Object-oriented Simulation Model of Rangeland Grasshopper Population Dynamics

James S. Berry[1], Gary Belovsky[2], Anthony Joern[3], William P. Kemp[4], Jerome Onsager[4]

[1]USDA-APHIS-PPQ, c/o Rangeland Insect Lab, Bozeman, MT 59717
[2]Dept. of Fisheries and Wildlife, Utah State University, Logan, UT 84322
[3]School of Biological Sciences, University of Nebraska, Lincoln, NE 68588
[4]USDA-ARS, Rangeland Insect Lab, Bozeman, MT 59717

Abstract

We are developing an object-oriented simulation model of rangeland grasshopper population dynamics. Individual objects in the model are aggregated to the landscape level. The model can simulate a community of any number of grasshopper species acting autonomously. In addition, spatial and temporal dynamics are easily included using object-oriented programming techniques.

Introduction

Over 130 species of grasshoppers inhabit rangeland ecosystems in the western United States. Each grasshopper species and its associated nymphal stages may have unique development, mortality and feeding rates, behavior, and food preferences. In addition, rangelands are not homogeneous and grasshoppers may move locally between habitats in response to environmental conditions. Some predators of grasshoppers, such as birds, are highly mobile and may search diverse habitats spread over relatively large areas. Grasshopper diseases affect species and life stages differently. Also, their impact on a grasshopper community depends on inoculum spreading over distance. A computer model that can simulate this complexity may allow us to better understand rangeland grasshopper community ecology and develop innovative strategies for managing rangeland grasshoppers.

In the past, computer simulations of rangeland grasshoppers have used cohort development algorithms [4, 5, 6], difference equations [7], energy flow models [3, 10] and other procedural-based programming techniques [1]. All of these techniques are severely limited for simultaneously simulating spatial relationships and characteristics of multiple species.

To overcome the weakness of these traditional modeling techniques, some researchers have turned to the object-oriented programming paradigm (OOP). For example, Stone [12] simulated a simple insect predator/prey system where individuals were modeled as autonomous objects. A similar approach has been used for moose/habitat interactions [11]. Olson & Wagner [8] used OOP to simulate an insect community on cotton. Plant & Stone [9] provide a general description of object-oriented modeling. OOP allows objects (insects, plants, etc.) to function uniquely and independently in a simulation. Therefore, complex systems can be modeled using OOP by initializing many types of objects, and modified copies of similar objects and then permitting each object to respond uniquely to current conditions in the model. The modeler does not need to generalize system behavior at the population level.

In this paper we describe the use of OOP to build an ecosystem model (GHSim) that focuses on rangeland grasshoppers and related biotic and abiotic components. We describe important data structures and techniques that facilitate ecosystem modeling using an OOP language. Also, we describe the objects in the model (e.g., grasshopper, predator) and how they can interact.

Methods

GHSim is written in Borland Pascal 7.0 (Borland International, Scotts Valley, Calif.) which provides object-oriented extensions to Pascal such as encapsulation (data and procedures can both be contained in an object) and inheritance (inheritance allows common object behaviors, procedures, and attributes to be defined at or near the top of object hierarchies). For example, the basic attributes of an insect such as six legs, and compound eyes could be defined in an object called Insect. A second object might be Beetle. Beetle would be declared as a type of Insect. Therefore, Beetle, without additional programming, would then share the attributes defined in Insect. Inheritance is

very useful for modeling complex systems where general classes of objects can be defined. More specific subclasses of objects then can inherit most attributes, and programming is needed only to refine or add to the inherited suite of attributes and procedures.

Data structures and many useful routines in GHSim are provided by a third party programming library (Object Professional, Turbo Power Software, Colorado Springs, Colo.). The most important data structure is a singly-linked list (SLL). A SLL can serve as a container object that can store essentially unlimited numbers and types of objects (similar to a Bag in Smalltalk). For example, an object called Population may be a SLL that contains all grasshopper objects at a site. Any object in the model that is defined as a SLL node can be stored in a SLL. In reality the object itself is not stored, but rather, a pointer to that object. Objects in a SLL are not sorted nor indexed. However, there may be occasions when a specific object must be found and used in the model, and an index to objects may be important. For these type of situations GHSim also maintains an index of the objects by using a string dictionary supplied in the Object Professional library. A string (e.g., "grasshopper1") may be associated with a specific grasshopper object. That grasshopper object can be accessed again by supplying the index "grasshopper1" to the string dictionary.

To provide attributes and functionality for objects in the model, we have developed a top object called simObj to serve as a template for all other objects simulated in GHSim. The top object, simObj is a descendent of Object Professional's SingleListNode object, and thus inherits the attributes of a SLL node. These attributes are required for any object to be stored in a SLL. Therefore, simObj and all of its descendants can be stored in a SLL.

Some uniformity of objects' variable and procedure names is required to simplify the programming and keep objects autonomous. Therefore, simObj has key variables and procedures defined. The variables can have unique values, and procedures can be redefined in the descendent objects. simObj defines two variables:

objID	:	integer,
cohortSize	:	real.

objID can be a unique identifier for an object and can be converted to a string and use as an index to the object in a string dictionary. GHSim can use cohortSize to model individuals (cohortSize = 1) or cohorts of individuals (cohortSize > 1). Four methods (procedures) are defined in simObj:

Init
Initializes the SLL node, and can be customized by descendent objects to initialize any variables in the object

Act
This procedure is called for each object during each iteration of the simulation. Because every object has the procedure **Act**, the program can traverse a SLL, access each object (regardless of object type and unknown to the program) and call its **Act** procedure. Each object type may have a completely different **Act** procedure which provides for autonomous and unique behavior for each object. In simObj, **Act** is empty and serves only as a template for descendent objects.

Display
Prints the object's current state to the screen

Print(*toFile*)
Prints the object's current state to disk file: *toFile*

Each descendent object type must have these procedures defined to provide the functionality intended for that object. These are the only procedures called in a simulation control loop where object is by design unknown and does not need to be known.

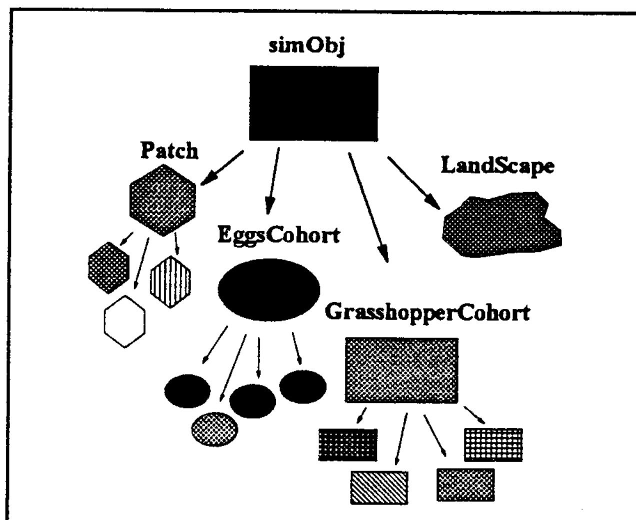

Figure 1. Object hierarchy for GHSim.

Currently, there are only five object types defined in GHSim and each of these are descendants of simObj: **LandScape**, **Patch**, **GrasshopperCohort**, **EggsCohort**, (Fig. 1) and **Predator**. As development progresses these object types will serve as ancestors to more specific types. These objects are the core of the simulation and will be described in detail. Objects will be in bold face.

The overall structure of the model is shown in Fig. 2.

The **LandScape** contains all the **Patches**. Each **Patch** produces food for the grasshoppers, contains **GrasshopperCohorts** and **EggsCohorts**, and calls the Act procedure each day of the simulation for every **GrasshopperCohort** and **EggsCohort** in the **Patch**. The main program is very simple. A **LandScape** is initialized, which will initialize all other objects is the model. Then **LandScape**'s Act procedure is called to run the simulation. Finally, the **LandScape**'s done procedure is called to release all memory and destroy all existing objects.

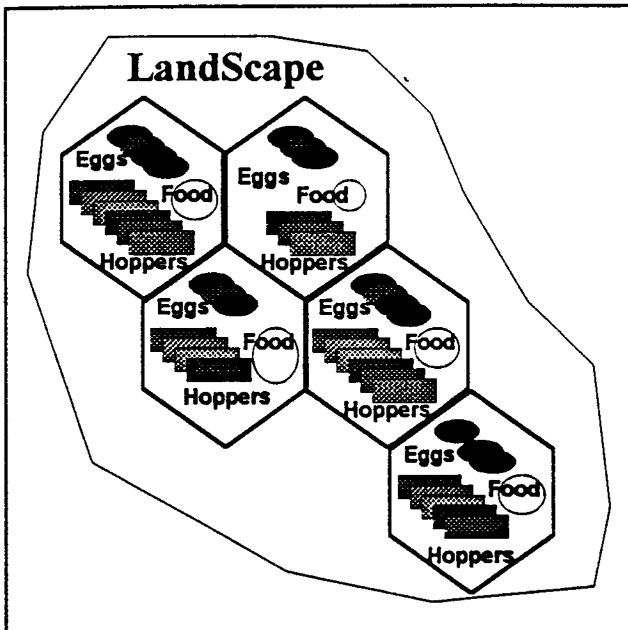

Figure 2. Overall model structure showing the relationship of objects in the system.

Object: LandScape

A **LandScape** is considered to be the smallest area that is essentially a closed system for the purpose of understanding grasshopper ecology. Therefore, a **LandScape** could represent a large valley, where grasshoppers and predators neither immigrate nor emigrate.

Variables:

- PatchesPtr
  - a SLL that contains all **Patches** (habitat sites) in the LandScape
- PatchLocDic
  - a string dictionary used to index all **Patches** in the **LandScape** by x,y coordinate location
- Precip
  - real type variable for total precipitation for the current day
- Temperature
  - real type variable for average temperature for the current day

Methods:

- init(*infile*)
  - does preliminary setup, opens output files, and initializes all objects in the **LandScape** for the first day of the simulation, may do initialization from file: *infile*
- Act
  - controls the daily loop for the simulation, calls Act for each **Patch** in the PatchesPtr list, sets the amount of grasshopper food (vegetation) in each **Patch** (in the future each **Patch** will simulate its own food growth)
- Display
  - not yet implemented, may be used to call Display for all **Patches**, or summaries and displays data from the **Patches**
- Print
  - not yet implemented, may be used to call Print for all **Patches**, or summarizes and prints data from the **Patches**
- Done
  - calls Done for each **Patch**, calls Done for PatchLocDict to dispose of its memory, closes the output files

Object: Patch

The **Patch** is defined a hexagon that contains only one grasshopper habitat type (i.e, the plant community does not change within the **Patch** borders). All **Patches** are the same size and, because of the hexagon shape, can be packed together without gaps. To create larger areas of identical habitat, several **Patches** that have the same habitat can be aggregated together. Grasshoppers reside within a **Patch**. However, they may migrate from one **Patch** to another.

Variables:

- eggPopulation
  - a SLL that contains all of the grasshopper egg objects (**EggsCohorts**) in the **Patch**
- population
  - a SLL than contains all of the motile grasshopper objects in the **Patch**
- Habitat
  - a string descriptor of the Habitat type of the **Patch**, not currently used
- XLoc, YLoc
  - location coordinates of the **Patch** within the

LandScape

totalFoodReq

total food required (mg) for all grasshoppers in the **Patch** for the current day

eggPopSize

sum of number of grasshopper eggs in all **EggsCohorts** in the **Patch**

food

amount of food for grasshoppers (mg) currently available in the **Patch**

Methods:

init(*IDNumber*)

calls simObj.init, sets objId = *IDNumber*, and creates and initializes eggPopulation and population

Display

prints to the screen summary information about the **Patch**

Print(*toFile*)

prints to *toFile* summary information about the **Patch**

AddEggs(*eggs*)

creates a new grasshopper egg cohort (**EggsCohort**) with *eggs* in the cohort, eggPopSize is increased by *eggs*

AddGrasshopperCohort(*aCohort*)

appends an existing grasshopper cohort object (*aCohort*) to population

purgeCohorts(*aPopPtr*)

deletes any empty grasshopper cohorts (cohorts whose size is equal to zero) in population (*aPopPtr*)

SortByBodyMass

sorts population by bodyMass so that the cohorts with the largest individuals are at the head of the list (population), important for allocating resources and dealing with competition in other parts of model

foodAvail

a function that returns the amount of food (mg) currently available in the **Patch** for grasshoppers

removeFood(*foodConsumed*)

decreases food in the **Patch** by *foodConsumed*, this may result in food being negative and is used in GrasshopperCohort.Act to impose competition and weight loss

Act

purgeCohorts is called to delete any empty cohorts from population, totalFoodReq (mg) is calculated by summing getFoodNeed for n, removeFood is called to remove totalFoodReq from food, each **GrasshopperCohort** in population is called to Act, lastly each **EggsCohort** in eggPopulation is called to Act

Done

deletes from memory all objects in population and eggPopulation

Object: Predator

The **Predator** object is not implemented yet, but will be first used for insectivorous birds. A **Predator** will probably reside in the **LandScape** with a **Patch** as a "home" base. Birds are highly mobile and will be allowed to move and forage in many **Patches** each day.

Object: GrasshopperCohort

The **GrasshopperCohort** is the most important and best defined object in GHSim. **GrasshopperCohort** provides methods for grasshopper growth, phenological development, feeding, starvation and weight loss, food consumption, egg resorption by the females when food is insufficient, reproduction, and movement from **Patch** to **Patch**. Competition for food results from the current **Patch** reporting negative food for the current day. The potential effects of competition are reduced reproduction, egg resorption by gravid females, weight loss, and starvation.

Variables:

podSize

parameter, initial number of eggs in a newly forming egg pod in a female grasshopper, may be species-specific

minPodSize

parameter, minimum number of eggs in forming egg pod before the female will resorb all remaining eggs and potentially try to start a new pod

eggSize

current mass of the individual eggs in a forming pod (mg)

age

age (days) of the cohort since hatching

minEggSize

parameter, mass (mg) of an egg when ready to be laid as part of a pod

daysLoss

number of consecutive days of weight loss for the cohort

massLoss

cumulative mass loss/individual for the cohort (mg)

eggWtLaid
: mass of an egg when it is laid (mg)

eggWtHatch
: mass of an egg at hatch (mg)

bodyMass
: fresh weight body mass (mg)/individual

maxBodyMass
: maximum bodyMass attained at any time during the simulation for the cohort

foodForGain
: surplus food (mg)/individual, amount beyond that needed for maintenance, can be used for growth and reproduction

cGrowth
: parameter, conversion factor used to convert food to grasshopper mass (0.05)

cEggs
: parameter, conversion factor used to convert food to grasshopper egg mass (0.1)

adultStage
: parameter, life stage number to indicate adult (6.0)

reproStage
: parameter, life stage number for stage at first reproduction (6.2)

stage
: current life stage

excessFood
: amount of food beyond that needed by the **GrasshopperCohort** for growth or maintenance, if negative grasshoppers lose weight

sexRatio
: parameter, to determine the how many new eggs are male or female (0.5)

gender
: male or female

currentPatch
: a pointer to the **Patch** where the **GrasshopperCohort** resides, provides access to information about the **Patch** (i.e., location, foodAvail)

Methods

init(*IDNumber, inPatch, Mass, Size*)
: sets: objID = *IDNumber*, currentPatch = *inPatch*, bodyMass = *Mass*, cohortSize = *Size*

Display
: prints to the screen summary information about the **GrasshopperCohort**

Print(*toFile*)
: prints to *toFile* summary information about the **GrasshopperCohort**

DoReproduction
: eggs in a reproductive female will increase in size:

```
eggSize = eggSize +
    (excessFood/eggNumber) * cEggs,
```

then currentPatch.removeFood(excessFood * cohortSize) is called; an egg pod will be laid when eggSize is greater than minEggSize, eggSize is set to zero, and eggNumber is then set to podSize to start a new egg pod

GetFoodNeed
: a function that calculates the total food required by the cohort/d (mg) as a function of foodReq and maxFoodIntake

calcStage
: a function that calculates the life stage number as a function of bodyMass

foodReq
: a function that calculates the food requirements/d/individual as a function of maxBodyMass (mg)

maxFoodIntake
: a function that calculates the maximum food that can be consumed/d/individual as a function of maxBodyMass, later the effect of temperature can be included

Act
: This procedure controls the basic life functions and actions of a grasshopper for a single day. Most of the Pascal code is included here to show the detail. The grasshoppers are called by their **Patch** to Act in order from those with the greatest bodyMass to least.

```
age := age + 1;

{ Do phenological development as a function of food availability.
Therefore, calculate stage as a function of bodyMass. }
    stage := calcStage;

{ Keep track of largest bodyMass attained. Used later for
starvation death and potential for weight gain. }
 if maxBodyMass < bodyMass then
      maxBodyMass := bodyMass;

{ Set excessFood to food in Patch after basic metabolic needs
for food are met and that amount of food has been removed from
the patch. This occurs in Patch.Act. If there was not enough food
in then patch for basic metabolic needs the Patch.food and
Patch.foodAvail will be negative until next day's food growth.
excessFood will be negative and negative growth (wt loss) or egg
resorption will occur.}
excessFood :=
      currentPatch^.foodAvail/cohortSize;
```

```
{ If excessFood < 0 then hoppers will lose wt. They should
lose weight in proportion to their demand or need for food. Per
capita shortfall for food for this cohort. }
  if excessFood < 0.0 then
    excessFood := excessFood *
      (GetFoodNeed/currentPatch^.totalFoodReq);

{ GetfoodNeed already eaten and removed from Patch, by the
Patch. Hoppers cannot eat all available food, only up to
maxFoodIntake. If maxFoodIntake < foodReq the excessFood
will be negative and hoppers will lose weight or resorb eggs.}
if excessFood > 0.0 then
      excessFood := (maxFoodIntake - foodReq);

{Grow the immature stages. Adults do not grow.}
if (stage < reproStage) then
begin
   bodyMass := bodyMass +  (excessFood *
      cGrowth);

   {Hoppers eat the food required for growth. If excessFood is
   negative then Hoppers lose weight and no food consumed.}
 if excessFood > 0 then
      currentPatch^.removeFood(excessFood *
           cohortSize);
end;

{ Adult females can regain weight lost or weight loss is
prevented by resorbing eggs. }
if (Gender = female) and (stage >= reproStage)
   and (excessFood < 0) and (eggSize > 0) then
   begin
   {calculate the number of eggs that must be resorbed to
   maintain bodyMass }
   {add 0.49 so that all fractional eggs round up to one}
   resorpNum := round(abs(excessFood) *
      (cEggs/EggSize) + 0.49);
   if resorpNum > eggNumber then
          resorpNum := eggNumber;

   {convert eggs back to food equivalents }
   excessFood := excessFood + ((EggSize/cEggs) *
      resorpNum);
  eggNumber := eggNumber - resorpNum;

   { Female will resorb rest of pod if number of eggs left is
   less than minPodSize. }
   if eggNumber < minPodsize then
    begin
     excessFood := excessFood + ((EggSize/cEggs)
                   * eggNumber);
     { Reset the eggs to start a new clutch. }
     eggSize := 0;
     eggNumber := PodSize;
    end;
 end;  { end of female weight gain and egg resorption }

{ Adults can only grow if they need to regain weight (i.e., they
have lost weight ).}
if (stage >= adultStage) and (bodyMass <
   maxBodyMass) and (excessFood > 0) then
     begin
      foodForGain := (1/cGrowth) *
           (maxBodyMass - bodyMass);

      { If true: Not enough intake for total recovery from wt
      loss; if false: enough intake for total recovery and
      maybe also reproduction. }
      if foodForGain > excessFood then
           foodForGain := excessFood;

      bodyMass := bodyMass + (foodForGain *
           cGrowth);

      { Hoppers eat the food required for wt gain. }
      currentPatch^.removeFood(foodForGain
           *cohortSize);

      { Some food is left for reproduction: egg production. }
      excessFood := excessFood - foodForGain;
end;  Adults gain weight }

{ Adults lose weight because excessFood is negative. }
if (stage >= adultStage) and (excessFood < 0)
  then
    begin
      bodyMass := bodyMass + (excessFood *
          cGrowth);
    end;

{ Keep track of mass loss and for how many consecutive days.
Used later for starvation. }
if excessFood < 0 then
      begin
        daysLoss := daysLoss + 1;
        massLoss := maxBodyMass - BodyMass;
      end
     else
      begin
        daysLoss := 0;
        massLoss := maxBodyMass - BodyMass;
      end;

{ Impose starvation if more than 30% of body mass has been
lost. The Patch will delete any grasshopper cohorts that are
empty. }
if (massLoss/maxBodyMass) > 0.30 then
  cohortSize := 0;

DoReproduction;
if bodyMass < 0.0 then bodyMass := 0.0;
```

Object: EggsCohort

Currently, **EggsCohort** only has the features in inherited from simObj. No other functionality has been implemented. In the future, overwinter mortality and spring egg hatch may be added.

Results

GHSim (58 kb for the executable code) currently simulates any number of **Patches** within a **LandScape** (54 bytes). The only limitation on the number of objects in the model is available memory. The virtual memory capabilities of OS/2 2.1 (International Business Machines Corp., Armonk, New York) and Micorosoft Windows (Microsoft Corp., Seattle, Wash.) allow GHSim to be limited only by disk space on 80386 or more sophisticated microcomputers. Each **Patch** (312 bytes) may contain any number of **GrasshopperCohorts** (127 bytes) and **EggsCohorts** (14 bytes). Each **GrasshopperCohort** and **EggsCohort** is unique and can, therefore, represent any grasshopper species.

Fig. 3a shows the population dynamics of seven **GrasshopperCohorts** in a single **Patch** where food becomes limited and competition for food causes some of the grasshoppers to starve. Grasshoppers with larger body

sizes will win a competitive interaction with smaller grasshoppers. However, Fig. 3a shows that the smaller grasshoppers can eat enough to hinder growth and reproduction by the larger grasshoppers when food is limited (Fig. 3b). This feature is especially apparent in the last **GrasshopperCohort** which exhibits slow growth and cannot enter the reproductive stage until the five smaller **GrasshopperCohorts** starve and are removed from the system.

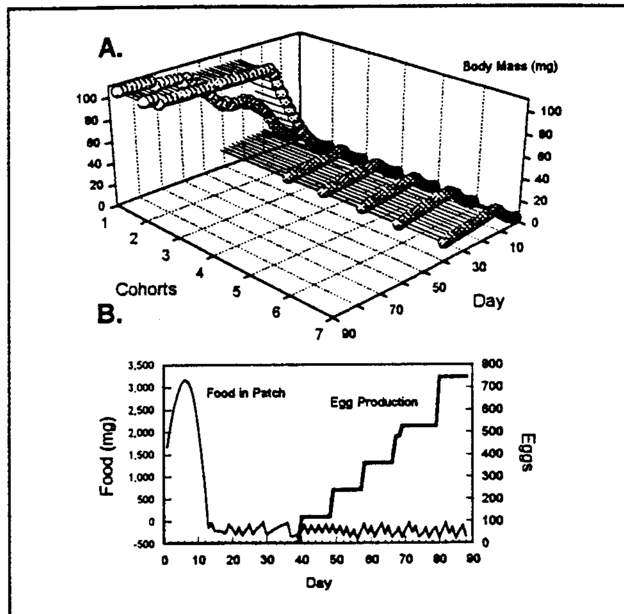

Figure 3. A) Simulation of 7 grasshopper cohorts in a single patch, and B) food and grasshopper eggs.

Discussion

OOP techniques are very useful for modeling complex ecological systems. The programming required to create systems that can contain multiple species and features is relatively simple. However, as a result of creating more complex models, two main problems have surfaced. First, because the model is complex (multiple patches, each with multiple cohorts and species of grasshoppers, and eggs), so is the potential output. Therefore, the results of a simulation are often difficult to comprehend. They must be sampled and summarized creating many potentially subjective views of the simulation. However, a similar problem arises with sampling and comprehending natural ecosystems.

The second problem also is related to model complexity. How can the model results be visualized or displayed? Time is an important element in the simulation. Therefore, the time series of these very complex data should be displayed so that system dynamics can be understood and evaluated.

We hope that analyses of GHSim and components that will be included in the future (predation, grazing management, and weather) will be useful for guiding research for innovative and ecologically sound rangeland grasshopper management. OOP and robust analysis of model output will be useful tools during this process.

Acknowledgments

This work was funded by the Grasshopper Integrated Pest Management Project (USDA Animal and Plant Health Inspection Service, Plant Protection and Quarantine) and USDA Agricultural Research Service, with support from the University of Nebraska and Utah State University.

References

1. Berry, J. S., W. P. Kemp & J. A. Onsager. 1993. Within-year population dynamics model for rangeland grasshopper (Orthoptera: Acrididae). Environ. Entomol. submitted.
2. Capinera, J. L., W. J. Parton, & J. K. Detling. 1983. Application of a grassland simulation model to grasshopper pest management on the North American shortgrass prairie. *In* W K. Lauenroth, G. V. Skogerboe, and M. Flug (eds.), Analysis of ecological system: state-of-the-art in ecological modelling. pp. 335-344.
3. Gyllenberg, G. 1974. A simulation model for testing the dynamics of a grasshopper population. Ecology **55**:645-650.
4. Hardman, J. M., W. A. Charnetski, & M. K. Mukerji. 1985. A model simulating grasshopper damage to wheat planted in infested stubble. Journal of Applied Ecology **22**:373-394.
5. Hardman, J. M., & M. K. Mukerji. 1982. A model simulating the population dynamics of the grasshopper (Acrididae) *Melanoplus sanguinipes* (Fabr.), *M. packardii* Scudder, and *Camnula pellucida* (Scudder). Res. Pop. Ecol. **24**:276-301.
6. Hilbert, D. W., & J. A. Logan. 1983. A simulation model of the migratory grasshopper *Melanoplus sanguinipes*. pp. 323-334.
7. Mann, R., R. E. Pfadt, & J. J. Jacobs. 1986. A simulation model of grasshopper population. Dynamics and results for some alternative control strategies. Agric. Exp. Sta. Univ. WY., Sci. Monogr. **51**:1-48.
8. Olson, R. L., & T. L. Wagner. 1992. WHIMS, A knowledge-based system for cotton pest management. AI Applications **6**:41-58.
9. Plant, R. E., & N. D. Stone. 1991. Object-oriented models. Pages 364 *in* Knowledge-based systems in agriculture. McGraw-Hill, New York. p. 364.
10. Rodell, C. F. 1977. A grasshopper model for a grassland ecosystem. Ecology **58**:227-245.
11. Saarenmaa, H., N. D. Stone, L. J. Folse, J. M. Packard, W. E. Grant, M. E. Makela, & R. N. Coulson. 1988. An artificial intelligence modelling approach to simulating animal/habitat interactions. Ecol. Modelling **44**:125-141.
12. Stone, N. D. 1990. Chaos in an individual-level predator-prey model. Natural Resource Modelling **46**:529-553.