WILLIAM & MARY
CHARTERED 1693

Dissertations, Theses, and Masters Projects   Theses, Dissertations, & Master Projects

1999

Artificial Societies: A Computational Approach to Studying Combat

Rachel J. Moore
College of William & Mary - Arts & Sciences

Follow this and additional works at: https://scholarworks.wm.edu/etd

Part of the Computer Sciences Commons, and the Military and Veterans Studies Commons

Recommended Citation

ARTIFICIAL SOCIETIES :

A Computational Approach to Studying Combat

A Masters Thesis

Presented to

The Faculty of the Department of Computer Science

The College of William & Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree of

Master of Science

---

by

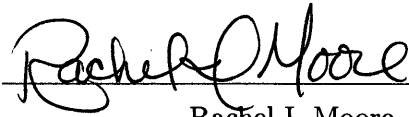Rachel I. Moore

1999

APPROVAL SHEET

This thesis is submitted in partial fulfillment of

the requirements for the degree of

Master of Science

Rachel I. Moore
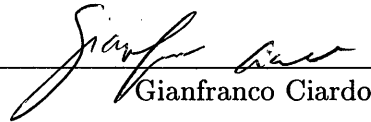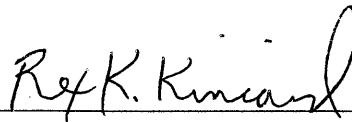Rachel I. Moore

Approved, July 1999

Steve Park
Steve Park
Thesis Advisor

Gianfranco Ciardo
Gianfranco Ciardo

Rex K. Kincaid
Rex Kincaid
Department of Mathematics

ii

Table of Contents

ACKNOWLEDGMENTS

List of Tables

List of Figures

ABSTRACT

Although the modeling of social processes via computer simulation is a growing trend among social scientists, there is a lack of computational explanation of these models in the literature. In an attempt to rectify this situation, this computer science dissertation is based on a complex "artificial society" model developed at three levels — conceptual, specification, and computational — with special attention to the object-oriented approach used at the computational level. In this society there are two groups of "agents" — hunters and gatherers — which move across a "landscape," reproduce, and interact via combat. This dissertation focuses on the population dynamics that results from combat between these two social groups.

ARTIFICIAL SOCIETIES :

A Computational Approach to Studying Combat

Chapter 1

Introduction

Simulation models were first used in the 1970s to study social dynamics [8]. In recent years, these social simulation models have become increasingly more complex as computational resources have become more accessible to researchers [5, 8]. The useful applications of social modeling via computer simulation are now numerous. These applications include, but are not limited to, the study of extinct societies, proposed Social Security reforms, economic trading systems, and the transmission of disease [3, 4, 5, 6].

A compelling reason to simulate social processes is that a computational environment can be used to execute "experiments" that would be impossible for social scientists to study otherwise. For example, these experiments might span many generations of a population in a town; this long period of time would be difficult to study otherwise. Social simulation models can also be used to study the emergence of societal behavior resulting from individuals following certain rules [4].[1]

---

[1]It should be noted that some skeptics claim social modeling is a "fad with limited potential" [13]. Contrary to my opinion, they believe complex human behaviors cannot be accurately simulated or predicted by a simulation model.

The behavior and characteristics of individuals within a group or society can have a profound impact upon the behavior and characteristics of the group as a whole. Micro-simulation is the common process of using a micro-scale "bottom-up" approach to study macro-scale behavior. *Individuals* follow specific social rules and, as time evolves, *group* behaviors and characteristics appear. For example, a study of disease transmission between individuals over many generations might demonstrate the development of a group immunity to particular diseases. Another example would be movement rules implemented on an individual level creating group migration over time.

A cellular automata (CA) approach to modeling began in 1970 with John Conway who used the now popular Game of Life to model interactions between cells in a 2-dimensional grid [6]. Another early example of a CA approach is Thomas Schelling's 1971 model of ethnic segregation [6, 7]. CA-based models have been used to explore the abstract properties of interacting agents via micro-simulation [6]. CA-based models exhibit characteristics of 1) cells arranged in a regular grid, 2) a finite set of cell states, 3) discrete time evolution, 4) cells changing their state according to local rules, 5) the same rules applying to all cells, and 6) simultaneous or sequential cell updates each time step.

The term "artificial societies" originated with Builder and Bankes in 1991 [1]. An artificial society is a micro-simulation of mobile agents interacting with a CA-based landscape. Though some life science researchers have recently begun to study artificial societies, the majority of interest in this type of simulation comes from social scientists. Epstein and Axtell have used an artificial society model to produce interesting results that emulate many areas of human social behavior [4].

Epstein and Axtell's study demonstrated that different emerging combat patterns can be

produced by changing agent attributes and behavior rules [4]. Unfortunately, they did not discuss which attributes were changed to produce these different combat patterns. In fact, I was unable to find any well-documented combat models in the artificial society literature. Since this is a computer science dissertation, I will carefully document the simulation model I developed at three levels — conceptual, specification and computational — paying close attention to the algorithms and data structures I used.

Epstein and Axtell allude to several extensions of their artificial society combat model, one of which is tribal conflict. Although the authors do not discuss tribal conflict in any depth, I found it an interesting scenario to simulate and study. Accordingly, the combat model in this dissertation incorporates two distinct groups of agents, hunters and gatherers, each group with different rules and characteristic attributes.

The goals of this dissertation are to design an artificial society simulation model of combat between hunters and gatherers, explain it well, conduct several experiments, and explore the "emergence" of macro-level phenomena from micro-level activity. In particular, the experiments presented in this dissertation will study how well a stable gatherer population can withstand the injection of predatory hunters.

The simulation model is presented in Chapter two, followed by a brief explanation of the visual interface for the simulation software in Chapter three. The simulation experiments and results are described and discussed in Chapter four. Some conclusions and suggestions for future research are provided in Chapter five.

Chapter 2

Model

Three levels of a simulation model are prescribed by Park and Leemis [10]. First is the *conceptual level*. The artificial society model developed in this dissertation is discussed at the conceptual level in Section 2.1 in terms of state variables and high-level interactions between the agents and the landscape. Second is the *specification level*. Section 2.2 describes the rules and algorithms of the model at this level. Third is the *computational level* — a computer program that is developed to be consistent with the model at the conceptual and specification levels. Section 2.3 describes the object-oriented hierarchy used in this dissertation at the computational level.

2.1 Conceptual Level

The simulation model upon which all results in this paper are based is an "artificial society" — a landscape and agents that jointly evolve over time according to simple rules of evolution and interaction. The landscape represents a "world" and the agents represent the "people"

in this world [4]. Agents are divided into two groups — *hunters* and *gatherers* — each with distinct behavioral characteristics. Time evolves in a discrete manner with agent events and landscape events synchronized at each step.

The landscape is a two-dimensional grid with varying amounts of a resource called "sugar" [4]. Each landscape location is defined by two static attributes and two dynamic states. The attributes are resource capacity and resource growback rate; the states are the current level of resource and agent occupancy status. As time evolves, the resource is removed from the landscape locations visited by the gatherers. The resource then grows back over time according to the growback rate at these locations until it is again gathered by an agent or the resource capacity is reached.

Gatherers travel the landscape, searching for and gathering resource from the landscape. Hunters travel the landscape in search of gatherers from which they steal resource. The defining characteristics of hunters and gatherers are their static attributes — field of view (FOV), rate of resource metabolism, gender, lifetime (life expectancy), the start and end of their fertile period, and the group to which they belong — and their dynamic states — wealth (current resource level), landscape position, age, and pregnancy status. Hunters are also characterized by an initial wealth attribute.

A gatherer interacts with the landscape by moving to a landscape location within its FOV, collecting all the resource there, then moving to another landscape location, etc. A gatherer accumulates wealth by collecting more resource than it metabolizes. A hunter also moves about the landscape, but does not collect the resource directly from the landscape. Instead, hunters interact with gatherers through combat. For a hunter to accumulate wealth, it must defeat gatherers in combat and collect more resource than it metabolizes.

High wealth is an advantage when combat occurs because, although combat outcomes are stochastic, the wealthy are favored. Only hunters will initiate combat; gatherers will fight to retain their wealth. Both gatherers and hunters can die of starvation. The onset of starvation is determined by a low wealth. If an agent's wealth drops to zero, death occurs immediately. Unless it is starving, a hunter will only fight a gatherer with less wealth. However, if it is starving, a hunter will risk fighting a wealthier gatherer. When combat occurs, the losing agent relinquishes all of its wealth to the winner and then necessarily dies of starvation.

When an agent dies of starvation or old age, it is removed from the landscape. The only way for the dead agent's group to "replace" it is by reproduction. An agent will mate with another nearby agent if both agents are fertile, of the opposite sex, of the same group, and if there is a nearby landscape location for the baby to occupy when it is born. Aggressive male hunters will also attempt to mate with nearby female gatherers if there are no fertile female hunters nearby. Multiple pregnancies during one time step are not possible. The baby will always belong to the same group as its father.

For all the experiments discussed in this dissertation, the artificial society model begins with a stable population of gatherers and no hunters. As time evolves, a stochastic number of hunters are "injected" into the landscape each time step. The time evolution of the society is simulated for a long period of time and statistics about the hunter and gatherer populations are accumulated. The experimental details and results are provided in Chapter four.

2.2 Specification Level

2.2.1 Discreteness of Time

In this simulation model, time is discrete and denoted $t = 0, 1, 2, \ldots, T$ where T is the stopping time. Each time step represents one "year" in the artificial society. As described in more detail later, agent movement, agent reproduction and landscape replenishment are synchronized each time step.

2.2.2 The Landscape

The landscape is a $X \times Y$ grid, with wraparound. That is, opposite edges of the landscape are "pasted" together to form a torus. A torus shape is used because it prevents congestion at landscape boundaries where agent movement would otherwise be limited. A landscape cell at location (x, y) is shown in Figure 2.1; x and y are the respective row and column coordinates.

Each landscape cell (x, y) has a real-valued maximum resource capacity $c(x, y)$ defined as

$$c(x, y) = 5e^{-((x-0.5X)/\sigma_x)^2 - ((y-0.5Y)/\sigma_y)^2}$$

with $\sigma_x = 0.54X$, $\sigma_y = 0.54Y$ and $x = 0, 1, 2, \ldots, X - 1$, $y = 0, 1, 2, \ldots, Y - 1$. This capacity function produces a resource "peak" at the center of the landscape. Cells near the center of the landscape have high resource capacity (approximately 5 units) and those further from the center have low resource capacity (nearly zero). At each landscape cell (x, y) the resource grows back at a real-valued rate of $\alpha(x, y) > 0$ units per time step, up to the resource capacity of that cell.
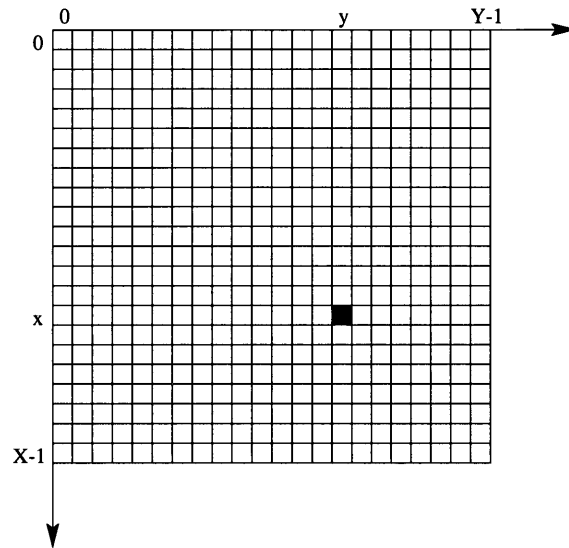
Figure 2.1: The Landscape

2.2.3   Agent Vision

The field of view (FOV) of each agent spans four directions on the landscape: north, south, east, and west. An agent with FOV parameter v can look in each of these four directions, but can only see landscape cells that are $1, 2, \ldots, v$ cells away from its current (x, y) cell position. For example, the FOV of an agent at landscape cell (x, y), with FOV parameter 4, is illustrated in Figure 2.2.
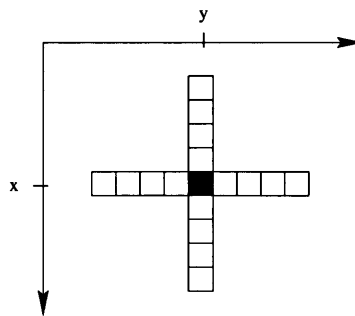
Figure 2.2: Agent Field of View ($v = 4$)

2.2.4 Combat

Given a hunter h with wealth (resource level) w_h and initial wealth w_{h_0} and a gatherer g with wealth w_g, the probability that the hunter will defeat the gatherer in combat is

$$\Pr\left(h \text{ defeats } g\right) = 0.15 + 0.30p(w_h/w_g) + 0.55p(w_h/w_{h_0})$$

where

$$p(z) = \frac{z^n}{z^n + 1} \quad z > 0.$$

The first term in the probability equation represents the probability that a starving hunter will defeat a wealthy gatherer in combat. If the hunter has wealth much greater than the gatherer, the second term will be close to 0.30. The third term will be close to 0.55 if the hunter's current wealth is significantly higher than its initial wealth.
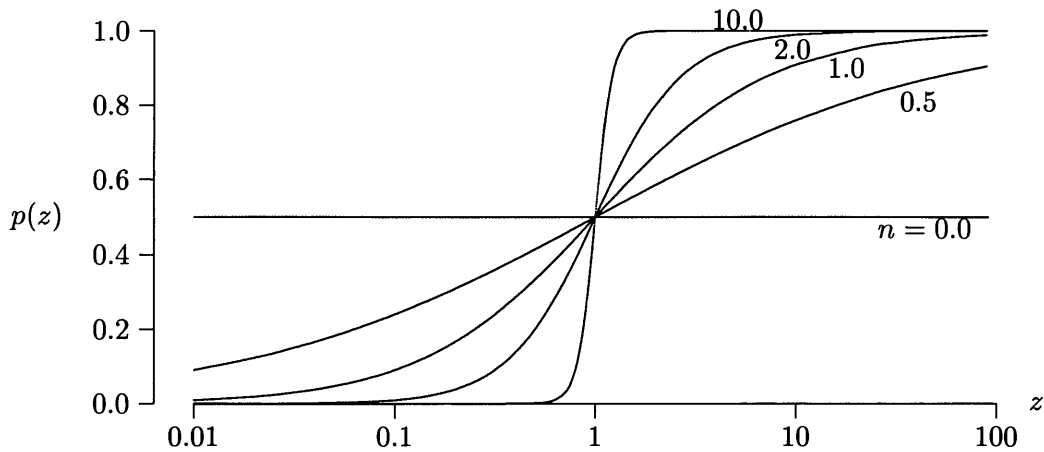
Figure 2.3: Effect of Parameter n and Wealth Ratios on Combat Outcome

As Figure 2.3 illustrates, $n > 0$ is a parameter in the $p(z)$ equation used to "shape" the likelihood of a hunter defeating a gatherer in combat. A hunter with greater wealth than a gatherer $(w_h/w_g > 1)$ and with greater wealth than its initial wealth $(w_h/w_{h_0} > 1)$ will

defeat a gatherer with probability close to 1.0. A hunter with low wealth relative to its initial wealth ($w_h/w_{h_0} < 1$) and relative to the gatherer's wealth ($w_h/w_g < 1$) will defeat a gatherer with probability close to 0.15. [1]

2.2.5 Agent Movement

Gatherers forage for large amounts of nearby resource, rather than search for resource at distant cells. A gatherer will always move to the unoccupied cell within its FOV with maximum resource. If there is more than one cell with this (local) maximum amount in the gatherer's FOV, the gatherer will chose the cell closest to its current landscape location. In the uncommon case that two or more cells having this (local) maximum amount of resource are equidistant from the agent's current cell location, the tie is broken stochastically. Each time step, each gatherer moves about the landscape according to the following rules.

1. The gatherer examines its FOV.

   - All occupied cells within the FOV are not feasible.
   - All unoccupied cells remaining within the FOV are feasible.

2. The gatherer moves to the closest feasible cell with maximum resource and gathers the resource present. If there are no feasible cells, the gatherer does not move.

Hunters are active, roaming the landscape and relying on their ability to prey on gatherers in order to survive. Hunters who are starving move differently than those who are not. Each hunter determines whether or not it is starving by comparing its current wealth to a wealth threshold parameter γ. Each time step, each hunter moves according to the following rules.

---

[1]As $n \to 0$, the ratios w_h/w_g and w_h/w_{h_0} become irrelevant. The probability that h will defeat g approaches 0.575, independent of the two ratios. Conversely, as $n \to \infty$ the two ratios become more significant in calculating the probability of a hunter defeating a gatherer.

1. The hunter examines its FOV.

    - All unoccupied cells within the FOV are not feasible.

    - All cells within the FOV occupied by other hunters are not feasible.

    - If the hunter is not starving, all cells within the FOV occupied by wealthier gatherers are not feasible. If the hunter is starving, all cells within the FOV occupied by gatherers are feasible, independent of the gatherer's wealth.

2. If there are no feasible cells, the hunter moves as far as possible within its FOV in a random direction to an unoccupied cell. If there is at least one feasible cell then the *starving* hunter moves to the furthest feasible cell occupied by the most vulnerable (least wealthy) gatherer. A hunter that is not starving moves to the furthest feasible cell occupied by the wealthiest gatherer. If there are no feasible or unoccupied cells within its FOV, the hunter does not move.

3. If the hunter moved to a feasible cell, combat occurs with the gatherer there and the winner is determined consistent with the probability function defined in Section 2.2.4. The winning agent gathers all of the losing agent's wealth. The losing agent then dies of starvation and is removed from the landscape.

Each time step, all gatherers move about the landscape. Then all the hunters move in search of the gatherers. Because agents that move first within a group have an advantage, the order of agent movement for both groups is randomized (shuffled) each time step. At the end of each time step, each agent's wealth is decreased by its metabolism rate. (The amount subtracted from its wealth per time period is the metabolism rate of the agent.)

2.2.6 Landscape and Agent Updates

Each time step, the resource level of each landscape cell (x, y) is increased by the growback rate $\alpha(x, y)$ up to its resource capacity $c(x, y)$. This growback occurs regardless of a cell's occupancy status. Before mating occurs, agents with wealth less than or equal to zero and those who have died of old age are removed from the landscape.

2.2.7 Agent Mating and Reproduction

An agent is fertile if its age is between the start and end of its fertile period and its wealth is greater than the reproduction threshold for its group (θ_h for hunters and θ_g for gatherers). In addition to being fertile, the following criteria must be met in order for an agent to mate on the landscape.

1. A fertile, potential mate must be within the agent's Moore neighborhood on the landscape. The Moore neighborhood of an agent occupying the darkened cell (x, y) in Figure 2.4 consists of the surrounding eight gray cells.
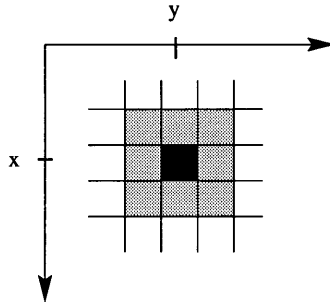
Figure 2.4: Moore Neighborhood

2. The potential mate must be of the opposite sex.

3. The potential mate must not already be pregnant.

4. A male potential mate must be of the same group as the female agent searching for a mate. A male hunter that initiates mating can "rape" a female gatherer; however, a male gatherer may not mate with a female hunter.

5. A cell must be vacant in either mate's Moore neighborhood for the baby to occupy when it is born.

When a baby is born, each parent contributes half of its wealth to the child. The baby's metabolism, vision, and lifetime are inherited from its parents. A coin flip for each attribute determines which parent will pass its attribute to the baby agent.

2.2.8 End of Time Period

After all mating and reproduction is complete, all agents are aged one year.

2.3 Computational Level

With t as the current time index and T as the stopping time index, the following pseudo-code

summarizes the model at the computational level.

```
Landscape world;
AgentMgr agents(&world);
world.buildLandscape();
agents.initializeAgents();
t = 0;
while (t < T && agents.numGatherers()) {
  agents.moveAgents();        /* shuffling and moving */
  agents.metabolizeResource()
  world.growbackResource();
  agents.flushAgents();
  agents.reproduceAgents();
  agents.ageAgents();
  t++;
}
```

(All constants used in Section 2.3 correspond to parameters used in some or all experiments in Chapter four. The stochastic functions used are summarized and implemented in Appendix A.)

2.3.1 Landscape Initialization

The landscape is encapsulated by a **Landscape** object, instantiated in the pseudo-code above

as **world**, which builds and replenishes the landscape. The **Landscape** object maintains an

$X \times Y$ array of **Cell** objects (see Figure 2.5). The attributes of each **Cell** are **capacity**

and **growback**. The states of each **Cell** are **resource** and **occupant**.

Landscape
cells : Cell[X][Y]
buildLandscape() : void getCell(x : int, y : int) : Cell* growbackResource() : void agentArrive(x : int, y : int, agent : Agent*) : void agentLeave(x : int, y : int) : void growback(x : int, y : int) : float capacity(x : int, y : int) : float

Cell
resource : float capacity : float growback : float occupant : Agent*
buildCell(cap : float, grow : float) : void agentArrive(agent : Agent*) : void agentLeave() : void growbackResource() : void getResource() : float isOccupied() : int getOccupant() : Agent*

Figure 2.5: Landscape and Cell Objects

The attributes and the states of the landscape cells are initialized in the function

buildLandscape(). This function initializes the attributes capacity and growback of

each Cell by calling the Landscape functions that correspond to $c(x,y)$ and $\alpha(x,y)$ —

capacity(x,y) and growback(x,y) . The states of each Cell are initialized within the

function Cell::buildCell() by setting its resource level to its resource capacity and its

occupant to NULL.

2.3.2 Agent Initialization

The hunters are represented by a linked list of Hunter objects; gatherers by a linked list

of Gatherer objects. These objects are derived from the Agent object, as illustrated in

Figure 2.6. Initially, there are A agents. At starting time $t = 0$, the hunter list has no

Hunter objects, and the gatherer list has A Gatherer objects.

The attributes of each Hunter are vision, metabolism, gender, group, lifetime,

begin_fertility, end_fertility, and init_wealth. The states of each Hunter are x,

y, wealth, age, and pregnancy (status). The attributes of each Gatherer are vision,

```
                                              ┌──────────────────────────────────────────────────┐
                                              │                      Hunter                        │
                                              ├──────────────────────────────────────────────────┤
                                              │ moves : HMoveList*                                 │
                ┌─────────────────────────┐   │ init_wealth : float                                │
                │         Gatherer         │   ├──────────────────────────────────────────────────┤
                ├─────────────────────────┤   │ Hunter()                                           │
                │ moves : GMoveList*       │   │ initAgent( wlth : float, mtblsm : float, vsn : int, newx : int, newy : int, fert_start : int, │
                ├─────────────────────────┤   │          fert_end : int, lifetime : int, gender : gender_type, agent_age : int ) : void │
                │ Gatherer()               │   │ checkCell( checkx : int, checky : int, scape : Landscape* ) : float │
                │ checkCell( checkx : int, checky : int, scape : Landscape* ) : float │   │ find Move( scape : Landscape *) : void │
                │ findMove( scape : Landscape* ) : void │   │ getGroup() : agent_type                            │
                │ getGroup() : agent_type  │   │ isStarving() : int                                 │
                │ addWealth( value : double ) : void │   ├ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─┤
                │ isStarving() : int       │   │ fightGatherer( gath : Gatherer* ) : int            │
                └─────────────────────────┘   └──────────────────────────────────────────────────┘
```


Figure 2.6: Derivation of Hunter and Gatherer Objects

metabolism, **gender**, **group**, **lifetime**, **begin_fertility**, and **end_fertility**. The states

of each **Gatherer** are **x**, **y**, **wealth**, **age**, and **pregnancy** (status).

The hunter list is maintained by a **HunterMgr** object and the gatherer list is maintained

by a **GathererMgr** object (Figure 2.7). Both of these list-managing objects are controlled

by a **AgentMgr** object named **agents** in the pseudo-code at the beginning of Section 2.3.

The attributes and states of all *A* agents are initialized at the start of the simulation in

the **AgentMgr** function **initializeAgents()** which makes calls to **initializeHunters()**

and **initializeGatherers()** via its **HunterMgr** and **GathererMgr** objects.

The function **HunterMgr::initializeHunters()** creates no hunters. It keeps a pointer

GathererMgr
gathererList : Gatherer* numGatherers : int scape : Landscape*
GathererMgr() ~GathererMgr() initializeGatherers(number : int, landscape : Landscape*) : void moveGatherers() : void reproduceGatherers() : void flushGatherers() : void getNumGatherers() : int ageGatherers() : void drawGatherers() : void
metabolizeResource() : void shuffleGatherers() : void createGatherer() : Gatherer* createBaby(x : int, y : int, parent1 : Gatherer*, parent2 : Gatherer*) : Gatherer* findMaleNeighbor(x : int, y : int, x1 : int&, y1 : int&) : void findFemaleNeighbor(x : int, y : int, x1 : int&, y1 : int&) : void findSpotForBaby(x : int&, y : int&, x1 : int, y1 : int, x2 : int, y2 : int) : void

HunterMgr
hunterList : Hunter* numHunters : int scape : Landscape*
HunterMgr() ~HunterMgr() initializeHunters(landscape : Landscape*) : void moveHunters() : void reproduceHunters() : void flushHunters() : void getNumHunters() : int ageHunters() : void addHunters(num : int) : void drawHunters() : void
metabolizeResource() : void shuffleHunters() : void createHunter() : Hunter* createBaby(x : int, y : int, parent1 : Hunter*, parent2 : Hunter*) : Hunter* findMaleNeighbor(x : int, y : int, x1 : int&, y1 : int&) : void findFemaleNeighbor(x : int, y : int, x1 : int&, y1 : int&) : void findSpotForBaby(x : int&, y : int&, x1 : int, y1 : int, x2 : int, y2 : int) : void

AgentMgr
hunters : HunterMgr gatherers : GathererMgr
AgentMgr(agentNum : int, scape : Landscape*) moveAgents() : void metabolizeResource() : void flushAgents() : void reproduceAgents() : void ageAgents() : void getNumHunters() : int getNumGatherers() : int drawAgents() : void

Figure 2.7: Agent Managing Objects

to the landscape and initializes numHunters to zero and hunterList to NULL.

The function GathererMgr::initializeGatherers() is shown in the pseudo-code below. A total of *A* gatherers are created in this function, as follows.

```
gathererList = NULL;
num_gatherers = 0;
Gatherer *gath;
for (i = 0; i < A; i++) {
  gath          = createGatherer();
  gath->next    = gathererList;
  gathererList = gath;
  num_gatherers++;
}
```

Each new gatherer is created within the function `createGatherer()` where its attributes

and initial states are generated. (See Appendix A for stochastic function implementation.)

The function `createGatherer()` initializes the states `x` and `y` of the new gatherer using

functions `Equilikely(0,X-1)` and `Equilikely(0,Y-1)`. These two calls are made until an

unoccupied (x, y) cell position is generated. The continuous state `wealth` is initialized by

a call to `Uniform(20,50)`. The continuous attribute `metabolism` is generated by a call to

`Uniform(3,7)`. The FOV parameter `vision` is a discrete attribute generated by a call to

`Equilikely(1,4)`. The `gender` is generated by a call to `Bernoulli(0.5)` which returns

0 (female) and 1 (male) with equal probability. The `lifetime` attribute is discrete and is

generated by a call to `Equilikely(80,100)` for male and female gatherers. The start and

end of a gatherer's fertility period is determined by making calls to `Equilikely(12,15)` and

`Equilikely(50,70)` if female and `Equilikely(15,19)` and `Equilikely(50,70)` if male.

The `age` of the gatherer is initialized by a call to `Equilikely(0, 60)`. The `GathererMgr`

then calls the function `Agent::initAgent()` for each created `Gatherer`.

The function `Agent::initAgent()` completes the agent initialization. The `pregnancy`

status of the agent is initialized to zero (false) and the agent parameters are copied into the

corresponding `Agent` class member variables.

A pointer to the new `Gatherer` is returned to function `initializeGatherers()`. The

new `Gatherer` is then inserted into the gatherer list (`gathererList`).

2.3.3   Agent Movement

At the beginning of each time step, the agents are permitted to move. The function

`AgentMgr::moveAgents()` makes two calls to move the agents, as follows.

```
gatherers.moveGatherers();
hunters.moveHunters();
```

Before the agents actually move, each agent list is shuffled. Shuffling prevents agents from remaining near the head of the list where they would have an early-move advantage. Gatherers are randomized by a call to **shuffleGatherers()** made within the function **GathererMgr::moveGatherers()**. Hunters are randomized using the same shuffling algorithm by a function call to **shuffleHunters()** from within **HunterMgr::moveHunters()**.

2.3.3.1 Gatherer Movement

The gatherers forage first. After each gatherer has been given the chance to move and has gathered the resource in the cell it occupies, the hunters pursue the gatherers.

The function **Gatherer::findMove()** is used to support the calling **GathererMgr** function, **moveGatherers()**, and is called for each gatherer in the list. A pointer to a **GMoveList** object is kept for each gatherer while it searches for a new landscape cell. This **GMoveList** object is derived from **MoveList**, a linked list of **moveNode** structures. (See Figure 2.8.) **GMoveList** sorts the **moveNode** list from highest to lowest resource return and secondly from closest to furthest landscape position in the gatherer's FOV.

```
struct moveNode {
    int        x;          /* target column */
    int        y;          /* target row    */
    float      resource;   /* total reward for moving to this cell */
    moveNode*  next;       /* pointer to next feasible move */
};
```

A gatherer examines all unoccupied cells in its FOV and passes the feasible moves to the **GMoveList** for sorted insertion into the **moveNode** list. The gatherer moves to the cell
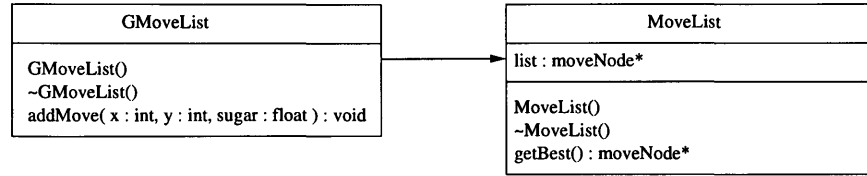
Figure 2.8: Gatherer Move List Object Derivation

with maximum resource return closest to its current cell location and gathers the resource.

If there are no unoccupied cells within the gatherer's FOV, the gatherer does not move.
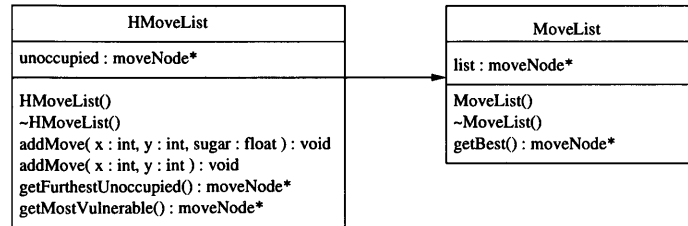
The function `GathererMgr::moveGatherers()` shuffles the gatherers and iterates over all the gatherers moving them one by one, as follows.

```
shuffleGatherers();
Gatherer* ptr = gathererList;
while (ptr != NULL) {
  ptr->findMove();   /* Each gatherer moves itself.       */
  ptr = ptr->next(); /* Retrieves next gatherer in list. */
}
```

2.3.3.2   Hunter Movement

The function `Hunter::findMove()` is used to support the calling `HunterMgr` function, `moveHunters()`. A pointer to a `HMoveList` object is kept for each hunter while it searches for a new landscape cell. This `HMoveList` object is derived from `MoveList` (Figure 2.9) and maintains two linked lists for cells in the hunter's FOV — one comprised of unoccupied cells sorted from furthest to closest with **resource** level always zero, and the other comprised of cells occupied by gatherers sorted from highest resource level to lowest and furthest to closest. The structures in both lists are **moveNode** structures which hold the landscape position and resource level of each feasible move.

HMoveList
unoccupied : moveNode*
HMoveList() ~HMoveList() addMove(x : int, y : int, sugar : float) : void addMove(x : int, y : int) : void getFurthestUnoccupied() : moveNode* getMostVulnerable() : moveNode*

MoveList
list : moveNode*
MoveList() ~MoveList() getBest() : moveNode*

Figure 2.9: Hunter Move List Object Derivation

A hunter examines all cells within its FOV which are occupied by gatherers and passes the feasible cells to the `HMoveList` to insert into the occupied `HMoveList::list` inherited from `MoveList`. All unoccupied cells within its FOV are added to the second member list, `HMoveList::unoccupied`. As described in Section 2.2.5, if the hunter is not starving, it ignores cells with wealthier gatherers and moves to the furthest cell with maximum resource level — the location returned by a call to `list->getBest()`. (Recall that a hunter's resource consumption as a result of combat is the wealth of the gatherer it defeats.) A starving hunter, instead, does not ignore wealthier gatherers and moves to the cell within its FOV occupied by the weakest gatherer — the location returned by a call to `list->getMostVulnerable()`. If no cells occupied by gatherers remain in the hunter's FOV, the hunter (starving or not) moves as far as possible within its FOV in a random direction to an unoccupied cell retrieved from the `MoveList` of unoccupied, visible cells. This location is returned by a call to `unoccupied->getFurthestUnoccupied()`. If both lists are empty, the hunter does not move.

If combat occurs, the losing agent surrenders all of its wealth to the winning agent and then dies of starvation. The winner's wealth is then increased by the loser's wealth.

The function, `HunterMgr::moveHunters()`, shuffles the hunters and iterates over all the

hunters moving them one by one, as follows.

```
shuffleHunters();
Hunter* ptr = hunterList;
while (ptr != NULL)
{
  ptr->findMove();   /* Each hunter moves itself.   */
  ptr = ptr->next(); /* Retrieves next hunter in list. */
}
```

2.3.4 Metabolization

The `AgentMgr` next makes function calls to `GathererMgr::metabolizeResource()` and

`HunterMgr::metabolizeResource()`. Within the function `metabolizeResource()` below,

the `GathererMgr` object traverses the gatherer list and calls `Agent::metabolize()` for each

`Gatherer`. This function call updates the gatherer's `wealth` by subtracting its `metabolism`.

```
Gatherer* ptr = gathererList;
while (ptr) {
  ptr->metabolizeResource();
  ptr = (Gatherer*)ptr->getNext();
}
```

The `HunterMgr::metabolizeResource()` function similarly traverses the `hunterList`,

forcing each `Hunter` to metabolize its resource.

2.3.5 Hunter Injection

After agents metabolize their resource, the `AgentMgr` object injects hunters into the land-

scape according to the following pseudo-code in `AgentMgr:moveAgents()`. The value of `p` is

discussed in detail in Chapter four. The function `HunterMgr::addHunters(num)` makes `num`

calls to `createHunter()` and inserts each returned `Hunter` pointer into the `hunterList`.

```
if (p) {
    int num = Geometric(p);
    hunters.addHunters(num);
}
```

The function `createHunter()` instantiates each new `Hunter` object by several stochastic

function calls. The states `x` and `y` are initialized to be on the landscape periphery by the

following pseudo-code.

```
do {
    x = Equilikely(0,39);
    y = Equilikely(0,39);
    if (x > 19)
        x += X - 40;
    if (y > 19)
        y += Y - 40;
} while (scape->getCell(x,y)->isOccupied());
```

The loop terminates when an unoccupied (x, y) cell position is generated. `wealth` is a

continuous state initialized by a call to `Uniform(50,70)`. The continuous hunter attribute

`metabolism` is generated by a call to `Uniform(2,7)`. The FOV parameter, `vision`, is a

discrete attribute generated by a call to `Uniform(50,70)`. The hunter's **gender** is gener-

ated by a call to `Bernoulli(0.5)`, returning 0 (female) and 1 (male) with equal probability.

The `lifetime` attribute is discrete and is generated by a call to `Equilikely(90,100)` for

females and `Equilikely(100,110)` for males. The start and end of the hunter's fertility

period are generated by two calls — `Equilikely(5,7)` and `Equilikely(60,80)` if female or

`Equilikely(7,9)` and `Equilikely(70,80)` if male. The age of the hunter is initialized by

a call to `Equilikely(0,60)`. The `HunterMgr` object calls each new hunter's `initAgent()`

function with the initialized states and generated attributes as parameters. In this func-

tion, each `Hunter` sets its `init_wealth` attribute to `wealth` and then calls parent function

`Agent::initAgent()` to complete the initialization. (See Section 2.3.2.)

2.3.6 Updates to Landscape and Agents

During each time step, the landscape cells are replenished in the `Landscape` function `growbackResource()`. If a `Cell` has resource level less than its `capacity`, growback resource units are added to `resource`, not exceeding `capacity`.

Inside function `AgentMgr::flushAgents()`, two function calls are made to flush dead agents. Functions `GathererMgr::flushGatherers()` and `HunterMgr::flushHunters()` remove agents with no remaining wealth and agents which have met their life expectancy (`age == lifetime`).

2.3.7 Agent Reproduction

Within `reproduceAgents()`, the `AgentMgr` calls `GathererMgr::reproduceGatherers()` and `HunterMgr::reproduceHunters`. These functions permit living agents with ample wealth to attempt reproduction.

The process of hunter reproduction will be described in detail, followed by a short explanation of the differences between gatherer and hunter reproduction. Each function in the `HunterMgr` class related to reproduction is also in the `GathererMgr` class. (See Figure 2.7.)

In the function `HunterMgr::reproduceHunters()`, the `HunterMgr` object iterates through the list of hunters. The fertility of the hunter is first established. If the hunter is not fertile, reproduction cannot take place. Second, the hunter's wealth is compared to θ_h. The hunter is prohibited from reproduction if its wealth is not above θ_h. Third, the `HunterMgr` object determines whether the hunter is already pregnant (if female). If not, the `HunterMgr` calls `findMaleNeighbor()` if the hunter is female or `findFemaleNeighbor()` if the hunter is

male.

These functions check the Moore neighborhood (Figure 2.4) of the hunter's location to find a feasible mate for the hunter. If the hunter is male and no feasible female hunter is found, `findFemaleNeighbor()` searches the same neighborhood for a feasible female gatherer. This is the allowance made in the conceptual and specification levels of the model for a male hunter to "rape" a female gatherer.

If a feasible mate is found, the `HunterMgr` object searches to determine whether there is a cell on the landscape for a baby. Within the function `reproduceHunters()`, a call to function `findSpotForBaby()` is made to search the Moore neighborhoods of both parent agents for an empty cell.

A baby hunter is created by a call to `createBaby()` if an empty cell was returned by `findSpotForBaby()`. In function `createBaby()`, the baby hunter's gender is determined by a call to `Bernoulli(0.5)`. The baby receives half of each parent's wealth. The wealth of each parent is consequently halved by calling `Agent::madeBaby()` for each parent. The baby's metabolism, vision, and lifetime are inherited directly from its parents. A call to `Bernoulli(0.5)` is made for each remaining attribute. If the result is zero, the baby assumes its mother's attribute; if the result is one, the baby assumes its father's attribute. The start and end of its fertility period are generated by calls to `Equilikely(5,7)` and `Equilikely(60,80)` if it is female or `Equilikely(7,9)` and `Equilikely(70,80)` if it is male. The discrete **age** state of the baby is set to zero. The function `initAgent()`, described in Section 2.3.2, is then called by the `HunterMgr` object for this new baby `Hunter`. The landscape is updated by a call to `agentArrive()` with the baby's x and y states as function parameters. The `HunterMgr` object adds this new hunter to the beginning of the hunter

list.

The reproduction process for gatherers is identical to the hunter algorithm described above, with three exceptions. The parameters for establishing the fertility period of a female baby are (12,15) and (50,70); a male baby's are (15,19) and (50,70). Gatherers may not initiate mating with hunters, and the reproduction threshold θ_h is replaced by θ_g.

2.3.8 End of Time Period

Within `ageAgents()`, the `AgentMgr` calls functions `GathererMgr::ageGatherers()` and `HunterMgr::ageHunters()`. These aging functions simply iterate over the corresponding agent list and call function `Agent::ageAgent()` for each agent. The function `ageAgent()` forces the `age` of each agent to increase by one. Finally, the time step t is incremented by one.

Chapter 3

Visualization

A graphical interface is an important tool for analyzing agent movement patterns in this simulation. An interface is also useful for verifying correctness of mating procedures and providing hints into some population dynamics. The interface used in this dissertation was written in Tcl. As shown in Figure 3.1, the interface displays the landscape and its occupying agents at specified time steps during the simulation.

Pressing on the plot button at the top right of the interface generates a graph of hunter and gatherer populations from the beginning of the simulation to the current time. When pressed, the step buttons advance time a corresponding number of years and update the agents on the landscape after the advance. Agents on the landscape appear as colored cells, allowing distinction between hunters and gatherers. Baby agents are also distinguishable by color. Black cells are unoccupied. The current time of the simulation is displayed above the landscape, after the interface is updated.
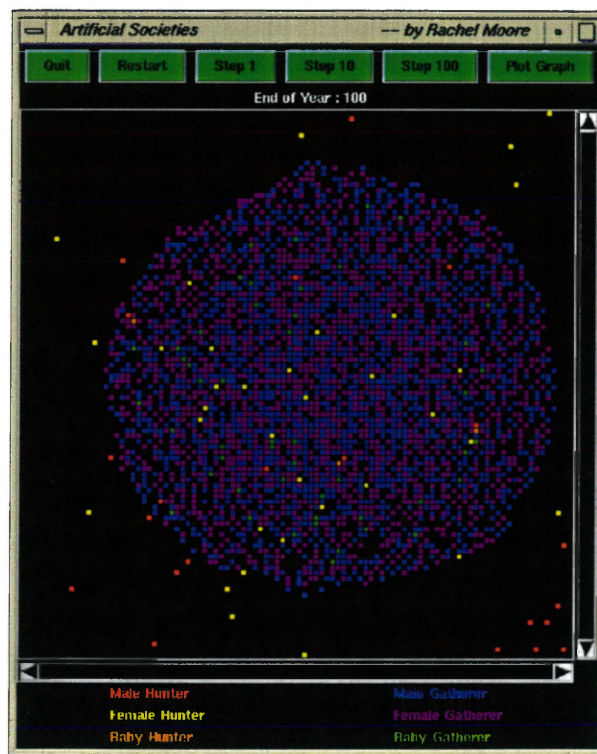
Figure 3.1: Interface Snapshot

Chapter 4

Experiments

This chapter describes the simulation experiments performed using the model in Chapter two. Representative time histories of population dynamics are shown and some interface snapshots are included to demonstrate movement patterns. Discussion of these experimental results is interleaved throughout the chapter.

Epstein and Axtell discuss a landscape "carrying capacity" of agents in their artificial society model [4]. They use this term to describe the steady-state (long-term) number of agents on the landscape. For example, if the simulation begins with more agents than the landscape's carrying capacity, as time evolves the agent population will be reduced by starvation. Conversely, if the simulation begins with less agents than the carrying capacity, as time evolves the process of agent reproduction will increase the population level to the carrying capacity.

In artificial society models with death and reproduction, as time evolves the agent populations will typically exhibit oscillatory behavior. This behavior makes it difficult to draw conclusions about the carrying capacity from just one realization (replication) of the

simulation. In my discussion, a carrying capacity is a steady-state mean population size. The simulation is replicated many times and these individual means are averaged to form the *steady-state carrying capacity* .

Based on a large amount of experimentation, I concluded that if the simulation is initialized with a large population of hunters and gatherers, one of the two outcomes below will always occur.

1. The hunters wipe out the entire gatherer population through combat and then the hunters slowly die of starvation. A typical example is illustrated in Figure 4.1, where the initial number of hunters and gatherers is 750 on a 100 x 100 landscape.
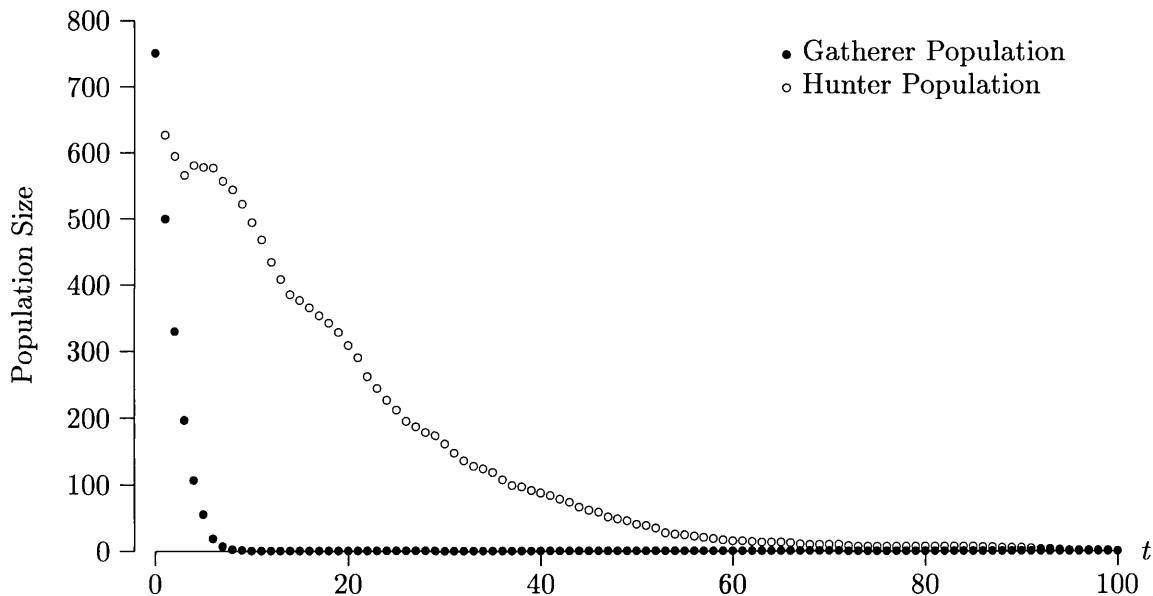
Figure 4.1: Extinction of Both Populations

2. The gatherers avoid extinction through combat because of their high and increasing wealth. Accordingly, the hunters die out and the gatherers flourish "forever". Figure 4.2 is a typical illustration. Note that the vertical axis is extended in Figure 4.2

relative to Figure 4.1 to show the dramatic increase in the gatherer population.
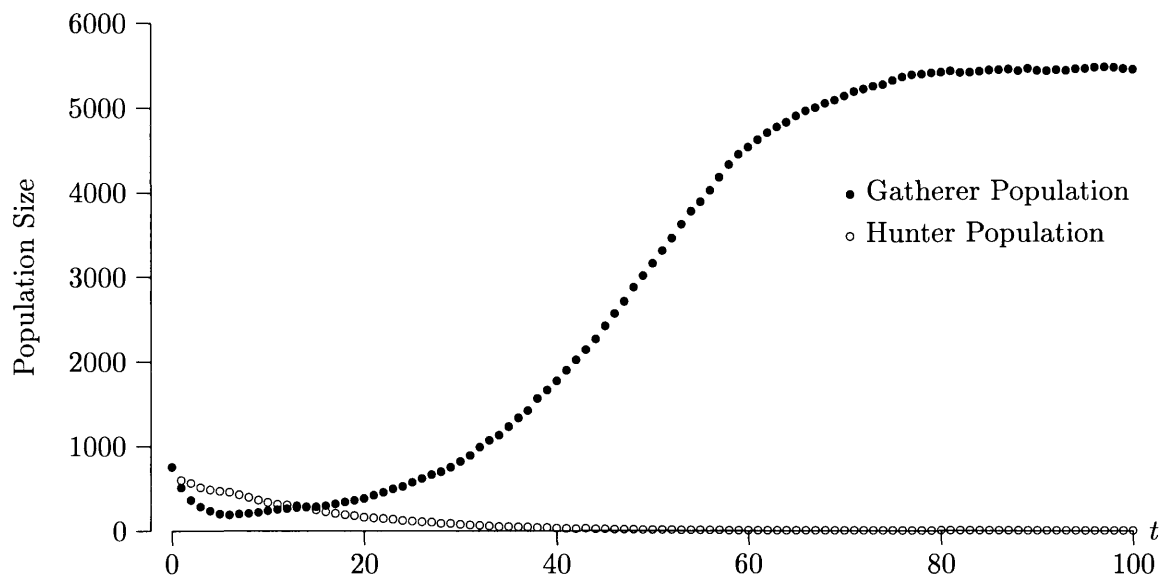
Figure 4.2: Gatherers Withstand Hunter Attack

Although I attempted to find agent and landscape attributes that would produce a steady-state interdependence of hunters and gatherers, I was not able to do so. Consequently, I altered the simulation model to begin with gatherers only and then "inject" hunters near the "edges" of the (2-D) landscape at each time step. This change has resulted in the interesting population dynamics discussed in Section 4.2 and Section 4.3.

Two sets of experiments are described in this chapter. In each set, as time evolves, a population of gatherers (initial size A) is subjected to a stochastic "injection" of hunters. The hunters are injected (created) randomly on the periphery of the landscape (no more than 20 cells from the "edge" in each direction). At each time step, a call to Geometric(p) ($0 \leq p < 1$) is made which determines the number of hunters to inject onto the landscape. (See Appendix A for stochastic function implementation.) The mean number of hunters injected per time step is $\mu = p/(1 - p)$. The following discussions will be in terms of the

injection rate μ rather than the corresponding probability parameter p.

The first set of experiments studies the effects of injecting hunters into a population of "weak" gatherers. In the second set, a "strong" population of gatherers is simulated. The characteristics of the hunters and the landscape are the same in both sets of experiments.

4.1 Setup

For all experiments, the dimensions of the landscape are $(X, Y) = (100, 100)$ and the stop time T is 2000.[1] The growback rate $\alpha(x, y)$ is a constant 2.4 for all (x, y). I chose this constant to be less than the average gatherer metabolism. If the growback rate were high relative to the gatherers' metabolism and cell capacities, starvation would be infrequent. If the growback rate were much smaller than the gatherers' metabolism and landscape cell capacities, gatherer starvation would be very frequent. As described in Section 2.2, the capacity function for cell (x, y) is

$$c(x, y) = 5e^{-((x-0.5X)/\sigma_x)^2 - ((y-0.5Y)/\sigma_y)^2}$$

with $\sigma_x = 0.54X, \sigma_y = 0.54Y$ and $x = 0, 1, 2, \ldots, X - 1, y = 0, 1, 2, \ldots, Y - 1$. For all experiments, the combat parameter n, used to shape the likelihood of a hunter defeating a gatherer in combat, is 1.0. (See Figure 2.3.)

All hunters in both experiment sets are characterized by the following tables of parameters. These parameter values determine the attributes and the initial states of the hunters.[2]

---

[1]For statistical purposes, if a gatherer population does not reach extinction by time T, it is assumed to survive "forever" with extinction time ∞.

[2]Not included in Table 4.2 are attributes `group = hntr` and `gender = Bernoulli(0.5)`.

The two deterministic attributes have no "Range" in Table 4.2; their values are listed in the corresponding "Expected Value" column. Stochastic parameters have a range (a, b) in both tables. These values are used in function calls to `Equilikely(a,b)` if integer-valued or to `Uniform(a,b)` if real-valued. The real-valued parameters are noted by an *.

Parameter	Variable Name	Range (a, b)	Expected Value
Initial Age	age	0 – 60	30
Initial Wealth*	wealth and init_wealth	50 – 70	60

Table 4.1: Parameters for Initial Hunter States

Parameter	Variable Name	Range (a, b)	Expected Value
Starvation Threshold (γ)	GAMMA		15.0
Reproduction Threshold (θ_h)	THETA_H		21
Vision (FOV)	vision	4 – 9	6.5
Metabolism*	metabolism	2 – 7	4.5
Female Fertility Begin	begin_fertility	5 – 7	6
Female Fertility End	end_fertility	60 – 80	70
Male Fertility Begin	begin_fertility	7 – 9	8
Male Fertility End	end_fertility	70 – 80	75
Female Lifetime	lifetime	90 – 100	95
Male Lifetime	lifetime	100 – 110	105

Table 4.2: Parameters for Hunter Attributes

4.2 Experiment Set 1 — "Weak" Gatherers

This set of experiments is based on "weak" gatherers, described by the parameter values in Tables 4.3 and 4.4.[3] The deterministic parameters do not have "Range" entries; their values are in the "Expected Value" column. The "Range" entries for stochastic parameters are used in calls to function `Equilikely(a,b)` if integer-valued or to function `Uniform(a,b)` if real-valued. The two real-valued parameters are designated with an *.

Parameter	Variable Name	Range (a,b)	Expected Value
Initial Number of Agents (A)	NUM_AGENTS		2500
Initial Age	age	0 — 60	30
Initial Wealth*	wealth	20 — 50	35

Table 4.3: Parameters for "Weak" Gatherer Initial States

| Parameter	Variable Name	Range (a,b)	Expected Value
Reproduction Threshold (θ_g)	THETA_G		15
Vision	vision	1 — 4	2.5
Metabolism*	metabolism	3 — 7	5
Female Fertility Begin	begin_fertility	12 — 15	13.5
Female Fertility End	end_fertility	50 — 70	60
Male Fertility Begin	begin_fertility	15 — 19	17
Male Fertility End	end_fertility	50 — 70	60
Female Lifetime	lifetime	80 — 100	90
Male Lifetime	lifetime	80 — 100	90

Table 4.4: Parameters for "Weak" Gatherer Attributes

I experimented to determine an appropriate initial number of gatherers A for the simulation experiments in this set. When no hunters are injected into the landscape, the gatherer

---

[3]Not included in Table 4.4 are attributes `group = gthrr` and `gender = Bernoulli(0.5)`.

population size oscillates "forever" about a steady-state mean value — the carrying capacity. A typical realization of an initial gatherer population of $A = 2500$ with $\mu = 0.0$ is shown in Figure 4.3. (The dashed line illustrates the steady-state mean gatherer population based on one hundred replications. It is not the mean of this particular realization.)
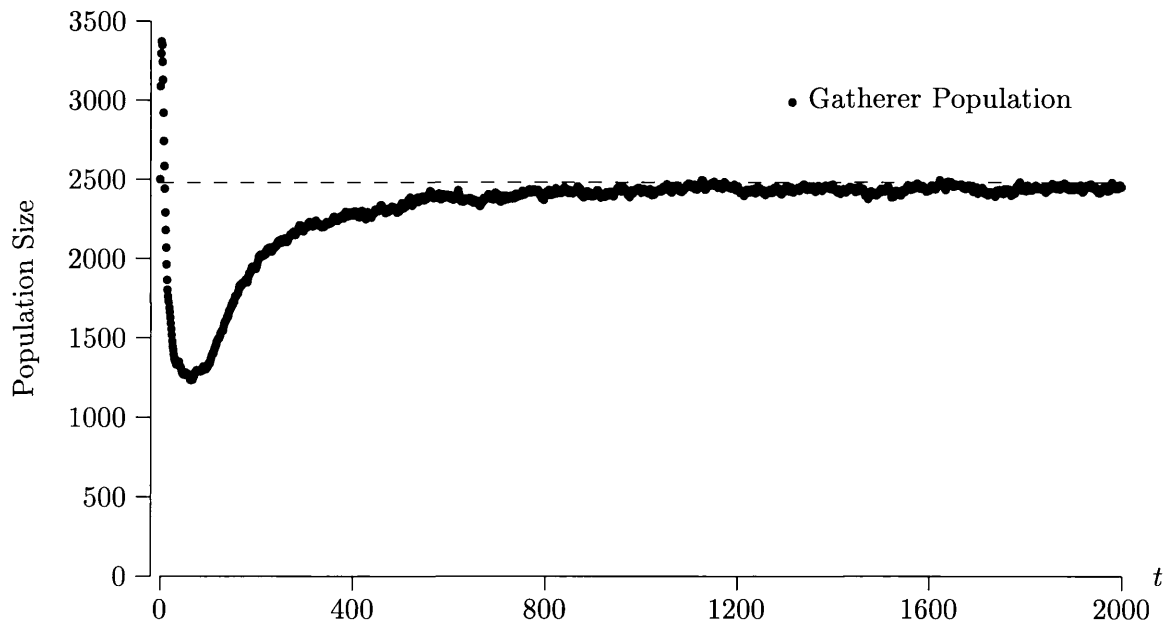

Figure 4.3: "Weak" Gatherers with No Hunters

For each point in Figure 4.4, the simulation was replicated one hundred times with stopping condition $T = 2000$.[4] For each replication, to avoid the effects of the initial transient, the sample mean of the gatherer population was computed between times $t = 300$ and $t = 2000$. A • point, with an associated vertical bar, corresponds to the 95% confidence interval calculated from one hundred mean gatherer populations. (The dashed lines are the mean population sizes.) As illustrated, as the initial gatherer population size increases, the

---

[4]Each replication used a different random number generator seed. See Appendix A for `rngs` library implementation.

steady-state gatherer population also increases, albeit slowly. This demonstrates that there

is not a well-defined carrying capacity in this case, independent of A.

If there were a well-defined carrying capacity, the steady-state gatherer population would

be the same regardless of initial population size and I would choose the initial number of

gatherers to be this steady-state population value. All experiments in this set use an initial

gatherer population of $A = 2500$, with a corresponding steady-state gatherer population of

2480.82 ± 2.41. If, in Figure 4.4, I were to construct the curve intersecting the midpoint of

the generated interval estimates, the point of intersection between it and the diagonal line

shown would be a justification for choosing A close to 2500.
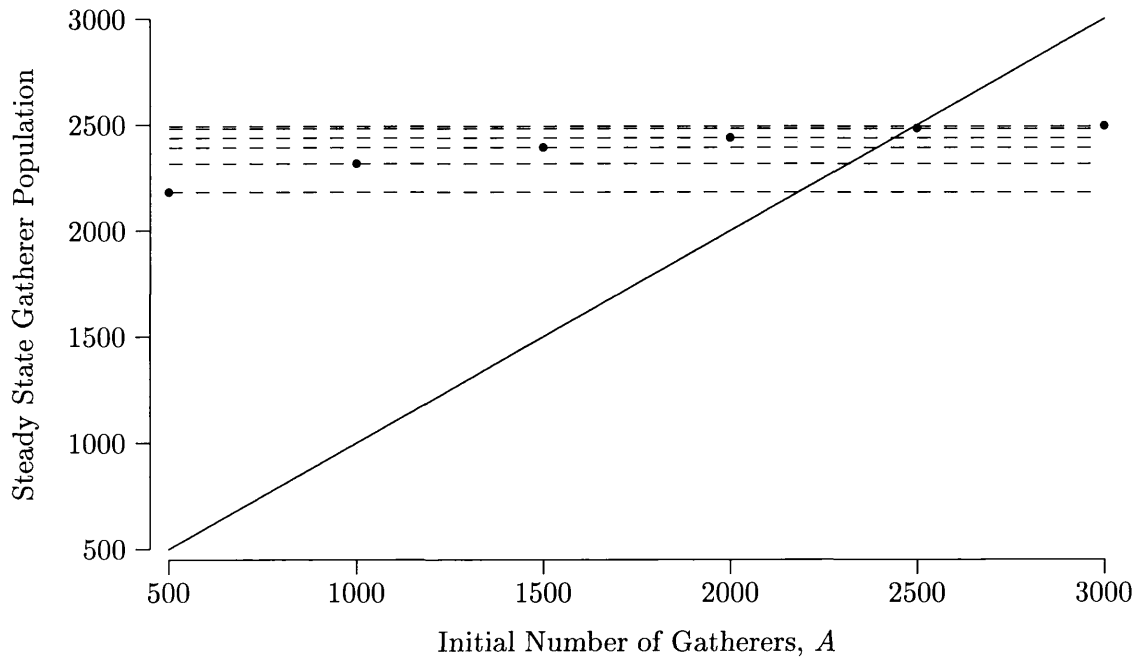
Figure 4.4: Steady-State "Weak" Gatherer Population

The only parameter varied in the following experiments is μ, the rate at which hunters

are injected into the landscape. Recall that $\mu = p/(1 - p)$ where p is the parameter in

stochastic function `Geometric(p)`. The value of μ ranges from 0 to 10 in this experiment set.

Table 4.5 summarizes the results of these experiments. These results suggest there are two threshold values for μ. A value of μ less than the low threshold indicates that, with probability approaching 1.0, a "weak" gatherer population will not reach extinction. A value of μ higher than the upper threshold indicates that, with probability approaching 1.0, a "weak" gatherer population will reach extinction. Because of time constraints I have not attempted to find precise values for these two thresholds.

μ	Gatherer Carrying Capacity	Average Extermination Time
0.00	2480.82 ± 2.41	∞
1.00	2377.51 ± 1.85	∞
2.00	2227.72 ± 2.40	∞
3.00	2018.41 ± 2.49	∞
4.00	1709.18 ± 5.12	∞
4.25	1546.94 ± 64.66	∞
4.50	1220.17 ± 90.03	∞
4.75	956.56 ± 117.25	∞
5.00	688.71 ± 119.54	∞
5.25	378.22 ± 101.05	∞
5.50	216.33 ± 89.10	∞
5.75	58.28 ± 50.84	∞
6.00	7.51 ± 14.91	∞
6.50	0	313.30 ± 40.93
7.00	0	220.49 ± 17.42
8.00	0	157.12 ± 7.21
9.00	0	134.58 ± 6.19
10.00	0	116.54 ± 4.14

Table 4.5: Summary of Results for "Weak" Gatherers

Figure 4.5 illustrates the probability of a gatherer population reaching extinction (as 95% confidence interval estimates) for the values of μ used in these experiments. The S-

shaped curve in Figure 4.5 indicates that the lower threshold is between $\mu = 4.00$ and

$\mu = 4.25$ and that the higher threshold is between $\mu = 6.00$ and $\mu = 6.50$.
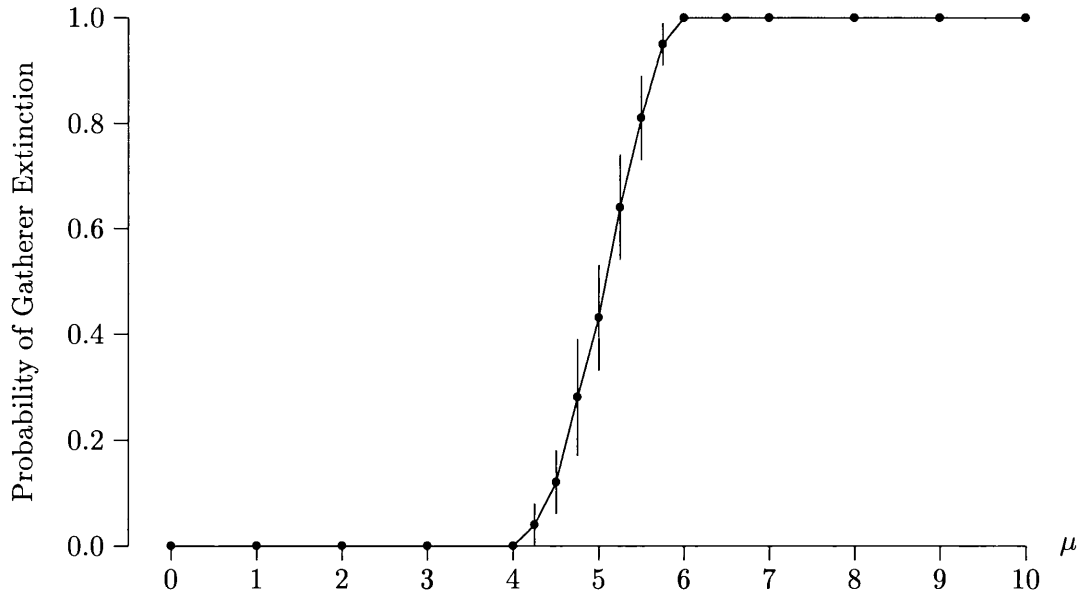
Figure 4.5: Probability of "Weak" Gatherer Extinction by $t = 2000$

As expected, I found the same behavior when I increased the stop time of the simulation

to $T = 10000$. Figure 4.6 illustrates that some of the populations which were still alive at

time $T = 2000$ did reach extinction by $T = 10000$. Based on the results in Figure 4.6, as T is

increased to infinity, I expect there will still be two extinction probability thresholds which

generate a similar S-shaped curve. This curve will be shifted to the left when compared

with results using a finite value for T. Because infinity cannot be computationally modeled,

I did not increase the stop time beyond $T = 10000$. All results in Sections 4.2 and 4.3, with

the exception of Figure 4.6, are based on a stop time of $T = 2000$.

Figure 4.7 shows the steady-state gatherer populations and mean extinction times based

on one hundred replications. The • points and corresponding vertical bars on the left-hand
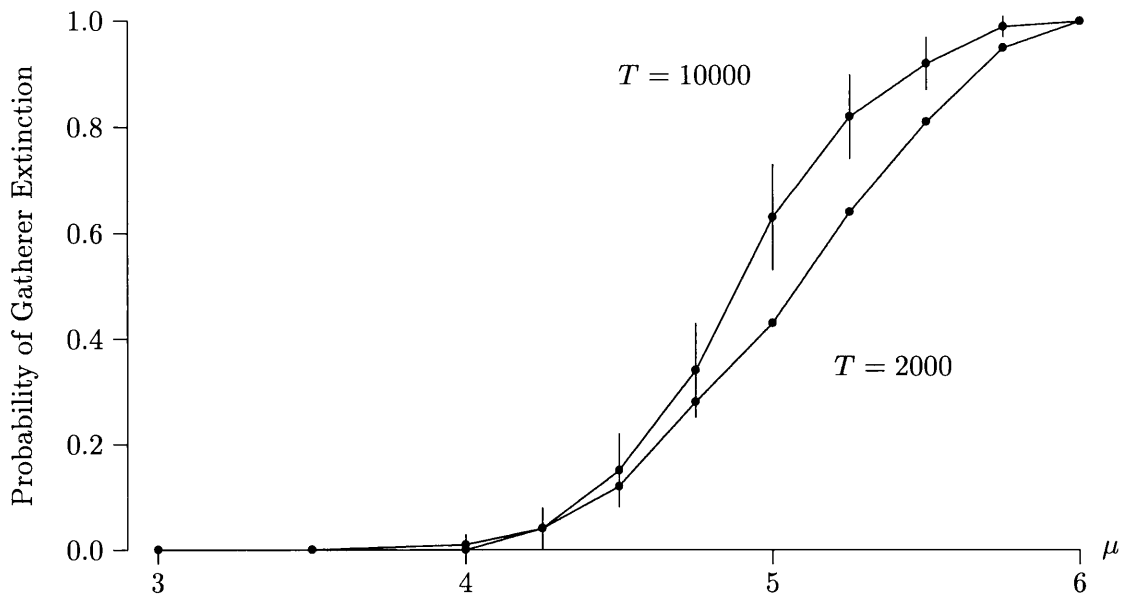
Figure 4.6: Probability of "Weak" Gatherer Extinction by $t = 10000$

curve are 95% confidence interval estimates for steady-state gatherer populations. For values of μ where all replications produced gatherer extinction, the corresponding • point for the steady-state gatherer population is zero. In Figure 4.7, the • points and corresponding vertical bars on the right-hand curve are 95% confidence interval estimates for the mean gatherer extinction times. For values of μ when at least one replication did not produce extinction (in $T = 2000$ time steps), there is no • point for an extinction time.

Before I began these experiments, I expected that a high rate of hunter injection into the landscape would extinguish the gatherer population quickly. I also expected the probability of this extinction to increase as the injection rate μ increased. However, I had no idea for what range of μ's this probability increase would occur or what the slopes of the curves in Figure 4.5 and Figure 4.7 would be. Consistent with the results in Figure 4.7 and Table 4.5, as μ increases, the mean gatherer population and extinction times decrease. In addition,
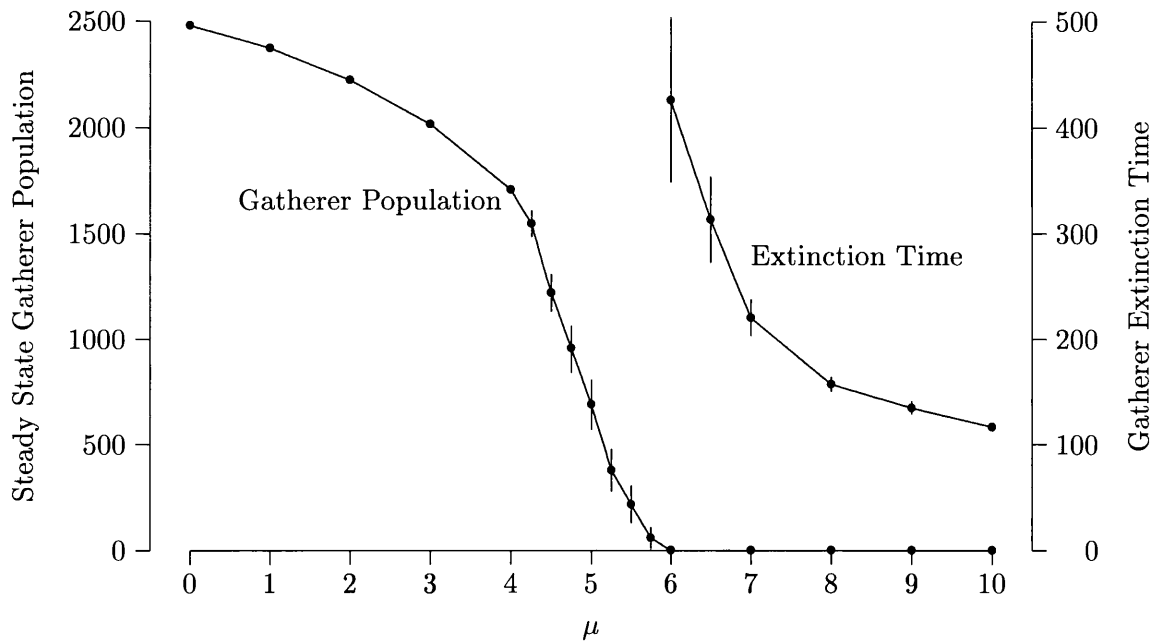
Figure 4.7: Steady State "Weak" Gatherer Populations and Extinction Times

the probability of "weak" gatherer extinction increases rapidly if μ is larger than the lower threshold value of μ.

Figures 4.8 through 4.10 illustrate representative time histories for $\mu = 2.0, 5.0$ and 9.0. All three vertical axes have the same scale, but the horizontal axes have been shortened in the two cases where the gatherer population became extinct.

Figure 4.8 confirms that an average injection of $\mu = 2.0$ hunters per time period decreases the gatherer population, but does not cause extinction. The gatherer population size oscillates dramatically, when compared to the gatherer carrying capacity in Figure 4.3 with no hunter injection. This is an example of gatherers withstanding a modest rate of hunter injection and the resulting combat that occurs. Note that there is a small hunter population sustained over time by new hunter injections and reproduction. If hunter injections were

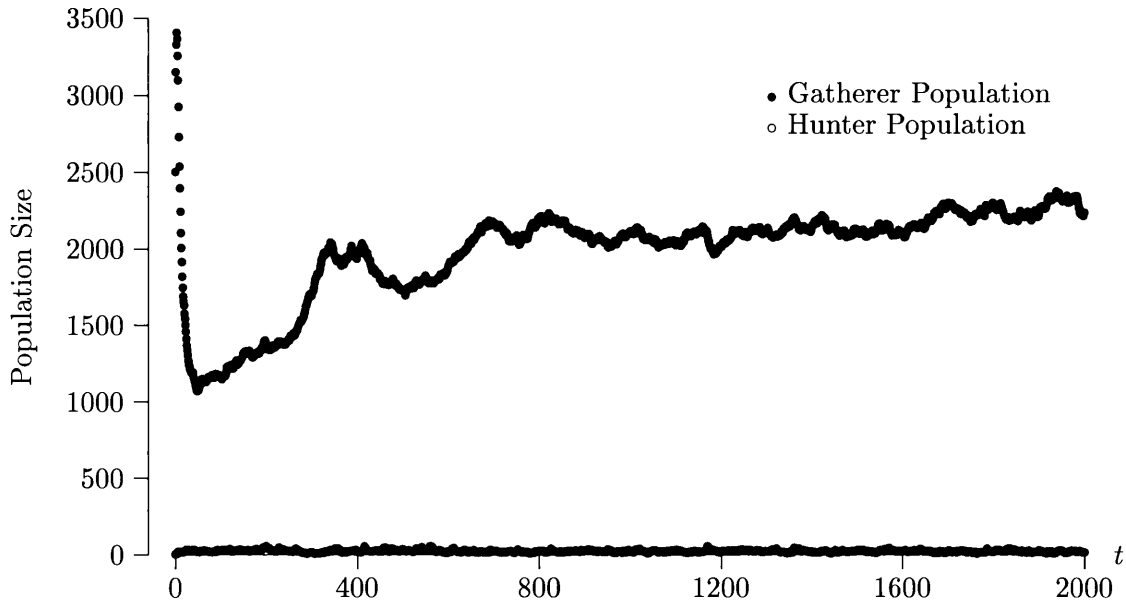terminated, this hunter population would drop to zero.

Figure 4.8: "Weak" Gatherer Time History — $\mu = 2.0$

As μ increases, the gatherer population is more adversely affected. In Figure 4.9, with

$\mu = 5.0$ the gatherer population reaches extinction at time $t = 476$. This is one of forty-

three replications in a set of one hundred where injections of five hunters (on average) per

time step drove the gatherer population to extinction. (See Figure 4.5.)

With a higher value of μ, the gatherer population dies out rapidly, as illustrated in

Figure 4.10 with $\mu = 9.0$. The first few time steps show significant gatherer reproduction,

but the hunters then quickly begin to defeat the gatherers in combat. Notice that as the

gatherers die, the hunter population increases. The wealth accumulated by the hunters

from the defeated gatherers and the higher injection rate allows the hunters to live longer

and mate more often. If hunter injection is terminated, the hunter population will soon
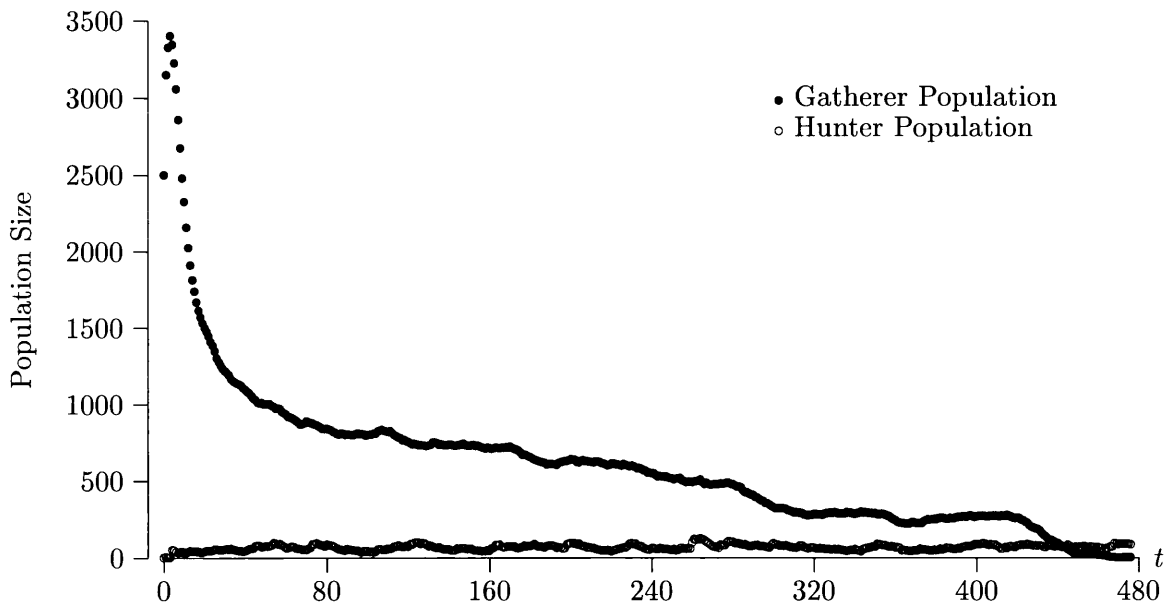
become extinct.

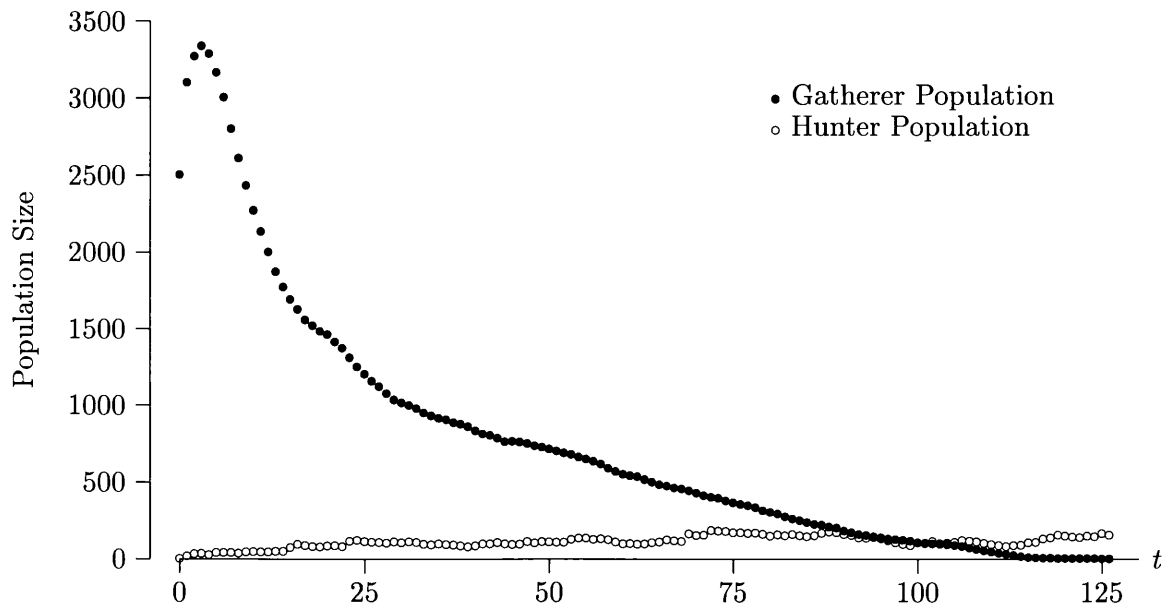**Figure 4.9**: "Weak" Gatherer Time History — $\mu = 5.0$

Figure 4.10: "Weak" Gatherer Time History — $\mu = 9.0$

4.3 Experiment Set 2 — "Strong" Gatherers

This set of experiments is on "strong" gatherers. On average, strong gatherers have larger FOVs and higher initial wealth than the "weak" gatherers. Also, their average metabolism rate is lower and both the male and female gatherers begin their fertility period earlier than the weak gatherers. The initial states and attributes of the gatherers are listed below in Tables 4.6 and 4.7.[5] The values of the two deterministic parameters are shown in their corresponding "Expected Value" entry. Real-valued variables are marked with an *. The same characteristics are used for hunters and the landscape as were discussed in section 4.1.

Parameter	Variable Name	Range	Expected Value
Initial Number of Agents (A)	NUM_AGENTS		5000
Initial Age	age	0 – 60	30
Initial Wealth*	wealth	30 – 50	40

Table 4.6: Parameters for "Strong" Gatherer Initial States

Parameter	Variable Name	Range	Expected Value
Reproduction Threshold (θ_g)	THETA_G		20
Vision (FOV)	vision	2 – 5	3.5
Metabolism*	metabolism	2 – 4	3
Female Fertility Begin	begin_fertility	5 – 7	6
Female Fertility End	end_fertility	50 – 70	60
Male Fertility Begin	begin_fertility	5 – 7	6
Male Fertility End	end_fertility	50 – 70	60
Female Lifetime	lifetime	80 – 100	90
Male Lifetime	lifetime	80 – 100	90

Table 4.7: Parameters for "Strong" Gatherer Attributes

---

[5]Not included in Table 4.7 are attributes group = gthrr and gender = Bernoulli(0.5).

I experimented to determine an appropriate initial number of strong gatherers A for the simulation experiments in this set. When no hunters are injected into the landscape, the gatherer population will oscillate "forever" around a steady-state mean value. Figure 4.11 is a typical example where $A = 5000$. (The dashed line is the steady-state mean gatherer population based on one hundred replications. It is not the mean of this particular realization.) If the size of the left axis were magnified, population oscillations would be more noticeable.
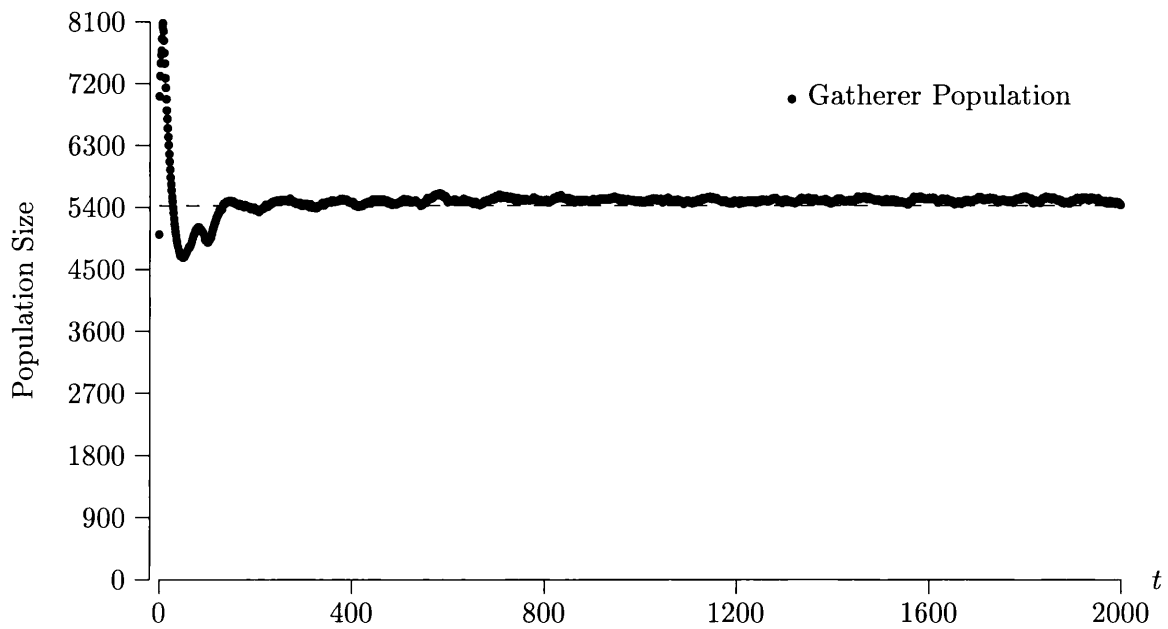
Figure 4.11: "Strong" Gatherers with no Hunters

For each point in Figure 4.12, the simulation was replicated one hundred times with stopping condition $T = 2000$. For each replication, to avoid the effects of the initial transient, the sample mean of the gatherer population was computed between times $t = 300$ and $t = 2000$. A • point with an associated vertical bar corresponds to the 95% confi-

dence interval calculated from one hundred gatherer population means. As in the first set of experiments, the initial number of gatherers does have an impact on the mean steady-state gatherer population. I chose the initial gatherer population size $A = 5000$ for all experiments in this set because it is closest multiple of 1000 to the intersection of the diagonal line shown and the curve connecting the interval estimates. The corresponding mean steady-state gatherer population is 5422.32 ± 1.01.
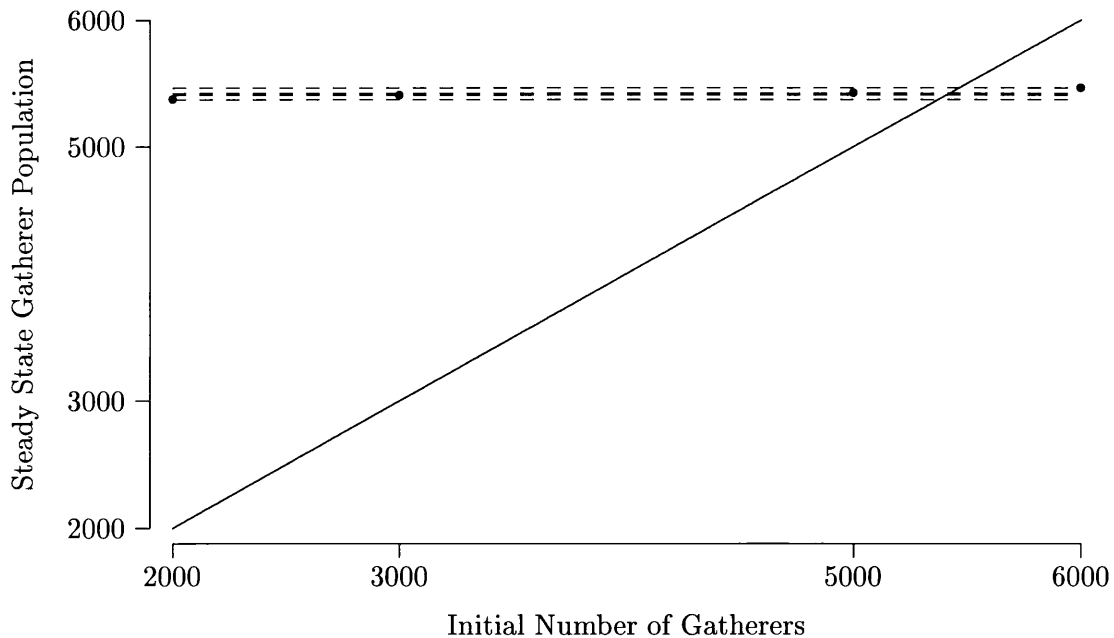
Figure 4.12: Steady State "Strong" Gatherer Population

The only parameter varied in this set of experiments is μ, the rate at which hunters are injected. The value of μ ranges from 0 to 32. Table 4.8 shows the results of these experiments. The interval estimates in Table 4.8, Figure 4.13, and Figure 4.14 are all based on one hundred replications.

The results in Table 4.8 indicate that there are two threshold values for "strong" gath-

μ	Gatherer Carrying Capacity	Average Extermination Time
0.00	5422.32 ± 1.01	∞
4.00	5072.23 ± 2.95	∞
8.00	4294.22 ± 2.64	∞
9.00	3673.70 ± 3.76	∞
10.00	3084.19 ± 16.47	∞
11.00	2474.33 ± 135.97	∞
11.25	2200.43 ± 169.95	∞
11.50	1826.31 ± 205.01	∞
11.75	1342.43 ± 223.40	∞
12.00	779.47 ± 207.51	∞
12.50	263.15 ± 130.40	∞
13.00	0	935.54 ± 68.95
14.00	0	529.66 ± 41.83
15.00	0	379.50 ± 30.14
16.00	0	285.57 ± 22.57
20.00	0	135.01 ± 7.82
24.00	0	104.30 ± 3.65
28.00	0	86.82 ± 2.88
32.00	0	77.89 ± 2.24

Table 4.8: Summary of Results for "Strong" Gatherers

erers. A value of μ less than the lower threshold indicates that any "strong" gatherer population, with probability approaching 1.0, will not reach extinction. A value of μ above the higher threshold indicates that any "strong" gatherer population, with probability approaching 1.0, will reach extinction. I generated the interval estimates shown in Figure 4.13 to determine an approximation of the μ threshold values. As in the "weak" gatherer experiment set, there is a dramatic rise in the probability of gatherer extinction between two estimated thresholds. In this set, the lower threshold is between $\mu = 10.0$ and $\mu = 11.0$ and the upper threshold is between $\mu = 12.5$ and $\mu = 13.0$.

Although I did not increase the stop time (T) for comparison in this set of experiments,
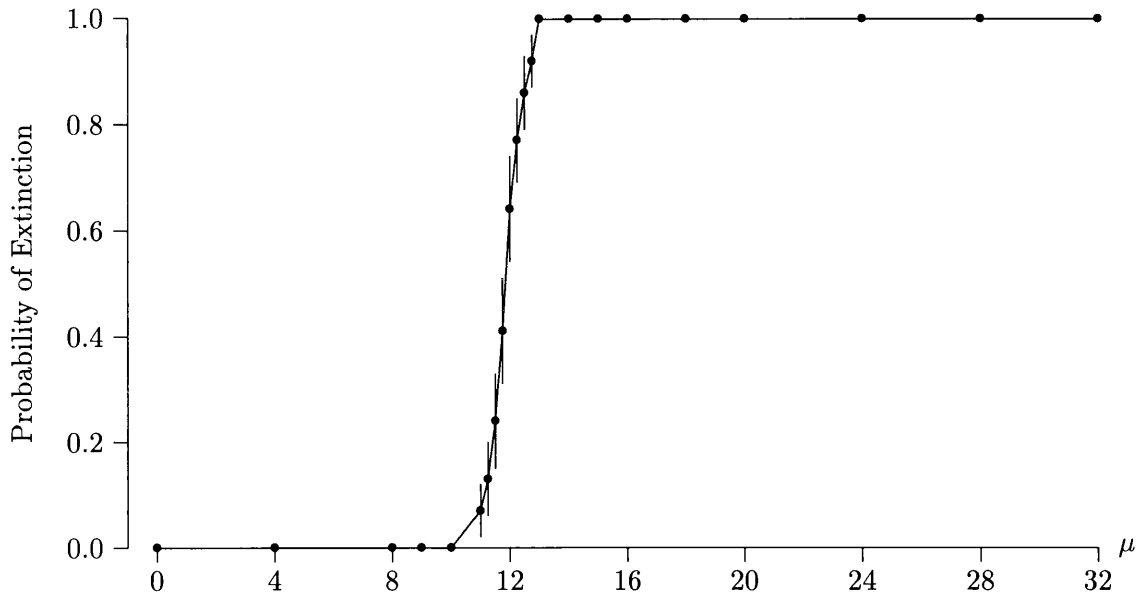
Figure 4.13: Probability of "Strong" Gatherer Extinction by $t = 2000$

I expect similar results as in the experiment set with "weak" gatherers. That is, I expect that if T were increased significantly, the extinction probability curve would be shifted left from the curve that is shown in Figure 4.13.

Figure 4.14 shows the steady-state gatherer populations and the extinction times. Each • point and corresponding vertical bar on the left-hand curve is an interval estimates of the mean steady-state gatherer population for the particular value of μ. Each • point and corresponding vertical bar on the right-hand curve is an interval estimate of the mean extinction time for the particular value of μ. For values of μ when in at least one replication a gatherer population did not reach extinction (by $T = 2000$), there is no • point for an extinction time. For values of μ where all replications produced gatherer extinction, the corresponding • point for the steady-state gatherer population is zero.

Before I began these experiments, I expected a high rate of hunter injection to abolish
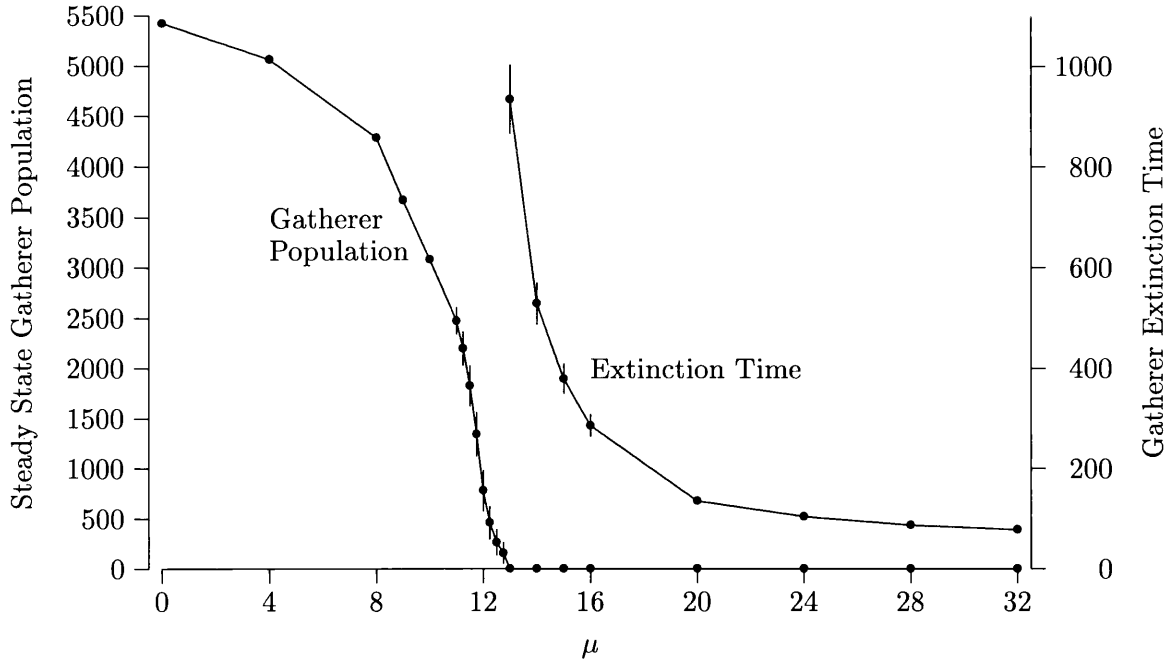
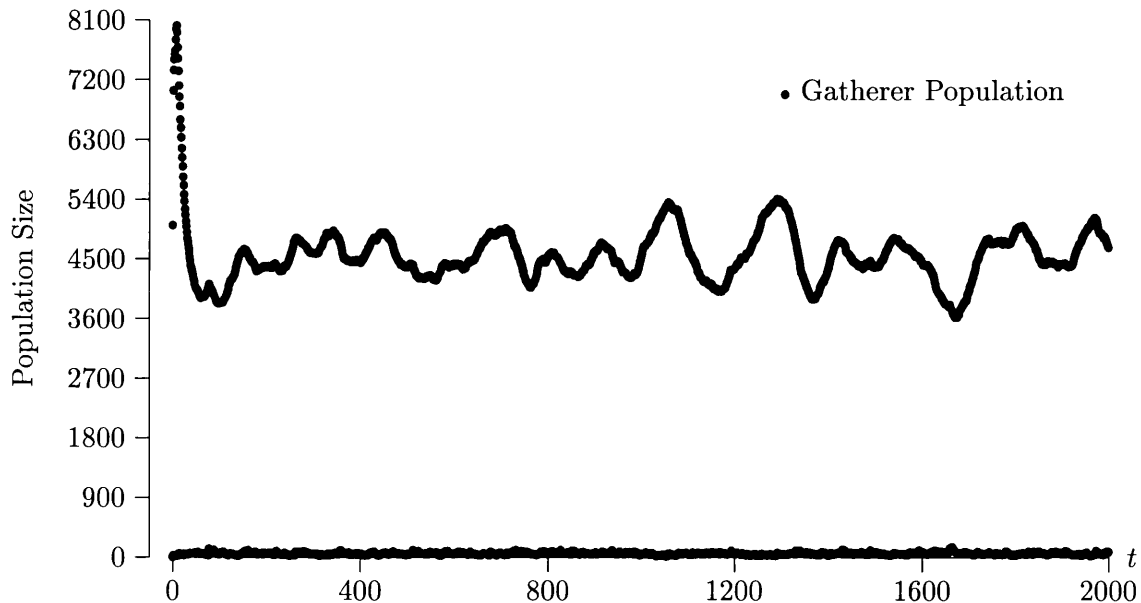**Figure 4.14**: Steady State "Strong" Gatherer Populations and Extinction Times

the "strong" gatherer population quickly. I expected this rate to be higher than the rate

at which "weak" gatherer populations are abolished. I expected the probability of "strong"

gatherer extinction to increase as the injection rate increases, but I did not know for what

range of μ's this would happen or what the "slopes" in Figure 4.13 and Figure 4.14 would

be. Consistent with the results in Table 4.8, Figure 4.13, and Figure 4.14, higher values of

μ produce a smaller steady-state gatherer population (if steady-state occurs) and an earlier

extinction time. The probability of "strong" gatherer extinction increases rapidly if μ is

between the two threshold values of μ.

The values of μ studied in this experiment set are higher than in the first experiment

set because the "strong" hunters are less vulnerable to combat with hunters. Regardless,

the same societal behavior occurs and the curves represented in Figure 4.7 and Figure 4.13

are similar. The extinction time curve suggests that there is a limiting extinction time —

the hunters can defeat gatherers only so fast.

Figures 4.15 through 4.18 are representative time histories of experiments in this set for $\mu = 8.00$, 11.75, 15.00 and 32.00. All four vertical axes have the same scale, but the horizontal axes have been shortened in two cases where the gatherer population became extinct before $t = 2000$.

Figure 4.15 shows a typical replication in which the value of μ is less than the low threshold discussed above. The gatherer population in this figure does reach a steady-state population size. At the rate $\mu = 8.00$, the gatherer population withstands hunter attack "forever". If compared with the gatherer population in Figure 4.11 ($\mu = 0$), the gatherers in Figure 4.15 have a lower steady-state population size. Thus, an average of eight hunters injected per time step does have a slight, adverse affect on the gatherer population but does not drive the gatherers to extinction.

Figure 4.15: "Strong" Gatherer Time History — $\mu = 8.00$

The gatherer population in Figure 4.16 oscillates much more dramatically than the population in Figure 4.15. Note that $\mu = 11.75$ is between the two extinction probability thresholds and so the gatherer population will reach extinction before $T = 2000$ with probability greater than zero. For this realization, the gatherer population reaches extinction at time $t = 1488$.
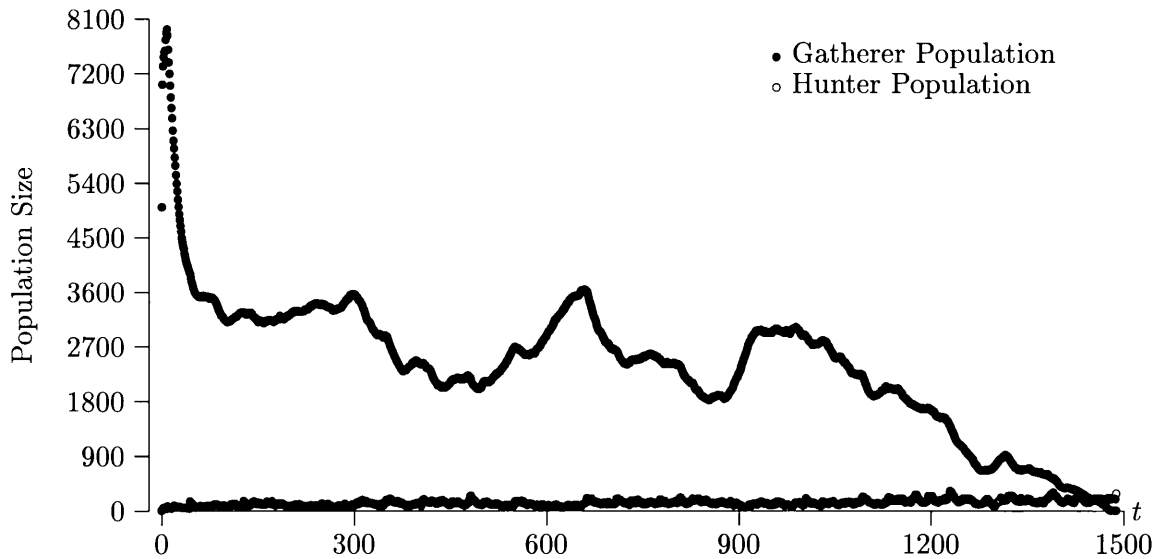
Figure 4.16: "Strong" Gatherer Time History — $\mu = 11.75$

Figure 4.17 is a representative time history where $\mu = 15.00$ is higher than the upper thresholds and so any "strong" gatherer population will reach extinction with probability close to 1.0. Note that the gatherers are extinct at time $t = 376$ after a somewhat gradual decline in population size.

Extremely large values of μ do not allow gatherer populations to oscillate or gradually decline as in Figure 4.15 and Figure 4.16 where μ is moderate. Figure 4.18 shows a steep decline of gatherers, reaching extinction at time $t = 80$. As expected, if μ is large the resulting extinction occurs much sooner than if μ is small.
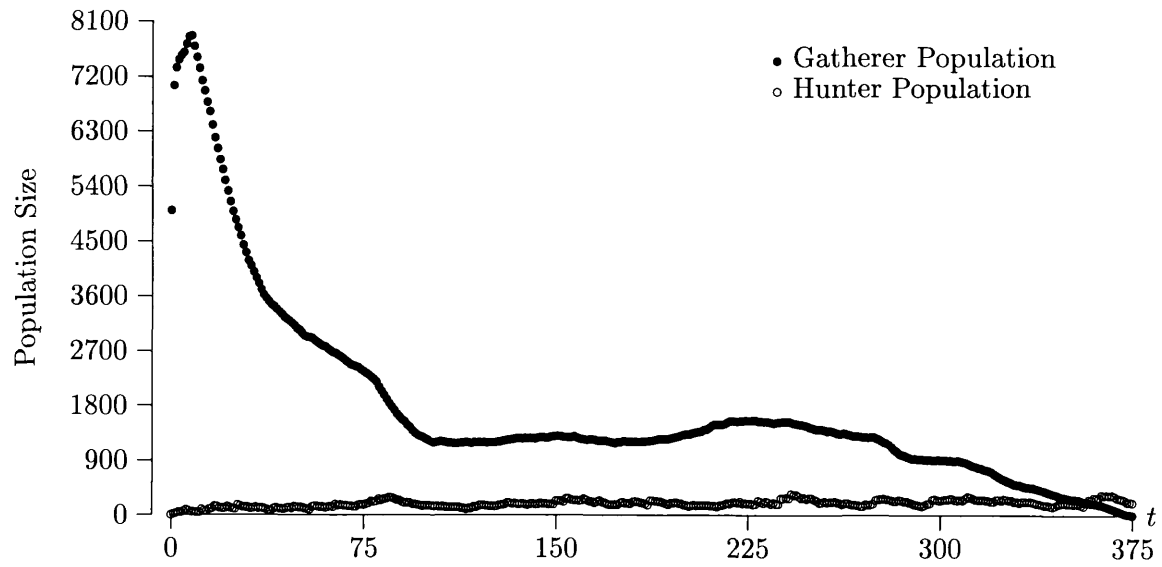
Figure 4.17: "Strong" Gatherer Time History — $\mu = 15.00$

Figure 4.18: "Strong" Gatherer Time History — $\mu = 32.00$

4.4   Interface Results

Recall that Chapter three discussed the visualization interface. I used this interface to verify several of the time history figures in Sections 4.2 and 4.3. Figure 4.19 depicts "strong" gatherers with an average of $\mu = 15.00$ hunters injected per time step. The corresponding time history is illustrated in Figure 4.17. The agent movement trends in this example are common with "weak" and "strong" gatherers and occur regardless of the value of μ.

Time evolves in Figure 4.19 from left to right and top to bottom. The interface snapshots are taken at times $t = 10$, 50, 100, 200, 300 and 372. Figure 4.19 clearly verifies that gatherers move towards the center of the landscape over time. They congregate at the center because the highest resource amounts are at the center of the landscape, where the cell resource capacity is largest. Recall that hunters are injected onto the landscape periphery. Some hunters are able to migrate to the center of the landscape in search of vulnerable gatherers to defeat in combat.

In the realization show in Figure 4.17 and Figure 4.19, after $t = 15$ the number of baby agents (both hunters and gatherers) created via reproduction was less than eighty per time period (see Figure 4.20). The horizontal axis begins at time $t = 15$ because the previous reproduction levels are large and, if displayed in the same figure with births after $t = 15$, variations in the latter would be hard to resolve. As expected, in Figure 4.20 the number of reproductions per time period decreases as the number of living agents decreases.

The gatherers near the center of the landscape are too densely packed for babies to be created since an empty cell neighboring one or both parents is required for an agent birth. The gatherers on the "edge" of the cluster are those most likely to engage in reproduction,

but they are also the most vulnerable to combat and starvation because they face the in-
coming hunters and are located at cells with lower resource capacities. Hunter reproduction
is limited because of the sparseness of the hunter population. Hunters that do not reach
the gatherer cluster often do not get a chance to reproduce because they have metabolized
their wealth to an amount lower than the hunter reproduction threshold. Male hunters that
do infiltrate the cluster can choose to "rape" the female gatherers, but only if there is an
empty cell for the baby. Hunters at or inside the gatherer cluster also face the challenge of
finding a fertile hunter of the opposite sex.

4.5   Simulation Verification

When simulating any system, it is crucial to verify that the model at the computational
level and simulation results are consistent with the model at the conceptual and specification
levels [10]. I used many consistency checks to ensure simulation correctness. The following
are just some of the consistency checks for this complex simulation model.

- Because multiple births are not possible, the total number of births in a time step
  must be less than or equal to of the total number of female agents at the start of that
  time step. The number of baby agents visible on the Tcl interface must equal the
  number of births in that time step.

- The number of combats in a time step must be no more than the number of hunters.

- The average of the stochastic attributes generated should equal the mean of the
  stochastic distribution used. For example, hunter vision was generated in all experi-
  ments with a function call to `Equilikely(4,9)`. At any time step in the population,

the mean of the hunter visions should be close to $6.5 = (4 + 9)/2$. As more hunters are included in the average, the mean vision will approach 6.5.

- I used the Tcl interface to determine whether hunters are injected on the periphery of the landscape. Because there is no way to determine which are the most recently injected hunters, this consistency check was helpful only if there were not too many hunters to track.

- The Tcl interface was also useful in checking reproduction proximities. A baby gatherer must be next to at least one gatherer, which must be next to its mate. A baby hunter must be next to a hunter or a "raped" female gatherer, which must also be next to its mate.
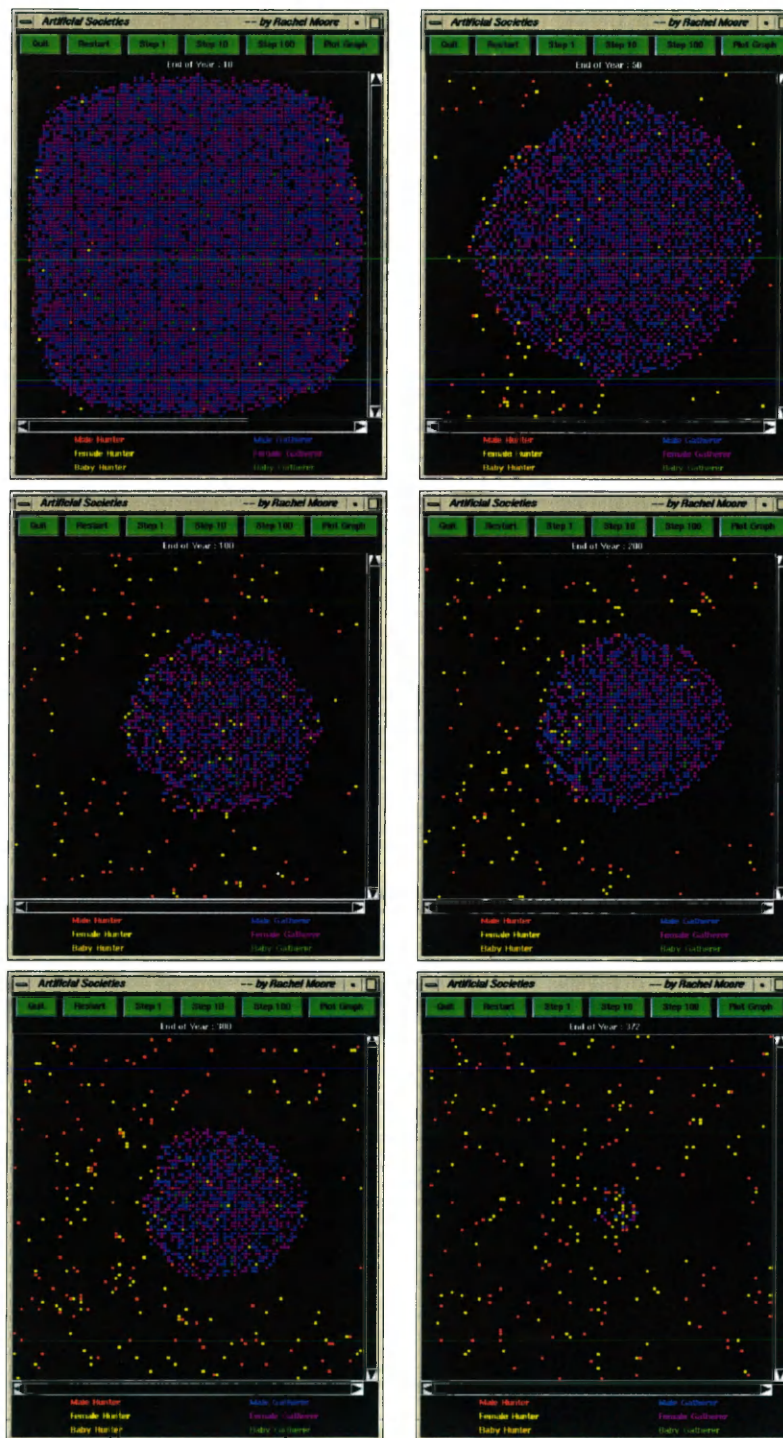
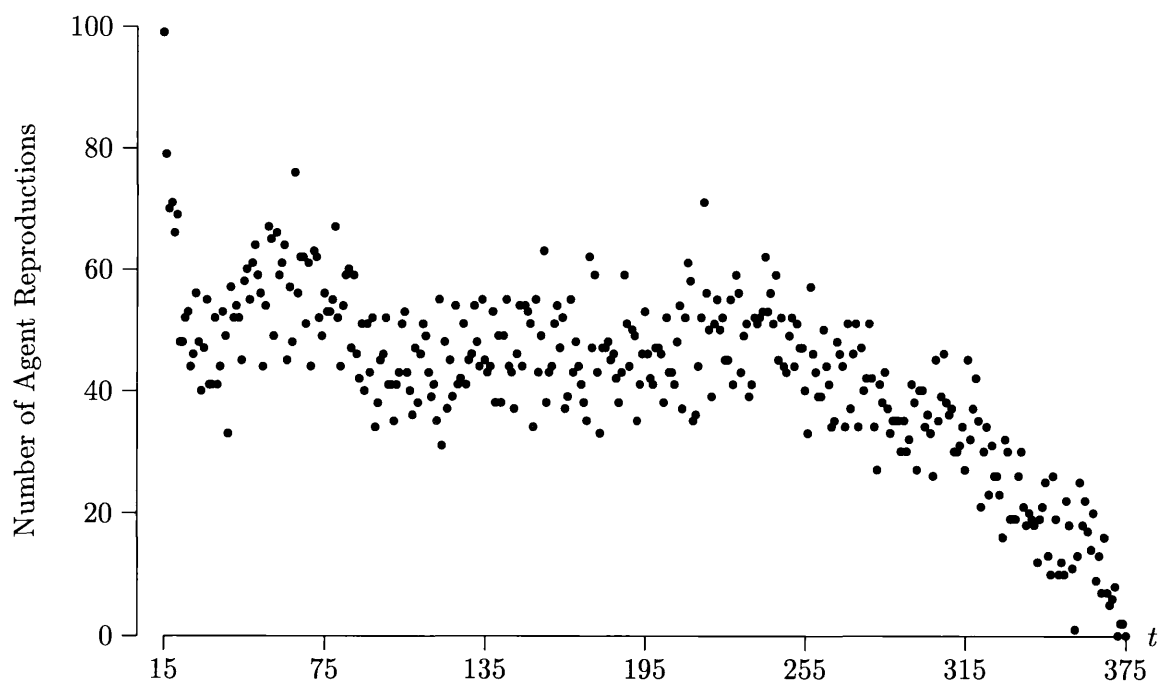**Figure 4.19**: Evolution of "Strong" Gatherers — $\mu = 15.00$

Figure 4.20: Agent Reproduction Levels — $\mu = 15.00$

Chapter 5

Conclusion

There is much more detail and complexity in the artificial society model developed in this dissertation than can be discussed here. I focused primarily on population dynamics and the natural rise and fall in populations due to agent combat. The following three points summarize key results from this simulation study.

- "Stronger" gatherers are better able to withstand hunter attack. They have a higher tolerance for hunters and, if the hunter injection rate is zero, their mean population size is much higher than that of "weaker" gatherers.

- The higher the hunter injection rate μ, the more rapidly the mean gatherer population size will diminish. If gatherer extinction occurs, a higher injection rate will force gatherer extinction to occur sooner, on average, than lower injection rates.

- The probability of the gatherer population successfully withstanding hunter attack and the probability of gatherer extinction are determined by the thresholds for μ. If μ is above the higher threshold, gatherer extinction is nearly certain. If μ is below the

lower threshold, gatherer populations are nearly certain to withstand hunter injection forever. If μ is between the thresholds the probability of extinction is positive and less than 1.0. Stronger gatherers have higher thresholds than weaker gatherers.

An issue for further study with the current model is to determine the earliest possible gatherer extinction time using large values of μ. Determining precise threshold values for "weak" and "strong" gatherers is another suggested study.

An extension to the current model is to modify the interface such that cell and agent states and attributes are accessible by clicking on an occupied landscape cell. Clicking on an unoccupied cell would only display the cell states and attributes. This modification would allow for easier and more thorough consistency checks.

Without changing the focus on population dynamics resulting from combat, there are many aspects and assumptions of this model that can be changed. Two significant aspects are the landscape's cell resource capacity function and the method used to "strengthen" gatherer populations. A resource capacity function with a different topology may change the population dynamics significantly. Strengthening gatherer populations by altering stochastic parameters other than those altered in this model could also provide variant combat trends and probability thresholds. For example, the "weak" gatherers had a lower average vision and initial wealth, higher average metabolism, and later times for the start of their fertility periods than "strong" gatherers. Eliminating some of these differences or adding others could change the vulnerability of gatherer populations to hunter attack.

Another extension to the model is to inject hunters onto a different region of the landscape. It may be true that there are waves of hunters (injected on the periphery) that

migrate towards the center — some successful and others not. Based on this movement trend, if the hunters were not injected on the periphery of the landscape but were injected close to the center or spread evenly throughout the landscape, the effects of the hunters on both "weak" and "strong" gatherers would probably be more severe and rapid.

Appendix A

Stochastic Functions

The following stochastic functions and excerpt from libraries **rngs** and **rvgs** were written by Steve Park and David Geyer [10]. The function `Random()` uses a Lehmer random number generator to produce a floating point value between 0.0 and 1.0. The function `PutSeed()` is used to initialize the generator. Each stochastic component in the computational model is allocated one of 256 disjoint random streams. A call to the function `SelectStream()` defines the active stream for the next random number to be generated. This allows for a unique source of randomness for each component [10]. The remaining functions are stochastic distributions described in Table A.1.

```
#define MODULUS    2147483647
#define MULTIPLIER 48271
#define CHECK      399268537
#define STREAMS    256          /* # of streams    */
#define A256       22925        /* jump multiplier */
#define DEFAULT    123456789    /* initial seed    */

static long seed[STREAMS] = {DEFAULT}; /* current state of each stream  */
static int  stream        = 0;         /* stream index, 0 is the default */
static int  initialized   = 0;         /* test for stream initialization */
```

```c
  double Random(void)
/* ==================================================================
 * Random returns a pseudo-random real number uniformly distributed
 * between 0.0 and 1.0.
 * ==================================================================
 */
{
  const long Q = MODULUS / MULTIPLIER;
  const long R = MODULUS % MULTIPLIER;
        long t;

  t = MULTIPLIER * (seed[stream] % Q) - R * (seed[stream] / Q);
  if (t > 0)
    seed[stream] = t;
  else
    seed[stream] = t + MODULUS;
  return ((double) seed[stream] / MODULUS);
}


  void PutSeed(long x)
/* ==================================================================
 * Use this function to set the state of the current random number
 * generator stream according to the following conventions:
 *    if x > 0 then x is the state (unless too large)
 *    if x < 0 then the state is obtained from the system clock
 *    if x = 0 then the state is to be supplied interactively
 * ==================================================================
 */
{
  char ok = 0;

  if (x > 0)
    x = x % MODULUS;                     /* correct if x is too large  */
  if (x < 0)
    x = ((unsigned long) time((time_t *) NULL)) % MODULUS;
  if (x == 0)
    while (!ok) {
      printf("\nEnter a positive integer seed (9 digits or less) >> ");
      scanf("%ld", &x);
      ok = (0 < x) && (x < MODULUS);
      if (!ok)
        printf("\nInput out of range ... try again\n");
    }
  seed[stream] = x;
}
```

```
   void SelectStream(int index)
/* ==================================================================
 * Use this function to set the current random number generator
 * stream -- that stream from which the next random number will come.
 * ==================================================================
 */
{
  stream = ((unsigned int) index) % STREAMS;
  if ((initialized == 0) && (stream != 0))    /* protect against       */
    PlantSeeds(DEFAULT);                       /* un-initialized streams */
}


   long Equilikely(long a, long b)
/* ==================================================================
 * Returns an equilikely distributed integer between a and b inclusive.
 * NOTE: use a < b
 * ==================================================================
 */
{
  return (a + (long) ((b - a + 1) * Random()));
}


   long Geometric(double p)
/* =========================================================
 * Returns a geometric distributed non-negative integer.
 * NOTE: use 0.0 < p < 1.0
 * =========================================================
 */
{
  return ((long) (log(1.0 - Random()) / log(p)));
}


   long Bernoulli(double p)
/* ==========================================================
 * Returns 1 with probability p or 0 with probability 1 - p.
 * NOTE: use 0.0 < p < 1.0
 * ==========================================================
 */
{
  return ((Random() < (1.0 - p)) ? 0 : 1);
}
```

```
      double Uniform(double a, double b)
/* ================================================================
 * Returns a uniformly distributed real number between a and b.
 * NOTE: use a < b
 * ================================================================
 */
{
  return (a + (b - a) * Random());
}
```

Distribution	Mean	Standard Deviation
Equilikely(a,b)	$\dfrac{a+b}{2}$	$\sqrt{\dfrac{(b-a+1)^2-1}{12}}$
Geometric(p)	$\dfrac{p}{1-p}$	$\dfrac{\sqrt{p}}{1-p}$
Bernoulli(p)	p	$\sqrt{p(1-p)}$
Uniform(a,b)	$\dfrac{a+b}{2}$	$\dfrac{b-a}{\sqrt{12}}$

Table A.1: Stochastic Function Descriptions

Bibliography

[1] C. H. BUILDER AND S. C. BANKES. Artificial Societies: A Concept for Basic Research on the Societal Impacts of Information Technology. *RAND Report P-7740*. RAND Corporation, Santa Monica, California, 1991.

[2] FREDERICO CECCONI AND DOMENICO PARISI. Individual versus social survival strategies. In *Journal of Artificial Societies and Social Simulation*, vol. 1, no. 2, 1998.

[3] JIM DORAN, MIKE PALMER, AND PAUL MELLARS. The EOS Project: Modelling Upper Paleolithic Change. In *Simulating Societies: The Computer Simulation of Social Phenomena*, edited by Nigel Gilbert and Jim Doran, pp. 195-222. UCL Press, London, 1994.

[4] JOSHUA M. EPSTEIN AND ROBERT AXTELL. *Growing Artificial Societies: Social Science From the Bottom Up*. Brookings Institution Press, Washington, D.C., 1996.

[5] NIGEL GILBERT. Computer Simulation of Social Processes. In *Social Research Update*, issue 6, 1993.

[6] NIGEL GILBERT. Simulation: An Emergent Perspective. In *Proceedings of the Conference on New Technologies in the Social Sciences*, 1995.

[7] RAINER HEGSELMANN AND ANDREAS FLACHE. Understanding complex social dynamics: A plea for cellular automata based modelling. In *Journal of Artificial Societies and Social Simulation*, vol. 1, no. 3, 1998.

[8] MICHAEL INBAR AND CLARICE S. STOLL. *Simulation and Gaming in Social Science*. Free Press, New York, 1972.

[9] MICHAEL W. MACY. Social order in artificial worlds. In *Journal of Artificial Societies and Social Simulation*, vol. 1, no. 1, 1998.

[10] STEVE PARK AND LAWRENCE LEEMIS. *Discrete-Event Simulation: A First Course*. Unpublished.

[11] DWIGHT W. READ. Kinship based on demographic simulation of societal processes. In *Journal of Artificial Societies and Social Simulation*, vol. 1, no. 1, 1998.

[12] GÉRARD WEISBUCH AND GUILLEMETTE DUCHATEAU-NGUYEN. Societies, cultures and fisheries from a modeling perspective. In *Journal of Artificial Societies and Social Simulation*, vol. 1, no. 2, 1998.

[13] JEFFREY R. YOUNG. Using Computer Models to Study the Complexities of Human Society. In *The Chronicle of Higher Education*, 1998.

VITA

Rachel Inez Moore was born in Pittsburgh, Pennsylvania, November 15, 1976. She graduated from Mary Schenley High School in that city, June 1994. Rachel earned her Bachelor of Science degree from the College of William and Mary in 1997. She is currently a Master of Science candidate at the College of William and Mary, with a concentration in Computer Science. All course requirements have been completed. Remaining work is the thesis — Artificial Societies : A Computational Approach to Studying Conflict.