(){
       return (float)(eggDevelPerDay() * 55f);
   }//end float eggProductionPerDay

   /**returns the name for the parent adult life stage
   *@return    the name for this stage, 'parent'
   */
   public String name(){
       return "parent";
   }//end name
}//end class ParentAdult
```

B.2.12 Egg.java

```
/**Egg.java is a subclass of Stage.  It is one of the SPB developmental stages.
*This class provides information about SPB eggs.
*/

package JavaSPB1;

/**contains attributes and methods for the egg life stage of SPB
*@see Spot
*@see Stage
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class Egg extends Stage{

   /**constructor
   *@param spot    the SPB infestation spot
   */
   public Egg(Spot spot){
      //call the constructor of the base class, Stage
      super(spot);
   }//end constructor

   /**returns the initial population of eggs
   *@return    the initial SPB egg population
   */
   public float initialPop(){
      return (float)_spot.egg();
   }//end initialPop

   /**returns the density of eggs per square decimeter
   *@return     density of SPB eggs per square decimeter of bark
   */
   public float density(){
      return (float)_spot.eggDensity();
   }//end density

   /**returns the fractional development rate for eggs per time step at the
   *current temperature
   *@return    fractional development rate for eggs per time step at the
   *           current temperature
   */
   public float develRate(){
      float develRate =  (float)(eggDevelPerDay() / 24f);
      return develRate;
   }//end develRate

   /**returns the rate at which developing eggs will not survive to become
   *larvae/pupae at each time step during the current day
   *@return    the mortality rate for eggs on the current day
   */
   public float mortRate(){
      int day = (int)_spot.jDay(_jHour);         //get the current day
```

```
        float eggMortRate = (float)Math.min(0.1474f,
            Math.exp(-3.37044786f - 0.03688004f * _spot.pineBA() -
            0.04255650f * _spot.standAvgDBH() + 0.00864398f * day));
        if(eggMortRate <= 0)
            eggMortRate = 0.001f;
        return eggMortRate;
    }//end mortRate

    /**returns the name for this stage
    *@return    the name for the SPB egg life stage, 'egg'
    */
    public String name(){
        return "egg";
    }//end name
}//end class Egg
```

B.2.13 Immature.java

```
/**Immature.java is a subclass of Stage.  It is one of the SPB developmental
*stages.  This class provides information about SPB larvae and pupae.
*/

package JavaSPB1;

/**contains methods and attributes of the immature life stage of SPB, which
*includes larvae and pupae
*@see Spot
*@see Stage
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class Immature extends Stage{

   /**constructor
   *@param spot    the SPB infestation spot
   */
   public Immature(Spot spot){
      super(spot);
   }//end constructor

   /**returns the initial population of larvae and pupae
   *@return    initial population of SPB larvae and pupae
   */
   public float initialPop(){
      return (float)_spot.immature();
   }//end initialPop()

   /**returns the density of larvae and pupae per square decimeter
   *@return    density of SPB larvae and pupae per square decimeter of bark
   */
   public float density(){
      return (float)_spot.immatureDensity();
   }//end float density()

   /**returns the fractional development rate for larvae/pupae per time step
   *@return    fractional development rate for SPB larvae/pupae per time step
   *           at the current temperature
   */
   public float develRate(){
      float develRate = (float)(immatureDevelPerDay() / 24f);
      return develRate;
   }//end float develRate

   /**returns the rate at which developing larvae and pupae will not survive
   *to become brood adults at each time step at the current temperature
   *@return    mortality rate for the larvae/pupae for the current time step
   *           at the current temperature
   */
   public float mortRate(){
      float immatureMortRate = (float)Math.min(0.9193f, Math.max(0.4183f,
           (0.4174929f - 0.01014249f * (_spot.hardwoodBA() + _spot.pineBA()) +
           0.00361542f * _spot.standAvgDBH() + 0.00554016f *
           _spot.jDay(_jHour) - 0.00001458f * _spot.jDaySquared(_jHour) +
           0.12780644f * _spot.loblollyRatio())));
      return immatureMortRate;
   }//end mortRate

   /**returns the fractional development rate per day for larvae and pupae
   *@return     the fractional development rate per day for larvae and pupae
   */
   protected float immatureDevelPerDay(){
      return (float)(eggDevelPerDay() * 0.23f);
   }//end immatureDevelPerDay

   /**returns the name for this stage
   *@return    the name for the larva/pupa SPB life stage, 'immature'
   */
   public String name(){
      return "immature";
   }//end name
}//end class Immature
```

B.2.14 BroodAdult.java

```
/**BroodAdult.java is a subclass of Stage.  It is one of the SPB developmental
*stages.  This class provides information about SPB brood adults.
*/

package JavaSPB1;

/**contains methods and attributes of the brood adult life stage of SPB
*@see Stage
*@see Spot
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class BroodAdult extends Stage{

   /**constructor
   *@param spot    the SPB infestation spot
   */
   public BroodAdult(Spot spot){
      //call the constructor of the base class, Stage
      super(spot);
   }//end constructor

   /**returns the initial population of brood adults
   *@return    the initial population of SPB brood adults
   */
   public float initialPop(){
      return (float)_spot.brood();
   }//end initialPop

   /**returns the density of brood adults per square decimeter
   *@return    the density of brood adult SPB per square decimeter of bark
   */
   public float density(){
      return (float)_spot.broodDensity();
   }//end density

   /**returns the fractional development rate of brood adults per time step
   *@return    fractional development rate of SPB brood adults per time step
   *           at the current temperature
   */
   public float develRate(){
      float broodAdultDevelRate = (float)(broodAdultDevelPerDay() / 24f);
      return broodAdultDevelRate;
   }//end develRate

   /**returns the rate at which developing brood adults will not survive to
   *become emerging adults at each time step
   *@return    mortality rate for SPB brood adults per time step during the
   *           current day
   */
   public float mortRate(){
      float broodAdultMortRate = (float)(Math.min(0.8344f, Math.max(0.1394f,
         (-.38424717f - 0.01768258f * _spot.pineBA() +
         0.01049436f * _spot.standAvgDBH() +
         0.00122082f * _spot.pinePerHectare() +
         0.00508942f * _spot.jDay(_jHour) -
         0.00001107f * _spot.jDaySquared(_jHour)))));
      return broodAdultMortRate;
   }//end mortRate

   /**returns the fractional development rate for brood adults per day
   *@return    fractional development rate for brood adults per day
   */
   protected float broodAdultDevelPerDay(){
      //interpolate BroodAdultDevelPerDay from array
      double[] temperatures = {0d, 54.4, 54.5, 59d, 60.98, 68f, 69.8, 71.96,
         77d, 80.06, 80.6, 86d, 87.98, 87.99, 130d};
      double[] rates = {0d, 0d, 0.054, 0.069, 0.088, 0.16, 0.156, 0.158,
         0.152, 0.177, 0.161, 0.153, 0.189, 0d, 0d};
      float broodAdultDevelPerDay = interpolate(temperatures, rates, _temperature);
      return (float)(1.7f * broodAdultDevelPerDay);
   }//end broodAdultDevelPerDay

   /**returns the name of the Stage
   *@return    the name of the brood adult infestation stage, 'brood'
   */
   public String name(){
      return "brood";
   }//end name
}//end class BroodAdult
```

B.2.15 EmergingAdult.java

```
/**EmergingAdult.java is a subclass of Stage.  It is one of the SPB
*developmental stages.  This class provides information about SPB emerging adults.
*/

package JavaSPB1;

/**provides methods and attributes for the emerging adult life stage of SPB
*@see Spot
*@see Stage
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class EmergingAdult extends Stage{
   /**constructor
   *@param spot    the SPB infestation spot
   */
   public EmergingAdult(Spot spot){
      //call the base constructor, Stage
      super(spot);
   }//end constructor

   /**returns the initial population of emerging adults
   *@return    the initial population of emerging SPB adults
   */
   public float initialPop(){
      return (float)_spot.emerging();
   }//end initialPop

   /**returns the density of emerging adults (not a true density,
   *but important for infested tree calculations in Simulator)
   *per square decimeter
   *@see Simulator
   *@return    density of emerging adults
   */
   public float density(){
      return (float)_spot.emergingDensity();
   }//end float density

   /**returns the fractional development rate for emerging adults per time step
   *@return    the fractional development rate for emerging adults per time
   *           step at the current temperature
   */
   public float develRate(){
      //half life reached after 0.125 days for emerging adults
      float rearrival = 5.55f;
      if(_jHour - _spot.start() == 35){
         int i = 0;
      }
      if(((_jHour + 1) % 24 < 8) || ((_jHour + 1) % 24 >= 20) ||
         (_temperature < 57)){
         rearrival = 0f;
      }
```

```
        float rearrivalRate = (float)(1f - Math.exp(-rearrival / 24f));
        return rearrivalRate;
    }//end float develRate

    /**returns the rate at which developing emerging adults will not survive to
    *become attacking adults at each time step (checks hardwood BA conditions
    *and range of mortality rate)
    *@return    mortality rate of emerging SPB adults per time step during the
    *           current day
    */
    public float mortRate(){
        float emergingMortality;

        //check hardwood BA conditions
        if(_spot.hardwoodBA() > 20){
            emergingMortality = (float)(emergingMortalityHelper() * 0.9f + 0.1f);
        }   else{
            emergingMortality = emergingMortalityHelper();
        }//end else

        //check range conditions
        if(emergingMortality <= 0){
            emergingMortality = 0.001f;
        } else if(emergingMortality >= 1){
            emergingMortality = 0.999f;
        }
        return emergingMortality;
    }//end mortRate

    /**checks the pine BA conditions to help find the value for emerging adult
    *mortality
    *@return    an initial value for emerging adult mortality per time step for
    *           the current day
    */
    protected float emergingMortalityHelper(){
        float mortConstant = 0.3227f;
        float emergingMortality;

        //check pine BA conditions
        if(_spot.pineBA() < 21){
            emergingMortality = (float)(0.93f * mortConstant + 0.07f);
        }  else if(_spot.pineBA() > 30){
            emergingMortality = (float)(1.07f * mortConstant - 0.07f);
        }  else{
            emergingMortality = (float)mortConstant;
        }//end else
        return emergingMortality;
    }//end emergingMortalityHelper

    /**return the name for this stage
    *@return    the name for the emerging adult SPB life stage, 'emerging'
    */
    public String name(){
        return "emerging";
    }//end name
}//end class EmergingAdult
```

B.2.16 TemperatureCalculator.java

```
/**TemperatureCalculator.java calculates the temperature for a given hour
*according to the mean maximum and minimum temperatures for the day of the year,
*and according to the time of sunrise.  Three sine waves of day versus
*temperature are used for this calculation.
*/
package JavaSPB1;
import java.io.*;

/**calculates the temperature for a given hour according to the mean maximum
*and minimum temperatures for that day, and according to the hour of sunrise
*@see TemperatureFinder
*@see Sunrise
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class TemperatureCalculator{
    final float PI = 3.14159f;              //the constant pi

   /*holds the maximum temperatures of the previous, current, and next days*/
   private float[] _maxima;

   /*holds the minimum temperatures of the previous, current, and next days*/
   private float[] _minima;

   /*the time of daybreak for the current day at the current longitude and
       latitude*/
   private float _daybreak;

   /*holds the times of daybreak for the current and next days*/
   private float[] _daybreaks;

   /*holds temperatures for every hour of the current day*/
   private float[] _temperatureAtTimeT;

   /*the following 3 arrays hold the constants for 3 sine waves to describe the
   temperature, depending on the time of day and whether the sun is above the
   horizon.  The sine waves have temperature as the y axis and time as the
   x axis. */

   /*the amplitudes of the temperature sine wave*/
   private float[] _amplitude;

   /*the frequencies of the temperature sine wave*/
   private float[] _frequency;

   /*the translations or shifts of the temperature sine wave up or down the
       y axis */
   private float[] _verticalShift;

   /*the object to obtain daily mean maximum and minimum temperatures*/
   private TemperatureFinder _tempFind;
private Sunrise _sunrise;          //the object to obtain the time of sunrise

/**constructs a new TemperatureCalculator; initializes arrays to hold
*variables for 3 sine wave functions of day versus temperature
*@see TemperatureFinder
*/
public TemperatureCalculator(){
   //instantiate a new TemperatureFinder object
   _tempFind = new TemperatureFinder();

   //initialize arrays
   _daybreaks = new float[3];
   _temperatureAtTimeT = new float[24];
   _amplitude = new float[3];
   _frequency = new float[3];
   _verticalShift = new float[3];
   _maxima = new float[3];
   _minima = new float[3];
}//end constructor

/**Sets the lattitude and longitude so that the time of sunrise may be
*calculated.  It is possible that negative latitudes or longitudes will
*give an incorrect time of sunrise.
*@param longitude   the approximate longitude for the geographical state
*@param latitude    the approximate latitude for the geographical state
*@see Sunrise
*/
public void setLatAndLong(float longitude, float latitude){
   _sunrise = new Sunrise(longitude, latitude);
}//end setLatAndLong

/**calculates the array of temperatures (holds 1 temperature for every hour)
*for the passed-in Julian day
*@param jDay    current Julian day for which an array of temperatures must
*               be calculated; must be a positive integer
*@see TemperatureFinder
*@see Sunrise
*/
public void calculateTemperatures(int jDay){
   boolean minimum;
   for(int i = 0; i < 3; i ++){
      minimum = true;
      _minima[i] = _tempFind.temperature(jDay + (i - 1), minimum);
      minimum = false;
      _maxima[i] = _tempFind.temperature(jDay + (i - 1), minimum);
   }//end for

   _daybreaks[0] = 0f;     //_daybreaks[0] is not used
   _daybreaks[1] = _sunrise.sunrise(jDay);
   _daybreaks[2] = _sunrise.sunrise(jDay + 1);
   _daybreak = _daybreaks[1];
   if(_daybreak < 1 || _daybreak > 12){
      System.out.println("Error: daybreak out of bounds.");
   }

   /*fill the sine wave variables*/
```

```
        _verticalShift[0] = (float)((_maxima[0] + _minima[1]) / 2f);
        _amplitude[0] = (float)((_maxima[0] - _minima[1]) / 2f);
        _frequency[0] = (float)(PI / (10f + _daybreaks[1]));

        _verticalShift[1] = (float)((_maxima[1] + _minima[1]) / 2f);
        _amplitude[1] = (float)((_maxima[1] - _minima[1]) / 2f);
        _frequency[1] = (float)(PI / (14f - _daybreaks[1]));

        _verticalShift[2] = (float)((_maxima[1] + _minima[2]) / 2f);
        _amplitude[2] = (float)((_maxima[1] - _minima[2]) / 2f);
        _frequency[2] = (float)(PI / (10f + _daybreaks[2]));

        /*fill the temperature array for each hour of the day*/
        for(int i = 0; i < 24; i ++){
            if(i <= _daybreak){
                _temperatureAtTimeT[i] = (float)(_verticalShift[0] +
                    _amplitude[0] * (float)Math.cos((i + 10) * _frequency[0]));
            }   else if(i > _daybreak && i < 15){
                _temperatureAtTimeT[i] = (float)(_verticalShift[1] - _amplitude[1] *
                    (float)Math.cos((i - _daybreaks[1]) * _frequency[1]));
            }   else{
                _temperatureAtTimeT[i] = (float)(_verticalShift[2] +
                    _amplitude[2] * (float)Math.cos((i - 14) * _frequency[2]));
            }//end else
        }//end for
    }//end calculateTemperature

    /**returns the temperature for the current hour of the day
    *@param hourOfDay   the current hour of the day; must be within range 0-23
    *@return    the temperature at the current hour
    */
    public float temperatureAtTimeT(int hourOfDay){
        float temperature = 0f;
        try{
            temperature = _temperatureAtTimeT[hourOfDay];
        }   catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Error: hour of day not between 0 and 23.");
        }//end catch
        return temperature;
    }//end temperatureAtTimeT
}//end TemperatureCalculator
```

B.2.17 TemperatureFinder.java

```
/**TemperatureFinder.java retrieves the mean maximum and mean minimum
*temperatures for any day of the year.
*/
package JavaSPB1;
import java.io.*;

/**finds the mean maximum and minimum temperatures for any day of the year.
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class TemperatureFinder{
   /*initialize very large arrays for holding maximum and minimum temperatures
      for each day of the year*/
   double[] dailyHigh = {0,52,53.3,51.4,52.2,48.5,51.3,50.8,49.3,51.8,50.2,
      50.3,50.9,47.3,51.9,53.6,54.3,53.7,56.3,56.3,54.9,57.2,59.6,60,56.3,
      56.8,58.8,56.7,60.9,57.2,58.7,57.6,59.1,55.7,54.1,55.1,62.8,62.9,59.4,
      52.5,55.9,56.5,60.5,59.5,56.7,60.9,59.6,54.1,54.7,60.1,61.3,61.6,61.4,
      57.8,56.3,60.3,60.1,64.8,63.4,64.7,63.5,66.2,70,65.7,63.6,62.1,64,64.5,
      67.5,68.7,65.3,68.6,67.5,67.7,65.9,69.9,67.9,68.7,69.5,70.9,71.4,69,
      69.3,70.4,66.7,61.9,65.4,69.5,73.5,73.2,70.5,71.6,74.1,76.3,76.8,75.6,
      71.3,70.7,75.7,77.1,74.9,75.7,77.1,79,78.6,76.1,78.7,80.3,82,78.2,80.1,
      81.6,84.1,78.3,80.4,79.3,75.3,76.5,76.1,77,77.2,77,77.4,75.8,79.5,78.5,
      79.9,81.2,81.7,81.8,82.4,80.1,81.9,83.7,81.5,81.2,81.6,82,84.1,82.5,
      83.3,85.3,85.2,84.3,85.7,86.6,86.9,87.3,85.9,83.3,83.8,86.4,85.2,85.9,
      84.5,85.7,86.3,87.5,87.1,89.9,88.1,90.1,90.5,90.5,91.4,91,90.6,90.6,90,
      87.9,88.9,90,90.5,90.9,90.2,89.9,90.1,90.1,90.7,90.7,92.3,92.8,93.1,
      93.3,91.8,92.7,93.7,92.5,91.7,92.1,93.7,92.9,93.7,94.2,93.4,92.1,93.3,
      93.5,92.9,93.1,93.2,93.9,93.7,92.5,92.4,93,93.5,93.3,93.1,91.7,97,91.9,
      91.3,89.3,90.3,91.5,90.6,91.2,90.7,91.8,93.3,93.1,93.3,94,91.1,90.1,90,
      90.8,91.6,92,93.2,92.9,90.9,92.6,92.2,92.3,91.9,89.4,90.7,91.3,91.1,
      91.1,89.9,90.1,89.9,88.6,89.9,89,87.5,87.4,86.6,87.1,87.9,92.8,88.4,
      86.2,87.3,86.5,86.5,85.5,85.3,84.8,84.9,86,85.3,85.1,83.6,82.5,82.9,
      81.7,82.3,82.4,80,78.9,81.6,81.8,82.8,83.3,81.4,79.7,78.7,79.7,79.3,
      79.4,79.1,79.3,80.3,80.9,79.7,80.7,77.4,73.7,73.7,74.8,72.7,75.9,77.3,
      77.9,75.5,74,71.9,74.6,72.7,70.9,72.5,74.5,77.3,70.6,69.7,67.3,67,68.2,
      69.7,68.4,68.7,68.3,67.1,66.8,65.7,66.9,63.5,66.5,67.9,67.1,66.7,64.5,
      65.7,64.2,61.5,61.5,63.7,64.2,66.5,60.7,56.7,58.8,55.7,60.3,60.1,62.2,
      60.9,62.4,55.3,55.5,56.3,54.4,54.3,56.4,58.5,57.4,55.8,52.8,54.5,54.5,
      57.7,61.4,54.6,53.4,56.8,60.7,58.1,52.7,53.3,55.4,56.8,56.1,52.9};

   double[] dailyLow = {0,31.7,32.6,33.8,28.9,27.6,31.1,27,28.5,27.8,27.8,28.9,
      27.9,27.9,27.3,26.3,26.7,27.2,32.8,33.3,31.3,31.3,37.2,37.7,35,33.7,
      32.2,30.9,34.1,34.6,34.4,35.6,36.1,34.8,28.8,29.1,31.9,34.8,33.5,30.6,
      31.3,30.7,32.2,32.9,32.9,34.7,36.7,36.8,33.5,33.5,33.3,31.8,32.9,31.1,
      31.7,32.9,32,35.3,34.8,37.6,32.8,41.2,40.6,43.5,41.9,38.1,36.9,38.7,
      38.8,38.7,41.2,45.5,44,43.5,43.6,41.7,41.3,41,42.5,44.5,43.3,40.6,38.9,
      39.4,40.9,42.9,41.7,40.7,46.9,48.5,46,44.9,47.1,46.1,49.1,49.3,45.4,
      44.1,46.8,48.3,47.8,48.5,50.7,52.1,52.7,53.6,51,52.1,54.6,57.7,57.3,
      57.2,55.6,54.7,58.3,54.5,52.7,51.6,53,54.4,55.2,55.8,55.7,56.6,53,54.5,
      53.9,58.6,59.2,59.3,58.1,56.3,60.3,60.9,58.7,58.2,58.1,58.6,59.3,58.7,
      58.5,59.9,60.4,60.5,61.3,61.7,61,61.9,60.8,62.3,60.4,61.5,61,61.1,61.5,
      60.3,62.3,63.4,64.7,65.1,63.9,64.9,66.1,66.7,67.3,68.4,72.7,68.1,67.9,
      66.5,67.2,67.5,66.7,67.8,67.9,68.2,68.1,68.2,68.1,68.7,67.7,69.1,70.2,
      69.9,69.7,70.2,69.1,68.7,67.9,68.7,69.5,71,70.6,70.3,69.5,67.9,68.3,
      69.3,70.3,69.2,69.7,70,71.3,70.9,70.1,70.5,66.7,70.7,71.7,71.9,71.4,
      71.1,69.6,69.1,70.8,68.6,67.5,67.3,67.5,68.4,69.4,69.2,69.7,69.6,68.1,
      67.9,68.1,68.3,69.1,68.8,68.7,68.6,68.4,68.9,69.7,68.4,67.8,67.4,67.9,
      67.7,67.5,66.6,66.1,66.3,66.7,65.5,65,67,68.1,67.5,66.3,65.4,65.9,65.7,
      65.1,63.6,63.2,62.3,61.7,61.2,63.1,65.9,64.1,63.7,61.9,61.9,60.1,59.4,
      58.5,59.1,59.2,59.5,57.7,55.9,53.6,53.4,53.9,53.7,54.7,52.5,52.4,52.8,
      52.7,49.7,49.8,50.7,51.7,54,60.1,54.2,51.9,48.5,46.5,45.4,43.6,44.5,
      46.9,48,50.3,46.3,44.3,46.8,45.1,45.5,45.7,46.1,48,43.8,44.3,44.1,43.5,
      42.7,44.6,43.5,44.4,41.4,40.8,40.3,37.8,38.3,41.2,40.3,41.9,41.6,40.6,
      44.1,40.5,36.9,35.3,41.3,35.9,38.6,41,38,33.9,34.4,32.5,38.9,37.3,35.2,
      37.1,34.7,34.7,28.2,31.6,36.6,38.2,34.3,36.5,30.8,31.2,33.1,30.8,32.5,
      31.8,36,31.3,32.5,29.2,31.7,31.2,29.3,31.4,31.2,31.8,36.9,34.8};

   /**default constructor*/
   public TemperatureFinder(){
   }

   /**returns the mean minimum temperature for the day passed to this method,
   *or returns the mean maximum temperature for that day if the boolean minimum
   *is false; uses values from arrays dailyHigh[] and dailyLow[]
   *@param jDay    the Julian day to find the maximum or minimum temperature for
   *@param minimum     if true, indicates a minimum is sought, if false,
   *                  indicates a maximum is sought
   *@return    the mean maximum or minimum temperature for a day of the year
   */
   public float temperature(int jDay, boolean minimum){
      float temperature =0f;
      try{
          if(minimum){
              temperature = (float)dailyLow[jDay];
          }   else{
              temperature = (float)dailyHigh[jDay];
          }//end else
      }   catch(ArrayIndexOutOfBoundsException e){
          System.out.println("Error: day is not between 2 and 366.");
      }//end catch
      return temperature;
   }//end temperature
}//end TemperatureFinder
```

B.2.18 Sunrise.java

```
/**Sunrise calculates the time of sunrise for a given latitude and longitude
*on a given day of the year, using complex trigonometrical equations.
*/

package JavaSPB1;

/**calculates the time of sunrise for a given day of the year
*@see Declination
*@see Time
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class Sunrise{
    final float PI = 3.14159f;              //the constant pi

    /*the number of radians in a circle*/
    final float RADIANS_IN_CIRCLE = (float)(2f * PI);

    /*constant by which you multiply degrees to get radians*/
    final float CONVERT_TO_RADIANS = (float)(PI / 180f);

    /*altitude which the sun should cross to be considered risen, in degrees*/
    final float RISEN_ALTITUDE = -0.83333f;

    final float DEGREES_LONG_IN_HOUR = 15f;  //the number of degrees longitude in one hour
    final float DAYS_IN_YEAR = 365f;   //the number of days in a year
    final float HOURS_IN_DAY = 24f;    //the number of hours in a day
    final float HOURS_IN_HALF_DAY = 12f;    //the number of hours in half a day
    final float SECONDS_IN_HOUR = 3600f;    //the number of seconds in an hour
    private float _altitudeRadian;          //altitude, in radians
    private float _latitudeRadian;          //latitude, in radians
    private float _declinationRadian;       //declination, in radians

    /*the number of whole hours difference between the current longitude and
       0 longitude (the Prime Meridian)*/
    private int _hoursFrom0Long;

    /*any fraction of an hour more than _hoursFrom0Long away from 0 longitude*/
    private float _hourFraction;

    /*the fraction of the year up until the day before the current day,
       multiplied by 2*pi; for use in the sine functions in classes Declination
       and Time*/
    private float _yesterday;

    private Declination _declination;   //the object to obtain the current declination of the sun
    private float _floatDeclination;  //value for the declination
    private Time _time;                       //the object to obtain the function of time
    private float _floatTime;                //value for the time
    private float _cosOfDiurnalArc;      //the cosine of the diurnal arc
    private float _cosOfDiurnalArcSquared;  //_cosOfDiurnalArc squared
    private float _tangent;                    //the tangent of the diurnal arc
```

```
   private float _sunrise;                    //the time of sunrise

   /*the arc the sun travels to meet the RISEN_ALTITUDE, measured in radians*/
   private float _diurnalArc;

   /**sets the longitude and latitude for the calculation of the time of
   *sunrise at a certain location and day of the year
   *@param longitude   the longitude of the location
   *@param latitude    the latitude of the location
   */
   public Sunrise(float longitude, float latitude){
       _declination = new Declination();
       _time = new Time();
       _latitudeRadian = (float)latitude * CONVERT_TO_RADIANS;
       _altitudeRadian = (float)RISEN_ALTITUDE * CONVERT_TO_RADIANS;
       _hoursFrom0Long = (int)(longitude / DEGREES_LONG_IN_HOUR);
       _hourFraction = (float)((longitude/DEGREES_LONG_IN_HOUR) - (float)_hoursFrom0Long);
   }//end constructor

   /**calculates the time of sunrise for a given day at a given location
   *@param jDay    the current Julian day, a positive number in range 1-365
   *@return        the time of sunrise on the current day
   */
   public float sunrise(int jDay){
       _yesterday = (float)((jDay - 1) * RADIANS_IN_CIRCLE / DAYS_IN_YEAR);
       _floatDeclination = _declination.declination(_yesterday);
       _floatTime = _time.time(_yesterday);
       _declinationRadian = (float)_floatDeclination * CONVERT_TO_RADIANS;

       /*Compute the diurnal arc that the Sun traverses to reach the
           RISEN_ALTITUDE, by first calculating the cosine of the diurnal arc*/
       _cosOfDiurnalArc = (float)(((float)Math.sin(_altitudeRadian) -
          (float)(Math.sin(_latitudeRadian) * Math.sin(_declinationRadian))) /
          (Math.cos(_latitudeRadian) * Math.cos(_declinationRadian)));

       /*use the trigonometric identities sec^2 t = 1 + tan^2 t, and sec^2 t = 1/(cos^2 t) to obtain
         the tangent of the diurnal arc from the cos^2 of the diurnal arc*/
       _cosOfDiurnalArcSquared = (float)(_cosOfDiurnalArc * _cosOfDiurnalArc);
       _tangent =  (float)Math.sqrt((float)(1f / _cosOfDiurnalArcSquared)  - 1f);

       //then find the diurnal arc in radians
       _diurnalArc = (float)Math.atan(_tangent);

       /*ensure the diurnal arc is above the horizon; it should be in range
           0 to PI/2 or in range 3PI/2 to 2PI*/
       if(_cosOfDiurnalArc < 0){
           _diurnalArc = (float)(PI - _diurnalArc);
       }

       //find the time of sunrise in hours
       _sunrise = (float)(HOURS_IN_HALF_DAY - (float)((_diurnalArc * HOURS_IN_DAY / PI) / 2f) -
           (float)(_floatTime / SECONDS_IN_HOUR) + _hourFraction);
       return _sunrise;
   }//end sunrise
}//end class Sunrise
```

B.2.19 Declination.java

```
/**Declination.java calculates the declination of the sun according to the day
*of the year.  This is a Fourier calculation.
*/
package JavaSPB1;

/**calculates the declination of the sun on a given day by means of a Fourier
*calculation
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,  6 June 2002
*/
public class Declination{
   //declare arrays for the Fourier calculation
   private double[] A = {.4019997, -22.90387, -.396668, -.1572202, .0003482348,
       .00116435, -.01175, -.00586506, .009377923, .006095152, -.007914253,
       -.008301031, .006554838, .008157138, -.004251391, -.007036179,
       .001792343, .010032, -.001344027, -.0120317, -.001172711, .01122712,
       .001065523, -.01036945, -.002162149};
   private double[] B = {0., 4.036158, .05449076, .07382506, .0002655289,
       .01361031, .0071535, -.009600434, -.006319217, .009471375, .007141262,
       -.00636192, -.007715654, .005755398, .00976881, -.003298255, -.01041775,
       .002329703, .01210598, .0004879257, -.01042857,.0002261123, .01050115,
       .002028243, -.009079814};

   /**default constructor*/
   public Declination(){
   }

   /**calculates the declination of the sun on a given day
   *@param yesterday   the fraction of the year up until the day before the
   *                 current Julian day, multiplied by 2*pi
   *@return        the declination of the sun
   */
   public float declination(float yesterday){
       float declination = (float)A[0];
       for(int i = 1; i < 25; i ++){
          declination += (float)(A[i] * (float)Math.cos(i * yesterday) +
          B[i] * (float)Math.sin(i * yesterday));
       }//end for
       return declination;
   }//end declination
}//end Declination
```

B.2.20 Time.java

```
/**Time.java is a Fourier function to calculate the equation of time.
*/
package JavaSPB1;

/**uses a Fourier function to calculate the equation of time
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01, 6 June 2002
*/
public class Time{
   //declare arrays for the Fourier calculation
    private double[] A = {.122271, 25.44992, -200.4817, -5.6131, -8.630279,
       -1.200097, -.7681252, -.1276106, .1367468, .3244089, .2796091, .3083768,
       .6542435, .3246704, -.03410427, -.4181312, -.4980704, -.3663179,
       -.1377198, -.07871771, -.2612189, -.1191331, .1775613, .2020183,
       -.02654251};
    private double[] B = {0., -441.3921, -561.3943, -19.7549, -11.29022,
       -1.414121, -.9893808, -.5066575, -.1080109, .1207151, -.06266683,
       -.302649, -.2564405, -.563018, -.2570775, -.146313, -.2087959,
       -.03617579, .410778, .5364594, .3401139, .03505065, -.1786814,
       -.3602988, -.3240202};

   /**default constructor*/
   public Time(){
    }

   /**Uses a Fourier function to calculate the equation of time, using values
    *   from arrays A and B.
   *@param yesterday    The fraction of the year up until the day before the
    *   current Julian day, multiplied by 2*pi
   *@return   the current time, as adjusted by the Fourier equation
   */
   public float time(float yesterday){
       //initialize the time
       float time = (float)A[0];

       for(int i = 1; i < 25; i ++){
           time += (float)((float)(A[i] * (float)Math.cos(i * yesterday)) +
               (float)(B[i] * (float)Math.sin(i * yesterday)));
       }//end for
       return time;
    }//end time
}//end Time
```

B.3 Package servletHelper -Assists the servlet.

The package servletHelper contains the code for reading information from an HTML form, and printing the output in numerical or graphical form. Additional packages from Apache JServ, KavaChart, and Acme GifEncoder, were used to run the servlet, print a graph, and encode the graph as a .gif, respectively.

B.3.1 PrintForm.java

```
/**Printform.java prints the HTML form that the user may use to enter
*   information about a southern pine beetle infestation.
*/

package servletHelper;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**prints the form for the user to input data about the SPB infestation
* @see <a href = http://java.apache.org/jserv/index.html> Apache JServ</a>
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,        8 July 2002
*/
public class PrintForm{

  /**creates an object to print the HTML form for the user to enter data
  *@param a   a generic variable
  */
  public PrintForm(int a){
  }

  /**prints the form for the user to input data about the SPB infestation;
  *    after the form is submitted, prints the form with updated data
  *@param out           the output stream (in this case, text)
  *@param theSpotData a variable to indicate whether the form has changed,
  *                       if theSpotData == 1, then the form has changed
  *@param theSpotID       the name of the infestation spot for which the simulation is run
  *@param theMonth       the month the spot was investigated in the field
  *@param theDay        the day the spot was investigated in the field
  *@param theYear       the year the spot was investigated in the field
  *@param theState       the state in which the spot is located
  *@param deadTrees       the initial number of trees killed by SPB
  *@param infestedTrees   the initial number of trees infested with SPB
  *@param underAttack     the number of trees initially under SPB attack
  *@param withEggs          the number of trees initially containing SPB eggs
  *@param withImm        the number of trees initially containing SPB larvae
  *@param withBrood       the number of trees initially containing SPB brood
  *@param loblollyRatio   the decimal proportion of trees in the stand that are loblolly
  *@param standAvgDBH       the average diameter at breast height of the trees in the stand
  *@param dbhUnits    the units of standAvgDBH, either cm or inches
```

```
 *@param pineBA       the pine basal area of the stand
 *@param thePBAunits       the units of pineBA, either sq.m/hectare or sq.ft/acre
 *@param hardwoodBA       the hardwood basal area of the stand
 *@param theHBAunits       the units of hardwoodBA, either sq.m/hectare or sq.ft/acre
 *@param theDuration       the length of the simulation, in hours
 *@param javahogURL       web address of the the javahog simulator
 *@param goButtonURL       web address of the GO button image
 *@param spbiccToolboxURL       web address of the SPBICC toolbox
 *@param spbiccHomeURL       web address of the SPBICC home page
 *@exception IOException     an error may occur reading from or printing to the HTML page
 */
public void printForm(ServletOutputStream out, int theSpotData,
   String theSpotID, int theMonth, int theDay, int theYear, char theState,
   int deadTrees, int infestedTrees, int underAttack, int withEggs,
   int withImm, int withBrood, float loblollyRatio, float standAvgDBH,
   String dbhUnits, float pineBA, String thePBAunits, float hardwoodBA,
   String theHBAunits, int theDuration, String javahogURL,
   String goButtonURL, String helpButtonURL, String helpURL,
   String spbiccToolboxURL, String spbiccHomeURL)
throws IOException{
   String title = "Javahog Data Entry Form";
   out.print("<HTML><HEAD><TITLE>");
   out.print(title + "</TITLE>");
   out.println("<script language=\"JavaScript\"> ");
   out.println("<!-- allow for dynamically changing targets " +
      "for the form");
   out.println("//.gif output is  printed in target \"CHART\", ");
   out.println("//    text output is printed in target \"_blank\" ");
   out.println();
   out.println("var initialFormatSelection = \"Graphical output\";" +
      " //.gif format initially selected");
   out.println("var theFormatSelection = initialFormatSelection; ");
   out.println();
   out.println("function changeFormatSelection(selection){");
   out.println("    //if the output format selection box has been");
   out.println("    //    changed, change the output format");
   out.println("         theFormatSelection=selection;");
   out.println("}");
   out.println();
   out.println("function changeTarget(){");
   out.println("    if ((theFormatSelection == \"HTML table\") " +
      " || (theFormatSelection == \"Comma-delimited text\")) {");
   out.println("    //target long text to a new window");
   out.println("        window.document.spot.target = \"_blank\"; ");
   out.println("    } else {");
   out.println("    //if theFormatSelection==\"Graphical output\", ");
   out.println("    //target the .gif to the frame it fits perfectly");
   out.println("        window.document.spot.target = \"CHART\"; " );
   out.println("    }");
   out.println("}");
   out.println("// -->");
   out.println("</script>");
   out.println("</HEAD><BODY bgcolor=\"#FFFFFF\" >");
   out.println(" Mouse over or click on the " +
      "question marks for more information. Click on the GO button " +
      "to run the simulation. " +
      "<font color = \"#007000\"><H2>Input</H2><H3>Please enter " +
      "the following information about your southern pine beetle " +
      "infestation: </font></H3><br> ");
   out.println("<FORM NAME=spot METHOD=GET ACTION=" + javahogURL +
      " TARGET='CHART'>");
   out.println("<INPUT TYPE=hidden name=spotData value=" + theSpotData +
      ">");
   out.println("<a href =" + helpURL + "#spotID" +
      " target = \"_blank\"><img src=" + helpButtonURL +
      " alt=\"Enter up to 15 characters; characters may include spaces \""+
      " border = 0></a> " +
      " ID name for your SPB spot:");
   out.println("<UL><INPUT TYPE=TEXT maxlength = 15 NAME=spotID value=" +
      theSpotID + "></UL>");
   out.print("<a href =" + helpURL + "#date" +
      " target = \"_blank\"><img src=" + helpButtonURL +
      " alt=\"Enter a date between April and October from the years" +
      " 1500 to 3000 A.D.\" border = 0></a> " +
      " Date the spot was ground-checked: <UL>");

   /*This may be a round-about way of ensuring the month passed in is
      displayed in the form, but it works. Other <select> lists are done
      the same way, by printing out "selected" in the html file when the
      <option> item matches the one passed in.        */

   out.println("<select name = month>");
   out.println("<option value = 4");
   if(theMonth == 4){
      out.print("selected");
   }
   out.print("> April </option>");
   out.println("<option value = 5");
   if(theMonth == 5){
      out.print("selected");
   }
   out.print("> May </option>");
   out.println("<option value = 6");
   if(theMonth == 6){
      out.print("selected");
   }
   out.print("> June </option>");
   out.println("<option value = 7");
   if(theMonth == 7){
      out.print("selected");
   }
   out.print("> July </option>");
   out.println("<option value = 8");
   if(theMonth == 8){
      out.print("selected");
   }
   out.print("> August </option>");
   out.println("<option value = 9");
   if(theMonth == 9){
      out.print("selected");
   }
   out.print("> Sept.  </option>");
```

```
out.println("<option value = 10");
if(theMonth == 10){
   out.print("selected");
}
out.print("> Oct. </option>");
out.println("</select>");

out.println("<select name = day >");
out.println("<option value = 1 ");
if(theDay == 1) out.print("selected");
out.print("> 1 </option>");
out.println("<option value = 2");
if(theDay == 2) out.print("selected");
out.print("> 2 </option>");
out.println("<option value = 3");
if(theDay == 3) out.print("selected");
out.print(">3 </option>");
out.println("<option value = 4 ");
if(theDay == 4) out.print("selected");
out.print("> 4 </option>");
out.println("<option value = 5");
if(theDay == 5) out.print("selected");
out.print("> 5 </option>");
out.println("<option value = 6");
if(theDay == 6) out.print("selected");
out.print(">6 </option>");
out.println("<option value = 7");
if(theDay == 7) out.print("selected");
out.print("> 7 </option>");
out.println("<option value = 8");
if(theDay == 8) out.print("selected");
out.print(">8 </option>");
out.println("<option value = 9");
if(theDay == 9) out.print("selected");
out.print("> 9  </option>");
out.println("<option value = 10");
if(theDay == 10) out.print("selected");
out.print("> 10 </option>");
out.println("<option value = 11");
if(theDay == 11) out.print("selected");
out.print("> 11 </option>");
out.println("<option value = 12");
if(theDay == 12) out.print("selected");
out.print("> 12 </option>");
out.println("<option value =13");
if(theDay == 13) out.print("selected");
out.print(">13 </option>");
out.println("<option value =14");
if(theDay == 14) out.print("selected");
out.println(">14 </option>");
out.println("<option value = 15");
if(theDay == 15) out.print("selected");
out.print("> 15 </option>");
out.println("<option value = 16");
if(theDay == 16) out.print("selected");
out.print(">16 </option>");
out.println("<option value = 17");
if(theDay == 17) out.print("selected");
out.print("> 17 </option>");
out.println("<option value = 18");
if(theDay == 18) out.print("selected");
out.print(">18 </option>");
out.println("<option value =19");
if(theDay == 19) out.print("selected");
out.print("> 19  </option>");
out.println("<option value = 20");
if(theDay == 20) out.print("selected");
out.print("> 20 </option>");
out.println("<option value = 21");
if(theDay == 21) out.print("selected");
out.print("> 21 </option>");
out.println("<option value =22");
if(theDay == 22) out.print("selected");
out.print("> 22 </option>");
out.println("<option value = 23");
if(theDay == 23) out.print("selected");
out.print(">23 </option>");
out.println("<option value = 24");
if(theDay == 24) out.print("selected");
out.print("> 24 </option>");
out.println("<option value = 25");
if(theDay == 25) out.print("selected");
out.print("> 25 </option>");
out.println("<option value = 26");
if(theDay == 26) out.print("selected");
out.print(">26 </option>");
out.println("<option value = 27");
if(theDay == 27) out.print("selected");
out.print(">27 </option>");
out.println("<option value = 28");
if(theDay == 28) out.print("selected");
out.print(">28 </option>");
out.println("<option value = 29");
if(theDay == 29) out.print("selected");
out.print(">29  </option>");
out.println("<option value = 30");
if(theDay == 30) out.print("selected");
out.print("> 30 </option>");
out.println("<option value = 31");
if(theDay == 31) out.print("selected");
out.print("> 31 </option>");
out.println("</select>");
out.println("<INPUT TYPE=text name=year size=4 maxlength=4 value =" +
   theYear + "></UL>");
out.println("<a href =" + helpURL + "#state" +
   " target = \"_blank\"><img src=" + helpButtonURL +
   " alt=\"Choose a Southeastern U.S. state\" border = 0></a>" +
   " State: <UL>");
out.println("<select name=state>");
out.println("<option value='A' ");
if(theState == 'A') out.print("selected");
out.print(">AL</option>");
out.println("<option value='R' ");
if(theState== 'R') out.print("selected");
out.print(">AR</option>");
out.println("<option value='F' ");
if(theState== 'F') out.print("selected");
out.print(">FL</option>");
out.println("<option value='G' ");
if(theState == 'G') out.print("selected");
out.print(">GA</option>");
out.println("<option value='L' ");
if(theState == 'L') out.print("selected");
out.print(">LA</option>");
out.println("<option value='M' ");
if(theState == 'M') out.print("selected");
out.print(">MS</option>");
out.println("<option value='N' ");
if(theState == 'N') out.print("selected");
out.print(">NC</option>");
out.println("<option value='O' ");
if(theState == 'O') out.print("selected");
out.print(">OK</option>");
out.println("<option value='S' ");
if(theState == 'S') out.print("selected");
out.print(">SC</option>");
out.println("<option value='T' ");
if(theState == 'T') out.print("selected");
out.print(">TN</option>");
out.println("<option value='X' ");
if(theState == 'X') out.print("selected");
out.print(">TX</option>");
out.println("<option value='V' ");
if(theState == 'V') out.print("selected");
out.print(">VA</option>");
out.println("</select></UL>");
out.println("<a href =" + helpURL + "#deadTrees" +
   " target = \"_blank\"><img src=" + helpButtonURL +
   " alt=\"Enter the number of previously infested (now dead) trees," +
   " an integer in the range 0 to 9000\" border = 0></a>");
out.println("Number of previously infested trees: <UL>");
out.println("<INPUT TYPE=TEXT NAME=dead size=4 maxlength=4 value=" +
   deadTrees + "></UL>");
out.println("<a href =" + helpURL + "#infestedTrees" +
   " target = \"_blank\"><img src=" + helpButtonURL +
   " alt=\"Enter the number of currently infested trees," +
   " an integer in the range 1 to 9000\" border = 0></a>");
out.println("Number of currently infested trees: <UL>");
out.println("<INPUT TYPE=TEXT NAME=infested size=4 maxlength=4 value=" +
   infestedTrees +"></UL>");
out.println("<a href =" + helpURL + "#stage" +
   " target = \"_blank\"><img src=" + helpButtonURL +
   " alt=\"Enter the number of trees infested with each developmental" +
   " stage of southern pine beetle, " +
   " in the range 0 to 9000\" border = 0></a>");
out.println("Optional: Number of trees currently: <UL>");
out.println("<INPUT TYPE=TEXT NAME=attack size =4 maxlength =4 value=" +
   underAttack +"> under attack <br>");
```

```
out.println("<INPUT TYPE=TEXT NAME=eggs size = 4 maxlength =4 value=" +
   withEggs + "> with eggs<br>");
out.println("<INPUT TYPE=TEXT NAME=immatures size=4 maxlength=4 " +
   "value= " + withImm +
   "> with larvae or pupae<br>");
out.println("<INPUT TYPE=TEXT NAME=brood size =4 maxlength=4 value= " +
   withBrood + "> with brood adults<br></UL>");
out.println("<a href =" + helpURL + "#loblolly" +
   " target = \"_blank\"><img src=" + helpButtonURL +
   " alt=\"Enter the proportion of loblolly trees in the stand," +
   " calculated as loblolly/total, " +
   " a real number in the range 0.0 to 1.0\" border = 0></a>");
out.println("Decimal proportion of loblolly in the stand: ");
out.println(" <UL><INPUT TYPE=TEXT " +
   "NAME=loblolly size=5 maxlength=5 value=" + loblollyRatio +
   "></UL>");
out.println("<a href =" + helpURL + "#DBH" +
   " target = \"_blank\"><img src=" + helpButtonURL +
   " alt=\"Enter the mean diameter at breast height of the trees " +
   " in the stand; be sure to select the correct units.  Ranges for" +
   " this real number value are 12.0 to 77.0 cm or 5.0 to 30.0 " +
   " inches.\" border = 0></a>");
out.println("Mean diameter at breast height (DBH) of the stand: " +
   "<UL>");
out.println("<INPUT TYPE=TEXT NAME=DBH size=6 maxlength=6 value = " +
   standAvgDBH +">");
out.println("<select name=DBHunits>");
out.println("<option value=\"in\" ");
if(dbhUnits.equals("in")) out.print("selected");
out.print(">inches</option>");
out.println("<option value=\"cm\" ");
if(dbhUnits.equals("cm")) out.print("selected");
out.print(">cm</option>");
out.println("</select></UL>");
out.println("<a href =" + helpURL  + "#pineBA" +
   " target = \"_blank\"><img src=" + helpButtonURL +
   " alt=\"Enter the pine basal area of the stand; be sure to " +
   " select the correct units.  Ranges for this real number value " +
   " are 2.0 to 69.0 square meters per hectare or 10.0 to 300.0 " +
   " square feet per acre.\" " + " border = 0></a>");
out.println("Pine basal area:<UL>");
out.println("<INPUT TYPE=TEXT NAME=pine size=6 maxlength=6 value = " +
   pineBA + ">");
out.println("<select name=pBAunits>");
out.println("<option value=\"ft\" ");
if(thePBAunits.equals("ft")) out.print("selected");
out.print(">sq.ft/acre </option>");
out.println("<option value=\"m\" ");
if(thePBAunits.equals("m")) out.print("selected");
out.print(">sq.m/hectare</option>");
out.println("</select></UL>");
out.println("<a href =" + helpURL + "#hardwoodBA" +
   " target = \"_blank\"><img src=" + helpButtonURL +
   " alt=\"Enter the hardwood basal area of the stand; be sure to " +
   " select the correct units.  Ranges for this real number value " +
   " are 0.0 to 69.0 square meters per hectare or 0.0 to 300.0 " +
   " square feet per acre.\" " + " border = 0></a>");
out.println("Hardwood basal area:");
out.println("<UL><INPUT TYPE=TEXT " +
   "NAME=hardwood size=6 maxlength=6 value = " + hardwoodBA + ">");
out.println("<select name=\"hBAunits\" >");
out.println("<option value=\"ft\" ");
if(theHBAunits.equals("ft")) out.print("selected");
out.print(">sq.ft/acre </option>");
out.println("<option value=\"m\" ");
if(theHBAunits.equals("m")) out.print("selected");
out.print(">sq.m/hectare</option>");
out.println("</select></UL>");
out.println("<a href =" + helpURL + "#duration" +
   " target = \"_blank\"><img src=" + helpButtonURL +
   " alt=\"Enter the number of days for which the simulator should " +
   " predict values for dead and infested trees, an integer in the " +
   " range 1 to 92.\" " + " border = 0></a>");
out.println("Number of days for simulation:");
out.println("<UL><INPUT TYPE=TEXT " +
   "NAME=duration size =3 maxlength=3 value = " + theDuration +
   "></UL>");
out.println("<INPUT TYPE=RESET value=Reset onClick = " +
   " \"changeFormatSelection(initialFormatSelection);\"><br> ");
out.print("<font color = \"#007000\"><H2>Output</H2><H3>Choose what " +
   "you want the model to output: </font></H3>");
out.println("<a href =" + helpURL + "#trees" +
   " target = \"_blank\"><img src=" + helpButtonURL +
   " alt=\"Check the \'Trees\' box and select dead and/or infested " +
   " trees for graphing.\" " +
   " border = 0></a>");
out.println("<INPUT TYPE=radio name=theAction value = " +
   "run checked>Trees:");
out.println("<UL><INPUT TYPE=checkbox " +
   "name=deadChecked value=yes checked> Dead Trees" +
   "<br><INPUT TYPE=checkbox name=infestedChecked " +
   " value=yes checked> Infested Trees </UL>");
out.println("<a href =" + helpURL + "#populations" +
   " target = \"_blank\"><img src=" + helpButtonURL +
   " alt=\"Check the \'Beetle Populations\' box and select " +
   " any or several beetle populations for  " +
   " graphing.\" border = 0></a>");
out.println("<INPUT TYPE = radio " +
   "name=theAction value = pops >Beetle populations: <UL>");
out.println("<INPUT TYPE= checkbox name = oldAttacking value = yes>" +
   " Initial Attacking Adults <br>" +
   "<INPUT TYPE=checkbox name=attacking value=yes checked>" +
   " Attacking Adults <br><INPUT TYPE " +
   "= checkbox name=parent value=yes> Parent Adults <br>" +
   " <INPUT TYPE=checkbox name=egg value=yes> Eggs <br>" +
   " <INPUT TYPE=checkbox name=imm value=yes> Larvae and Pupae " +
   " <br><INPUT TYPE=checkbox name=broodAdult value=yes> " +
   " Brood Adults <br><INPUT " +
   "TYPE=checkbox name=emerge value=yes> EmergingAdults </UL>");
out.println("<a href =" + helpURL + "#format" +
   " target = \"_blank\"><img src=" + helpButtonURL +
   " alt=\"Select among the different options for viewing " +
          " the output.\" " +
          " border = 0></a>");
      out.println("Choose an output format: <br><UL>");
      out.println("<select name=outFormat onChange= ");
      out.println("\"changeFormatSelection" +
          "(window.document.spot.outFormat.options[selectedIndex].text); " +
          " \" >");
      out.println("<option value=\"graphical\" selected>Graphical output");
      out.println("<option value=\"table\">HTML table");
      out.println("<option value=\"comma\">Comma-delimited text</select>");
      out.println("</UL><br><input " + "type=\"image\" " +
          " alt=\"run the simulation\" src=" +
          goButtonURL + " border = 0  + onClick = ");
      out.println(" \"changeTarget();\" >");
      out.println("<Br></FORM>");
      out.println("</BODY>");
      out.println("<p> Last updated: July 17, 2002 ");
      out.println("<br> <a HREF = \"mailto: sarahs@vt.edu\"> " +
          "Sarah Satterlee</a><br>");
      out.println("<a href=" + spbiccToolboxURL +
          " TARGET=\"_top\">Back to " +
          "SPBICC's Toolbox</a><br>");
      out.println("<a href=" + spbiccHomeURL +
          " TARGET=\"_top\">SPBICC home</a></HTML>");
   }//end printForm1
}//end PrintForm
```

B.3.2 TestBounds.java

```
/**TestBounds.java checks user input for errors and makes some simple
*   conversions before sending the data to the javahog simulator.
*/

package servletHelper;
import java.util.*;

/**checks user input for errors, ensures data is in the right format for
*   use by the javahog simulator
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,       8 July 2002
*/
public class TestBounds{

  /**creates a new object to test the user input for errors
  *@param a   a generic variable
  */
  public TestBounds(int a){
  }

  /**checks that duration of the simulation is within the bounds for
  *    which the simulation model has been validated -- between 1 and 92 days.
  *@param theDuration       the number of time steps in the simulation, in hours
  *@param errorString       the String containing all of the error messages about the user input
  *@return        the number of time steps in the simulation, in hours
  */
  public int durationBounds(int theDuration, StringBuffer errorString){
     if (theDuration < 1 || theDuration > 92){
        errorString.append("Error: the number of days for " +
           "simulation is not within range: 1-92. <br>");
     }//end if
     return theDuration;
  }//end durationBounds

  /**checks that the numbers of trees at different stages of SPB infestation
  *   sum to number of infested trees.
  *@param underAttack       the number of trees currently under SPB attack
  *@param withEggs        the number of trees currently containing SPB eggs
  *@param withImm        the number of trees currently containing SPB larvae
  *@param withBrood        the number of trees currently containing SPB brood adults
  *@param infested        the total number of infested trees
  *@param errorString        the String containing all of the error messages about the user input
  *@return       the total number of infested trees
  */
  public int infested(int underAttack, int withEggs, int withImm,
     int withBrood, int infested, StringBuffer errorString){
     int sumStages = underAttack + withEggs + withImm + withBrood;
     //if no trees are infested, then there can be no infestation simulation!
     if(infested < 1){
        errorString.append("Error: the number of total currently " +
           "infested trees must be at least 1. <br>");
        infested = 1;
```

```
    } else if((infested != sumStages) && (sumStages != 0)){
      errorString.append("Error: the number of infested trees does " +
           "not equal the sum of the trees under attack, with eggs, " +
        "with larvae or pupae, and with brood adults. For default " +
        "distribution of infested trees, enter zero for numbers of " +
        "trees under attack, with eggs, with larvae or pupae, " +
        "and with brood adults. <br>");
        infested = 1;
    }//end else if

    return infested;
  }//end infested

  /**checks that the value for loblolly proportion is between 0 and 1
  *@param ratio        the decimal proportion of loblolly in the stand
  *@param errorString        the String containing all of the error messages
  *                      about the user input
  *@return         the decimal proportion of loblolly in the stand
  */
  public float ratioBounds(float ratio, StringBuffer errorString){
     if(ratio <= 0 || ratio >1)
       errorString.append("Error: the loblolly proportion must be > 0 " +
         "and < 1. <br>");
    return ratio;
  }//end ratioBounds

  /**checks that the value for DBH is within bounds according to whether
  *       its units are metric or English
  *@param standAvgDBH     the mean diameter at breast height of all the
  *                          trees in the stand
  *@param dbhUnits       the units of standAvgDBH, either cm or inches
  *@param errorString       the String containing all of the error messages
  *                      about the user input
  *@return        the mean diameter at breast height of the stand, in cm
  */
  public float dBHbounds(float standAvgDBH, String dbhUnits,
    StringBuffer errorString){

    if(dbhUnits.equals("cm")){
       if(standAvgDBH < 12 || standAvgDBH > 77){
         errorString.append("Error: the stand average DBH is not " +
           "within range: 12-77 cm. <br>");
       }
    } else{
       if(standAvgDBH < 5 || standAvgDBH > 30){
         errorString.append("Error: the stand average DBH is not " +
           "within range: 5-30 inches. <br>");
       }//end if
         standAvgDBH *= 2.54;
    }//end else
    return standAvgDBH;
  }//end dBHbounds

  /**checks that the value for pineBA is within bounds according to whether
  *       its units are metric or English
  *@param pineBA       the pine basal area of the stand
```

```
*@param pBAunits       the units for pineBA, either sq.m/hectare or sq.ft/acre
*@param errorString       the String containing all of the error messages about the user input
*@return       the pine basal area of the stand, in sq.m/hectare
*/
public float pBAbounds(float pineBA, String pBAunits,
    StringBuffer errorString){

    if(pBAunits.equals("m")){
        if(pineBA < 2 || pineBA > 69){
            errorString.append("Error: the pine basal area is not within " +
                "range: 2-69 square meters per hectare. <br>");
        }
    }//end if
    else{
        if(pineBA < 10 || pineBA > 300){
            errorString.append("Error: the pine basal area is not within " +
                "range: 10-300 square feet per acre. <br>");
        }
        pineBA *= 0.229681;
    }//end else
    return pineBA;
}//end pBAbounds

/**checks that the value for hardwoodBA is within bounds, according
*       to whether its units are metric or English
*@param hardwoodBA       the hardwood basal area of the stand
*@param hBAunits       the units for hardwoodBA, either sq.m/hectare or sq.ft/acre
*@param errorString       the String containing all of the error messages about the user input
*@return       the hardwood basal area of the stand, in sq.m/hectare
*/
public float hBAbounds(float hardwoodBA, String hBAunits,
    StringBuffer errorString){

    if(hBAunits.equals("m")){
        if(hardwoodBA < 0 || hardwoodBA > 69){
            errorString.append("Error: the hardwood basal area is not " +
                "within range: 0-69 square meters per hectare. <br>");
        }
    } else{
        if(hardwoodBA < 0 || hardwoodBA > 300){
            errorString.append("Error: the hardwood basal area is not " +
                "within range: 0-300 square feet per acre. <br>");
        }
        hardwoodBA *= 0.229681;
    }//end else
    return hardwoodBA;
}//end hBAbounds

/**checks that the year is a reasonable value (although it is only
*       important in the leap year calculation)
*@param year    the year when the infestation occurred
*@param errorString       the String containing all of the error messages about the user input
*@return       the year when the infestation occurred
*/
public int yearBounds(int year, StringBuffer errorString){
    if(year > 3000 || year < 1500){
        errorString.append("Error: value for year is not " +
            "within range: 1500-3000. <br>");
    }//end if
    return year;
}//end yearBounds

/**gets approx. latitude and longitude for geographic centers of each state
*@param state       the geographic state where the SPB-infested spot is
*@param errorString       the String containing all of the error messages about the user input
*@return       approximate latitude and longitude for the geographic state
*/
public double[] estimateLatLong(char state, StringBuffer errorString){
    double latitude;
    double longitude;
    switch(state){
        case 'A':       latitude = 32.7;        //Alabama
                longitude = 86.5;       break;
        case 'R':       latitude = 33.3;        //Arkansas
                longitude = 92.6;       break;
        case 'F':       latitude = 31.6;        //Florida
                longitude = 82.5;       break;
        case 'G':       latitude = 32.8;        //Georgia
                longitude = 82.9;       break;
        case 'L':       latitude = 31.5;        //Louisiana
                longitude = 93.4;       break;
        case 'M':       latitude = 32.1;        //Mississippi
                longitude = 89.8;       break;
        case 'N':       latitude = 35.5;        //North Carolina
                longitude = 79.2;       break;
        case 'O':       latitude = 34.6;        //Oklahoma
                longitude = 94.8;       break;
        case 'S':       latitude = 34.1;        //South Carolina
                longitude = 80.1;       break;
        case 'T':       latitude = 35.2;        //Tennessee
                longitude = 86.7;       break;
        case 'X':       latitude = 30.7;        //Texas
                longitude = 95.0;       break;
        case 'V':       latitude = 37.8;        //Virginia
                longitude = 77.7;       break;
        default:        latitude = 35;          //shouldn't happen
            longitude = 81;
        errorString.append("Error finding latitude and longitude. <br>");
    }//end switch
    double[] location = {latitude, longitude};
    return location;
}//end estimateLatLong

/**gets Julian day for a day of the year within April and October
*@param month       the month of the beginning of the simulation
*@param day         the day of the beginning of the simulation
*@param year        the year of the beginning of the simulation
*@param errorString       the String containing all of the error messages about the user input
*@return       the Julian date of the beginning of the simulation (days since January 1)
*/
public int convertToJulian(int month, int day, int year,
    StringBuffer errorString){
        int jDay = 150; //julian days, default value
        //if month is Apr., June or Sept., there are only 30 days
        if((month == 4) || (month == 6) || (month == 9)){
            if(day == 31){
                errorString.append("Error: there are not 31 days in the " +
                    "chosen month. <br>");
            }
        }//end if

        /*find Julian day for the last day of the previous month and add to
            that the day of the current month*/
        switch(month){
            case 4:     jDay = 90 + day;    break;    //April
            case 5:     jDay = 120 + day;   break;    //May
            case 6:     jDay = 151 + day;   break;    //June
            case 7:     jDay = 181 + day;   break;    //July
            case 8:     jDay = 212 + day;   break;    //August
            case 9:     jDay = 243 + day;   break;    //Sept.
            case 10:    jDay = 273 + day;   break;    //Oct.
            default:    errorString.append("Error converting date to Julian" +
                            "<br>");
        }//end switch

        //if February had an extra day this year, add 1 day
        if(year % 4 == 0){
            if(year % 100 != 0 || year % 1000 == 0)
                jDay++;  //this is a leap year
            }//end if
        return jDay;
    }//end convertToJulian

    /**checks that the number of trees infested in any stage is within
    *       bounds for which the simulation has been validated
    *@param trees       number of trees in an infestation stage
    *@param errorString     the String containing all of the error messages
    *                   about the user input
    *@return        the number of trees in an infestation stage
    */
    public int treeBounds(int trees, StringBuffer errorString){
        if( trees > 9000){
            errorString.append("Error: the number of trees in any stage of " +
                "infestation should not exceed 9000. <br>");
        } else if(trees < 0){
            errorString.append("Error: numbers of trees in any stage of " +
                "infestation cannot be negative. <br>");
        }
        return trees;
    }//end treeBounds
}//end TestBounds
```

B.3.3 PrintTrees.java

```
/**PrintTrees.java prints the output of the javahog simulator as a
*   graph, showing the predicted numbers of dead and infested trees
*   over time.
*/

package servletHelper;
import javaSPB1.*;
import java.io.*;
import java.awt.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import Acme.JPM.Encoders.GifEncoder;
import javachart.chart.*;

/**prints numbers of dead and infested trees over time as a graph,
*   according to values predicted by the javahog simulation model
* @see <a href = http://java.apache.org/jserv/index.html> Apache JServ</a>
*@see <a href = http://www.acme.com/java/software/Package-Acme.JPM.Encoders.html>GifEncoder</a>
*@see <a href = http://www.ve.com>KavaChart</a>
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,        8 July 2002
*/
public class PrintTrees{

   /**creates a new object to print the numbers of dead and infested trees over time
    *@param a       generic variable
   */
   public PrintTrees(int a){
    }

   /**prints numbers of dead and infested trees over time as a graph, as predicted by the javahog model
    *@param sim           the Simulator to run the simulation
   *@param duration       the duration of the simulation, in hours
   *@param out           the servlet output stream, in this case, as a .gif
   *@param yesDead       is true if dead trees have been selected for graphing
   *@param yesInfested is true if infested trees have been selected for
    *                     graphing
   *@param spotID       the identification name of the infestation spot
   */
   public void printTrees(Simulator sim, int duration,
      ServletOutputStream out, boolean yesDead, boolean yesInfested, String spotID){

      Frame frame = null;
      Graphics g = null;
      int WIDTH = 380;
       int HEIGHT = 260;
      try{
         int yDepth = 0;
          int xDepth = 0;
          int topY = 0;
         boolean showDepth = true;
```

```
            boolean yesNo = true;
            frame = new Frame();
            frame.addNotify();
            Image image = frame.createImage((WIDTH - 25), (HEIGHT - 5));
            RotateString rotate = new RotateString(frame);
            Globals myGlobals = new Globals(topY, xDepth, yDepth, showDepth, rotate, image);
            Plotarea myPlot = new Plotarea(myGlobals);
            double[] blankData = new double[duration];
            double[] infestedData = new double[duration];
            double[] deadData = new double[duration];
            double[] dayData = new double[duration];
            String[] labels = new String[duration];
            for(int i = 0; i < duration; i++){
                infestedData[i] = sim.infestedElement(i);
                deadData[i] = sim.deadElement(i);
                dayData[i] = i;
                labels[i] = "";
                blankData[i] = 0;
            }//end for
            Dataset dead = new Dataset("Dead", dayData, deadData, labels, 1, myGlobals);
            Dataset infested = new Dataset("Infested", dayData, infestedData, labels, 2, myGlobals);
            Dataset blank = new Dataset("Blank", dayData, blankData, labels, 3, myGlobals);
            LineChart chart = new LineChart("Dead and Infested");
            if(!yesDead && !yesInfested){
                chart.addDataset(blank);
            } else{
                if(yesDead){
                    chart.addDataset(dead);
                }
                if(yesInfested){
                    chart.addDataset(infested);
                }
                chart.getBackground().setTitleFont(new Font("Serif", Font.PLAIN, 12));
                chart.getBackground().setTitleString(spotID + ", Dead & Infested Trees");
                chart.setLegendVisible(true);
                chart.getLegend().setLlX(0.4);
                chart.getLegend().setLlY(0.75);
                chart.getLegend().setIconHeight(0.04);
                chart.getLegend().setIconWidth(0.04);
                chart.getLegend().setIconGap(0.02);
                chart.getLegend().setVerticalLayout(false);
                chart.getXAxis().setTitleString("Day of Simulation");
                chart.getYAxis().setTitleString("Trees");
                chart.resize(WIDTH, HEIGHT);
                g = image.getGraphics();
                chart.drawGraph(g);
                GifEncoder encoder = new GifEncoder(image, out);
                encoder.encode();
            }//end else
        } catch(IOException e){
        } finally{
            if(g != null) g.dispose();
            if(frame != null) frame.removeNotify();
        }//end finally
    }//end printTrees
}//end PrintTrees
```

B.3.4 PrintPops.java

```
/**PrintPops.java prints the output of the javahog simulator as a
*   graph, showing the predicted populations of southern pine beetles
*   over time.
*/
package servletHelper;
import javaSPB1.*;
import java.io.*;
import java.awt.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import Acme.JPM.Encoders.GifEncoder;
import javachart.chart.*;

/**prints southern pine beetle populations over time as a graph,
*   according to values predicted by the javahog simulation model
* @see <a href = http://java.apache.org/jserv/index.html> Apache JServ</a>
*@see <a href = http://www.acme.com/java/software/Package-Acme.JPM.Encoders.html>GifEncoder</a>
*@see <a href = http://www.ve.com>KavaChart</a>
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,        8 July 2002
*/
public class PrintPops{

   /**creates a new object to print the southern pine beetle populations over time
    *@param a        generic variable
   */
   public PrintPops(int a){
    }

   /**prints southern pine beetle populations over time as a graph,
    *        as predicted by the javahog simulation model
    *@param sim           the Simulator to run the simulation
    *@param duration       the duration of the simulation, in hours
    *@param out           the servlet output stream, in this case, as a .gif
    *@param yesOldAttack    is true if initial attacking adults have been selected for graphing
    *@param yesAttack       is true if attacking adults have been selected for graphing
    *@param yesParent       is true if parent adults have been selected for graphing
    *@param yesEgg       is true if eggs have been selected for graphing
    *@param yesImm       is true if larvae have been selected for graphing
    *@param yesBrood       is true if brood adults have been selected for graphing
    *@param yesEmerge       is true if emerging adults have been selected for graphing
    *@param spotID       the identification name of the infestation spot
   */
   public void printPops(Simulator sim, int duration, ServletOutputStream out, int yesOldAttack,
      int yesAttack, int yesParent, int yesEgg, int yesImm, int yesBrood, int yesEmerge, String spotID){

      int yesAdults = (int)(yesOldAttack + yesAttack + yesParent + yesEmerge);
       int yesImmatures = (int)(yesEgg + yesImm + yesBrood);
       Frame frame = null;
       Graphics g = null;
      int WIDTH = 355;
```

```
 int HEIGHT = 260;
 try{
    Axis axisY;
    int yDepth = 0;
    int xDepth = 0;
    int topY = 0;
    boolean showDepth = true;
    boolean yesNo = true;
    boolean isXAxis;
    frame = new Frame();
    frame.addNotify();
    Image image = frame.createImage(WIDTH, HEIGHT);
    RotateString rotate = new RotateString(frame);
    Globals myGlobals = new Globals(topY, xDepth, yDepth, showDepth, rotate, image);
    Plotarea myPlot = new Plotarea(myGlobals);
    double[] oldAttackData = new double[duration];
    double[] attackData = new double[duration];
    double[] parentData = new double[duration];
    double[] eggData = new double[duration];
    double[] immData = new double[duration];
    double[] broodData = new double[duration];
    double[] emergeData = new double[duration];
    double[] dayData = new double[duration];
    double[] blankData = new double[duration];
    String[] labels = new String[duration];
    for(int t = 0; t < duration; t++){
       oldAttackData[t] = sim.beetleElement(0, t * 24);
       attackData[t] = sim.beetleElement(1, t * 24);
       parentData[t] = sim.beetleElement(2, t * 24);
       eggData[t] = sim.beetleElement(3, t * 24);
       immData[t] = sim.beetleElement(4, t * 24);
       broodData[t] = sim.beetleElement(5, t * 24);
       emergeData[t]= sim.beetleElement(6, t * 24);
       blankData[t] = 0;
       dayData[t] = t;
       labels[t] = "";
    }//end for
    LineChart chart = new LineChart("Adult Populations");
    LineChart chart2 = new LineChart("Immature Populations");
    Dataset oldAttack = new Dataset("Initial Attackers", dayData,
       oldAttackData, labels, 0, myGlobals);
    Dataset attack = new Dataset("Attackers", dayData, attackData, labels, 1, myGlobals);
    Dataset parent = new Dataset("Parents",dayData, parentData, labels, 2, myGlobals);
    Dataset egg  = new Dataset("Eggs", dayData, eggData, labels, 3, myGlobals);
    Dataset imm = new Dataset("Larvae", dayData, immData, labels, 4, myGlobals);
    Dataset brood = new Dataset("Brood", dayData, broodData,labels, 5, myGlobals );
    Dataset emerge = new Dataset("Emergers", dayData, emergeData, labels, 6, myGlobals);
    Dataset empty = new Dataset("", dayData, blankData, labels, 7, myGlobals);
    chart.addDataset(empty);
    chart2.addDataset(empty);

    if(yesImmatures != 0 || yesAdults != 0){
       if(yesOldAttack == 1) chart.addDataset(oldAttack);
       if(yesAttack == 1) chart.addDataset(attack);
       if(yesParent == 1) chart.addDataset(parent);
       if(yesBrood == 1) chart.addDataset(brood);
```

```
            if(yesEmerge == 1) chart.addDataset(emerge);
            if(yesEgg == 1) chart2.addDataset(egg);
            if(yesImm == 1) chart2.addDataset(imm);
         }//end if

         chart.getBackground().setTitleFont(new Font("Serif", Font.PLAIN, 12));
         chart.getBackground().setTitleString(spotID + ", Beetle Populations");
         chart.setLegendVisible(true);
         chart.getLegend().setLlX(0.20);
         chart.getLegend().setLlY(0.60);
         chart.getLegend().setIconHeight(0.06);
         chart.getLegend().setIconWidth(0.08);
         chart.getLegend().setIconGap(0.0000);
         chart.getLegend().setVerticalLayout(true);
         chart.getLegend().setBackgroundVisible(false);
         chart2.setLegendVisible(true);
         chart2.getLegend().setLlX(0.60);
         chart2.getLegend().setLlY(0.60);
         chart2.getLegend().setIconHeight(0.06);
         chart2.getLegend().setIconWidth(0.08);
         chart2.getLegend().setIconGap(0.0000);
         chart2.getLegend().setVerticalLayout(true);
         chart2.getLegend().setBackgroundVisible(false);
         chart2.getLegend().setLabelColor(Color.blue);
         Gc gc = new Gc(myGlobals);
         chart.getDatasets()[0].getGc().setLineColor(gc.TRANSPARENT);
         chart2.getDatasets()[0].getGc().setLineColor(gc.TRANSPARENT);
         isXAxis = false;
         axisY = new Axis(chart.getDatasets(), isXAxis, myPlot);
         chart2.getYAxis().setSide(axisY.RIGHT);
         chart2.getYAxis().setTitleString("Immatures");
         chart2.getYAxis().setLabelColor(Color.blue);
         chart2.getYAxis().setTitleColor(Color.blue);
         chart.getXAxis().setTitleString("Day of Simulation");
         chart.getYAxis().setTitleString("Adults");
         chart.resize((WIDTH), HEIGHT);
         g = image.getGraphics();
         chart2.resize((WIDTH), HEIGHT);
         //layering 2 charts on top of each other, set one transparent
         chart.getBackground().getGc().setFillColor(gc.TRANSPARENT);
         chart.getPlotarea().getGc().setFillColor(gc.TRANSPARENT);
         chart2.drawGraph(g);
         chart.drawGraph(g);

         GifEncoder encoder = new GifEncoder(image, out);
         encoder.encode();
      } catch(IOException e){
      } finally{
         if(g != null) g.dispose();
         if(frame != null) frame.removeNotify();
      }//end finally
   }//end printPops
}//end PrintPops
```

B.3.5 PrintTable.java

```
/**PrintTable.java prints the output of the javahog simulation as
*  an HTML table
*/

package servletHelper;
import javaSPB1.*;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**prints the simulation output as an HTML table
* @see <a href = http://java.apache.org/jserv/index.html> Apache JServ</a>
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,        8 July 2002
*/
public class PrintTable
{
   /**creates a new object to print the simulation output as a table
   *@param a   a generic variable
   */
   public PrintTable(int a){
   }

   /**prints the simulation output as an HTML table
   *@param req           provides user request information for the servlet
   *@param out           the output stream (in this case, text)
   *@param theSpotID       the name of the infestation spot for which the simulation is run
   *@param theMonth       the month the spot was investigated in the field
   *@param theDay       the day the spot was investigated in the field
   *@param theYear       the year the spot was investigated in the field
   *@param theState       the state in which the spot is located
   *@param deadTrees       the initial number of trees killed by SPB
   *@param infestedTrees   the initial number of trees infested with SPB
   *@param underAttack     the number of trees initially under SPB attack
   *@param withEggs           the number of trees initially containing SPB eggs
   *@param withImm       the number of trees initially containing SPB larvae
   *@param withBrood       the number of trees initially containing SPB adults
   *@param loblollyRatio   the decimal proportion of trees in the stand that are loblolly
   *@param standAvgDBH       the average diameter at breast height of the trees in the stand
   *@param theDBHunits the units of standAvgDBH, either cm or inches
   *@param pineBA       the pine basal area of the stand
   *@param thePBAunits       the units of pineBA, either sq.m/hectare or sq.ft/acre
   *@param hardwoodBA       the hardwood basal area of the stand
   *@param theHBAunits       the units of hardwoodBA, either sq.m/hectare or sq.ft/acre
   *@param theDuration       the length of the simulation, in hours
   *@param sim              the Simulator to run the simulation
   *@exception IOException an exception may occur reading from or printing to HTML
   */
   public void printTable(HttpServletRequest req, ServletOutputStream out,
      String theSpotID, int theMonth, int theDay, int theYear,
      char theState, int deadTrees, int infestedTrees, int underAttack,
      int withEggs, int withImm, int withBrood, float loblollyRatio,
      float standAvgDBH, String theDBHunits, float pineBA, String thePBAunits,
      float hardwoodBA, String theHBAunits, int theDuration, Simulator sim)
      throws IOException{

      HttpUtils hu = new HttpUtils();
      StringBuffer newPage = hu.getRequestURL(req);
      newPage.append("?"+ req.getQueryString());
      String newURL = newPage.toString();
      int i;
      String state;
      String pbaUnits;
      String hbaUnits;
      switch(theState){
         case 'A': state = "AL";      break;
         case 'R': state = "AR";      break;
         case 'F': state = "FL";      break;
         case 'G': state = "GA";      break;
         case 'L': state = "LA";      break;
         case 'M': state = "MS";      break;
         case 'N': state = "NC";      break;
         case 'O': state = "OK";      break;
         case 'S': state = "SC";      break;
         case 'T': state = "TN";      break;
            case 'X': state = "TX";      break;
         case 'V': state = "VA";      break;
         default: state = "?";
      }//end switch

      if(thePBAunits.equals("m")){
         pbaUnits = "sq.m/hectare";
      } else {
         pbaUnits = "sq.ft/acre";
      }
      if(theHBAunits.equals("m")){
         hbaUnits = "sq.m/hectare";
      } else{
         hbaUnits = "sq.ft/acre";
      }
      out.println("<HTML><TITLE>" + theSpotID +
         " tabular output</TITLE>" +
         "<BODY BGCOLOR=\"#FFFFFF\" LINK=\"#007000\" VLINK=\"#009000\"> ");
      out.println("<H3><font color = \"#007000\">Model input</font></H3>");
      out.println("<table border=2 bordercolor = \"#007000\"><tr>" +
         "<td bgcolor=\"88ee88\" >");
      out.println("ID </td><td>" + theSpotID +
         "</td><td bgcolor=\"88ee88\">");
      out.println("Date </td><td>" + theMonth + "/" + theDay + "/" + theYear +
         "</td></tr><tr><td bgcolor=\"88ee88\">");
      out.println("Killed trees </td><td>" + deadTrees);
      out.println("</td><td bgcolor=\"88ee88\"> State </td><td>" + state +
         "</td></tr><tr><td bgcolor=\"88ee88\">");
      out.println("Infested trees </td><td>" + infestedTrees);
      out.println("</td><td bgcolor=\"88ee88\">Loblolly ratio </td><td>" +
         loblollyRatio);
      out.println("</td></tr><tr><td bgcolor=\"88ee88\">Under attack " +
            "</td><td>" + underAttack);
        out.println("</td><td bgcolor=\"88ee88\">DBH</td><td>" + standAvgDBH +
            " " + theDBHunits);
        out.println("</td></tr><tr><td bgcolor=\"88ee88\">With eggs </td><td>" +
             withEggs);
        out.println("</td><td bgcolor=\"88ee88\"> Pine BA </td><td>" + pineBA +
            " " + pbaUnits + "</td></tr>");
        out.println("<tr><td bgcolor=\"88ee88\">With larvae</td><td>" +
            withImm);
        out.println("</td><td bgcolor=\"88ee88\">Hardwood BA</td><td>" +
            hardwoodBA + " " + hbaUnits);
        out.println("</td></tr><tr><td bgcolor=\"88ee88\">With brood</td><td>" +
               withBrood);
        out.println("</td><td bgcolor=\"88ee88\">Days</td><td>" + theDuration +
            "</td></tr></table>");

        out.println("<H3><font color = \"#007000\">Model output</font></H3>");
        out.println("<table border=2 bordercolor = \"#007000\"><tr>" +
            "<td bgcolor=\"88ee88\">");
        out.println("Day</td><td bgcolor=\"88ee88\">Killed Trees</td>" +
            "<td bgcolor=\"88ee88\">Infested Trees");
        out.println("</td><td bgcolor=\"88ee88\">Initial Attackers</td>" +
            "<td bgcolor=\"88ee88\">Attackers");
        out.println("</td><td bgcolor=\"88ee88\">Parents</td>" +
            "<td bgcolor=\"88ee88\">Eggs</td><td bgcolor=\"88ee88\">Larvae");
        out.println("</td><td bgcolor=\"88ee88\">Brood</td>" +
            "<td bgcolor=\"88ee88\">Emergers</td></tr>");

        for(int t = 0; t < (theDuration + 1); t++){
            out.println("<tr><td>" + t + "</td><td>");
            out.println(sim.deadElement(t) + "</td><td>");
            out.println(sim.infestedElement(t) + "</td>");
            for(i = 0; i < 7; i++){
                out.println("<td>"+ sim.beetleElement(i,t) + "</td>");
            }
            out.println("</tr>");
        }//end for
        out.println("</table></BODY>" );
        out.println("<h3>" +
            "<a href=\"http://ultra.isis.vt.edu/~sarah/javahog.html\">" +
            "Back to Javahog</a></h3></HTML>");
    }//end printTable
}//end PrintTable
```

B.3.6 Delimited.java

```
/**Delimited.java prints the output of the javahog simulator as comma-
 *   delimited text.
 */
package servletHelper;
import javaSPB1.*;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** prints simulation output as comma-delimited text
* @see <a href = http://java.apache.org/jserv/index.html> Apache JServ</a>
* @author Sarah Satterlee <br> Copyright Virginia Tech Entomology <br>
*    All rights reserved
* @version 1.01,        8 July 2002
*/
public class Delimited{

   /**creates a new Delimited, to print output as comma-delimited text*/
   public Delimited(int a){
   }

   /**prints the simulation output as lines of comma-delimited text
   *@param req           provides user request information for the servlet
   *@param out           the output stream (in this case, text)
   *@param theSpotID       the name of the infestation spot for which the simulation is run
   *@param theMonth       the month the spot was investigated in the field
   *@param theDay       the day the spot was investigated in the field
   *@param theYear       the year the spot was investigated in the field
   *@param theState       the state in which the spot is located
   *@param deadTrees       the initial number of trees killed by SPB
   *@param infestedTrees   the initial number of trees infested with SPB
   *@param underAttack     the number of trees initially under SPB attack
   *@param withEggs           the number of trees initially containing SPB eggs
   *@param withImm       the number of trees initially containing SPB larvae
   *@param withBrood       the number of trees initially containing SPB brood adults
   *@param loblollyRatio   the decimal proportion of trees in the stand that are loblolly
   *@param standAvgDBH       the average diameter at breast height of the trees in the stand
   *@param theDBHunits the units of standAvgDBH, either cm or inches
   *@param pineBA       the pine basal area of the stand
   *@param thePBAunits       the units of pineBA, either sq.m/hectare or sq.ft/acre
   *@param hardwoodBA       the hardwood basal area of the stand
   *@param theHBAunits       the units of hardwoodBA, either sq.m/hectare sq.ft/acre
   *@param theDuration       the length of the simulation, in hours
   *@param sim               the Simulator to run the simulation
   */
   public void delimited(HttpServletRequest req, ServletOutputStream out,
      String theSpotID, int theMonth, int theDay, int theYear, char theState,
      int deadTrees, int infestedTrees, int underAttack, int withEggs,
      int withImm, int withBrood, float loblollyRatio, float standAvgDBH,
      String theDBHunits, float pineBA, String thePBAunits, float hardwoodBA,
      String theHBAunits, int theDuration, Simulator sim) throws IOException{
```

```
HttpUtils hu = new HttpUtils();
StringBuffer newPage = hu.getRequestURL(req);
newPage.append("?"+ req.getQueryString());
String newURL = newPage.toString();
int i;
String state;
String pbaUnits;
String hbaUnits;
switch(theState){
   case 'A': state = "AL";       break;
   case 'R': state = "AR";       break;
   case 'F': state = "FL";       break;
   case 'G': state = "GA";       break;
   case 'L': state = "LA";       break;
   case 'M': state = "MS";       break;
   case 'N': state = "NC";       break;
   case 'O': state = "OK";       break;
   case 'S': state = "SC";       break;
   case 'T': state = "TN";       break;
   case 'X': state = "TX";       break;
   case 'V': state = "VA";       break;
   default: state = "?";
}//end switch

if(thePBAunits.equals("m")){
   pbaUnits = "sq.m/hectare";
} else {
   pbaUnits = "sq.ft/acre";
}
if(theHBAunits.equals("m")){
   hbaUnits = "sq.m/hectare";
} else {
   hbaUnits = "sq.ft/acre";
}

out.println("<HTML><TITLE>" + theSpotID +
   " comma-delimited output</TITLE>" +
   "<BODY BGCOLOR=\"#FFFFFF\" LINK=\"#007000\" VLINK=\"#009000\"> ");
out.println("<H3><font color=\"#007000\"> You can copy this text, " +
   "and paste it into your favorite spreadsheet program.</H3></font>");
out.println("Model input <br>");
out.print("ID," + theSpotID);
out.print(",Date," + theMonth + "/" + theDay + "/" + theYear);
out.println("<br>Killed trees," + deadTrees);
out.print(",State," + state);
out.println("<br>Infested trees," + infestedTrees);
out.print(",Loblolly ratio," + loblollyRatio);
out.println("<br>Under attack,"+ underAttack);
out.print(",DBH," + standAvgDBH + " " + theDBHunits);
out.println("<br>With eggs," + withEggs);
out.print(",Pine BA," + pineBA + " " + pbaUnits);
out.println("<br>With larvae," + withImm);
out.print(",Hardwood BA," + hardwoodBA + " " + hbaUnits);
out.println("<br>With brood," + withBrood);
out.print(",Days," + theDuration);
out.println("<br>Model output");
```

```
        out.print("<br>Day,Killed Trees,Infested Trees,Initial Attackers,");
        out.print("Attackers,Parents,Eggs,Larvae,Brood,Emergers");

        for(int t = 0; t < (theDuration + 1); t++){
            out.println("<br>" + t + ",");
            out.print(sim.deadElement(t) + ",");
            out.print(sim.infestedElement(t) + ",");
            for(i = 0; i < 7; i++){
                out.print(sim.beetleElement(i,t) + ",");
            }
        }//end for
        out.println("</BODY><h3> " +
            "<a href=\"http://ultra.isis.vt.edu/~sarah/javahog.html\"> " +
            "Back to Javahog</a></h3></HTML>");
    }//end delimited
}//end Delimited
```

Vita

Sarah Melissa Satterlee was born in 1979 in East Lansing, Michigan. She has lived in half a dozen states across the country, but she now calls Virginia home.

Sarah attended Nandua High School on the Eastern Shore of Virginia. She received her B.S. from The College of William & Mary in chemistry and biology, achieving honors in biology for her thesis entitled "The chemical attraction of mites to their passalid beetle hosts."

Sarah and her husband James have recently moved into their first house, in Blacksburg, Virginia.

Address:	**Phone:**	**Email:**
503 Woodbine Drive Blacksburg, VA 24060	540-552-6131	smsatt@cs.com

Degrees:

B.S. with Honors in Biology, B.S in Chemistry
May 2000, The College of William and Mary, Williamsburg, VA 23187

M.S. in Entomology
December 2002, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061

Recent Employment:

Research Assistant. Supervisor: Nicholas Stone, Associate Professor. Jun '00-present.
Virginia Tech, Department of Entomology. 216A Price Hall, Blacksburg VA 24061. 540-231-4010.
Nicholas Stone is currently the Director of Education, International Institute of Information Technology, Virginia Tech, 1101 King Street, Suite 610, Alexandria, VA 22314. 703-518-8066.
Responsibilities: Reconstruction of an old Fortran population dynamics model into a new Java model. Placement of the new Java model online via a servlet. Optimization of the model using genetic algorithms.

Research Assistant. Supervisor: Carey Bagdassarian, Assistant Professor. 757-221-2556. Jun '99-Jul '99
The College of William & Mary, Department of Chemistry. P.O. Box 8795, Williamsburg, VA 23186.
Responsibilities: Construction of a population dynamics simulation model in Fortran.

Research Assistant. Supervisor: David Kranbuehl, Professor. 757-221-2542. May '97-Aug '97
The College of William & Mary, Department of Chemistry. P.O. Box 8795, Williamsburg, VA 23186.
Responsibilities: Data entry and management, operation of chemical analysis equipment.

Publications:

Broeckling, C.D., M.E. McClanan, S.M. Satterlee, and K.L. Tabor. *In review. Bacillus thuringiensis* (*Bt*) Resistance Management Regulations are Inadequate. A student debate article. American Entomologist.

Satterlee, S.M. 2000. The chemical attraction of mites to their passalid beetle hosts. An honors thesis. The College of William & Mary.

Presentations:

Satterlee, S.M. 2001. Evolution of SPBMODEL: from FORTRAN to genetic algorithms. Southern Forest Insect Work Conference. Jekyll Island, GA. July 2001.

Satterlee, S.M. 2001. Evolution of the Hog Model: from FORTRAN to genetic algorithms. Entomological Society of America National Meeting. San Diego, CA. December 2001.

Satterlee, S.M. 2002. The Evolving Hog Model. Southern Forest Insect Work Conference. Roanoke, VA. July 2002.

Satterlee, S.M. and N.J. Fashing. 2002. A preliminary investigation of cues used by mesostigmatic mites to locate their passalid beetle hosts. XI International Congress of Acarology. Mérida, Yucatán, México. September 2002.

Professional Affiliations: Entomological Society of America, W.B. Alwood Entomological Society

Recognition: Monroe Scholar, National Merit Scholar