Modeling an "Able" Invader—the "Cane" Toad

By Angela B. Shiflet and George W. Shiflet
Wofford College, Spartanburg, South Carolina

1. Scientific Question

Introduction

At a time when biologists are concerned about a worldwide decline in amphibian populations, it is ironic that an introduced species of toad is rapidly increasing in abundance in Australia. In the first global assessment studies, scientists found 43% of known amphibian species are in decline (Stuart *et al*. 2004), while Australian populations of this toad have increased from an original population of 3000 to millions (up to 2000 per hectare (Freeland 1986)). And, they are expanding their range at up to 50 kilometers per year into habitats previously thought too restrictive for their survival (Phillips *et al*. 2006). These toads are voracious predators and nimble competitors, and their large populations have spread widely through several Australian states threatening native species and disrupting the existing biological communities.

The toad, commonly called the **marine** or **cane toad** (***Bufo* (*Chaunus*) *marinus***), was introduced into various countries, including Australia, but in Australia it has become a major worry (see Figure 1). Why would anyone introduce such an animal to a country that already had such a rich, unique fauna? To answer that question, we must examine Puerto Rico during the early part of the 20th Century. At that time, sugar cane growers there were desperately seeking something to control beetle grubs (larvae) that were destroying the roots of their crops. In response, the U.S. Department of Agriculture imported cane toads from Barbados. Within ten years, the beetle grub numbers were reduced to the level of a mere nuisance. This was a relatively rare example of a positive outcome from introducing species to new geography, and it encouraged other cane-growing regions to mimic this "successful" strategy. Cane toads were introduced to Australia in 1935 to fight the damage done by gray-backed (*Dermolepida albohirtum*) and Frenchi beetles (*Lepidiota frenchi*) (Freeland and Martin 1985; Alford *et al*. 1995) to cane crops. However, the toads proved to be ill-chosen as a cure—The beetles attack the top of the cane instead of the roots, and toads don't fly; beetles are active during the day, while toads are active at night; the toads do not like the hot cane fields, where there is a high danger of **desiccation**, or drying; and the amphibian had too many other tasty prey alternatives. Sadly, cane toad numbers continue to increase, and models predict that they will eventually occupy twice the 1 million km^2 of Australia they presently do (Urban *et al*. 2007).

Figure 1 Cane toad (Fuller 2011)

In Australia, female cane toads are prodigious producers of eggs (8,000-35,000 eggs/clutch (Hero and Stoneham 2009)), laying eggs once or twice each year, beginning during their second wet season (Cohen and Alford 1993). These eggs are produced in long gelatinous strings, attached to shallow vegetation. After approximately 48 hours, tadpoles emerge and initially feed on algae (Hinkley 1962). After 37-40 days, metamorphosis into toadlets normally occurs. This time is variable, however, dependent on various climatic factors, competition and predation (from previous tadpole cohorts). Growth rates are strongly density-dependent with higher growth rates and maturation at lower densities (Alford *et al*. 1995). Because young toadlets are small and poorly developed, they must initially stay near water to prevent desiccation (Cohen and Albert 1993). As they age and mature, young toads move further from the water, but the first dry season takes its toll on them with only 10-47% surviving (Alford *et al*. 1995). Freeland and Martin (1985) found that young toads are the primary colonizers with dispersion occurring at the edges of the toads' distribution.

Juvenile and mature toads are active at night, feeding on insects attracted to lighted areas (Wright and Wright 1949). They can often be found in gardens, around houses and other disturbed areas (Krakauer 1968). Cane toads are relatively aggressive and somewhat undiscerning predators. Although they favor certain beetles, they will also occasionally dine on other types of arthropods (e.g., ants, crabs, spiders) (Krakauer 1968, Easteal 1982). Various researchers have observed cane toads feeding on snakes, birds, small mammals, and other amphibians (Rabor 1952, Krakauer 1968, Oliver 1949, Bartlett and Bartlett 1999). Researchers are constantly investigating the actual impacts the toads may have on competitor, prey, and predator populations.

Because these animals seem to fancy relatively open, disturbed areas associated with human activity, residents in many areas of eastern and northern Australia are quite likely to encounter them. Imagine yourself and your family living in a quiet suburb near Cairns, Queensland. You might have a pet dog or cat you feed every evening on the back patio. If your pet doesn't eat all the food, an opportunistic cane toad might finish off the remains. If your dog happens upon and mouths a toad, it would be in for a nasty surprise, and you might find your pet drooling profusely or foaming at the mouth, staggering, twitching, seizing, and/or vomiting. Your dog has been exposed to one of the toad's major defenses against predation—toxic secretions.

The members of the genus *Bufo*, including the cane toad, are somewhat notorious for their capacity and propensity to produce bufadienolides and other toxic chemicals, which they secrete in quantity from prominent, specialized parotoid glands on their heads. Cane toads release these secretions defensively when trifled with by curious or potentially predaceous animals. The bufadienolides are cardiotoxic steroids that act much like digitalis, interfering with membrane sodium-potassium pumps and thereby increasing the contraction of the heart (Halliday *et al*. 2009). The toxins are absorbed rapidly by the mouth lining, so it is little wonder your dog soon regrets the folly of its appetite or curiosity. For heart patients, digitalis may restore normal heart muscle function, but in large enough quantities cane toad toxin may lead to cardiac failure. On the other hand, *Bufo* toxin (*Chansu*) has been used in China since the seventh century (Xiao 2002), and is still used in Chinese Traditional Medicine as an analgesic and to treat heart failure (Ma *et al*. 2007). Additionally, toad toxin preparations are used for certain types of cancer (Meng *et al*. 2009). It is little wonder that enterprising Australians are exporting *Bufo marinus* to the Chinese (BBC News 2010).

The Problem

Invasive species often do not expand along a continuous front, but satellite groups establish themselves at **invasion hubs**, such as advantageous habitat areas. In the vast parts of arid Australia during the dry season, **artificial water points** (**AWPs**), such as troughs or dams for livestock, can serve as invasion hubs for cane toads. Thus, targeting AWPs might help prevent the spread of this invasive species. To study this hypothesis, scientists erected toad-proof fences around AWPs in an experimental zone, collecting toads that were already in the AWPs and excluding others from these water sources. They also performed simulations to model the potential dispersal ability of toads under various climate conditions with and without AWPs (Florance *et al*. 2011). In this module, we are interested in developing a similar but simplified model to study the effect of fencing AWPs on adult cane toad invasion. Ignoring climate, topography, periods of sleep, and the cane toad life cycle for the basic model, we wish to examine the impact of fenced and unfenced AWPs on the migration of cane toads through an area.

2. Computational Model

Grid-Based Individual-Based Model

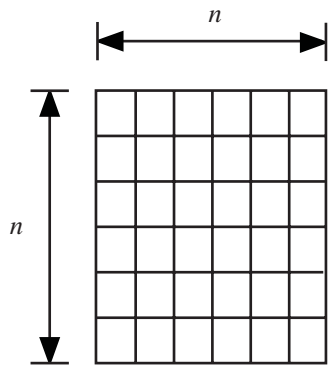
For our simulation, a **grid-based individual-based model** seems appropriate. An **individual-based model** (**agent-based model** or **entity-based model**) follows individuals of a population in an **environment**, considering the global consequences of their local interactions. Individuals are described in terms of their **behaviors** (**rules** or **transition rules**) and their **state** (or set of characteristic parameters). With a **grid-based model**, the environment consists of one or more **grids**, or rectangular arrangements of **cells** (**sites**), and an individual moves discretely from one cell to another instead of continuously to any point (Reynolds 1999).

Grid-based individual-based simulations are related to **cellular automaton simulations**, which are also dynamic computational models, or such models that change with time, that are discrete in space, state, and time. Cellular automaton grids represent environments, and rules regulate the behavior of the system by specifying local relationships and indicating how cells change states. The prime distinction between a grid-based individual-based model and a cellular automaton model is that a simulation for the former loops through all individuals one at a time, while a simulation for the latter loops through all grid elements one at a time.

Model of Environment

For our grid-based individual-based simulation of the movement of cane toads through an area with AWPs, the environment consists of two grids, *moisture* and *food*, to store wetness and nutrient values. Each grid is an n-by-n (or $n \times n$) two-dimensional array (matrix or lattice) of numbers (see Figure 2), and each cell in the lattice contains a real number value, typically between 0.0 and 1.0, representing a characteristic of the corresponding location.

Figure 2 Grid with cells to model area



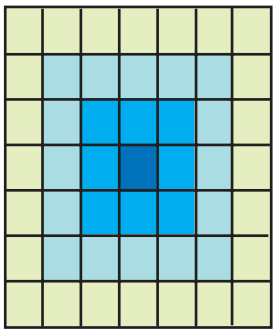
We can assume a fixed or random amount of nutrition at each site of the initial *food* array. For example, we may initialize all food values to be *FOOD_CELL* = 0.03. Alternatively, being careful to restrict the value to be between 0.0 and 1.0, we may initialize a cell to contain a normally distributed random number with mean *MEAN_FOOD* = 0.1 and standard deviation *STD_FOOD* = 0.02.

Throughout the moisture grid, we have a low, constant value, such as 0.0, except at or near AWPs; natural waterways, such as rivers; and moist sites, such as logs. AWPs and natural waterways have values of 1.0 (*AWP*). Cells immediately adjoining such water areas, whether fenced or unfenced, have lower positive values, such as 0.4 (*AWP_R1*), while those two locations away have even lower positive values, such as 0.2 (*AWP_R2*). Figure 3 displays part of a *moisture* matrix and a visualization of the corresponding area around an AWP. Moist areas that serve as refuges for the toads have values of *MOIST_AREA* = 0.5.

Figure 3 (a) Part of the *moisture* matrix and (b) visualization of area around an AWP

(a) **(b)**

$$\begin{bmatrix} & & & \vdots & & & \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.0 \\ 0.0 & 0.2 & 0.4 & 0.4 & 0.4 & 0.2 & 0.0 \\ \cdots\ 0.0 & 0.2 & 0.4 & 1.0 & 0.4 & 0.2 & 0.0\ \cdots \\ 0.0 & 0.2 & 0.4 & 0.4 & 0.4 & 0.2 & 0.0 \\ 0.0 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ & & & \vdots & & & \end{bmatrix}$$

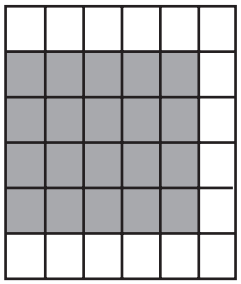


To prevent toads from going off the grid in certain directions, we give cells on those directional boundaries values of -1. Thus, if we only want to allow toads to migrate out of the environment to the west, we assign -1 to all cells in the first and last rows and last column of *moisture* and *food*. Figure 4 presents a 6-by-6 *food* matrix and a corresponding visualization with darker shades of gray representing larger values.

Figure 4 Example of (a) initial *food* matrix with (b) visualization

(a) **(b)**

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 \\ 0.03 & 0.03 & 0.03 & 0.03 & 0.03 & -1 \\ 0.03 & 0.03 & 0.03 & 0.03 & 0.03 & -1 \\ 0.03 & 0.03 & 0.03 & 0.03 & 0.03 & -1 \\ 0.03 & 0.03 & 0.03 & 0.03 & 0.03 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$



Toad's State

For each toad, we store certain characteristic parameters or attributes, such as the following, that represent the toad's state at that instance:

toadID - integer identification number

energy - value from 0.0 to 1.0 indicating toad's amount of energy. An energy value of 1.0 indicates that the toad is full. The lower the value, the hungrier the toad is.

water - value from 0.0 to 1.0 indicating toad's amount of water. A water value of 1.0 indicates that the toad is completely hydrated. The lower the value, the thirstier the toad is.

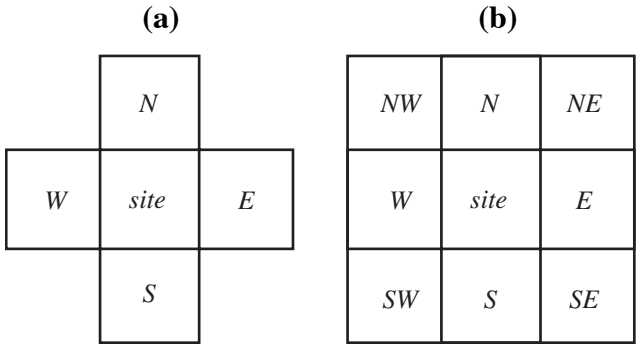
alive - integer value indicating if toad is alive (*ALIVE* = 1), dead (*DEAD* = 0), or has migrated off the grids (*MIGRATED* = 2)
row - matrix row location of the toad on the grids
col - matrix column location of the toad on the grids
rowLast - when looking for water, previous matrix row location of toad on grids
colLast - when looking for water, previous matrix column location of toad on grids
numTimeSteps - number of steps the toad has been in a moist area

For a toad variable *aToad*, we use the dot notation to represent parameters, such as *aToad.energy* for the toad's energy. For an array *toads*, we employ subscript notation, too, such as *toads*(*i*)*.energy* for the *i*th toad's energy.

Toad Behavior

A toad's behavior is often regulated by its own state, particularly its amount of water and energy, and the moisture and food conditions around it. With *site* being the current location of a toad on the *moisture* and *food* grids, our simulation might use the site's **von Neumann neighborhood**, which includes the four nearest neighbors to the north, east, south and west (see Figure 5a), or might employ the **Moore neighborhood** of all eight neighbors (see Figure 5b). Sometimes in determining a toad's behavior, we also consider the grid value of *site* itself.

Figure 5 (a) von Neumann and (b) Moore neighborhoods



*While going through the module, please complete the **Quick Review Questions**, which occur periodically, for a fast reinforcement of material comprehension. After completing a question part, refer to the section on "**Answers to Quick Review Questions**" towards the end of the module for immediate feedback.*

Quick Review Question 1 Suppose *site* is in row 30 and column 45. Suppose also that the first row of the grid is visualized as up, or to the far north.

a. Give the row and column of the site's neighbor to the south.
b. Give the row and column of its neighbor to the northwest.

The model of a toad's behavior involves an important simulation technique, **random walk**, which has many applications in the sciences. **Random walk** refers to the

apparently random movement of an entity, such as a cane toad. With this simulation technique, at any time step, an entity can move, perhaps under certain constraints, at random to a neighboring cell. Relevant **toad behavior rules**, which include aspects of a random walk in the movement rules, are as follows:

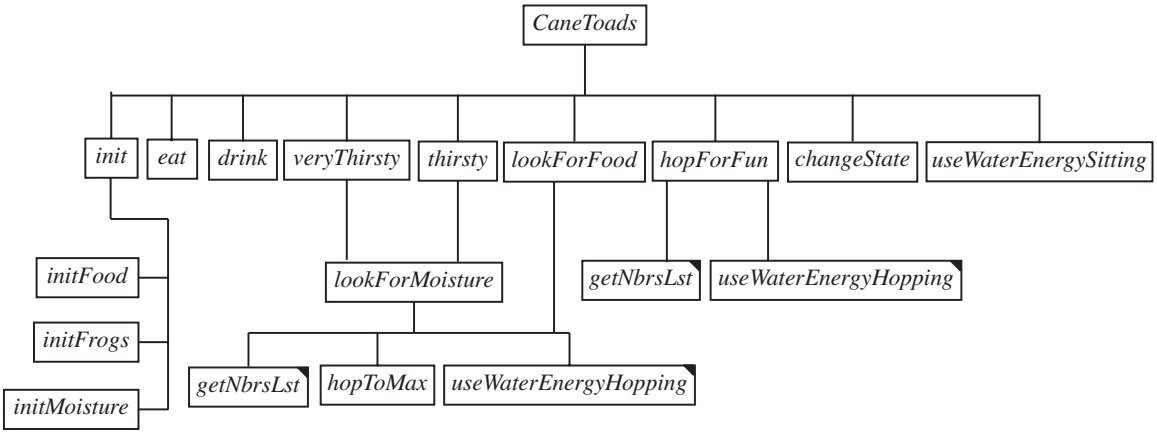
1. If the toad is alive and its energy value is such that it would like to eat (energy value below *WOULD_LIKE_EAT*), it eats;
2. else if the toad is alive and its water value is such that it would like to drink (water value below *WOULD_LIKE_DRINK*), with probability *MAY_EAT* it eats.
3. If the toad is in water and would like to drink, it does.
4. If the toad is very thirsty (water value below *NEED_TO_DRINK*) but is not in water, it hops to the nearest neighboring site with the most moisture, not including the current or previous location or locations on a north, east, or south boundary;
5. else if the toad is thirsty (water value below *WOULD_LIKE_DRINK*) but is not in water or a moist area (*moisture* grid cell's value less than *MOIST_AREA*), it hops to the nearest neighboring site with the most moisture, not including the current or previous location or locations on a north, east, or south boundary;
6. else if the toad is thirsty and is in a moist area (*moisture* grid cell's value greater than or equal to *MOIST_AREA* and less than *AWP*), the longer the toad has been in that location the more likely it is to move to the nearest neighboring site with the most moisture, not including the current or previous location or locations on a north, east, or south boundary;
7. else if the toad is hungry (energy value below *WOULD_LIKE_EAT*), it hops to the nearest neighboring site with the most food, including the current location but not including locations on a north, east, or south boundary;
8. else with probability *MAY_HOP*, the toad hops to a neighboring site that is not on a north, east, or south boundary.
9. A toad uses energy (*ENERGY_HOPPING*) hopping.
10. A toad uses some but less energy (50% of *ENERGY_HOPPING*) sitting.
11. A toad uses water (*WATER_HOPPING*) hopping into a dry area (*moisture* grid cell's value less than *MOIST_AREA*).
12. while a toad uses some but less water (50% of *WATER_HOPPING*) sitting in a dry area.
13. A toad gains energy (*amtEat*) eating. *amtEat* is at most *AMT_EAT* but no more than the amount of food in the toad's *food* site; moreover, the toad's energy value cannot exceed 1.0.
14. A toad gains some water (*FRACTION_WATER* of *amtEat*) eating. However, the toad's water value cannot exceed 1.0.
15. A toad gains water (*AMT_DRINK*) drinking. However, the toad's water value cannot exceed 1.0.
16. If the toad is dead, it does not change state.
17. If the toad has migrated to the west boundary, it does not change state.

3. Algorithms

Program Structure Diagram

Figure 6 contains the program structure diagram for a cane toad simulation, and the following sections describe individual routines, which often provide algorithms for all or parts of the toad behavior rules.

Figure 6 Program structure diagram for cane toads simulation



Constants and Initialization

The algorithms employ a number of constants, which Table 1 lists. Values on the *food* and *moisture* grids are between 0.0 and 1.0. Some constants, such as *DESICCATE* and *FRACTION_WATER*, are from the literature; although death from desiccation may occur at slightly different percentages, and the fraction of prey that is water varies depending on the source. Many of the values from Table 1 can be adjusted to explore alternative conditions. A procedure, *init*(), initializes these constants.

Table 1 Cane toad simulation constants

Constant	Value	Meaning
ALIVE	1	value indicating toad is alive
AMT_DRINK	0.05	maximum amount toad drinks in 1 time step
AMT_EAT	0.01	maximum amount toad eats in 1 time step
AWP	1	*moisture* value for water, such as an AWP
AWP_R1	0.4	*moisture* value of neighboring cell to water
AWP_R2	0.2	*moisture* value of cell 2 cells away from water
COLLST_AWP	such as (5, 13, 3, 9)	vector of column indices for AWPs
COLLST_MOISTAREA	such as (13, 15, 3)	vector of column indices for moist areas

DEAD	0	value indicating toad is dead
DESICCATE	0.6	level at which desiccation occurs
ENERGY_HOPPING	0.002	maximum amount of energy used by toad in a hop
FENCED	such as (1)	vector of *ROWLST_AWP* indices of fenced AWPs
FOOD_CELL	0.03	food value for initializing constant *food* grid
LIMITED	0.0	*moisture* value for limited amount of moisture
MAY_EAT	0.2	probability of eating if thirsty
MAY_HOP	0.5	probability of hopping if not thirsty or hungry
MEAN_ENERGY	0.94	mean energy for initialized toad
MEAN_FOOD	0.1	mean value for initializing random *food* grid
MEAN_WATER	0.94	mean water for initialized toad
MIGRATED	2	value indicating toad has migrated off the grid
MOIST_AREA	0.5	*moisture* value for moist area
NEED_TO_DRINK	0.8	water level at which toad really needs to drink
NUM_TOADS	such as 50	number of toads
NUM_NEIGHBORS	4 or 8	number of neighbors not including self
NUM_STEPS	such as 300	number of simulation steps
FRACTION_WATER	0.6	fraction of prey that is water
PERMANENT_WATER	1.0	*moisture* value for natural water, such as a river
ROWLST_AWP	such as (3, 11, 15, 9)	vector of row indices for AWPs
ROWLST_MOISTAREA	such as (7, 15, 17)	vector of row indices for moist areas
SIDE	such as 20	number of cells along one side of grid
STARVE	0.6	level at which starvation occurs
STD_ENERGY	0.02	standard deviation of energy for initialized toad
STD_FOOD	0.02	standard deviation for initializing random *food*
STD_WATER	0.02	standard deviation of water for initialized toad
WATER_HOPPING	0.002	maximum amount of water used by toad in a hop
WOULD_LIKE_DRINK	0.9	water level at which toad would like to drink
WOULD_LIKE_EAT	0.9	food level at which toad would like to eat

Several constants deserve special comment. To simulate the activity of toads over a period of a 12-hour night, if *NUM_STEPS* = 300, each time step is of length 2.4 min/time step:

$$\frac{12 \text{ hr}}{300 \text{ time steps}} = \frac{12 \text{ hr}}{300 \text{ time steps}} \times \frac{60 \text{ min}}{\text{hr}} = \frac{2.4 \text{ min}}{\text{time step}}$$

Another important constant is *SIDE*, which is the number of cells, such as 20, along one side of a square *food* or *moisture* grid. Thus, to represent a half-kilometer square field, which is 500 m on each side, each cell corresponds to a 500/20 = 25 m^2 area.

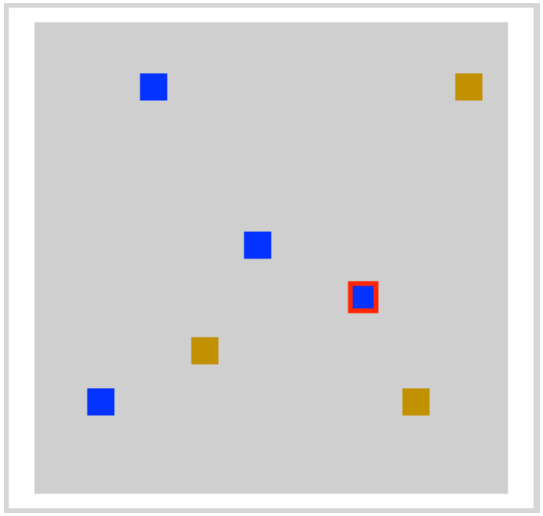
Programming languages typically number the rows and columns for matrices, such as *food* or *moisture*, beginning with 0 or 1. For this module, we will assume indexing begins with 0. Paired vectors (one-dimensional arrays) *ROWLST_AWP* and

COLLST_AWP contain the *moisture* row and corresponding column indices for AWPs. For example, suppose as in Table 2 *ROWLST_AWP* is (3, 11, 15, 9) and *COLLST_AWP* is (5, 13, 3, 9). Thus, the first of four AWPs occurs in row 3 and column 5 of the grids, and the cell in row 3 and column 5 is blue in the visualization of Figure 7. Using each value from *ROWLST_AWP* and the corresponding value from *COLLST_AWP* as row-column indices in *moisture* and parentheses around the row-column pair, we assign array elements *moisture*(3, 5), *moisture*(11, 13), *moisture*(15, 3), and *moisture*(9, 9) to have moisture amounts of 1.0. The vector *FENCED* indicates fenced AWPs by listing the *ROWLST_AWP* (or *COLLST_AWP*) indices. Thus, to indicate that only the second AWP is fenced, *FENCED* is assigned a vector with the index of the second element of *ROWLST_AWP* (or *COLLST_AWP*), which we write as (1). Table 2 shows the appropriate numbers in boldface, and Figure 7 displays the second AWP in row 11 and column 13 with a red outline indicating fencing. Similar to *ROWLST_AWP* and *COLLST_AWP*, *ROWLST_MOISTAREA* and *COLLST_MOISTAREA* are vectors with the row and corresponding column indices for moist areas. For example, if as in Table 2 *ROWLST_MOISTAREA* is (13, 15, 3) and *COLLST_MOISTAREA* is (7, 15, 17), *moisture*(13, 7), *moisture*(15, 15), and *moisture*(3, 17) have values *MOIST_AREA* and appear in yellow in Figure 7.

Table 2 Example *ROWLST_AWP*, *COLLST_AWP*, *FENCED*, *ROWLST_MOISTAREA*, and *COLLST_MOISTAREA* vectors with their indices; boldface numbers indicate that grid element in row 11 and column 13 is a fenced AWP

Vector Index	0	**1**	2	3
ROWLST_AWP	3	**11**	15	9
COLLST_AWP	5	**13**	3	9
FENCED	**1**			
ROWLST_MOISTAREA	13	15	3	
COLLST_MOISTAREA	7	15	17	

Figure 7 Visualization with AWPs, fenced, and moist areas as indicated in Table 2

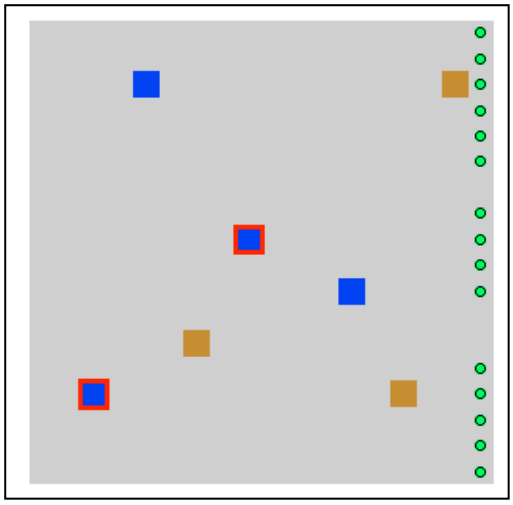


Quick Review Question 2 Suppose *ROWLST_AWP* = (75, 61, 34, 91, 53, 22).

a. Give the length of *COLLST_AWP*.
b. Give the maximum length of *FENCED*.
c. Suppose *COLLST_AWP* = (15, 36, 39, 58, 72, 80) and *FENCED* = (0, 4, 5). Give the row and column locations of fenced AWPs.

Besides defining constants, *init*() also calls other functions, *initFood*(), *initMoisture*(), and *initToads*() to initialize matrices *food* and *moisture* and a one-dimensional array *toads*, respectively. For the basic simulation of this module, all toads are initialized in random locations in the next-to-the-last grid column, which is towards the east, but not in the first or last rows of that column; and more than one toad can be in the same grid cell. To require migration for the full width of the terrain, toads are allowed to migrate off the grid to the west but not in any other direction. Thus, the north boundaries (row 0), east boundaries (column *SIDE* - 1), and south boundaries (row *SIDE* - 1) of *food* and *moisture* are initialized to be -1, a number less than any other food or moisture value. Figure 8 shows a visualization of initialization with 50 toads, some of which share cells, on a 20-by-20 grid.

Figure 8 Visualization of 50 initialized toads



Because of their ubiquitous nature, to avoid passing a number of parameters, the constants and the *food* and *moisture* arrays are global to functions that employ them. With the **left pointing arrow** (←) indicating assignment of the right hand side to the variable on the left, the descriptions of the initialization functions follow.

init()
Function to initialize global constants, *SIDE*-by-*SIDE* matrices *food* and *moisture*, and a vector, *toads*, of *NUM_TOADS* number of toads
Post:
Global constants in Table 1 have been initialized.
food has been initialized as a global *SIDE*-by-*SIDE* food matrix.
moisture has been initialized as a global *SIDE*-by-*SIDE* moisture matrix.

toads has been initialized as a global *NUM_TOADS* toad vector.
FENCED, *ROWLST_AWP*, *COLLST_AWP* are used by *initMoisture*(), *getNbrsLst*(), and visualization routines, while *ROWLST_MOISTAREA* and *COLLST_MOISTAREA* are employed in *initMoisture*() and a visualization procedure.

Algorithm:
initialize global constants listed in Table 1
food ← *initFood*()
toads ← *initToads*()
moisture ← *initMoisture*()

***food* ← *initFood*()**
Function to return an initialized *SIDE*-by-*SIDE* food matrix
Pre:
SIDE, *FOOD_CELL, MEAN_FOOD,* and *STD_FOOD* are global constants.
Post:
The function has returned a *SIDE*-by-*SIDE* food matrix.
Algorithm:
initialize each element of the last column and the first and last rows of *SIDE*-by-*SIDE* matrix *food* to be -1 and each other element to be *FOOD_CELL*

Alternative Algorithm:
initialize each element of the last column and the first and last rows of *SIDE*-by-*SIDE* matrix *food* to be -1 and each other element to be a normally distributed random number with mean *MEAN_FOOD* and standard deviation *STD_FOOD*, restricted to be in the interval 0.0 to 1.0

***moisture* ← *initMoisture*()**
Function to return an initialized *SIDE*-by-*SIDE* moisture matrix
Pre:
SIDE, *LIMITED*, *MOIST_AREA, AWP, AWP_R2*, *AWP_R1*, *FENCED*, *ROWLST_AWP*, *COLLST_AWP*, *ROWLST_MOISTAREA*, and *COLLST_MOISTAREA* are global constants initialized in *init*().
Post:
The function has returned a *SIDE*-by-*SIDE* moisture matrix.
Algorithm:
initialize each element of *SIDE*-by-*SIDE* matrix *moisture* to be *LIMITED*

for *i* going sequentially through *ROWLST_MOISTAREA* indices
moisture(*ROWLST_MOISTAREA*(*i*), *COLLST_MOISTAREA*(*i*)) ← *MOIST_AREA*

for *i* going sequentially through *ROWLST_AWP* indices
initialize *moisture* matrix two cells from (*ROWLST_AWP*(*i*), *COLLST_AWP*(*i*)) to *AWP_R2*

initialize *moisture* matrix adjacent to (*ROWLST_AWP*(*i*), *COLLST_AWP*(*i*)) to *AWP_R1*
moisture(*ROWLST_AWP*(*i*), *COLLST_AWP*(*i*)) ← *AWP*

initialize each element of the last column and the first and last *moisture* rows to be -1

Quick Review Question 3 Suppose *SIDE* = 100, *ROWLST_AWP* = (75, 61, 34, 91, 53, 22) and *COLLST_AWP* = (15, 36, 39, 58, 72, 80), as in Quick Review Question 2; and *AWP* = 1, *AWP_R1* = 0.4, *AWP_R2* = 0.2, and *LIMITED* = 0.0 as in Table 1. Immediately after the call to *initMoisture*(), give the values of the following:

a. *moisture*(0, 30)
b. *moisture*(75, 61)
c. *moisture*(61, 36)
d. *moisture*(35, 38)
e. *moisture*(22, 82)
f. *moisture*(42, 99)

***toads* ← *initToads*()**
Function to return an initialized *toads* vector of size *NUM_TOADS* with toads placed at random locations in the next-to-the last column, excluding the first and last row
Pre:
SIDE, *NUM_TOADS, MEAN_ENERGY, STD_ ENERGY*, *MEAN_WATER*, *STD_WATER*, and *ALIVE* are global constants.
Post:
The function has returned a toads vector of *NUM_TOADS* number of elements.
Algorithm:
for *i* going sequentially through *NUM_TOADS* sequential indices
toads(*i*).*toadID* ← *i*
toads(*i*).*energy* ← normally distributed random number with mean *MEAN_ENERGY* and standard deviation *STD_ ENERGY*, restricted to interval from 0.0 to 1.0
toads(*i*).*water* ← normally distributed random number with mean *MEAN_WATER* and standard deviation *STD_WATER*, restricted to interval from 0.0 to 1.0
toads(*i*).*alive* ← *ALIVE*
toads(*i*).*row* ← uniformly distributed random integer from 1 to *SIDE* - 2, picked from second to next-to-the-last row indices
toads(*i*).*col* ← *SIDE* - 2, the next-to-the-last column index
toads(*i*).*rowLast* ← -1
toads(*i*).*colLast* ← -1
toads(*i*).*numTimeSteps* ← 0

Quick Review Question 4 Suppose *SIDE* = 100. After execution of *initToads*(), give

a. the initial possible row(s) of a toad
b. *toads*(24).*col*
c. *toads*(24).*toadID*

Simulation

The function *CaneToads*() is the main driving function for the simulation. Moreover, at each time step, the function saves the food grid and the location of each toad for later animation.

CaneToads(), *thirsty*(), and *hopForFun*() all involve probabilities of certain occurrences. For example, Rule 8 states that even if a toad does not look for water or food, with probability *MAY_HOP* the toad may hop to a neighboring site. If *MAY_HOP* is 0.15, a 15% chance exists that under these circumstances the toad will hop. To process this probability, we generate a uniformly distributed random floating point number between 0.0 and 1.0. On the average, 15% of the time this random number is less than 0.15, while 85% of the time the number is greater than or equal to 0.15 (see Figure 9). Thus, if the random number is less than 0.15, we have the toad hop; otherwise, the toad does not change locations at this time step.

Figure 9 15% of floating point values between 0 and 1 are less than *MAY_HOP* = 0.15

Quick Review Question 5 Consider the following pseudocode that designates the direction a simulated toad moves:

```
randNum ← uniformly distributed random number between 0.0 and 1.0
if randNum < 0.12
        have the toad move to the north
else if randNum < 0.26
        have the toad move to the east
else if randNum < 0.69
        have the toad move to the south
else
        have the toad move to the west
```

Give the probability that the animal moves in each of the following directions:
a. north **b.** east **c.** south **d.** west

CaneToads() processes eating and drinking for all the toads. Then, for all these amphibians, the function handles toad movement, possible change of toad state to dead or migrated, and storage of toad locations for later animation. The *food* grid is also saved for animation purposes. The algorithm for *CaneToads*() and many of the functions that this driver calls follow. Comments, prefaced by "//", indicate, among other things, the rules that a function processes.

***CaneToads*()**
Function to perform a cane toad simulation

Algorithm:

```
toads ← init()  // initialize constants, food and moisture grid, and toads vector

// save initial toad locations and food grid for later visualization
for i going sequentially through NUM_TOADS toads indices
    rowPath(i) ← list containing one element, toads(i).row
    colPath(i) ← list containing one element, toads(i).col
foodGrids ← list containing one element, food

for step going sequentially through NUM_STEPS indices

    // toads eat and drink
    for i going sequentially through NUM_TOADS indices
        if toads(i).alive is ALIVE

            if toads(i).energy < WOULD_LIKE_EAT      // Rule 1: would like to eat
                toads(i) ← eat( toads(i) )
            // Rule 2:  would like to drink, may eat
            else if toads(i).water < WOULD_LIKE_DRINK and a uniformly distributed
                    random number between 0.0 and 1.0 is less than MAY_EAT
                toads(i) ← eat( toads(i) )

            // Rule 3:  water available and would like to drink
            if (moisture(toads(i).row, toads(i).col) ≥ AWP and
                    toads(i).water < WOULD_LIKE_DRINK )
                toads(i) ← drink( toads(i) )

    // toads move and change state; save locations for later animation
    for i going sequentially through NUM_TOADS indices
        if toads(i).alive is ALIVE

            // process hopping for water, food, or fun
            if (toads(i).water < NEED_TO_DRINK)  // Rule 4: very thirsty
                toads(i) ← veryThirsty( toads(i) )
            else if (toads(i).water < WOULD_LIKE_DRINK)          // Rules 5 and 6:
        thirsty
                toads(i) ← thirsty( toads(i) )
            else if (toads(i).energy < WOULD_LIKE_EAT)  // Rule 7: hungry
                toads(i) ← lookForFood(toads(i))
            else if a uniformly distributed random number between 0.0 and 1.0
                    is less MAY_HOP       // Rule 8: may hop for fun
                toads(i) ← hopForFun(toads(i))
            else    // Rule 10: doesn't hop but uses some of its water and energy sitting
                toads(i) ← useWaterEnergySitting(toads(i))

            toads(i) ← changeState(toads(i)) // Rules 16 and 17
```

```
        // save toad locations and food grid for later visualization
        append toads(i).row to rowPath(i)
        append toads(i).col to colPath(i)

    // save food grid for later visualization
    append food to foodGrids
```

Rule 1 states that a toad, *aToad*, who would like to eat does. The desired amount that the animal consumes in one time step is *AMT_EAT*. However, the toad cannot consumer more food than is available in its current location, *food*(*aToad.row*, *aToad.col*). Moreover, its energy level should not exceed 1, so the additional food cannot be more than 1 - *aToad.energy*. Thus, the toad eats *amtEat*, the minimum of *AMT_EAT*, *food*(*aToad.row*, *aToad.col*), and 1 - *aToad.energy*. This value is added to the energy value of the toad and subtracted from the food value of the *food* matrix cell. Food, such as a beetle, contains water, too; and we assume toad prey averages *FRACTION_WATER* = 0.60 = 60% water. Thus, when it eats, we add *FRACTION_WATER * amtEat* to *aToad*'s water quantity. However, we must again be careful that the sum does not exceed 1.0, so we change *aToad.water* by the minimum of *aToad.water* + *FRACTION_WATER* * *amtEat* and 1.0. The algorithm for *eat*() follows:

```
aToad ← eat( aToad )
    Function to update a toad's energy and water after it eats
Pre:
    food, AMT_EAT, FRACTION_WATER are global variables.
Post:
    The toad's energy and water levels have been adjusted after eating.
Algorithm:
// eat and get some water from eating to levels no more than 1.0
amtEat ← minimum of AMT_EAT, food(aToad.row, aToad.col), and 1 - aToad.energy
aToad.energy ← aToad.energy + amtEat
aToad.water ← minimum of aToad.water + FRACTION_WATER * amtEat and 1.0
food(aToad.row, aToad.col) ← food(aToad.row, aToad.col) - amtEat

return aToad
```

Quick Review Question 6 Suppose *AMT_EAT* = 0.01, *FRACTION_WATER* = 0.6, *aToad.row* = 5, and *aToad.col* = 5. Give the values of *aToad.energy*, *aToad.water*, and *food*(5, 5) after execution of *aToad* = *eat*(*aToad*) for each of the following situations:

- **a.** *aToad.energy* = 0.9, *aToad.water* = 0.8, and *food*(5, 5) = 0.03
- **b.** *aToad.energy* = 0.9, *aToad.water* = 0.8, and *food*(5, 5) = 0.005
- **c.** *aToad.energy* = 0.999, *aToad.water* = 0.8, and *food*(5, 5) = 0.03
- **d.** *aToad.energy* = 0.9, *aToad.water* = 0.999, and *food*(5, 5) = 0.03

Because we assume the amount a toad drinks from a water source is negligible relative to the water source, the algorithm for *drink*() is simpler than that for *eat*(). Thus, in one time step, a drinking toad adds no more than *AMT_DRINK* to its internal water amount, being careful that the total does not exceed 1.0.

If after possibly eating and/or drinking, the toad is still very thirsty with water value less than *NEED_TO_DRINK*, we call the function *veryThirsty*(), whose algorithm is below, to process the animal's next behavior. If this cane toad is in water so that the moisture value of its current location, *moisture*(aToad.*row*, aToad.*col*), greater than or equal to *AWP*, the cane toad is in water and so does not change locations. Otherwise, *veryThirsty*() calls *lookForMoisture*() to move the toad to the most advantageous neighboring cell.

***aToad* ← *veryThirsty*(*aToad*)**
Function to change the position of a very thirsty toad
Pre:
moisture, *AWP*, and *AMT_DRINK* are global variables.
Algorithm:
if (*moisture*(aToad.*row*, aToad.*col*) < *AWP*) // not found water
aToad ← *lookForMoisture*(*aToad*)

return *aToad*

If the toad is not as thirsty but would like to drink, we call *thirsty*() to process *aToad*'s next move. Again, if the toad is in water, it remains there. However, if the animal is in a dry area with moisture level less than *MOIST_AREA*, it looks for moisture as in *veryThirsty*() by calling *lookForMoisture*(). Cane toads take diurnal shelter in moist areas, such as soil cracks or logs, in part to minimize water loss (Florance *et al*. 2011). If this thirsty amphibian is currently in a moist but not wet location, with moisture value at least *MOIST_AREA* but not as great as *AWP*, Rule 6 says that the longer the toad has been there the more likely it is to move. To keep track of the number of time steps in that location, we use the toad's *numTimeSteps* attribute. For each time step at the location, we increment *aToad.numTimeSteps* by one. As *aToad.numTimeSteps* increases, its reciprocal decreases and the chance that a uniformly distributed random number between 0.0 and 1.0 would be greater than 1/*aToad.numTimeSteps* increases. Using this fact, for the *thirsty*() algorithm below, the probability that a toad looks for moisture is (1 - 1/*aToad.numTimeSteps*). When the *aToad* does move, we reinitialize its *numTimeSteps* attribute to 0.

***aToad* ← *thirsty*(*aToad*)**
Function to change state of a thirsty toad
Pre:
MOIST_AREA, *AWP*, and *MAY_HOP* are global constants.
moisture is a global variable.
Algorithm:
if (*moisture*(*aToad.row*, *aToad.col*) < *MOIST_AREA*) // currently in dry area
aToad ← *lookForMoisture*(*aToad*)

```
else if (moisture(aToad.row, aToad.col) < AWP)      // in moist area, not water
    aToad.numTimeSteps ← aToad.numTimeSteps + 1   // store number of time steps here
    // The longer the toad has been in moist area, the more likely it is to look for water
    if 1/aToad.numTimeSteps < uniformly distributed random number between 0.0 & 1.0
        aToad.numTimeSteps ← 0  // leaving moist area
        aToad ← lookForMoisture(aToad)
// else toad is already in water, so stays there

return aToad
```

Quick Review Question 7 Suppose *aToad.row* = 20, *aToad.col* = 30, *moisture*(20, 30) = *MOIST_AREA* = 0.5, *AWP* = 1.0, and *aToad.numTimeSteps* = 4 as *thirsty*(*aToad*) is called.

- **a.** Which part, if any, of the *if-then-else* statement executed?
- **b.** After execution of the first statement in the *else* block, what is the value of *aToad.numTimeSteps*?
- **c.** Under these circumstances, give the probability that the toad will leave its current location to look for water.
- **d.** If the uniformly distributed random number between 0.0 and 1.0 is 0.1, give the value of *aToad.numTimeSteps* upon end of execution of the function.
- **e.** If the uniformly distributed random number between 0.0 and 1.0 is 0.3, give the value of *aToad.numTimeSteps* upon end of execution of the function.

Even if not needing to drink, a toad may be hungry, in which case, we call *lookForFood*() (Rule 7). If not thirsty or hungry, this amphibian may still move, which *hopForFun*() processes (Rule 8). The functions *lookForFood*(), *lookForMoisture*(), and *hopForFun*() all call *getNbrsLst*() to obtain the locations to which the toad might move in one time step. The locations as row and column grid indices are returned in a matrix, *matLst*, with the row indices in the first column of *matLst* and the column indices in the second column. The parameters to *getNbrsLst*() are a toad (*aToad*), the number of elements in a neighborhood (*numNeighbors*), and a Boolean variable (*elimPrevious*) that is true if the previous location of the toad, (*aToad.rowLast*, *aToad.colLast*), is not a viable location. As Table 3 enumerates, the possible values for *numNeighbors* are 4 or 5 for a von Neumann neighborhood and 8 or 9 for a Moore neighborhood. If the current site is a possible site for the next time step, *numNeighbors* is 5 or 9.

Table 3 Possible arguments for *numNeighbors* in *getNbrsLst*()

Neighborhood	Not Including Current Site	Including Current Site
von Neumann	4	5
Moore	8	9

The function *getNbrsLst*() first forms corresponding vectors, *rowLst* and *colLst*, of the rows and columns of neighborhood cells. For example, suppose *aToad.row* is 11 and

aToad.col is 14 as for the middle cell with dark green in Figure 10a, and the call to the function is of the following form:

```
matLst  = getNbrsLst(aToad, 9, true)
```

Starting with the toad's current location and then including the cell to the north, the algorithm lists the cell indices in a clockwise fashion as follows:

rowLst = (11, 10, 10, 11, 12, 12, 12, 11, 10)
colLst = (14, 14, 15, 15, 15, 14, 13, 13, 13)

We then eliminate any fenced AWPs if it exists in the neighborhood. As Figure 10a shows, if *ROWLST_AWP* = (3, 11, 15, 9), *COLLST_AWP* = (5, 13, 3, 9), and *FENCED* = (1), a fenced artificial water point exists in row 11 and column 13. Thus, as follows we remove the pair 11, 13 from the lists indicating possible next locations for the toad:

rowLst = (11, 10, 10, 11, 12, 12, 12, 10)
colLst = (14, 14, 15, 15, 15, 14, 13, 13)

Then, we eliminate boundary cells and if *elimPrevious* is true, the toad's last location. The neighborhood in Figure 10a is not on a boundary. If, however, *aToad.rowLast* is 12 and *aToad.colLast* is 14, the cell with light green, we eliminate the pair from *rowLst* and *colLst*, as follows:

rowLst = (11, 10, 10, 11, 12, 12, 10)
colLst = (14, 14, 15, 15, 15, 13, 13)

The function returns a matrix, *matLst*, like the one formed from these vectors, as follows:

$$matLst = \begin{matrix} rowLst & colLst \end{matrix} \begin{bmatrix} 11 & 14 \\ 10 & 14 \\ 10 & 15 \\ 11 & 15 \\ 12 & 15 \\ 12 & 13 \\ 10 & 13 \end{bmatrix}$$

Depending on the computer language and the algorithm, the location pairs may appear in a different order (on different rows of *matLst*), but this arrangement is irrelevant in the execution of the program. We only need a matrix of possible neighboring locations to which the toad can move.

As another example, suppose *aToad.row* is 1 and *aToad.col* is 18 as for the middle cell in Figure 10b, which gives *food* values in the cells; and the call to the function is of the following form:

```
matLst  = getNbrsLst(aToad, 4, false)
```

The initial values for *rowLst* and *colLst* include the corresponding row and column indices for cells to the north, east, south, and west of the toad's current location, as follows:

rowLst = (0, 1, 2, 1)

colLst = (18, 19, 18, 17)

Suppose *SIDE* = 20 so that the *food* and *moisture* grids are of size 20 × 20 and array indices begin with 0. Then, in Figure 10b, the cells in white with *food* values of -1, which neighbor the toad's location, are on the boundary and are out-of-bounds for the toad's next move. For the module's basic simulation, we allow a toad to migrate off the grids to the west, but not to move to a north, east, or south boundary. Thus, we eliminate the location pairs with row 0 (north), column (*SIDE* - 1) = 19 (east), or row (*SIDE* - 1) = 19 (south), as follows:

rowLst = (2, 1)
colLst = (18, 17)

A resulting returned matrix with possible next locations for the toad and the algorithm for *getNbrsLst*() follow:

$$matLst = \begin{matrix} & rowLst & colLst \\ & \begin{bmatrix} 2 & 1 \\ 18 & 17 \end{bmatrix} \end{matrix}$$

Figure 10 Example neighborhoods for middle cell

(a)

moisture

		Column		
		13	14	15
R	10	0.4	0.4	0.2
o	11	1.0	0.4	0.2
w	12	0.4	0.4	0.2

(b)

food

		Column		
		17	18	19
R	0	-1	-1	-1
o	1	0.02	0.03	-1
w	2	0.01	0.02	-1

***matLst* ← *getNbrsLst*(*aToad*, *numNeighbors*, *elimPrevious*)**

Function to return row and column coordinates of toad's neighbors in a matrix

Pre:

SIDE, *ROWLST_AWP*, *COLLST_AWP*, *FENCED* are global variables.

aToad is a toad.

numNeighbors is the number of neighbors, 4, 5, 8, or 9, in neighborhood of the toad as indicated in Table 3.

elimPrevious is a Boolean variable that is true if the previous location is not to be included in the neighborhood and false if the previous location is to be included.

Post:

matLst, a matrix of the row and column coordinates of *toad*'s next possible position, has been returned. The first column contains row coordinates, and the second column contains corresponding column coordinates of these neighbors. If *elimPrevious* is true, the previous location is eliminated. Fenced AWPs are not included as neighbors. Areas on the grid boundary to the north, east, and south are not included

as neighbors. If *numNeighbors* is 4 or 5, a von Neumann neighborhood is used. If *numNeighbors* is 8 or 9, a Moore neighborhood is used. If *numNeighbors* is 5 or 9, the toad's current location is included.

Algorithm:

// get row and column coordinates of all neighbors of (*aToad.row*, *aToad.col*)
if *numNeighbors* is 4
    rowLst ← list of rows for positions to N, E, S, W of (*aToad.row*, *aToad.col*)
    colLst ← list of columns for positions to N, E, S, W of (*aToad.row*, *aToad.col*)
else if *numNeighbors* is 5
    rowLst ← list with *aToad.row* and rows for positions to N, E, S, W
    colLst ← list with *aToad.col* and columns for positions to N, E, S, W
else if *numNeighbors* is 8
    rowLst ← list of rows for N, NE, E, SE, S, SW, W, NW positions
    colLst ← list of columns for N, NE, E, SE, S, SW, W, NW positions
else if *numNeighbors* is 9
    rowLst ← list with *aToad.row* and rows for N, NE, E, SE, S, SW, W, NW positions
    colLst ← list with *aToad.col* and columns for N, NE, E, SE, S, SW, W, NW positions

if length of *FENCED* is positive // eliminate *FENCED* areas
    eliminate pairs formed by *ROWLST_AWP*(*FENCED*) and *COLLST_AWP*(*FENCED*)
        from pairs formed by *rowLst* and *colLst*

eliminate boundary locations to the north, east, and south from *rowLst* and *colLst*

if *elimPrevious* is true // eliminate last location where toad was
    eliminate (*aToad.rowLast*, *aToad.colLast*) from pairs formed by *rowLst* and *colLst*

return *matLst*, a matrix with *rowLst* as the first column and *colLst* as the second column

Quick Review Question 8 Suppose *SIDE* = 100, *ROWLST_AWP* = (75, 61, 34, 91, 53, 22), *COLLST_AWP* = (15, 36, 39, 58, 72, 80), and *FENCED* = (0, 4, 5) as in Quick Review Question 2. For the following situations, give a *matLst* after execution of the following command:

```
matLst  = getNbrsLst(aToad, numNeighbors, elimPrevious)
```

a. *aToad.row* = 76, *aToad.col* = 16, *aToad.rowLast* = 76, *aToad.colLast* = 15, *numNeighbors* = 9, *elimPrevious* = true

b. *aToad.row* = 98, *aToad.col* = 16, *aToad.rowLast* = 97, *aToad.colLast* = 16, *numNeighbors* = 5, *elimPrevious* = true

To complete Rule 8, *CaneToads*() calls *hopForFun*(), which moves the toad by updating its position. With a command of the following form, *hopForFun*() calls to *getNbrsLst*() to obtain the possible neighbors excluding the current position, so that the toad does actually move, but allowing the toad to return to its last location:

```
matLst = getNbrsLst( aToad, NUM_NEIGHBORS, false )
```

One of these possibilities is selected at random for the toad's new position. As the following algorithm indicates, the toad's water and energy values are appropriately decreased with a call to *useWaterEnergyHopping*() before the revised *aToad* is returned.

***aToad* ← *hopForFun*(*aToad*)**
Function to update a toad's location to hop in a random "legal" direction
Pre:
NUM_NEIGHBORS is a global constant.
aToad is a toad.
Post:
aToad.row, *aToad.col*, *aToad.energy*, and perhaps *aToad.water* have been updated after a hop in a random direction.
Algorithm:
// get neighbors, including last place toad was and excluding current position
matLst ← *getNbrsLst*(*aToad*, *NUM_NEIGHBORS*, false)
rowLst ← first column of *matLst* // vector of row indices of neighbors
colLst ← second column of *matLst* // vector of column indices of neighbors

// hop to random neighboring location
loc ← random integer index into *rowLst*
aToad.row ← *rowLst*(*loc*)
aToad.col ← *colLst*(*loc*)

// use energy and perhaps water after a hop
aToad ← *useWaterEnergyHopping*(*aToad*)

return *aToad*

As for a toad, *aToad*, to be "hopping for fun" with *hopForFun*(), looking for moisture with *lookForMoisture*(), or searching for food with *lookForFood*(), first call *getNbrsLst*() to obtain the row and column indices of possible neighbors. With *lookForMoisture*(), the call has the form

```
matLst = getNbrsLst( aToad, NUM_NEIGHBORS, true )
```

which eliminates the current and previous location from consideration. Eliminating these cells helps to prevent the toad from remaining in a moist location or alternating between moist locations that do not diminish in moisture but also do not contain enough water to drink. The call to *getNbrsLst*() from *lookForFood*() has the form

```
matLst = getNbrsLst( aToad, NUM_NEIGHBORS + 1, false )
```

which does not eliminate the current and previous locations. However, *food* values do decrease as toads eat, so the animal is less likely to remain in an uninviting area.

Both *lookForMoisture*() and *lookForFood*() store the current location in (*aToad.rowLast*, *aToad.colLast*) for subsequent time steps and call *hopToMax*() to move in a direction of maximum appropriate resource. While looking for food, the toad may

stay in the same location or may move. Thus, depending on whether it sits or hops, *lookForFood*() calls *useWaterEnergySitting*() or *useWaterEnergyHopping*() to diminish *aToad.water* and *aToad.energy* appropriately. While looking for moisture, the toad always moves, so *lookForMoisture*() calls *useWaterEnergyHopping*(). The algorithm for *lookForFood*() follows, and the algorithm for *lookForMoisture*() is similar except as just noted:

***aToad* ← *lookForFood*(*aToad*)**
Function to update toad location so that it hops to neighbor with most food
Pre:
food and *NUM_NEIGHBORS* are global constants.
Post:
aToad.row, *aToad.col*, *aToad.energy*, and perhaps *aToad.water* have been updated after goes to a neighbor, including current site, with maximum food.
Algorithm:

```
// get row and columns of neighbors, including current site
matLst ← getNbrsLst( aToad, NUM_NEIGHBORS + 1, false )
rowLst ← first column of matLst       // vector of row indices of neighbors
colLst ← second column of matLst    // vector of column indices of neighbors

// store current cell as last position
aToad.rowLast ← aToad.row
aToad.colLast ← aToad.col

// hop to "legal" neighboring location with maximum food
(aToad.row, aToad.col) ← hopToMax(rowLst, colLst, food)

// use energy and perhaps water
if (aToad.row is aToad.rowLast) and (aToad.col is aToad.colLast)   // toad is sitting
    aToad ← useWaterEnergySitting(aToad)
else     // toad has moved
    aToad ← useWaterEnergyHopping(aToad)

return aToad
```

While looking for moisture or food, the toad moves to a permissible neighborhood cell with the maximum amount of the desired resource. If more than one of these neighbors has this maximum amount, the toad selects one of these cells at random. For the example in Figure 10a, the maximum moisture amount of 0.4 occurs in acceptable neighbors to the north, southwest, and northwest of the toad. Thus, *lookForMoisture*() has the toad hopping at random to one of these locations. The functions *lookForMoisture*() and *lookForFood*() call *hopToMax*(), whose algorithm follows, to obtain the row and column for the toad's next grid location.

(row, col) ← hopToMax(rowLst, colLst, gridMat)
Function to return the row and column of new toad location with maximum grid value

Pre:
rowLst is a list of row coordinates for neighbors.
colLst is a list of corresponding column coordinates for neighbors.
gridMat is a 2D matrix, such as *food* or *moisture*, of values.
Post:
The function has returned the row and column of the new toad position, where the new position has the maximum *gridMat* value, selected at random in case of ties.
Algorithm:

```
// find grid levels of neighbors
for i going sequentially through indices of rowLst
    valuesAround(i) ← gridMat(rowLst(i), colLst(i))

maxValue ← maximum value in valuesAround
dirLst ← vector of indices i where gridMat(rowLst(i), colLst(i)) is maxValue

if the length of dirLst is more than 1
    ndx ← randomly selected value from dirLst
else     // only one direction has maximum value
    ndx ← only value in dirLst

row ← rowLst(ndx)
col ← colLst(ndx)

return (row, col)
```

The functions *useWaterEnergyHopping*() and *useWaterEnergySitting*() reduce the energy level of a toad, *aToad.energy*, by the amount *ENERGY_HOPPING* or half of *ENERGY_HOPPING*, respectively. If the toad is in a moist or wet area, we assume that its fluids do not diminish. For instance, many amphibians seek moist humid burrows to avoid desiccation. If, however, the toad is in a dry area, which is one with moisture value less than *MOIST_AREA*, *useWaterEnergyHopping*() decreases *aToad.water* by the amount *WATER_HOPPING*. In the case of diminishing water or energy, the toad will die before the danger of *aToad.water* or *aToad.energy* becoming negative. The algorithm for *useWaterEnergySitting*() is similar to that of *useWaterEnergyHopping*(), which follows, except subtraction is 50% of that involving a hop.

aToad ← useWaterEnergyHopping(aToad)
Function to update a toad's energy and perhaps water values after a hop
Pre:
moisture, *MOIST_AREA*, *WATER_HOPPING*, *ENERGY_HOPPING* are global variables.
Post:
aToad.energy has been reduced.
aToad.water has been reduced if the toad is not in a moist area or water.
Algorithm:

```
if moisture(aToad.row, aToad.col) < MOIST_AREA
    aToad.water ← aToad.water - WATER_HOPPING    // use water hopping
aToad.energy ← aToad.energy - ENERGY_HOPPING  // use energy hopping

return aToad
```

After eating, drinking, and moving, *CaneToads*() calls *changeState*() to adjust the state of a toad as necessary to be *DEAD* or *MIGRATED*, as follows:

```
aToad ← changeState(aToad)
    Function to perhaps change toad's state to dead or migrated
Pre:
    DESICCATE, STARVE, DEAD, and MIGRATED are global constants.
Post:
    If aToad has desiccated or starved, aToad.alive has been changed to DEAD and its
        position has been changed to (0, 0).
    If aToad has migrated off the grid to the west, aToad.alive has been changed to
        MIGRATED.
Algorithm:
if aToad.water < DESICCATE or aToad.energy < STARVE
    aToad.alive ← DEAD
    aToad.row ← 0
    aToad.col ← 0
else if aToad.col is 1
    aToad.alive ← MIGRATED

return aToad
```

Display Simulation

An animation can help us understand the behavior of toads in the environment. For each time step, we display the *food* grid in grayscale. For this scale, 0 results in black and 1 in white. To have the higher values appear darker, we subtract each food value from the maximum food value, *maxFoodValue*, and divide by *maxFoodValue* to obtain values between 0 and 1 indicating its degree of gray. For example, for a maximum food value, *maxFoodValue*, of 0.03, a cell's food value of 0.01 converts to a grayscale value of (0.03 - 0.01)/0.03 = 0.02/0.03 = 0.667, which displays as light gray. In contrast, a relatively high food value of 0.027 has a grayscale value of (0.03 - 0.027)/0.03 = 0.003/0.03 = 0.100 that appears as dark gray in a visualization. We also alter the border elements of each *food* grid so that they appear in white.

Many computational systems visualize the first matrix row on the bottom of the display. For these systems, we flip the matrix from top to bottom before visualization so that the graphics image is easier to compare with the matrix.

Superimposed on a visualization of each *food* grid, we display moist areas in yellow, AWPs in blue, and a fenced AWP with a red outline. After generating the

background for a time step, the toads are displayed as green dots. The sequence of visualizations of the outcomes for the time steps results in an animation.

Multiple Simulations

The stochastic nature of this model is such that we should not take any one simulation as indicative of what will happen in general. Thus, we also develop a function *multipleSimulations*() that calls *CaneToads*() a designated number of times, *numSimulations*. At the end of each simulation, a function *getNumAlive*() counts and returns the number of alive toads (number of *i* where *toads*(*i*).*alive* is *ALIVE*), number of dead toads (number of *i* where *toads*(*i*).*alive* is *DEAD*), and number of toads that have migrated (*MIGRATED*). Ongoing totals (*totalAlive*, *totalDead*, and *totalMigrated*) accumulate these counts, and finally *multipleSimulations*() returns the average of each total, or the totals divided by *numSimulations*.

4. Software Implementation

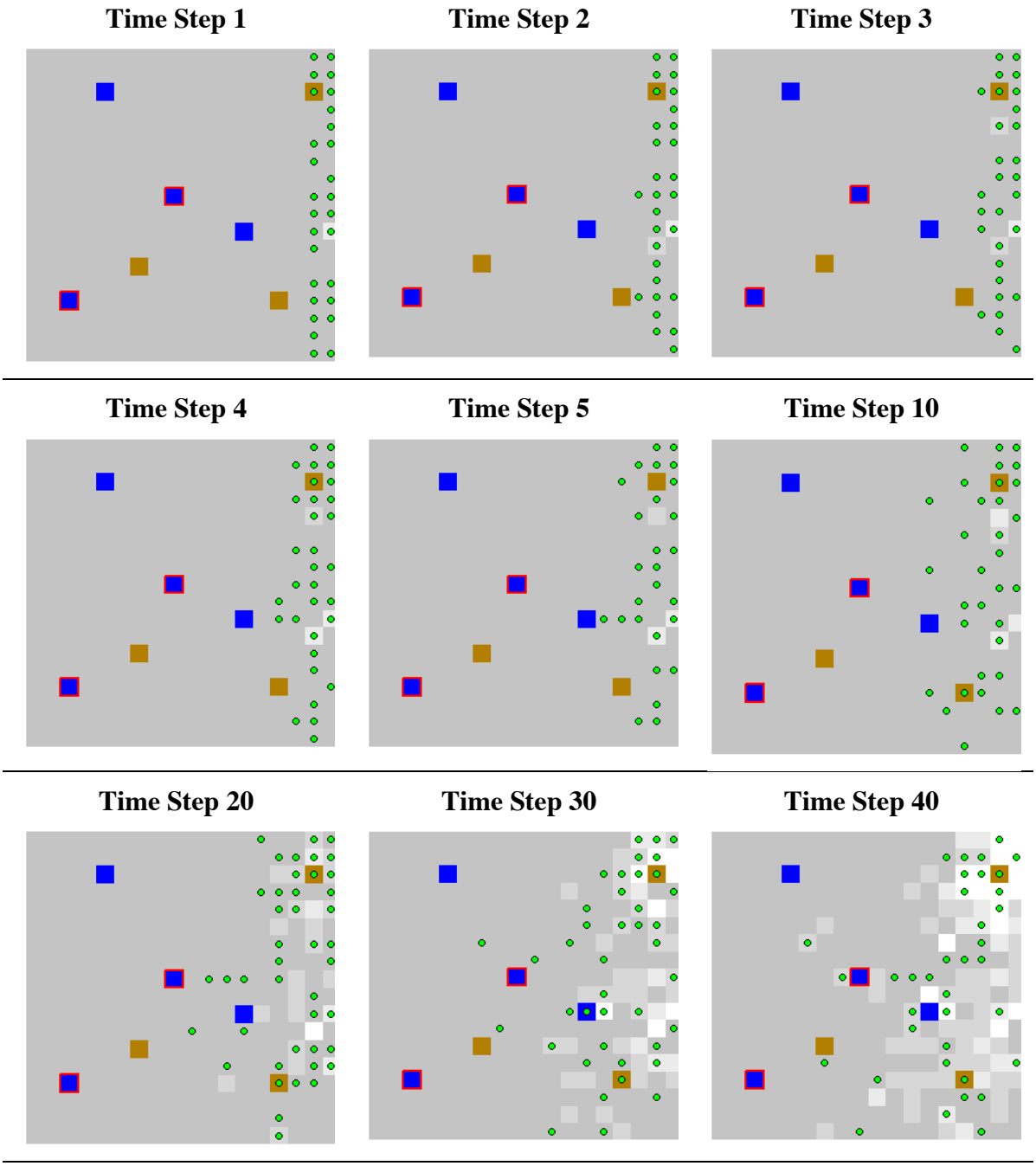
Associated files available for download contain sequential implementations in MATLAB and C and a parallel implementation in C with MPI.

5. Example Problem

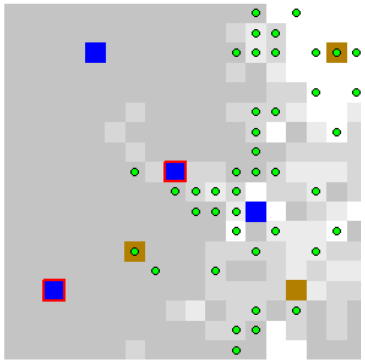
Visualization

With Figure 8 containing the initial configuration, Figure 11 contains several frames of an animation of a simulation with constant definitions as in Table 1, *NUM_NEIGHBORS* = 8, uniform initial food of *FOOD_CELL* = 0.03, and *FENCED* = (2, 3). Initially all toads are in column 19, and Figure 12 contains the row positions. After 300 time steps, 8 of the 50 original toads remain alive; 24 are dead; and 18 toads have migrated off the grid to the west. Notice, in time steps 81-86, a toad that needs water travels around the leftmost fenced AWP, occasionally eating and "frantically" looking for water. The toad "smells" that water is close but cannot get to the AWP. Similarly, at time step 300, all eight surviving toads, some of which share the same cells, are surrounding fenced AWPs in cells that have no food. The need for water and the non-diminishing *moisture* values are keeping them in the area.

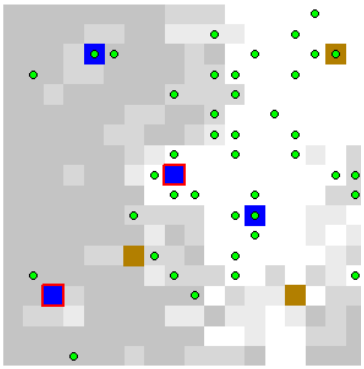
Figure 11 Several frames of the animation of one simulation



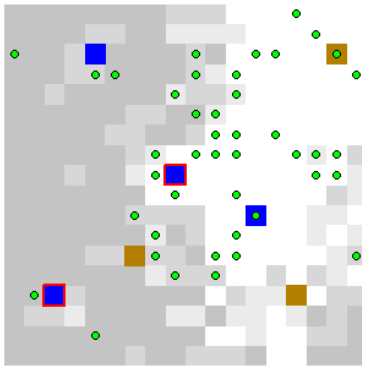
Time Step 50



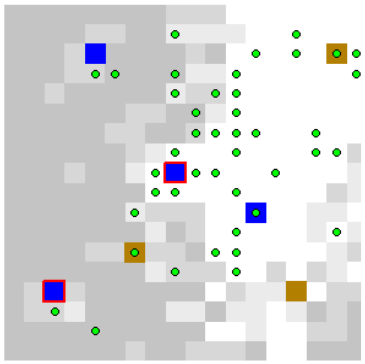
Time Step 81



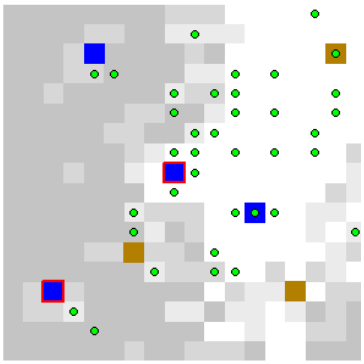
Time Step 82



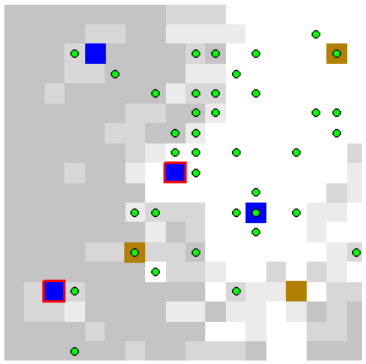
Time Step 83



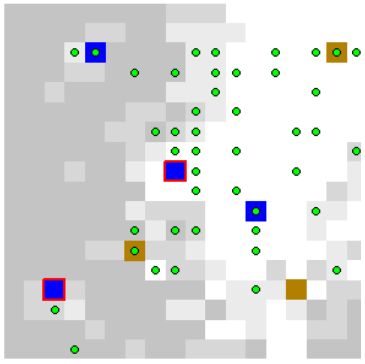
Time Step 84



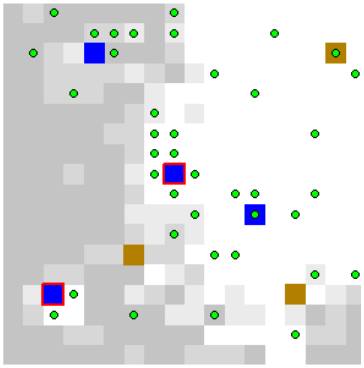
Time Step 85



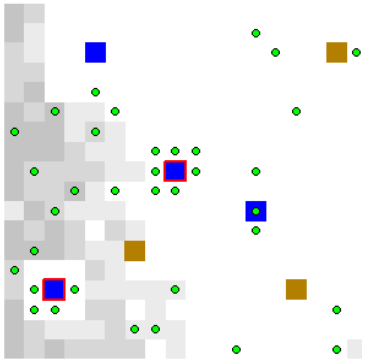
Time Step 86



Time Step 100



Time Step 150



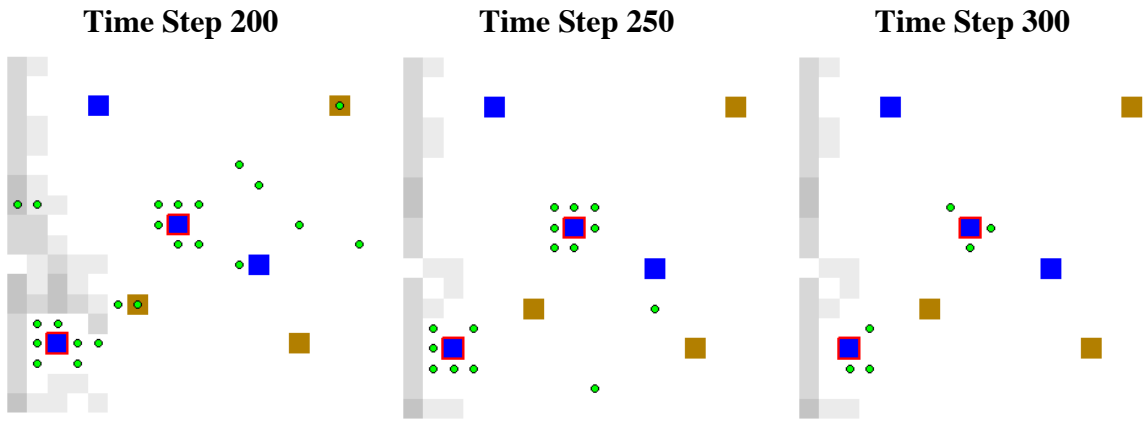
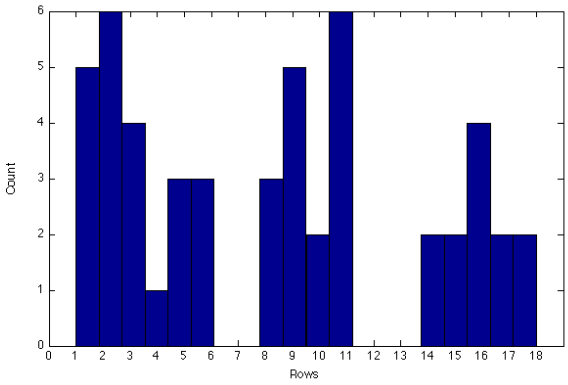


Figure 12 Initial row distribution of toads, which are in column 19, for simulation



Averaging Results of Multiple Simulations

The function call *multipleSimulations*(1000) invokes the simulation *CaneToads*() one-thousand times and returns the average number of toads that are alive, dead, and migrated after 300 time steps. For the values indicated for the simulation above with *FENCED* = (2, 3), one such call to *multipleSimulations*(1000) returns the following averages: 6.465 (12.93%) alive, 28.601 (57.20%) dead, and 14.934 (29.87%) migrated. Interestingly, we obtain different results depending on which AWPs we fence. Table 4 presents results, averaging over 100 simulations, for several fencing configurations. With one fenced AWP, the greatest impact occurs with fencing the AWP that is closest to the direction of initial invasion; and fencing all AWPs results in almost complete fatality.

Table 4 Results of *multipleSimulations*(100) using constants of Table 1, *NUM_TOADS* = 50, *NUM_NEIGHBORS* = 8, *FOOD_CELL* = 0.03, and indicated fenced AWPs at the following (row, column) positions: AWP 0 - (3, 5), AWP 1 - (11, 13), AWP 2 - (15, 3), AWP 3 - (9, 9)

0	1	2	3	Alive	Dead	Migrated

				3.45	9.95	36.60
x				5.36	18.25	26.39
	x			4.85	38.23	6.92
		x		6.50	17.04	26.46
			x	4.68	23.32	22.00
x	x	x	x	0.63	49.26	0.11

In another simulation, we release 21 toads along the east border of an enclosed area that does not allow migration and that has only one AWP in the middle. Averaging the results of 100 simulations, Table 5 shows almost all the toads (99.14%) survive a 12-hour night if we have no fencing around the AWP. However, when we fence this AWP, on the average just over 1 toad (6%) survives the night.

Table 5 Results of *multipleSimulations*(100) using constants of Table 1, *NUM_TOADS* = 21, *NUM_NEIGHBORS* = 8, *FOOD_CELL* = 0.03, and indicated fenced AWPs at the following (row, column) positions: AWP 0 - (9, 9)

0	**Alive**	**Dead**
	20.82 (99.14%)	0.18 (0.86%)
x	1.26 (6.00%)	19.74 (94.00%)

6. Assessment of Model

The results do show the impact of fencing all or some AWPs for a particular situation. Moreover, the toads do migrate from the direction of release on the east to the west. A search for food and water with diminishing food availability to the east drives the invasion front westward.

Our simulated results for a 12-hour period listed in Table 5 show general agreement with the experimental release of cane toads in an enclosed area with one AWP. In that experiment, with an unrestricted AWP and a control group of 20 toads, 19 toads survived a 72-hour period. Predation, which our simulation does not consider, accounted for the one toad's death. Our simulation also agrees with another part of the experiment in which 20 of 21 toads died overnight in an enclosed area with a fenced AWP (Florance *et al*. 2011).

However, it is probably unrealistic to consider our simulated food resources being depleted quite so completely without additional prey moving into the area. For instance, if we allow the simulation of the enclosed area with 20 toads to run for 900 time steps (three 12-hour nights), all toads die as they eat all the food. Even if we make the initial constant food amount (*FOOD_CELL*) 1.0 in the simulation with 50 toads and four AWPs, we obtain similar results to Table 4 for fencing AWP 1 with the average alive being 10.82, dead 33.48, and migrated 5.70 at the end of 300 time steps. Food still becomes completely depleted around the fenced AWP.

The simulation of this module makes many other simplifying assumptions. We have not considered the impact of climate, particularly wet and dry seasons and

temperature. Moreover, we have not adequately addressed the adult cane toad's inclination to sleep in moist places during the day and to travel, often to water, during the night. Our simulated toads are attracted to moist areas, but do not rest there for lengthy periods of time.

Additionally, we have used an arbitrary landscape for the simulation. A professional quality simulation used to make predictions upon which to take actions should probably incorporate a map of the distribution of permanent waters and AWPs, as in (Florance *et al*. 2011). With their experiments and their model that uses such a map as well as climate data, (Florance *et al*. 2011) predicts "that systematically excluding toads from AWP would reduce the area of arid Australia across which toads are predicted to disperse and colonize under average climatic conditions by 38 per cent from 2,242,000 to 1,385,000 km^2."

7. High Performance Computing Implementation

Why Parallelism?

Use of climate data, terrain maps, finer grain and more extensive grids that cover a continent, a significantly larger number of toads, and running numerous simulations to average the results add considerably to the computational burden. In such situations to be able to obtain meaningful results in a reasonable amount of time, the modeler can employ **high performance computing** (**HPC**), which uses multiple processes working in parallel.

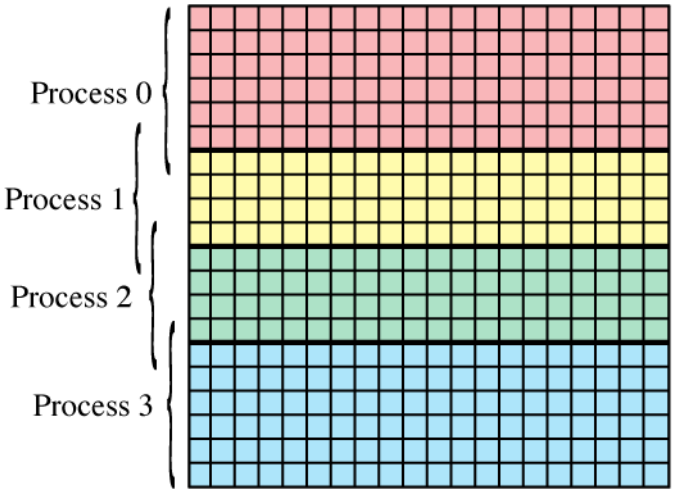
Approach to Parallelizing

Many approaches exist for running this simulation with multiple processes using high performance computing. When running a simulation multiple times to obtain, for example, the mean number of alive, dead, and migrated toads for a certain situation, we can have each process run several of the simulations and then gather their results. For example, for 100 simulations with 4 processes, each process can execute 25 simulations; and the master process can collect the individual results and calculate the means.

We can also parallelize the code for one simulation. For a language, such as C, that stores matrices in row-major order, a row at a time, one approach is to store several contiguous rows of the *moisture* and *food* grids on each process. For example, Figure 13 displays a 20 × 20 grid arrangement with Process 0 storing and managing rows 0-5; Process 1, rows 6-9; Process 2, rows 10-13; and Process 3, rows 14-19. Moreover, each process stores and manipulates the toads that are in its purview. Because the first and last rows of the grids store constant values and are areas in which toads do not travel for this simulation, for load balancing we might designate that Processes 0 and 3 each manage six rows, two of which have no activity, while Processes 1 and 2 manipulate 4 rows. Projects explore alternative partitions. For appropriate movement, toads on the local borders must examine rows managed by other processes. For example, a toad in row 5 is on Process 0 but must consider *moisture* and *food* cells in row 6, which is on Process 1, when looking for water and prey. Thus, Process 0 also stores, but does not manipulate,

row 6 of the *moisture* and *food* matrices. Similarly, Process 1 stores rows 5 and 10 for examination purposes only, while Process 0 is responsible for row 5 and Process 2 controls row 10. At each time step, the functions *eat*() and *drink*() are the only ones that alter the *food* matrix. Thus, it is appropriate to copy modified local border cells of *food* immediately after toads eat and drink and before they move. Because the *moisture* matrix does not change, we only need to communicate its local boundary values once, at the beginning of the simulation. After animal movement in each iteration, we should transfer control to the appropriate process of any toad that has moved from one grid segment to another.

Figure 13 One partition of 20 × 20 environment among 4 processes



Besides communication and partition of the grids and the toads, we should be careful of random number generation, which each process must do, to avoid different processes using the same random number sequence. HPC systems with MPI enabled architecture often have available a function library, such as **SPRNG**, for parallel random number generation.

Speedup and Scalability

Table 6 and Figure 14 present timings for a parallel version of the program (*caneToadsMPI-v1.c*) that has different processes perform complete simulations. The program employs 20 × 20 food and moisture grids using 50 or 500 toads. With each process running separate simulations, the most improvement occurs with the larger number of toads. For 500 toads, doubling from 2 to 4 processes almost cuts the execution time in half. However, the execution time is only diminished by one-third when employing 8 instead of 4 processes. We continue to see improvement as the number of processes increases, but communication hurts the speedup.

Table 6 Timings for a parallel version of the program (*caneToadsMPI-v1.c*) with 20 × 20 grids that which divides simulations among processes

# Simulations	# Toads	2 Processes	4 Processes	8 Processes
100	50	0:01.76	0:01.36	0:01.28
100	500	0:05.99	0:03.15	0:02.11

Figure 14 Graph of timings from Table 6

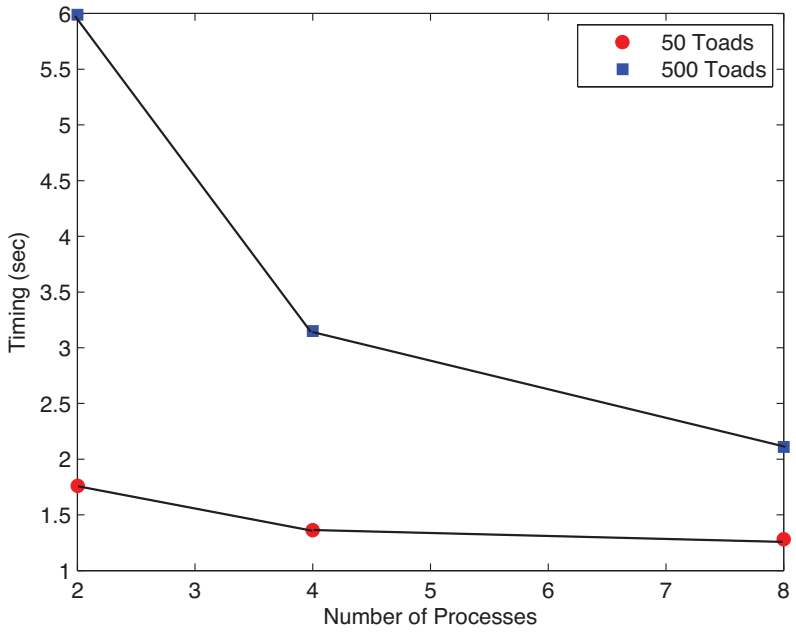
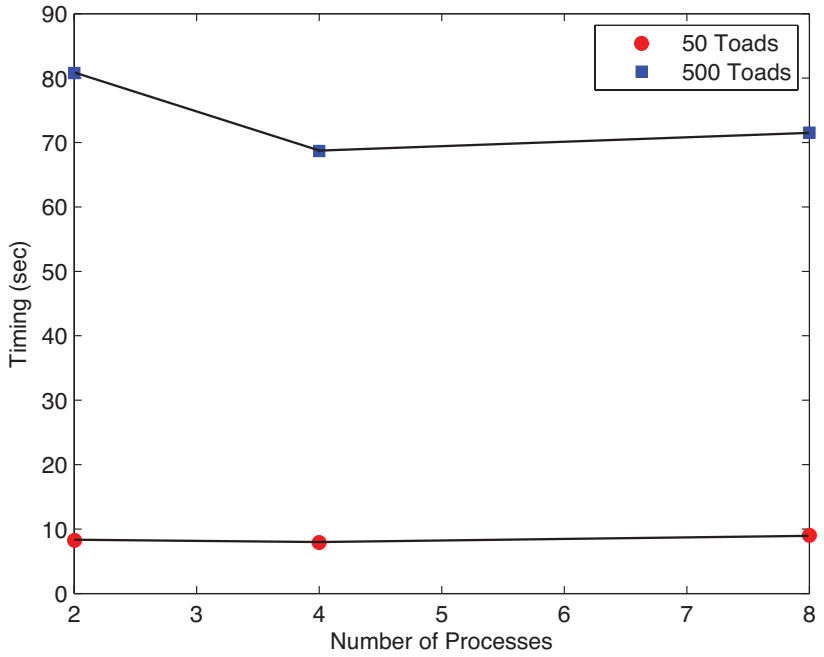


Table 7 and Figure 15 show timings for another version of the program (*caneToadsMPI-v2.c*), one that parallelizes a single simulation by splitting the grids and toads among the processes. Although we do see some improvement when using 4 instead of 2 processes, unfortunately, the execution time increases when using 8 processes. The communication of grid and toad information between processes limits the utility of using multiple processes. Thus, for 20 × 20 grids with a large number of toads, the first parallel version, which has processes running complete simulations, seems most advantageous.

Table 7 Timings for a parallel version of the program (*caneToadsMPI-v2.c*) with 20 × 20 grids that divides simulations among processes

# Simulations	# Toads	2 Processes	4 Processes	8 Processes
100	50	0:08.24	0:07.93	0:08.99
100	500	1:20.79	1:08.68	1:11.54

Figure 15 Graph of timings from Table 7



Additional Considerations

Other issues, examined in the projects, are appropriate to consider in the high performance computing implementation of this problem. For example, is it more efficient to implement parallelism by partitioning the physical grid among processes or by distributing toads among processes? Does each process always have the exact same amount of work to do, or can load balancing issues arise if, say, certain portions of the grid have more water and/or food sources than others? How does the parallel code scale as the physical problem increases? What are potential performance bottlenecks, and how might the parallel code be re-written to address these?

8. Projects

Develop the following projects using sequential programming or high performance computing, except as noted.

For Projects 1-9, adjust the simulation of this module as indicated. Visualize one simulation and perform multiple simulations to determine the mean number of toads that are alive, dead, or migrated under various fencing configurations. Discuss the impact of the change on the results.

1. Adjust the model so that toads are released at random times and not all at once.
2. Adjust the model to take into account density dependence, such as not allowing more than a certain number of toads to occupy a cell at the same time.
3. Adjust the model to allow for rapid food regeneration.

4. Write and incorporate a rule to help prevent toads from remaining indefinitely around a fenced AWP. For example, you might write a rule where occasionally a very thirsty toad might be allowed hop in a random direction or in the direction of minimum moisture.

5. Initialize the *food* matrix so that each internal cell is initialized with a normally distributed value with mean *MEAN_FOOD* and standard deviation *STD_FOOD*. However, restrict each value to be between 0.0 and 1.0. Using a diffusion algorithm as described in sections "Diffusion Model" and "Diffusion Algorithm" of the module "Biofilms: United They Stand, Divided They Colonize," diffuse food at each time step (Shiflet and Shiflet 2009). Visualize the results. Does the change have an impact on depletion of food around fenced AWPs?

6. The section "Sensitivity Analysis" of the module "Time after Time: Age- and Stage-Structured Models" provides the background for this project (Shiflet and Shiflet 2010). For several constants in Table 1, perform a sensitivity analysis to determine how sensitive death percentage is to each of the constants with value under one in Table 4.

7. Have a moisture grid that includes a larger body of permanent water, such as a river.

8. Explore the difference between dry and rainy seasons on the effectiveness of fencing AWPs. For rainy seasons, the moisture grid has more small puddles and larger ponds that tend to evaporate between rainstorms.

9. Develop a simulation for a random number of AWPs and moist areas in random locations, a random selection of fenced AWPs, and random number of toads for a grid with $SIDE \geq 100$.

10. Employing a HPC implementation, partition the grids into submatrices of size $p \times q$ instead of submatrices involving entire grid rows. Compare timings of this configuration and the one discussed in the module.

11. Using distributions of movement patterns, (Schwarzkopf and Alford 2002; 2005) developed individual-based, correlated random walk models to ascertain if adult cane toads move nomadically. At each iteration, a simulated toad does the following:

- Decide if it is going to move or not
- If it decides to move, decide if it returns to its most recent previous location
- If it decides to move but not return to its most recent previous location, decide the angle to turn and the distance to move

With this algorithm, develop a simulation following the movements of 10,000 toads and using a simulation time of 30 days with a time step of one-half a day. Consider each of three seasons (wet, late wet, dry) using movement parameters from those in Table 8, which are similar to values obtained from observing a group of tracked cane toads in the Heathlands region of Australia. Assume the wet season lasts four months, late wet lasts two, and dry lasts six months. Ignoring death, food, and moisture, do not use any grids. For each season, compute the following path characteristics: Mean total distance traveled, mean total displacement, and mean **straightness**, which is the total displacement divided by total distance traveled.

Table 9 gives approximate mean straightness and mean total displacement per day for a group of traced cane toads in the region. Discuss your results, including when toads tend to move straighter and further. Hypothesize reasons for these results.

Table 8 In three seasons (wet, late wet, dry), probabilities that a toad moves; if moves, that it returns to most recent previous location; if moves but does not return, that it goes various distances at various angles. Values approximated for Heathlands from (Schwarzkopf and Alford 2002)

Probability	Wet Season	Late Wet Season	Dry Season
moves	0.80	0.79	0.69
returns	0.03	0.10	0.10
distance 0 m	0.11	0.14	0.18
distance 5 m	0.25	0.34	0.16
distance 15 m	0.24	0.27	0.17
distance 35 m	0.16	0.15	0.18
distance 75 m	0.14	0.08	0.17
distance 155 m	0.05	0.02	0.12
distance 315 m	0.02	0.00	0.01
distance 635 m	0.02	0.00	0.01
distance 1275 m	0.01	0.00	0.00
angle -170°	0.12	0.22	0.28
angle -150°	0.03	0.08	0.05
angle -130°	0.03	0.07	0.03
angle -110°	0.03	0.02	0.03
angle -90°	0.03	0.03	0.03
angle -70°	0.03	0.03	0.03
angle -50°	0.02	0.02	0.03
angle -30°	0.07	0.02	0.03
angle -10°	0.06	0.02	0.04
angle 10°	0.05	0.04	0.02
angle 30°	0.06	0.02	0.03
angle 50°	0.06	0.03	0.02
angle 70°	0.06	0.02	0.04
angle 90°	0.03	0.04	0.02
angle 110°	0.05	0.04	0.04
angle 130°	0.07	0.04	0.03
angle 150°	0.05	0.08	0.05
angle 170°	0.15	0.18	0.20

Table 9 In three seasons (wet, late wet, dry), mean straightness and mean displacement (m) per day. Values approximated for Heathlands from (Schwarzkopf and Alford 2002)

Mean	Wet Season	Late Wet Season	Dry Season
Straightness	0.33	0.20	0.09

Displacement (m)	36	22	15

For Projects 12-16, use the information in the project description and the "Introduction" section, as necessary.

12. Sometime between September and March, Australian cane toads work their way toward aquatic sites to breed. Because the toads are so widely distributed, from New South Wales to the Northern Territory, the timing depends on the particular climatic zone and habitat conditions. The water can be temporary or permanent, brackish or fresh, but they prefer relatively clear water with rather neutral pH and sufficient emergent, submergent, and/or floating plants for cover to lay their eggs. As noted in the introduction, clutch size varies widely, but it is correlated positively with the size of the female.

If a female cane toad lays 10,000 eggs, about 7000 will survive to produce tadpoles. Tadpoles live normally for 10 to 100 days (average = 38), and the wide range results from various environmental factors (e.g., temperature), food levels, and density. For instance, development/growth rate can be delayed by high levels of intraspecific competition for food.

Tadpole survival is strongly density dependent (Hearnden 1991), but is also influenced by predation from older cohorts of *B. marinus* tadpoles, climatic conditions, and food levels. Field data suggests that the maximum survival for tadpoles (σ_{tmax}) to be about 0.35 tadpoles/L. Using a coefficient of intraspecific competition ($d = 0.5771$ tadpoles/L), we can fit the following function to field data to reveal the relationship between tadpole survival (σ_t) and initial density (T).

$$\sigma_t(T) = \sigma_{tmax}/(1 + d\,T) \qquad \text{(Lampo and de Leo 1998)}$$

Predation by older tadpoles can reduce survival from 88% to 1.7% (Hero and Stoneham 2005). Surviving tadpoles become **metamorphs** (**toadlets**), which must make the transition to a terrestrial lifestyle.

a. Develop an individual-based model with a food grid to simulate development of tadpoles in a pond. Initialize tadpoles in the pond to be of a variety of reasonable ages and locations. Running the simulation a number of times, determine the mean number of tadpoles surviving to become toadlets for various densities. Attempt to adjust parameters to match $\sigma_t(T)$. Indicate simplifying assumptions you make, and discuss your results.

b. Using the values from Part a, have tadpoles emerge at random locations and times around the edge of the pond. Running the simulation a number of times, determine the mean number of tadpoles surviving to become toadlets.

13. Tadpoles may survive at temperatures between 17°C and 42°C, with maximum survival at 29°C. Refine Project 12b to take into account the impact of pond temperature on tadpole survival. For simplicity, assume the water temperature is the same throughout the pond but is 1-3°C lower at night. Run the simulation for low, high, and optimum temperatures.

14. Because toadlets (see Project 12) are initially quite small (9-13 mm) and lack the extreme toxicity of other life stages, they are quite vulnerable to predation. Metamorphs grow very rapidly at first (0.647 mm/day (Zug and Zug 1979)), but the rate of growth is density-dependent. The earliest metamorphs are generally found within 1 M of the water (Cohen and Alford 1993). Survival is influenced by desiccation and predation, varying from 1.2 – 17.6% (Lampo and de Leo 1998). Susceptibility to desiccation is reduced with increased numbers of retreat sites available to the toadlets.

a. Develop an individual-based simulation of toadlets near a pond that includes a moisture grid and predators and that does not allow the toadlets migrate off the grid. Attempt to adjust parameters so that survival is as indicated. Running the simulation a number of times, determine the percent that survive and the mean toadlet size at the end of one year. Indicate your assumptions and discuss your results.

b. Produce an animation of a simulation.

15. Surviving toadlets are considered **juveniles** at 1 year (Lampo and de Leo 1998) and become breeding adults at 2 years. Adult survival depends on a number of environmental factors, especially desiccation. Toads obtain much of their water from their prey (~69%) and lose water via evaporation, respiration, and excretion (Kearney *et al*. 2008). Although these animals can sustain substantial water loss, if they lose ≥ 40% of their body mass, they are much more likely to die of dehydration (Florance *et al*. 2011). Adult survival rates vary between 30 and 70% (Lampo and de Leo 1998). Juveniles are assumed to have only 10% of the adult survival rates.

a. Using this information, develop a grid-based individual-based simulation involving juvenile and adult cane toads. Initialize the grid with juveniles and adults in random locations and juveniles of random ages. Have new juveniles entering the simulation at random times from around a pond. Have new adult toads entering the simulation at random times from grid boundaries. Allow toads to migrate out of the area in any direction. Because young toads are primary colonizers, young adults should be more likely to move than older toads and young toadlets. Running the simulation a number of times, determine the mean number of juveniles and adults that survive, die, and migrate.

b. Produce an animation of a simulation.

16. Scientists continue to search hard for effective measures to restrain cane toad populations. Dr. Rick Shine, a biologist at the University of Sydney, and his colleagues are experimenting with various control measures. One of Professor Shine's honor students, Georgia Ward-Fear, has come upon a remarkable possibility. Toadlets of this species, unlike those of other anuran species, are active by day. To avoid desiccation, they confine themselves to the areas around water. It so happens that these sites are also favored by a species of meat ant (*Iridomyrmex reburrus*), for foraging. Also, most other species of young frogs will hop away to avoid ants—not cane toadlets. Their ancestors never had to deal with such large, predatory ants, so the escape behavior has not evolved. Thus the toadlets often

provide a nutritious morsel for the ants, and these predators successfully reduce the young toad population (up to 90%) (Ward-Fear *et al*. 2009).

Develop a simulation that contains a grid with water, land, toadlets, and meat ants. The toadlets can stay where they are or move in random directions, but they stay close to or in water. Meat ants remain on land. When a meat ant is adjacent to a toadlet on land, with a certain probability the ant "eats" the toadlet, that is, the toadlet disappears from the simulation. Perform an animation of the simulation. Plot the number of toadlets versus time.

For the following projects, develop a simulation with an animation. Also, perform multiple simulations to determine the mean number of toads that are alive, dead, or migrated. Discuss the results.

17. Temperature can have a big impact on migration of *Bufo marinus* as the animal favors warmer weather but tends to desiccate faster under such conditions. The threshold temperatures for population growth are estimated as 14°C and 40°C, while the optimal temperature range for population growth is estimated as 31°-35°C (Sutherst *et al*. 1996). Incorporate a temperature gradient grid into your simulation, where temperatures are cooler to the south and gradually warm for cells further north, as generally happens in Australia. Write and incorporate rules using this grid. Have the toads released either gradually or all at once from the middle part of the south border and allow them to migrate off the grid anywhere. Save the temperatures where toads die and migrate, and display two histograms of the numbers of dead and migrated toads versus temperature. Also, use food and moisture grids.

18. Through experimentation and curve fitting, (Kearney *et al*. 2008) developed the following model for the hopping speed, S in km/hr, of the cane toad as a function of its core body temperature, T_b in °C, from 15°C to 35°C:
$S = -25.48396 + 4.51222\ T_b - 0.29052\ T_b^2 + 0.0082619\ T_b^3 - 0.000086431\ T_b^4$
Core body temperature is directly proportional to air temperature, and the two temperatures are almost equal on rainy nights. Moreover, the scientists estimated the proportion of time a toad moves as having a median of 3.84% and interquartile (middle) range of 1.4 to 6.8%. They assumed activity is limited only by temperature and not rainfall. Their analysis predicted foraging rates generally less than 1 g/hr throughout the present range. Assuming that a diet of crickets has 69% water content, they also predicted that toads would need to drink more than 1 L/yr in cooler areas and 10 L/yr in arid northern regions. Incorporating these findings from (Kearney *et al*. 2008), develop a grid-based individual-based model of the spread of toads.

19. Extend the previous project to account for cane toad metabolic rates. With experimentation and curve fitting, (Kearney *et al*. 2008) derived a formula for resting metabolic rate, M in watts or joules (J) per second, as a function of core body temperature, T_b in °C, and *mass* in grams, as follows:

$$M = 0.0056 - 10.0^{(0.038\ Tb\ -\ 1.771)}\ mass^{0.82}$$

Field metabolic rates for active toads was assumed to be 2.5 that for resting toads in the laboratory. Moreover, (Kearney *et al*. 2008) determined that a diet of crickets has an energy density of about 6.3 kJ/g wet mass and that cane toads can assimilate about 85% of this amount, or 5.355 kJ/g = 5355 J/g. For instance, a 120 g with body temperature 25°C has resting metabolic rate $M = 0.0429$ watts = 0.0429 J/sec. Dividing by the assimilated energy density of crickets, we find that this resting cane toad requires about 8×10^{-6} g/sec. Assume cane toad body masses between 50 g and 500 g. (Letnic *et al*. 2008) indicates that most of the cane toads in colonizing-front populations in the Northern Territory are adults. Moreover, the scientists estimated mean masses of 170 g for males and 290 g for females with some as large as 2 kg.

20. For free ranging cane toads, (Halsey and White 2010) obtained estimates of energetics, such as the rate of the change the volume of oxygen in the blood (rate of energy expenditure or metabolic rate), $\dot{V}O_2$ in ml/hr, calibrated to overall dynamic body acceleration (*ODBA*) in g, a metric for body motion. Using data and curve fitting, they developed the following equation for $\dot{V}O_2$ as a function of *ODBA* and *mass* in g at 25°C body temperature:

$$\dot{V}O_2 = 555.9\ ODBA + 0.372\ mass - 19.98$$

6.4 ml O2 hr^{-1} represents about 3200 J/day = 0.037 J/sec. Assume such a relationship is proportional. Moreover, (Kearney *et al*. 2008) determined that a diet of crickets has an energy density of about 6.3 kJ/g wet mass and that cane toads can assimilate about 85% of this amount, or 5.355 kJ/g = 5355 J/g. Dividing by the assimilated energy density of crickets, we find that a cane toad with $\dot{V}O_2$ of 6.4 ml/hr requires about 6.9×10^{-6} g/sec.

Studying eight cane toads deployed in the field, they determined the following values ± **standard error of the mean** (**SEM**):

- Mean body mass = 136 ± 13 g; minimum mass = 97 g; maximum = 204 g
- Mean *ODBA* over recording time = 0.0384 ± 0.0044 g; minimum = 0.0232 g; maximum *ODBA* = 0.054 g
- Maximum *ODBA* over 5 min. = 0.086 ± 0.016 g

They also determined the following proportions (percentages) of recording time:

- Proportion spent resting = 84.0%
- Proportion spent in low activity behavior = 13.87 ± 2.3 %
- Proportion spent hopping = 2.10 ± 0.7 %

Toads typically hop for less than 5% of the time, moving on the average 4% of the time at a rate of 18 m/hr.

Incorporating these findings from (Halsey and White 2010), develop a grid-based individual-based model of the spread of toads.

21. Extend the previous project to account for temperature as in Project 12. The equation estimated the rate of the change the volume of oxygen in the blood, $\dot{V}O_2$, for cane toad with body temperature 25°C (Halsey and White 2010). To obtain the rate at other body temperatures, we can employ a Q_{10} correction, as follows:

$$Q_{10} = \left(\frac{r_2}{r_1} \right)^{\frac{10}{t_2 - t_1}}$$

where r_1 and r_2 are metabolic rates, and t_1 and t_2 are corresponding temperatures. Thus, using the formula for $\dot{V}O_2$ from the previous project to estimate $r_1 = \dot{V}O_2$ at t_1 = 25°C and knowing the toad's body temperature t_2 and Q_{10}, we can calculate its r_2, or metabolic rate, at t_1. A Q_{10} of 2 results in a doubling of the metabolic rate with each increase in temperature of 10°C. For the eight toads in the study, Q_{10} values ranged from 2.0257 to 7.5960 and averaged 3.4426 (standard deviation = 1.8316). With body temperatures ranging from 13.4 to 19.7, the mean ± SEM for the eight toads was 17.1 ± 0.9°C.

The following projects examine several fundamental HPC issues. For each, develop tables and graphs similar to those in the section on "Speedup and Scalability" and discuss the results. The programs caneToadsMPI-v1.c *and* caneToadsMPI-v2.c *are available on the Blue Waters website.*

22. This project examines the question, "Is it more efficient to implement parallelism by partitioning the physical grid among processes or by distributing toads among processes?" Develop a parallel program that distributes toads among the processes and compare your results with *caneToadsMPI-v2.c*, which partitions the grid among processes.

23. This project examines the question, "Does each process always have the exact same amount of work to do, or can load balancing issues arise if, say, certain portions of the grid have more water and/or food sources than others?" Running *caneToadsMPI-v2.c* on multiple processes, consider various configurations of the food and moisture grids from most food and all moisture being on one process to equal distribution of food and AWPs among the processes. Use a parallel profiler to aid in your analysis.

24. This project examines the question, "How does the parallel code scale as the physical problem increases?" For *caneToadsMPI-v2.c*, consider scalability holding the number of toads fixed and increasing the grid size, holding the grid size fixed and increasing the number of toads, and then increasing both.

25. Repeat Project 24 for *caneToadsMPI-v1.c*.

26. This project examines the question, "What are potential performance bottlenecks, and how might the parallel code be re-written to address these?" Use a parallel profiler to analyze *caneToadsMPI-v2.c* to discover areas of code that consume the most execution time, and revise the code to improve the results.

Project 7 in the module "Time after Time: Age- and Stage-Structured Models" (Shiflet and Shiflet 2010) involves a stage-based model of cane toads.

9. Answers to Quick Review Questions

1. **a.** (31, 45)
b. (29, 44)

2. **a.** 6, the length of *ROWLST_AWP*

b. 6, the length of *ROWLST_AWP* and of *COLLST_AWP*
c. (75, 15), (53, 72), and (22, 80)

3. **a.** -1 because the cell is on the northern border
b. 0.0 because the cell is not close to an AWP
c. 1 because the cell is at an AWP
d. 0.4 because the cell is adjacent to an AWP
e. 0.2 because the cell is two cells away from an AWP
f. -1 because the cell is on the eastern border

4. **a.** Any row between 1 and 98, inclusively, because we omit border rows of 0 and 99
b. Column 98
c. 24 because the ID and index agree

5. **a.** 0.12, or 12%
b. 0.14, or 14%, = 0.26 - 0.12
c. 0.43, or 43%, = 0.69 - 0.26
d. 0.31, or 31%, = 1.0 - 0.69

6. **a.** *aToad.energy* = 0.91, *aToad.water* = 0.806, and *food*(5, 5) = 0.02 because *amtEat* = 0.01, so *aToad.energy* = 0.9 + 0.01, *aToad.water* = 0.8 + 0.6*0.01, and *food*(5, 5) = 0.03 - 0.01
b. *aToad.energy* = 0.905, *aToad.water* = 0.803, and *food*(5, 5) = 0.0 because *amtEat* = *food*(5, 5) = 0.005, so *aToad.energy* = 0.9 + 0.005, *aToad.water* = 0.8 + 0.6*0.005, and *food*(5, 5) = 0.005 - 0.005
c. *aToad.energy* = 1.0, *aToad.water* = 0.8006, and *food*(5, 5) = 0.029 because *amtEat* = 1 - *aToad.energy* = 0.001, so *aToad.energy* = 0.9 + 0.001, *aToad.water* = 0.8 + 0.6*0.001, and *food*(5, 5) = 0.03 - 0.001
d. *aToad.energy* = 0.91, *aToad.water* = 1.0, and *food*(5, 5) = 0.02 because *amtEat* = 0.01, so *aToad.energy* = 0.9 + 0.01, *aToad.water* = the minimum of 0.999 + 0.6*0.01 = 1.005 and 1.0, and *food*(5, 5) = 0.03 - 0.01

7. **a.** The *else* block because *moisture*(20, 30) is not less than *MOIST_AREA* but it is less than *AWP*
b. 5
c. 0.8 = 80% because 1/*aToad.numTimeSteps* = 1/5 = 0.2 and there is a 1 - 0.2 = 0.8 = 80% chance that a uniformly distributed random number between 0.0 and 1.0 will be greater than 0.2
d. 5 because 0.2 is not less than 0.1, so the toad does not look for water
e. 0 because 0.2 is less than 0.3, so the toad does look for water

8. **a.** $\begin{bmatrix} 76 & 16 \\ 75 & 16 \\ 75 & 17 \\ 76 & 17 \\ 77 & 17 \\ 77 & 16 \\ 77 & 15 \end{bmatrix}$ because (75, 15), a fenced AWP, and (76, 15), the previous position, are eliminated

b. $\begin{bmatrix} 98 & 16 \\ 98 & 17 \\ 98 & 15 \end{bmatrix}$ because (99, 16), a border cell, and (97, 16), the previous position, are eliminated

10. References

Alexander, T.R. 1964. Observations on the feeding behavior of *Bufo marinus* (Linne). Herpetologica 20:255–259.

Alford, R.A., M. Lampo and P. Bayliss. 1995. The comparative ecology of *Bufo marinus* in Australia and South America. CSIRO *Bufo* Project: An Overview of Research Outcomes, Unpublished report, CSIRO, Australia

Bartlett, R.D. and P.P. Bartlett. 1999a. A Field Guide to Florida Reptiles and Amphibians. Gulf Publishing Company, Houston, Texas.

BBC News. 2010. http://news.bbc.co.uk/2/hi/8480041.stm

Cohen, M.P. and R.A. Alford. 1993. Growth, survival and activity patterns of recently metamorphosed *Bufo marinus*. Wildlife Research 20:1–13.

Easteal, S. 1982. The genetics of introduced populations of the marine toad, *Bufo marinus* (Linnaeus), (Amphibia: Anura); a natural experiment in evolution. Ph.D. dissertation. Griffith University, School of Australian Environmental Studies, Brisbane, Queensland, Australia.

Easteal, S. 1985. The ecological genetics of introduced populations of the giant toad, *Bufo marinus*. III. Geographical patterns of variation. Evolution 39:1065–1075.

Easteal, S. 1986. *Bufo marinus*. Pp. 395.1–395.4. Catalogue of American Amphibians and Reptiles. Society for the Study of Amphibians and Reptiles, St. Louis, Missouri.

Florance, Daniel, Jonathan K. Webb, Tim Dempster, Michael R. Kearney, Alex Worthing and Mike Letnic. 2011. "Excluding access to invasion hubs can contain the spread of an invasive vertebrate" *Proc. R. Soc. B* published online 23 February 2011 doi: 10.1098/rspb.2011.0032

Freeland, W.J. and K.C. Martin. 1985. The rate of range expansion by *Bufo marinus* in Northern Australia, 1980–84. Australian Wildlife Research 12:555–559.

Freeland, WJ (1986) 'Populations of Cane Toad, Bufo marinus, in Relation to Time since Colonization', Australian Wildlife Research, 13: 321–329

Fuller, Pam 2011. cane_toad_72dpi.jpg, Cane toad (*Bufo marinus*) National Biological Information Infrastructure. http://www.nbii.gov/portal/server.pt/community/terrestrial_invasives/920 Accessed 08/28/11

Gao, Huimin, Martin Zehl, Alexander Leitner, Xiyan Wu, Zhimin Wang, Brigitte Kopp. J Ethnopharmacology 131, 368-376.

Halliday, Damien C.T., Daryl Venables, David Moore, Thayalini Shanmuganathan, Jackie Pallister, Anthony J. Robinson , Alex Hyatt. 2009. Cane toad toxicity: An assessment of extracts from early developmental stages and adult tissues using MDCK cell culture. Toxicon 53, 385–391.

Halsey, Lewis G. and Craig R. White. 2010. "Measuring Energetics and Behaviour Using Accelerometry in Cane Toads *Bufo marinus*," *PLoS ONE* 5(4): e10170. doi:10.1371/ journal.pone.0010170

Hayes, R.A., Piggott, A.M., Dalle, K., Capon, R.J., 2009. Microbial biotransformation as a source of chemical diversity in cane toad steroid toxins. Bioorganic and Medicinal Chemistry Letters 19, 1790–1792.

Hearnden M. N. 1991. *Reproductive and larval ecology of Bufo marinus (Anura: Bufonidae)*. Ph.D. thesis. Townsville (Australia): James Cook University of North Queensland

Hero, J. M. and M. Stoneham. 2009. "Bufo marinus." AmphibiaWeb. http://amphibiaweb.org/cgi/amphib_query?where-genus=Bufoandwhere-species=marinus Accessed 8/21/11

Hinckley, A.D. 1962. Diet of the giant toad, *Bufo marinus* (L.), in Fiji. Herpetologica 18:253–259.

Kearney, Michael, Ben L. Phillips, Christopher R. Tracy, Keith A. Christian, Gregory Betts and Warren P. Porter. 2008. "Modelling species distributions without using species distributions: the cane toad in Australia under current and future climates" *Ecography* 31: 423 434, 2008

Kearney, M. and Porter, W. P. 2004. Mapping the fundamental niche: physiology, climate, and the distribution of a nocturnal lizard. *Ecology* 85: 3119 3131

Krakauer, T. 1968. The ecology of the neotropical toad, *Bufo marinus*, in south Florida. Herpetologica 24:214–221.

Krakauer, T. 1970. Tolerance limits of the toad, *Bufo marinus*, in south Florida. Comparative Biochemistry and Physiology 33:15–26.

Lampo M., G. A. DeLeo 1998. The invasion ecology of the toad, *Bufo marinus*: From South America to Australia *Ecological Applications* 8: 88-396

Letnic, Mike, Jonathan K. Webb, and Richard Shine. Invasive Cane Toads (Bufo marinus) Cause Mass Mortality of Freshwater Crocodiles (Crocodylus johnstoni) in Tropical Australia. Biological Conservation (2008) 141:1773-1782

Lever, C. The Cane Toad: The History and Ecology of a Successful Colonist; Westbury Academic and Scientific: York, UK, 2001.

Ma, H.Y., Kou, J.P., Wang, J.R., Yu, B.Y., 2007. Evaluation of the anti-inflammatory and analgesic activities of Liu-Shen-Wan and its individual fractions. Journal of Ethnopharmacology 112, 108–114.

Meng, Z., Yang, P., Shen, Y., Bei, W., Zhang, Y., Ge, Y., Newman, R.A., Cohen, L., Liu, L., Thornton, B., Chang, D.Z., Liao, Z., Kurzrock, R., 2009. Pilot study of Huachansu in patients with hepatocellular carcinoma, nonsmall-cell lung cancer, or pancreatic cancer. Cancer 115, 5309–5318.

Oliver, J.A. 1949. The peripatetic toad. Natural History 58:30–33.

Phillips, BL, Brown, GP, Jonathan K. Webb, and Richard Shine (2006) 'Invasion and the evolution of speed in toads', Nature, 439: 803.

Phillips, BL, Brown, GP, Greenlees, Jonathan K. Webb, and Richard Shine (2007) 'Rapid expansion of the cane toad (Bufo marinus) invasion front in tropical Australia', Austral Ecology, 32: 169–176.

Queensland Government. Biosecurity Queensland Department of Employment, Economic Development and Innovation GPO Box 46, Brisbane 4001 February 2010 Anna Markula, Steve Csurhes and Martin Hannan-Jones http://www.dpi.qld.gov.au/documents/Biosecurity_EnvironmentalPests/IPA-Cane-Toad-Risk-Assessment.pdf

Rabor, D.S. 1952. Preliminary notes on the giant toad *Bufo marinus* (Linn.) in the Philippine Islands. Copeia 1952:281–282.

Reynolds, Craig. 1999. "Individual-Based Models, an annotated list of links" http://www.red3d.com/cwr/ibm.html Accessed 07/18/11

Rossi, J.V. 1983. The use of olfactory cues by *Bufo marinus*. Journal of Herpetology 17:72–73.

Schwarzkopf, Lin and Ross A. Alford. 2002. "Nomadic movement in tropical toads" *OIKOS* 96: 492–506

Schwarzkopf, Lin and Ross A. Alford. 2005. "Movement and dispersal in established and invading toad populations" 2005. Cane Toad Forum held in Kununurra. http://www.canetoads.com.au/forumprocedespp4.htm Accessed 8/9/11

Shiflet, Angela B. and George W. Shiflet. 2009. "Biofilms: United They Stand, Divided They Colonize" UPEP Curriculum Module http://shodor.org/petascale/materials/UPModules/biofilms/

Shiflet, Angela B. and George W. Shiflet. 2010. "Time after Time: Age- and Stage-Structured Models" UPEP Curriculum Module. http://shodor.org/petascale/materials/UPModules/ageStructuredModels/ Accessed 1/24/12

Stuart, Simon N**.,** Janice S. Chanson, Neil A. Cox, Bruce E. Young, Ana S. L. Rodrigues, Debra L. Fischman and Robert W. Waller (2004) Status and Trends of Amphibian Declines and Extinctions Worldwide. Science 306(5702):1783-1786.

Sutherst, Robert W., Robert B. Floyd, and Gunter F. Maywald. 1996. " The Potential Georgphical Distribution of the Cane Toad, *Bufo marinus* L. in Australia" *Conservation Biology*, Vol. 10, No. 1 (Feb., 1996), pp. 294-299

Urban, Mark C., Ben L. Phillips, David K. Skelly and Richard Shine. (2007) The cane toad's (Bufo marinus) increasing ability to invade Australia is revealed by a dynamically updated range model. Proc R Soc B 274(1616):1413-9.

Ward-Fear, G., G. P. Brown, M. Greenlees, and R. Shine. 2009. "Maladaptive traits in invasive species: in Australia, cane toads are more vulnerable to predatory ants than are native frogs." *Functional Ecology* 23:559-568

Wright, A.H. and A.A. Wright. 1949. Handbook of Frogs and Toads of the United States and Canada. Third edition. Comstock Publishing Associates, Ithaca, New York.

Xiao, P.G., 2002. Modern Chinese Materia Medica, vol. IV. Chemical Industry Press, Beijing, China, p. 253.

Zug, G. R. and P. B. Zug. 1979. *The marine toad, Bufo marinus: a natural history resume of native populations*. Smithsonian contributions to zoology. Washington, DC: Smithsonian Institution Press

11. Acknowledgements

We gratefully acknowledge the generous help of The National Computational Science Institute Undergraduate Petascale Education Program (UPEP) and Whitney Sanders. UPEP, which is an NCSA Blue Waters project in collaboration with the National Computational Science Institute (NCSI) and national HPC programs, funded development of the module and supported UPEP intern Whitney Sanders, who implemented the module's HPC programs.