SUBSISTENCE ALGORITHMS: WHY DO SOME SOCIETIES THRIVE WHILE OTHERS FADE?

6.0 Introduction

One of the largest topics in archaeological modeling is human subsistence and the myriad ways in which human groups adapt to different environments. Indeed, some of the defining features of *H. sapiens* are related to the flexibility in subsistence strategies, such as the ability to develop complex cultural and social solutions to acquire and manipulate what we consume (e.g., using fire, domestication of animals, etc.), to plan for the future, and to provision other members of one's community. While nonhuman animals do some of this, too—ants farm fungus, corvids store nuts for the future, nonhuman primates provision their young even past infancy—human societies do this at unprecedented scale, do it regularly, and have done it throughout history. Using an agent-based modeling approach, we can examine the relationship between people and their environments, which subsistence strategies were feasible in certain environments, how different food sharing strategies could develop over time, how subsistence and exchange interrelate, and how these can lead to the patterns we detect in the archaeological record.

OVERVIEW

▷ Algorithm zoo: subsistence and resilience
▷ Consumption, subsistence, and resilience strategies
▷ Foraging algorithms
▷ Population dynamics, evolutionary dynamics, and fission–fusion algorithms
▷ Tragedy of the commons
▷ Game theory
▷ Parameterization and model's input data

Most subsistence models are built up from the patch level. While much of the prior sections have focused on agent–agent interactions, subsistence questions engage patch–agent interactions (and even patch–agent–agent or patch–agent–patch interactions). This chapter is divided into the core aspects of subsistence models:

- general dynamics of resource acquisition;
- resilience and adaptive strategies;
- dynamics in foraging behavior;
- population growth and fission–fusion dynamics;
- fitness and evolutionary dynamics; and

- the *Tragedy of the Commons* model and fundamentals of game theory.

Although there isn't one canonical textbook for ABM in subsistence studies, Wilensky and Rand (2015) is a good place to start.

In this chapter, as in previous ones, we show how to code simple fundamental algorithms in multiple ways, working toward more complex models of subsistence. Instead of coding one particular model from beginning to end, we provide relevant sections of published simulations that include subsistence. We begin by covering the *Wolf–Sheep Predation* model in the NetLogo library. This model forms the basis for several other agent-based models, such as *AmphorABM*, a model of viticulture in the Pre-Roman Gaul (ch. 9), and *Ger Grouper*, a model of Mongolian pastoralism, which we also cover here. We introduce the *MedLanD* models to show how patch degradation can be incorporated into subsistence models. We also cover fission–fusion dynamics from Crema (2013) to show how population dynamics feed off models of subsistence. These fission–fusion dynamics are further explored in several other models, such as the *Cardial Neolithic* model. Finally, we discuss two important theoretical frameworks that are often considered when modeling subsistence: evolutionary dynamics, such as fitness-based evolution and social dynamics, including the tragedy of the commons and the fundamentals of game theory. You can find the full model bibliography and links to original code at the end of the chapter. Working versions of all models are also available in the ABMA Code Repo.[1]

Recall that ABMs are particularly useful for computational problems where the agents and environments are heterogeneous. This makes it a great tool for exploring evolutionary processes.

We follow these with a discussion on how to parameterize your model, that is, where to find values and their ranges that can be used in your simulations. Parameterization is one of the most critical aspects of model development, as it will impact how your model runs and how you validate it against the archaeological record.

This chapter covers quite a lot of ground, but in the end you will have several subsistence models in your repertoire, will understand how and where to build up model complexity in light of data availability, and will begin to examine the robustness of conclusions that emerge from a simulation.

[1]You can find all code written in this chapter in the ABMA Code Repo: https://github.com/SantaFeInstitute/ABMA/tree/master/ch6

6.1 Modeling Resource Acquisition

To model subsistence we must return to economic theory. A basic tenet of economics is that most material resources are limited. In chapter 4, agents targeted cells with a resource they needed, but we did not contend with the depletion of those resources except to mark a cell as "used." In this section, we will begin to look at how the simple fact of scarcity can impact human behavior. For this, patches will have variables representing the presence/absence or amount of a resource, as well as a way to regenerate, while agents using a patch will need procedures to consume the resources on those patches and from their own stores.

Model:
Wolf–Sheep Predation by Wilensky, based on the *Lotka–Volterra* model

ABMA Code Repo:
ch6_wolf_sheep

The fundamental algorithms that underlie most environmental productivity models can be found in Wilensky's 1997 model of *Wolf–Sheep Predation*. Wilensky's model is, itself, an agent-based implementation of the *Lotka–Volterra* model, which is based on a differential equation. This model illustrates a foundational principle of ecology; namely, that predators and prey oscillate in abundance over time as they engage in trophic interactions. In Wilensky's implementation, sheep subsist on available grass, while wolves eat available sheep. The feedback between them causes fluctuations of resources (sheep for wolves, grass for sheep) that drive sine wave-like oscillation patterns in their populations.

Choose the `sheep-wolves-grass` model version from a drop-down list in the INTERFACE.

This model can be accessed through the NetLogo Models Library. If you open the code and look at the `setup` procedure, you will immediately see regeneration times set for grass patches, and that not all patches are productive. Green patches start with the maximum regrowth time, and brown patches have their own grass regrowth clock, which is initialized randomly to avoid all brown patches turning green at the same time (fig. A.4).

Next, each entity consumes resources. As the grass grows, sheep eat grass and are, in turn, eaten by wolves. We will start with the grass:

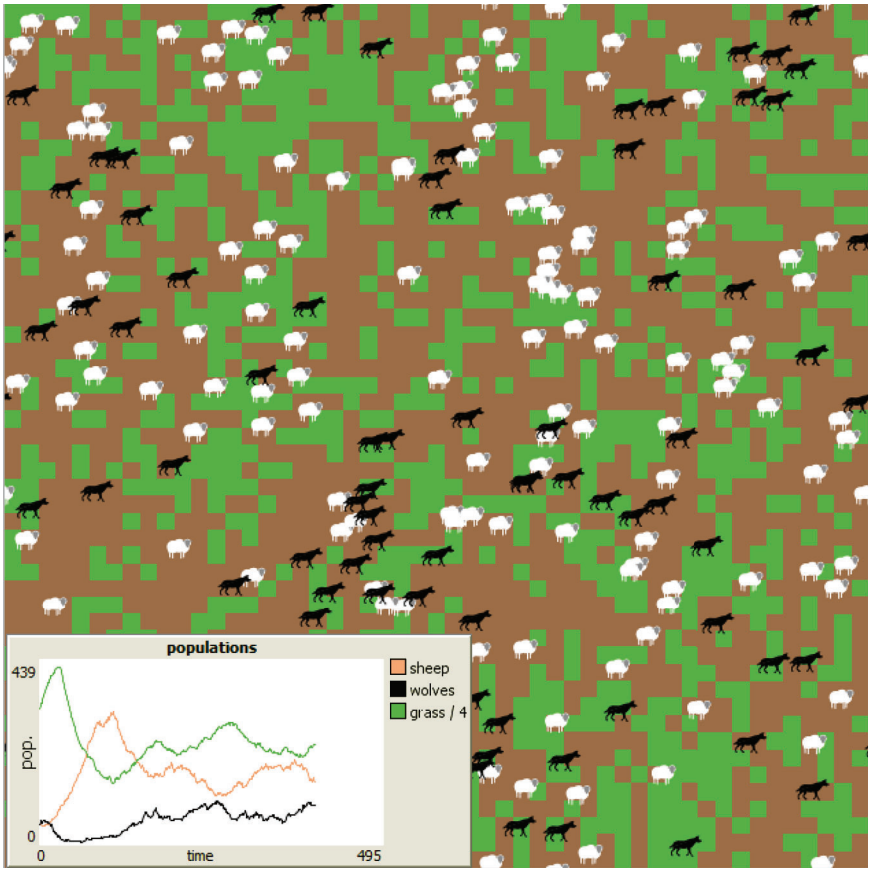

Figure 6.0. Screenshot of the *Wolf–Sheep Predation* model. The population sizes of wolves, sheep, and grass vary over time through linked consumption and regrowth/reproduction cycles. These cycles are known as *Lotka–Volterra* dynamics.

CODE BLOCK 6.0

```
to grow-grass
  if pcolor = brown [
    ifelse countdown <= 0 [
      set pcolor green
      set countdown grass-regrowth-time
    ]
    [ set countdown countdown - 1 ]
  ]
end
```

The built-in variables `color` and `pcolor` are often used to track the current state of the turtle or the patch. They also help to visualize the dynamics of the model.

When a patch becomes depleted (brown), it begins counting down ticks from the number defined on the `grass-regrowth-time` slider. When it reaches zero, the patch regrows and resets the countdown. In this model, the grass resource is binary—green or brown—and once eaten it is unavailable until the counter has reached zero and the grass turns green again.

In contrast, sheep eat the grass in a continuous manner.

CODE BLOCK 6.1

```
to eat-grass
  if pcolor = green [
    set pcolor brown
    set energy energy + sheep-gain-from-food
  ]
end
```

The sheep update their `energy` with the value from the `sheep-gain-from-food` slider. Here, the patch color is used to identify its state and to trigger consumption. If we didn't want to use color, another variable could replace it, for example, a binary `if grass? true`. We could also repurpose the counter since it is reset to equal `grass-regrowth-time` when it is productive (green): `if countdown = grass-regrowth-time [...]` (see code block 6.0).

TIP
NetLogo colors can be referred to by name (e.g., cyan, green, brown) or by number. Check COLOR SWATCHES in the TOOLS menu.

An alternative implementation would be to have a continuous patch variable for `grass-amount`, similar to the one we used in our replication of the *Artificial Anasazi* model in chapter 3. We would initialize patches with up to a `max-grass` maximum specified using a slider in the INTERFACE. Instead of waiting for the patch to fully recover, the grass could regrow by an incremental amount with each time step. You could also code the sheep to consume a specified amount of grass that would be subtracted from `grass-amount`.

CODE BLOCK 6.2

```
patches-own [ grass-amount ]
to setup
  ...
  ask patches [
    set grass-amount random max-grass
  ]
  ...
end

to grow-grass-cont
  ifelse grass-amount < max-grass - grass-amount [
    set grass-amount grass-amount + regrowth-amount
  ]
  [set grass-amount max-grass]
end

to eat-grass-cont
  if pcolor = green [
    ask patch-here [ set grass-amount grass-amount -
    sheep-gain-from-food ]
    set energy energy + sheep-gain-from-food
  ]
end
```

The decision of whether to treat patch resources as binary or continuous can be an important one. A binary variable is simpler in many ways, but may not be sufficient if the spatial or temporal heterogeneity of the resource base is an important factor. Equally, if different agents consume different amounts of resources, then the continuous version is more appropriate. Another consideration is whether data exists to inform these parameter values: we can get estimates of kilocalories obtained by sheep and wolves, but when modeling Neanderthals and mammoths these kind of numbers will be more difficult to estimate.

TIP
Agents who die are no longer available, so if you want to record any of their variable values, write them to a list before issuing the `die` command.

The sheep and wolves lose energy at each time step as they expend energy through metabolism and movement, and if that value is at or below zero, they die.

CODE BLOCK 6.3

```
to death
  if energy < 0 [ die ]
end
```

In some models, when energy drops to "dangerous" levels, it triggers a set of new behavioral rules aimed at survival.

A balance of consumption and regrowth needs to be met to create a stable equilibrium among all three populations such that they do not go extinct. To strike this balance is surprisingly difficult. For example, counterintuitively, if the `grass-regrowth-time` is very quick (e.g., 6 or less), the wolves will go extinct. This is because the abundance of grass causes sheep to become numerous, which in turn leads to soaring wolf populations. Wolves hunt the sheep to extinction, and their own demise follows closely. With fast grass regrowth, the boom-and-bust cycle of sheep numbers is so dramatic that the wolves struggle to recover.

Use the sliders to investigate under what conditions the wolves disappear. This can be triggered by different combinations of parameters. Look back to chapter 4 for equifinality.

In fact, in most agent-based models of human subsistence, the complete extinction of the population is very common; it is quite difficult to find a stable equilibrium. Thus, a primary goal of these kind of models is establishing the range of parameter values that results in a stable ecosystem, with humans and resources both thriving.

While humans and wolves are obviously quite different, the base model from *Wolf–Sheep Predation* forms the backbone for several models on human subsistence and the resilience of early farming populations. The particularities of these agent–patch interactions make ABMs particularly suited to working through the complex dynamics of human–environmental systems.

6.2 Population Resilience in the Face of Environmental Perturbation

One of the biggest questions for understanding human populations is about how they survive in years of low productivity. **Resilience** is the capacity of a system, such as an ecosystem or a social system, to quickly recover from hardship and external fluctuations. In models of subsistence, we see populations of agents increase or decrease according to local productivity. Sometimes populations are able to recover from productivity downturns, such as a drought; in other cases, households die or migrate elsewhere.

resilience: the capacity of a system to recover from a perturbation.

This is one of the biggest assets of agent-based modeling: while we may know the endpoint of an archaeological society, a model can show us the process that got the society to that place.

Agent-based models are a great tool to study the resilience of a past population, as they enable us to look at the feedback between the landscape productivity and the individuals who live there. By examining how agents respond to shifts in patch productivity, when they are able to survive, when external factors or soil depletion make survival difficult, and how populations can bounce back from such perturbations, we can better understand properties of resilient societies.

The perturbations that have an impact on the group subsistence are usually related to spatial and temporal variability. **Spatial variability** usually relates to the clustering of resources, or external factors such as substrates determining soil quality, which differ across the modeled area. **Temporal variability** may be related to climate oscillations, seasonal fluctuations, or environmental degradation (e.g., soil depletion). The spatial variability can be incorporated through a patch variable, either from imported data (e.g., a GIS layer, see ch. 7) or coded manually (e.g., using clustering algorithms, see ch. 3), while temporal variability may come from variables updating themselves every specific number of `ticks` or from data, such as climate curves. Here, we will look at the different aspects of spatiotemporal variability.

spatial and temporal variability: heterogeneous distribution (in time or space or both) of an attribute, such as soil type or average rainfall.

There are many ways to model climate oscillations, the simplest of which would be to modify the rate of regrowth of the resources. A simple multiplier `climate-state` with values between 0 and 2 can be used in a modified `grow-grass-cont` procedure. The value of the modifier can be taken from, for example, a random normal distribution at the start of `go` where the standard deviation denotes the variability of the climate or is taken directly from data. For example, climatic curves such as temperature or precipitation data could be imported from a CSV into a long list of values that you could iterate through with each tick.

If the value of the multiplier falls below 1, then the amount of resource is reduced as you multiply by a fraction (e.g., 0.7); but when it goes above 1, then a bumper crop happens (multiplication by, say, 1.4).

CODE BLOCK 6.4

```
  set climate-state random-normal 1 climate-variability
  ...
to grow-grass-climate
  let regrowth-climate regrowth-amount * climate-state
```

CODE BLOCK 6.4 (cont.)

```
  ifelse grass-amount < max-grass - regrowth-climate [
    set grass-amount grass-amount + regrowth-climate
  ]
  [set grass-amount max-grass]
end
```

Many of the more empirically derived ways to think about human subsistence, such as the dynamics of shifting weather patterns or the impact of substrates on growth rates of certain plants, were incorporated into the farming dynamics of the *Village Ecodynamics Project* (Kohler, Bocinsky, et al. 2012). This agent-based model simulated the farming system in the arid area of southwestern Colorado. The area was dry farmed; Puebloan farmers relied on rain to be able to grow their crops and rarely, if ever, watered their fields. The researchers compiled retrodicted (or hindcast) data on precipitation and heat patterns to model realistic farming production. This kind of data is not always available, but if we wanted to do something similar in a model's patch data, we might use a probability (e.g., `ifelse random-float 1 < rain-probability`) to determine whether an area would receive rain during a given time period.

Model:
Village Ecodynamics
by Kohler

Note that depending on where this is implemented, the rain could vary per patch or rain everywhere at once.

In many environments, there is a marked difference between seasons. For example, in societies engaging in transhumance, households move between summer and winter pastures, enabling them to provide a full food supply throughout the year (see the movement patterns in the *Ger Grouper* model discussed in ch. 4). To do this we use tick-based cycles:

CODE BLOCK 6.5

```
to cycle-productivity
  if remainder ticks 182 = 0 [set gain-from-food
  summer-gain-from-food]
  if remainder ticks 364 = 0 [set gain-from-food
  winter-gain-from-food]
end
```

`remainder` divides the two numbers, in this case the number of ticks and 182 (signifying days), and returns the remainder. If the remainder is 0, then ticks is a multiple of 182 and the code within the `if` command block is

run to switch to the summer value for per-patch food returns. The second line switches it back at the end of the year. Cycling procedures in the `go` code offer a flexible and dynamic way to adjust patch productivity over time. Wren et al. (2020) use two tick-based cycles, one a 14-day tidal cycle to represent coastal shellfish productivity and a second cycle of four seasons to represent caloric returns from edible plants.

Finally, modeling spatial differences in environmental conditions (for example, different crop yields on different substrates or decreasing yields due to soil degradation) can be done in a similar fashion to climate modeling—using simple multipliers. We can manipulate the harvest algorithm so that `harvest-amount` is multiplied by a variable denoting the soil quality specific to each patch.

CODE BLOCK 6.6

```
to harvest
  set harvest-amount harvest-amount * soil-quality
  ...
end
```

The `soil-quality` can be dynamically readjusted to mirror the loss of nutrients in consecutive farming seasons, or we can use an imported map with the soils of the region. In the *Village Ecodynamics Project* (Kohler and Varien 2012), agents examine the underlying productivity of their home patch and compare it against the productivity of each patch within a Moore neighborhood. If their patch is not as productive as neighboring patches, the agent moves. With productivity changing over time due to exploitation, drought/moist years, and warm/cold years, among other drivers, this causes agents to be reactive to environmental pressures, forcing them to move across the landscape. We can model this combined temporal-spatial variability by importing different seasonal productivity landscapes or by having separate values for each season.

Soil data is widely available through various databanks. See chapter 7 for methods on importing different types of geospatial data.

Model:
AgModel, version 0.3
by Ullah

In a quite different model related to the transition from foraging to agriculture, Barton and Ullah (2016) included not only regrowth of plant resources but also evolution from a wild morphotype toward a domesticated type of plant species. Patches that had been harvested by foragers would regrow slightly larger, easier to harvest, and at a higher density per patch due to the effects of human selection. This example, and the majority of those

above, are about adjusting the characteristics of patches due to either an external pressure or human activity on a patch. However, in *AgModel* there is an additional patch–patch interaction. Namely, their patches diffuse their "genetic material" (i.e., pollen) outward to neighboring patches, thus creating a system whereby a critical threshold of the domesticated morphotype must be met before it can begin to overtake the wild type even in the absence of continued human selection (Ullah 2015). Thus, they were able to look at the feedback loop between the long-term impacts of human activity on the environment and how that in turn increases the resilience of the society by improving their farming yields.

Patches are thus given "agency" in that they carry out actions within the system rather than just being passive entities.

6.3 Putting Energy to Use: Ways to Increase Resilience

The importance of storing surplus energy for the development of all kinds of human societies is an important topic in resilience studies (Testart et al. 1982) because that surplus energy allows people to act in new ways. Beyond the consumption and regrowth algorithms, models of agricultural subsistence usually involve other sets of actions to put that surplus energy to work, often in ways that invest some of that energy in the future: planting seeds, raising stock, or moving to better land. For example, storing grain enables agents to trade when they have a surplus or to increase their resilience by maintaining a buffer in case of adverse events. Most sedentary or semisedentary communities use storage to save some of their resources for the off-season or for lean years. In those cases, one needs to establish the limits to storage so that a modeled Bronze Age village does not end up with silos and silos of grain. Stored food may have a `decay-rate`, representing the rate of rot or pest damage.

Thus, another key aspect of modeling agricultural subsistence is conceptualizing energy expenditure: be it for regular farming, opening new fields, or for procreation. In each of these activities there is a trade-off between short-term investment and long-term gain and the associated risks related to unpredictable environmental factors (e.g., crop failure).

Crabtree (2016) used a patch productivity model to examine how subsistence agriculture and luxury agriculture interacted in southern France. In the *AmphorABM* model, there are two types of farmers: Gaulish subsistence farmers (wheat) and Etruscan/Greek luxury farmers (wine). Only wine can

Model:
AmphorABM by Crabtree

ABMA Code Repo:
ch6_amphorABM

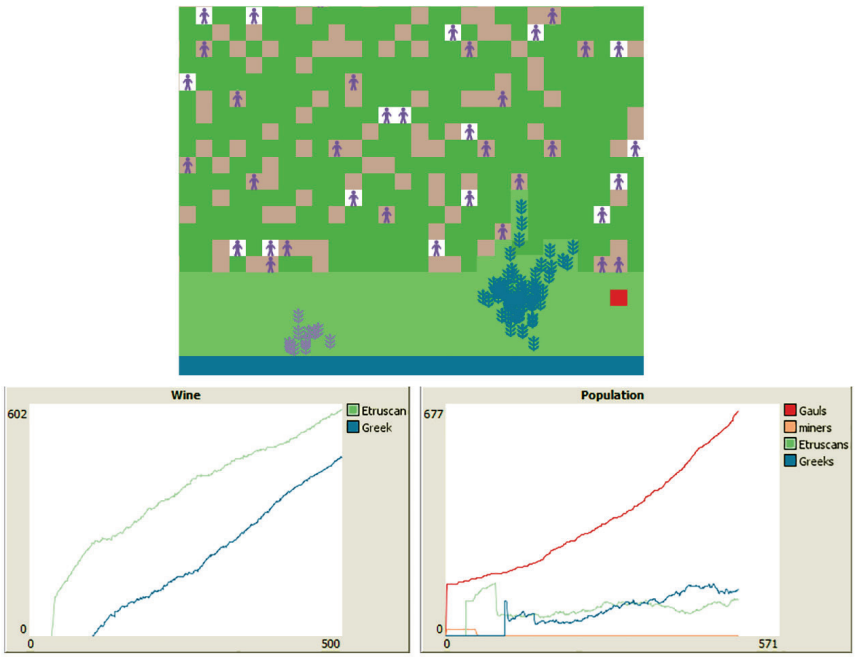

Figure 6.1. Screenshots of the *AmphorABM* model. Greek and Etruscan farmers compete for land for their vineyards in the littoral zone (vine icons), while Gaulish wheat farmers (person icons) occupy the interior and trade for their preferred wine. The two graphs show the evolution of the different groups and their agricultural outputs.

be grown along the littoral, but wheat can be grown on any farming patch (fig. 6.1). The following code shows the baseline algorithms for modeling how fields are planted and harvested, and how grain is eaten and stored.

CODE BLOCK 6.7

```
to plant
  if pcolor != brown [
    set pcolor cyan
    set energy energy - planting-calories
    if EtruscanWine > 1 [ set energy energy + 1 ]
  ]
end
```

```
to harvest
  if pcolor = cyan [
    set energy energy + harvest-amount
    set energy energy - harvest-calories
    if metal >= 1 [ set energy energy + 1 ]
  ]
end

to eat-grain
  set energy energy - 1
end

to store-grain
  set storage storage + energy
  set energy 0
end
```

CODE BLOCK 6.7 (cont.)

TIP
When testing code, follow each key variable by printing its value in every function to make sure the accounting is correct (e.g., grain does not go into negative values, etc.). INTERFACE plots may also help.

At each tick, Gaulish agents choose a fallow field and plant it with grain. Ethnographically, people often hosted beer parties to help with planting costs (e.g., McAllister 2006), so if the agents have wine available, they get an energy boost to reduce the net energy loss from planting. Farmers gain a certain amount of energy from harvested fields. For harvesting, having metal tools helps increase efficiency; thus, if metal is available, agents retain a certain amount of energy (Briggs 2003). Finally, they consume some of their grain, and then store whatever is left (those procedures are in the order listed above in *AmphorABM*'s `go` code).

This model uses colors to differentiate patches and patch states. This is handy when there are multiple agent types and makes it easier to account for all of the changes in foodstuffs/energy/time variables.

It is important to note that humans are quite particular in that their fate is not always linearly correlated to environmental shifts. We know from history that there were periods during which even the optimal levels of productivity could not maintain human populations. In those cases, we know that other systems were at play to ensure that group's survival, such as kin and nonkin networks or large-scale trade. In chapter 8, we explore some of the ways in which societal networks can make human groups resilient.

Equally important is the agent's ability to assess the optimal strategy when deciding where to use their energy, for example, during the selection

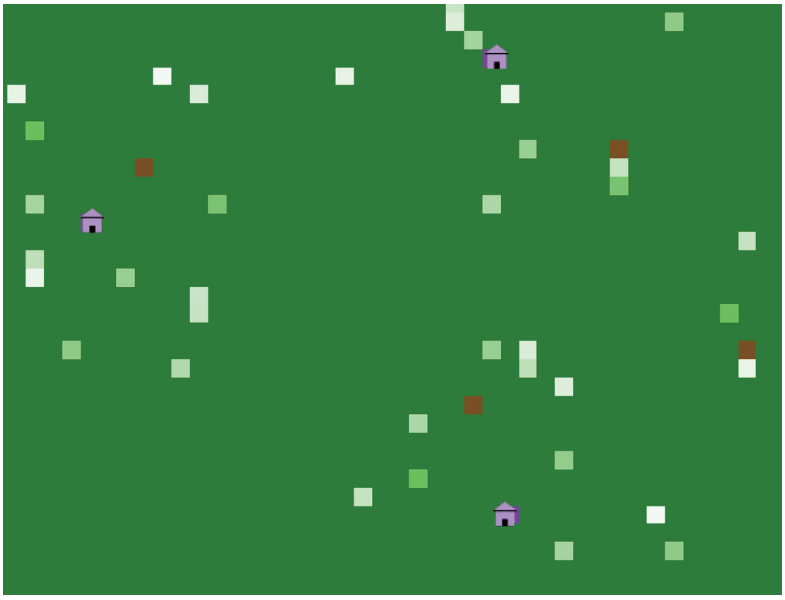

Figure 6.2. A screenshot of the *MedLanD* model. Farms cultivate patches as needed, while leaving previously used patches to regenerate in a swidden rotation system. Farms undergo fission if a high energy threshold is reached.

of a new patch to plant. To understand this, we will switch to a different agent-based model incorporating harvesting and soil fertility. In a study of Mediterranean landscape dynamics and Neolithic farming (*MedLanD*), Barton, Ullah, and Bergin (2010) developed a model of farming household agents. In their model, agents assess how many plots of land they need based on their household size and the previous year's yields. They also assess the soil quality and relative location of each patch to make choices about which plots will likely have the greatest yield. The selection algorithm includes the current soil fertility, which degrades with each year of cultivation but replenishes each year that the land is left fallow. The *MedLanD* ABM also models topsoil erosion and other spatial processes as a result of farming, with greater rates of erosion if land is not left fallow (fig. 6.2). This is done through a tight coupling to the geographic information system GRASS-GIS of Robinson et al. (2018).

NetLogo is often used as a "simple" version of a model to experiment on while the simulation is developed in a more computationally powerful framework.

The main *MedLanD* model was built in the Java-based DEVS-Suite (Sarjoughian and Zeigler 1998). However, Barton (2014) developed a closely related, though more abstract, NetLogo model. In Barton's NetLogo version, called *Swidden Farming*, `pcolor` is again used to demarcate patch

states with shades of green and brown for uncleared forest, active, or fallow fields, and purple and magenta for abandoned or dead farmsteads.[2] The `choose-land` procedure uses a combination of patch and global variables to rank and select new patches to cultivate:

Model:
Swidden Farming by Barton et al.

CODE BLOCK 6.8

```
to choose-land
  let hh_num 0
  ask households [
    set hh_num who
    ifelse any? other (patches in-radius
    swidden_radius) with [fertility > 0 and (owner =
    hh_num or owner = -1)]
      [
      ask one-of ((((patches in-radius swidden_radius)
      with [owner = hh_num or owner = -1])
        with-max [ (fertility * harvest * init_energy /
        100) - (farm_cost * init_energy / 100) -
        veg_clear_cost - ((distancexy pxcor pycor) /
        5) ]))
        [
        set xval pxcor
        set yval pycor
        ]
      farm xval yval
      ]
      [set energy energy - (0.1 * init_energy)]
    ]
end
```

Here, the agent `households` assess whether there are any available and fertile patches to move to in their search radius. The criteria assessed when choosing the patch with the highest potential yield include `fertility`, the patch's energy capacity (`init_energy`), the proportion of the patch that can be harvested (`harvest`) minus all the costs involved: costs of farming (`farm_cost`), preparing the land for farming

Some of these variables are parameters whose values must be derived from agricultural, anthropological, and historical data.

[2]For a link to the code, refer to the Model Zoo at the end of this chapter.

(`veg_clear_cost`), and the costs of traveling to it (`distancexy`). The `household` chooses the patch with the highest net energy yield.

Model: *Piaroa Swidden Farming* by Riris

Riris (2018) built on the concepts of Barton (2014) to develop his own model of swidden agriculture in Venezuela. He investigated mobility ranges and their effects on the pattern and extent of forest regeneration, effectively demonstrating that a high-mobility pattern of swidden farming could result in an Amazonian forest that was indistinguishable from "untouched" forest to observers on the ground. This group of three subsistence models shows that when the internal algorithms of an agent-based model are well conceptualized and described, it is possible to switch not just between coding languages (e.g., Java to NetLogo) but also back and forth between highly empirical models like *MedLanD* and *Piaroa Swidden* and more abstract models like Barton's *Swidden Farming* model.

Chapter 9 describes how to properly document your code for use by others.

6.4 Foraging Models

While so far we have been considering only farming models, many of the same algorithms at least loosely apply to foraging societies as well. Foraging agents can also select suitable habitat based on patch variables, move location, harvest resources, and expend energy or time on these tasks. There is also a rich literature on human foraging under the umbrellas of **optimal foraging theory** (Stephens and Krebs 1986) and human behavioral ecology (Winterhalder and Smith 1981). The many papers falling under these headings tend to be quantitative and numerical model-driven, making them an ideal source to draw upon for agent-based model design of human foragers (for a review, see Hawkes, O'Connell, and Jones 2018). This section will use a couple of abstract models by Barton to examine the two most commonly applied optimal foraging theory (OFT) models in archaeology: the *Patch-Choice* (Barton 2013) and *Prey-Choice* (Barton 2015) models.

optimal foraging theory: a model of resource acquisition behavior that maximises the net benefit (i.e., after costs have been accounted for).

PATCH-CHOICE MODEL

Model: *Patch-Choice* by Barton

ABMA Code Repo: *ch6_patch_choice*

Imagine you're standing in front of a bush while picking berries. At the start, the bush will be full and your calories-per-hour return will be high, but as the bush becomes more and more picked over, there will be a point when it is more profitable to leave that bush for the next bush. Mathematically, that point of departure should be when the rate of return on the first bush drops below the average return of the environment (also known

as the **marginal value theorem**; Charnov 1976). This, combined with the cost that is the travel time to the next patch, forms the logic of the baseline *Patch-Choice* model. To model this, we need the basics already covered in agent harvest, updating patches to have depleted resources, patch regrowth, and so forth. We also need a way to measure the average rate of caloric return for the agent over the last few time steps and the average return for the environment.

marginal value theorem: a model optimizing an individual's foraging rate by balancing the diminishing rate in their current patch against the travel time to a new patch. In an ABM, agents should leave when the current patch has a lower foraging rate than the average of their neighbors.

We can create a slider for the `resource-density` of the landscape such that when patches are created they have a certain probability of containing food.

CODE BLOCK 6.9

```
to setup_patches
  ask patches [
    set pcolor brown
    if random 100 < resource-density [
      set food food-value
      set pcolor green
    ]
  ]
end
```

Agents need to track their general experience on the landscape so that they know when their personal rate of return drops below the threshold of the `resource-density`. To do so, agents will use a rolling history of their food "encounters" over a certain number of previous time steps (`encounter-list`).

CODE BLOCK 6.10

A list is the best data structure for this. With each time step, `fput` adds each new cell's return to the list and `but-last` drops the oldest value.

```
to forage
  ask foragers [
    rt random 360
    forward 1
    ifelse food > 0 [
      let current-harvest random food
      set encounter-list fput current-harvest
      encounter-list
```

CODE BLOCK 6.10 (cont.)

```
      ask patch-here [
        set food food - current-harvest
        set pcolor pcolor - current-harvest
        if food <= 0 [set pcolor brown]
      ]
    ]
    [
      set encounter-list fput 0 encounter-list
    ]
    set encounter-list but-last encounter-list
    set encounter-rate mean encounter-list
  ]
end

to leave-patch
  ask foragers
    [
      if encounter-rate < (resource-density / 100) [
        rt random 360
        forward 10
        set encounter-list n-values 10 [food-value]
      ]
  ]
end
```

You can see how a mobility pattern similar to *Lévy Flights* (ch. 4) is coded here. OFT is used to determine how long each patch should be used before the agent moves on.

The `go` code calls `forage` first, where agents random-walk single steps and harvest some amount of their encountered patch's `food` amount. The agent updates its `encounter-list` with either the food amount or zero if the patch is empty, also dropping the last value off the list. Last, it measures its average return rate by taking the mean of the total returns. The `leave-patch` procedure is run next, where agents see if their average return has dropped below the `resource-density` of the landscape as a whole; if it has, they make a longer movement to a new part of the map.

In general, the agent's random-walk will slowly over-exploit one area of the resource landscape by recrossing the same patches repeat-

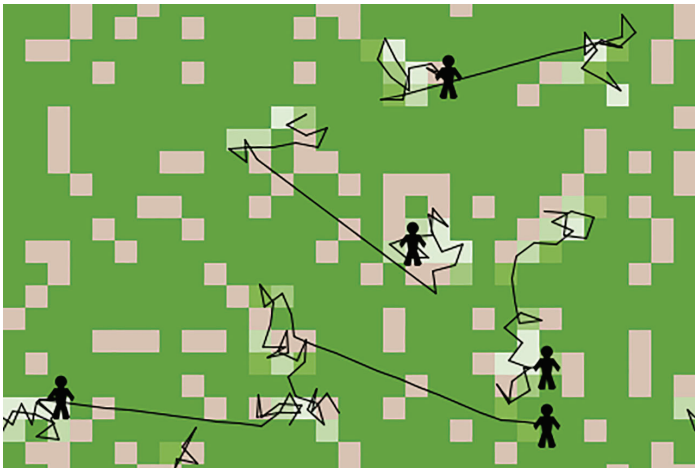

Figure 6.3. Screenshot of *Patch-Choice* model based on optimal foraging theory. Agent paths show their short-step random walk around a localized area. When the average return over the previous 10 time steps drops below the average for the environment, they make a longer step to a new part of the landscape.

edly. When their average return drops too low, moving a larger distance (`forward 10`) to an unexploited part of the landscape should improve their returns (fig. 6.3).

The code above simplifies the original *Patch-Choice* model that incorporated search and processing costs and agent energy levels, and divided the landscape into a grid of nine eco-patches, each of which contained a hundred or so of NetLogo's patches (Barton 2013). Rather than move a longer distance, agents would forage continuously within one eco-patch, and then jump to another when return rates decreased. This highlights an interesting distinction between what OFT and NetLogo mean by "patches." Resolving that distinction is important for your model design and depends on your model's ontology and scales.

Wren et al. (2020) uses a similar patch-choice ABM, called the *Paleoscape*ABM, to examine plant and shellfish foraging. In their model, patches represent different habitat types, each of which has a different return rate. In addition to having agents measure a rolling average return to decide when to move, foragers also look at all patches within a specified radius (`in-radius`) and make choices about where to move based on the anticipated return and the cost of moving to that patch (see sec. 4.3 or the *MedLanD* model's patch selection algorithm above). Choosing among

Model:
PaleoscapeABM by Wren

Note how OFT's simultaneous encounter (i.e., an of-the-moment choice) differs from the long-term dynamics we can model using ABM.

several different options and weighing their expected net returns (energy acquired after energy spent and/or time taken) is known as **simultaneous encounter** (Stephens et al. 1986; Waddington and Holden 1979) and makes a connection point to the next OFT model.

PREY-CHOICE MODEL

diet-breadth model: a model concerned with an individual's acquisition of specific resources within a larger set of resources. Agent actions are determined by the value and the cost associated with acquiring those resources.

The *Prey-Choice* model is used to understand why only certain types of foods end up in the diet, out of a longer list of available foods. The model is often referred to as the *Diet-Breadth* model in archaeology since we are interested in understanding the diversity of flora and fauna in a site's archaeological record (Kelly 2013).

Imagine you are in a landscape with an unknown distribution of lettuce, apple trees, and moose. As you walk along, you find a field of lettuce—should you stop for some? The lettuce is plentiful and easy to harvest, but not very calorically dense. If you stop to forage, you are not going to run into any apple trees or moose. If you do choose to move on, sure, a moose will have a huge caloric return, but only if you encounter one, manage to get close enough, hit it, track it, and process it. The math of this is relatively straightforward if we assume the forager should always try to maximize their net caloric returns. The model is known as the *Diet-Breadth* model because you can rank the species and determine the cutoff point at which certain species will not be worth the effort. Species within the diet-breadth should always be worth trying for when encountered. To model this, we need several more per-species variables than last time, including food values, processing costs, search times (based on density in the landscape), pursuit times, and the probability of a successful kill (especially for the moose). It is only after all the costs and risks involved are subtracted from the food value that we get a good sense of which species the forager should try to acquire (Stephens and Krebs 1986).

Model: *Diet-Breadth* by Barton

We will use a simple model by Barton (2015) as our starting point. Set up a hunter agent and a separate animal `breed` with four species types, each with their own density/abundance, `food-value`, and `processing-cost`. Their ranks, which determine whether the hunter should kill the species if it encounters one, can then be calculated by subtracting the `processing-cost` from `food-value`, sorting the list, and then extracting that species' position.

```
to setup_animals
  ...
  set rank-list (list (food-value1 - processing-cost1)
                      (food-value2 - processing-cost2)
                      (food-value3 - processing-cost3)
                      (food-value4 - processing-cost4))
  set rank-list sort-by > rank-list

  create-animals number1 [
    set species 1
    set food-value food-value1
    set processing-costs processing-cost1
    set rank position (food-value1 - processing-cost1)
    rank-list + 1
  ]
  ...
```

CODE BLOCK 6.11

Again, lists are the most computationally effective way to process all this information.

Again, one of ABM's strengths is enabling agents to differ from one another. Thus, Barton is able to depart slightly from the classic *Diet-Breadth* model by incorporating an agent `energy` level that determines how willing they are to take a lesser-ranked prey (fig. 6.4):

```
to forage
  let prey one-of animals-here
  if prey != nobody  [
    if (energy >= 85 and [rank] of prey = 1) or
       (energy < 85 and energy >= 70 and [rank] of prey
       < 3) or
       (energy < 70 and energy >= 55 and [rank] of prey
       < 4) or
       (energy < 55) [
       set energy energy + [food-value] of prey
    ]
  ]
end
```

CODE BLOCK 6.12

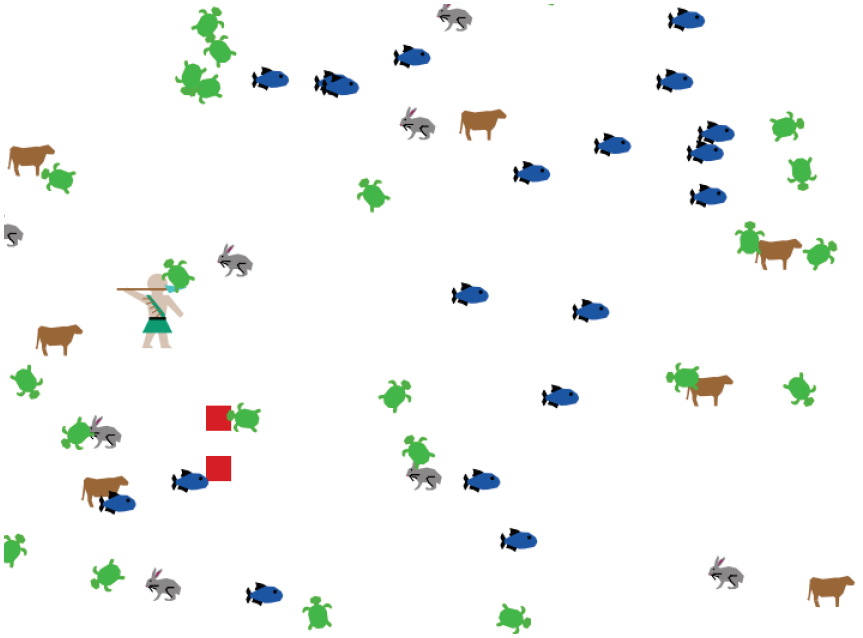

Figure 6.4. Screenshot of the *Diet-Breadth* model. The hunter agent pursues prey species. The likelihood of the hunter attempting to kill any prey encountered varies depending on how much energy the hunter has left.

Model:
Ache Hunting
by Janssen & Hill

The model also plots out a rolling list of recently killed prey and hunter energy levels as the five species randomly walk around the landscape. Greater detail is included in another *Prey-Choice* model simulating foraging practices among the Ache hunters in Paraguay (Janssen and Hill 2014). They follow a similar approach but also incorporate the additional prey parameters of pursuit time and probability of a successful kill, and different per-species encounter rates for different habitat types (e.g., meadow, riparian, high forest, etc.).

See chapter 4 for the *Ache Hunting* model's multilevel mobility algorithm.

6.5 Household-Level Production & Population Dynamics

The last element often included in subsistence models is population dynamics. While we discussed the death of agents when energy levels drop too low, new agents can also be created. Subsistence models often include procedures for household-level fissioning (splitting), or individual reproduction, when energy levels or local population sizes reach a certain threshold. Depending on the consumption and reproductive parameters, the result may either be a relatively stable oscillation of the population, extinction, or run-

away population growth. In chapter 4, we introduced the published model *Ger Grouper* to examine migration scenarios at the population level (Clark and Crabtree 2015). *Ger Grouper* examined how different levels of variability in the environment impact the survival of households. In the model, households fission when energy levels increase above a certain level. Households extract and consume energy from productive patches and share them according to certain probabilities with their relatives (Clark and Crabtree 2015); see fig. 6.5 in this book). Like in the *Wolf–Sheep Predation* model, the tension lies in finding a balance between productivity of patches and reproduction within realistic parameter ranges. If agents reproduce rapidly, they may outstrip their environment; if agent harvest is relatively low while environmental productivity is relatively high, the environment may have no impact on agents at all.

Model:
Ger Grouper by Crabtree

ABMA Code Repo:
ch6_GerGrouper

Recall that the Mongolian word *ger* means yurt-style house.

Although it is possible to model the exact process of marrying and leaving the house and then raising a child (e.g., Kohler, Bocinsky, et al. 2012), in most models it is common to simplify it down to fission–fusion dynamics, such that if a household-agent has enough energy stored it creates a new household, dividing the energy between the parent and the offspring.

TIP
You can set a `label` to show the current energy level of agents by adding `set label round energy` to the `go` code.

CODE BLOCK 6.13

```
to reproduce-gers
  if energy >= 20 [
    if random 100 < gerreproduce [
      set energy (energy / 2)
      hatch 1
    ]
  ]
end
```

Here, an offspring inherits the parent's variables. When an agent is asked to hatch, it passes its current values to the newly created agent, so we divide the parent's energy in two so both end up with half of the original energy.

In some cases, we don't want an agent to inherit values from the parent (e.g., imagine if there was a variable "age").

As new households fission from parent households, you may want to add movement commands to the hatch code (e.g., `hatch 1 [ move-to one-of neighbors ]`) to ensure the offspring ends up on an appropriate patch. However, note what comes next in the `go` code to ensure that new offspring aren't disadvantaged by, for example, having a higher chance of landing on a bad patch. As these agents join the population, we can track

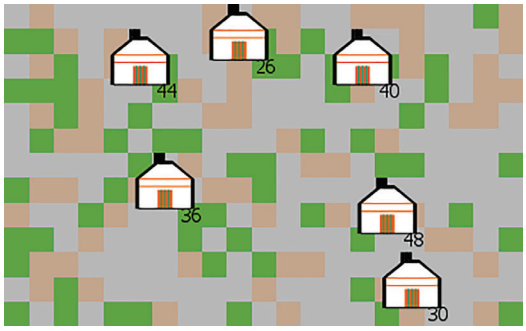

Figure 6.5. Screenshot of the *Ger Grouper* model. *Ger* agents are displayed on a landscape with patches in different states denoted using "pcolor". Agent labels show their current energy level.

how populations grow (or shrink) depending on exogenous dynamics of environmental fluctuations.

Ger Grouper explicitly simulates fissioning of households but leaves the fusion part of the dynamic implicit. As seen before, agents could simply die if their energy got too low. Depending on the scenario, this could literally represent a group starving to death, or we could just make the uncoded assumption that if a household was no longer viable, the remaining stragglers might go join another household. This process can instead be modeled explicitly. Crema (2014) explored fission–fusion dynamics in an abstract but highly scalable model. While other models focused on grasping the population dynamics in particular human populations known from the archaeological record (e.g., Kohler and Varien 2012; Crema 2013), this model identifies simple rules driving human groups to expand or contract.

Model:
Fission–Fusion by Crema

ABMA Code Repo:
ch6_FissionFusion

In Crema's model, agents, who represent households, form groups and respond to local environmental conditions (carrying capacity: K). The number of agents on a patch determines agent fitness (ϕ), which in turn is directly related to K. Fitness of agents drives reproduction (ρ), which leads to fission and fusion events. For agents whose individual fitness is lower than a desired threshold, it is advantageous to leave the group to seek a patch with higher productivity. They can then choose to do one of the following three behaviors:

The symbol K is used for carrying capacity in many numerical models, such as the Fisher–Skellam–KPP wave of advance model discussed in chapter 4.

- `solo`, where agents find an empty patch and forage on their own;
- `fusion`, where they find another group and join them; or

- `merge`, where they find a nearby agent and set out to an empty patch together.

Through these simple rules, Crema was able to demonstrate that some aspects of population spread detected in the archaeological record can be replicated through simple dynamics. While his model was built in R, here we simplify and replicate a small portion of the code to show how the fission–fusion dynamics work.

We use the parameter K to determine the maximum energy a turtle has available from a patch. When the patch's available energy is insufficient to support the number of turtles (i.e., exceeds K), fitness benefits decrease dramatically. While this simplifies Crema's asymptotically declining fitness, it replicates the general fission and fusion dynamics (fig. 6.6).

CODE BLOCK 6.14

```
extensions [ Rnd ]
turtles-own [ turtle_energy age ]
patches-own [ energy ]

to setup
  ca
  ask patches [
    set pcolor white
    set energy K
  ]
  crt 50 [
    set color 15 + (10 * random 12)
    set age 0
  ]
  reset-ticks
end
```

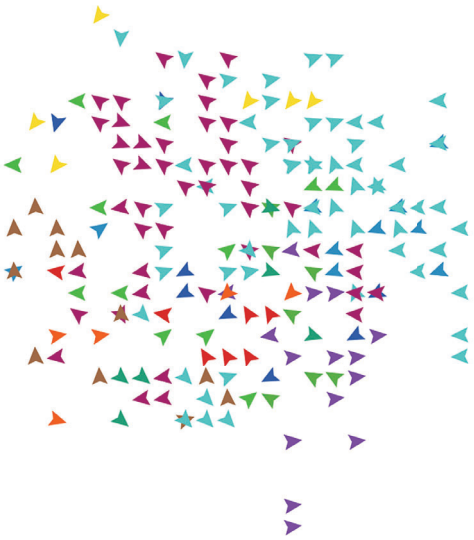

Figure 6.6. Screenshot of the *Fission–Fusion* model. Agents expand outward from their initial central point. Note the overlapping agents, which have undergone fusion or merge procedures.

CODE BLOCK 6.14 (cont.)

```
to go
  ask turtles [
    eat
    fission
    reproduce
    grow-old
  ]
  tick
end
```

Agents in the model consume resources locally (`eat`), sharing the energy available on the patch, K, among all local inhabitants. Finally, agents age and die with a probability proportional to their age.

CODE BLOCK 6.15

```
to eat
  set turtle_energy energy / count turtles-here
end
```

```
to grow-old
  set age age + 1
  if random 100 < age [die]
end
```

CODE BLOCK 6.15 (cont.)

The probability of dying is proportional to the age so no agent can exceed 100 years of life.

If there are too many agents on the patch to fulfill their needs, they move away (fission). Agents have a chance to reproduce if they have enough energy.

```
to reproduce
  if turtle_energy >= 1 [
    if random 100 < reproduction [
      hatch 1 [
        set age 0
        rt random 360
      ]
    ]
  ]
end
```

CODE BLOCK 6.16

We also discuss the weighted choice, a.k.a. roulette wheel algorithm, on page 223.

DON'T FORGET
Add sliders for `K` and `reproduction` to the NetLogo interface (range 0 to 30).

Now that we have coded the environment, the agents, and their population dynamics, we will add the three procedures for fission–fusion. Note that in code block 6.14, we have added the `Rnd` extension since we will be using a weighted random choice to decide among three types of fission.

We will reuse several other code pieces from chapter 4 (e.g., the targeted walk algorithm), so it should feel familiar.

```
to fission
  if turtle_energy < 1 [
    let items [ "fusion" "solo" "merge" ]
    let weights [ 0.5 0.3 0.2 ]
    let pairs (map list items weights)
    let selection first rnd:weighted-one-of-list
    pairs [ [p] -> last p ]
```

CODE BLOCK 6.17

Add a `print pairs` command after `let pairs` if you're not sure how this code works.

CODE BLOCK 6.17 (cont.)

```
    if selection = "fusion" [fusion]
    if selection = "solo" [solo]
    if selection = "merge" [merge]
]
end

to fusion
  let target min-one-of patches with [any?
    turtles-here] in-radius 19 [distance myself]
  if target != nobody [move-to target]
end

to solo
  let target min-one-of patches with [not any?
    turtles-here] in-radius 19 [distance myself]
  if target != nobody [move-to target]
end

to merge
  let friend min-one-of other turtles
    in-radius 2 [distance myself]
  let target min-one of patches with [not any?
    turtles-here] in-radius 10 [distance myself]
  if target != nobody and friend != nobody [
    move-to target
    ask friend [move-to [patch-here] of myself]
  ]
end
```

Note that the probabilities add up to 1. This is not actually necessary, but it helps to make sure the model's logic is sound. In the model repository we implemented these weights as sliders.

This model demonstrates how individual circumstances and choices shape large-scale population patterns.

During the fission events, agents choose from three different outcomes with a probability given by the values `[ 0.5 0.3 0.2 ]`.

Through these three algorithms, coupled with reproduction and death parameters, we can watch the agents expand and contract across the landscape through time. This model can also be used to simulate the changes in

the distribution of populations across a landscape in reaction to changes in patch productivity since it is impacted by K.

POPULATION STABILITY & GROWTH

In the `reproduce` and `grow-old` procedures in our replication of Crema's model, the parameterization of the population dynamics is quite abstract. Really, we are just comparing the ratio between the two values: probabilities of `reproduction` and `death` per time step. If the two are equal, the population will go extinct since available energy would limit reproductive rates. But as `reproduction` increases relative to `death`, the population may survive and expand. In other models, the reproduction and death rates are empirically derived, with values data-mined from ethnographic, historical, and other census records of societies. To give an example of how it works in practice, Wren and Burke (2019) looked through records of total fertility rates—the average number of births during women's lifetimes—in different societies to find a value to use for their reproductive rate. Pennington (2001) reported values between 2.8 and 8.0 births with a median of 4.3, and included a specific example that was ecologically similar to their case study with a value of 4.4. With an estimate of average reproductive years, a little math in `setup`, and taking into account that their chosen time step represented one month, that worked out to a probability of 1.467% chance of an agent reproducing each month:

Model:
LGM Ecodynamics
by Wren and Burke

ABMA Code Repo:
ch6_weightedsurvival

CODE BLOCK 6.18

```
to reproduce
  ask turtles [
    if random-float 1 < birthrate_month [ reproduce ]
  ]
end

to check_death
  ask turtles with [age > 0] [
    set deathrate_month birthrate_month
    if random-float 1 < deathrate_month [die]
  ]
end
```

Wren & Burke based their model on Barton et al.'s *Hominin Ecodynamics* model.

Initially, they also set the death rate to the same value such that the overall population growth rate would average out to be 0%. Note the additional condition that only agents with `age > 0` are at risk of death. In this model's `go` code, reproduction occurs before death, so there are actually more agents (i.e., the agents at the start of that time step plus the newly hatched ones), being subjected to the same probability of death. Without the caveat that newly hatched agents should be excluded, this would result in a net population decline.

From this point, a few different possibilities exist depending on the scope of the model. If you have an *a priori* reason to model a fixed population growth rate, you could just adjust the ratio between the reproductive rate and the death rate such that you have a top-down imposed population growth rate. Using a more bottom-up approach, Wren and Burke (2019) were interested in how the spatially variable ecological conditions might have affected the survival of groups in different parts of their landscape. To experiment with this, they adjusted the probability of death such that the patch `suitability` values, which ranged from 0 to 1, would affect how likely an agent was to die (fig. 6.7). They assumed an inverse linear relationship between probability of death and `suitability` and coded in a linear equation to determine the per-patch probability of death:

CODE BLOCK 6.19

This is a good example of a linear model, here: $y = death\_slope \times x + death\_yintercept$. The values for these variables were solved in setup.

```
to habitatsuitability_check_death
  ask turtles with [age > 0] [
    if random-float 1 < (death_slope * suitability) +
      death_yintercept [die]
  ]
end
```

For example, if this equation was applied to agents on the *SugarScape* hills in ch. 3, the agents at the bottom of the resource hills would be more likely to die, leaving the ones at the top of the hills more likely to have net population growth (fig. 6.7). The fixed probability per patch can also be replaced with a dynamic patch variable and equations to determine rates of reproduction and death, as in Crema (2013).

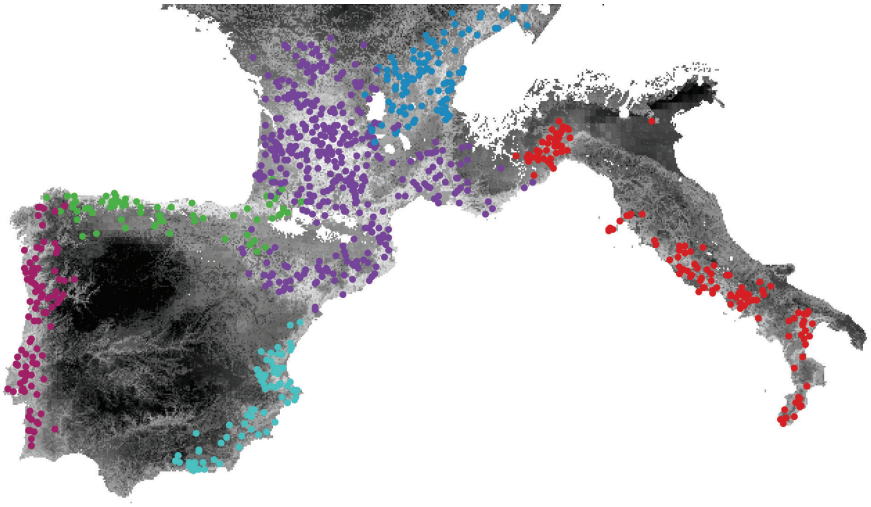

Figure 6.7. Screenshot of the *LGM Ecodynamics* model. Agents on lighter shaded cells (i.e., higher "suitability") have a higher probability of surviving, thereby driving increased net population growth in those regions. The structure of the landscape segments the population over space and drives population dynamics.

AGENT FITNESS & EVOLUTIONARY DYNAMICS

Differential rates of reproduction, based on a measured agent attribute, can lead to interesting dynamics of cultural and/or biological evolution. Agent attributes may be thought of as measure of their **fitnes** , where those with higher fitness have a higher likelihood of reproduction or survivability. How fitness is determined varies widely depending on research questions, but it can either directly or indirectly affect other agent characteristics (fig. 6.8).

fitness: a measure of an agent's "quality," usually relative to other agents, such as likelihood of reproduction or survivability in a biological system, or presence of preferred traits or behaviors that bring an economic or social benefit in a cultural system (see ch. 5).

A simple algorithm makes fitness equal to reproductive probability in a similar but inverse manner to the `check_death` procedure in code block 6.18:

CODE BLOCK 6.20

```
if random-float 1 < fitness [ reproduce ]
```

Another method is to use a weighted choice or **roulette wheel** approach through the `Rnd` extension. This will ask some proportion of the agents to reproduce, but we will make it more likely for agents to be asked to reproduce if their fitness is higher: `ask rnd:weighted-n-of 10 turtles [fitness] [reproduce]` .

roulette wheel: an algorithm that selects among multiple outcomes, each with a different probability. As in casino roulette, where getting a red pocket is more likely than a specific number.

Figure 6.8. Screenshot of the *Roulette Reproduction* model. The roulette wheel provides a weighted probability of reproduction based on a fitness measure. The average fitness of the population increases (y-axis) as a single lineage replaces the initially random population over time (x-axis).

As a simple example of cultural evolution, we will build a quick model around the roulette wheel approach to fitness-based reproduction:

CODE BLOCK 6.21

Model:
Roulette Reproduction
by Wren

ABMA Code Repo:
ch6_roulette_rep

We also used a roulette wheel in the `weighted-random-walk` in chapter 4 where agents were more likely to move to a patch with a higher attribute value.

```
extensions [ Rnd ]
turtles-own [ fitness age ]
to setup
  ca
  crt 100 [
     set fitness random-float 1
     set age 1
     setxy random-xcor random-ycor]
  reset-ticks
end

to go
  ask rnd:weighted-n-of 10 turtles
    [fitness] [reproduce]
  ask n-of 10 turtles with [age > 0] [die]
  ask turtles [
    set age age + 1
    set heading heading + 15
    fd 1
```

CODE BLOCK 6.21 (cont.)

```
  ask turtles [
    set age age + 1
    set heading heading + 15
    fd 1
  ]
  tick
end

to reproduce
  hatch 1 [ set age 0 ]
end
```

Here, we assign fitness as a random number in the setup rather than as a characteristic of the environment. The `weighted-n-of` line asks ten agents to reproduce, but is more likely to ask agents with a higher fitness value. We then kill off an equal number of turtles, though sparing those who just hatched.

Note how fitness determines the probability of reproduction while death is equally likely for everyone. Over time this leads to changes in the average fitness in the population.

Let's examine the effects this has on turtle demographics. In the INTERFACE, add three plots to measure: `count turtles`, `mean [fitness] of turtles`, and `mean [age] of turtles`. Run the model for about 500 ticks and examine the plots and agent population. You should notice first that the population size remains constant; our plot is just a check to make sure the model is doing what we expect it to do. The fitness plot will show that the `fitness` of the population steadily increases (fig. 6.8. The lower-fitness individuals are chosen to reproduce less often by the roulette wheel, yet they die just as often, so their lower fitness value does not get passed on to the next generation with as high a frequency as the more fit turtles. The third plot shows that the average age of agents quickly rises and then stabilizes at age 10. While some agents will be lucky and live to 50 ticks or so, since the death rate is constant and random, it is highly unlikely that an agent will live to be 100.

Line plots of avaerages are easiest here, but fitness and age could also be bar-plot-based histograms if you needed to see the variability at each time step rather than the trends over time.

To summarize, with this simple model we have mechanisms that produce a stable population size, a reasonable age structure in our population, and agent attributes that increase when under active positive selection. If

the fitness attributes were genetically inherited, then this would be a simulation of biological evolution under natural selection; however, if that attribute represented a cultural trait, we could equally be modeling cultural evolution. In that case, we could add in some transmission of cultural traits (see sec. 5.2 in the previous chapter).

The probability of dying is related to the number of times an agent underwent the `check-for-death` procedure. Although at each time step the probability of dying is the same, those agents that lived 11 years had 10 more chances of dying compared to a 1-year-old agent.

Whether genetic or cultural, the `fitness` value was the attribute under direct selection. However, what do you notice about the agents' colors over the course of a run? Since we did not assign a color when we created the turtles in `setup`, they were randomly assigned. We also did not assign a new random color within the `reproduce` procedure, so this means that the hatched agent inherits their parent's color.

We used a similar approach in chapter 1 to show color inheritance in the *Y&B Diffusion Model*.

Color has no direct connection to fitness in this model, but the colors assigned to agents with higher fitness values are more likely to reproduce, leading rapidly to only one or two colors being represented in the agent population. We can think of color as representing any other attribute of a population that "hitchhikes" along by accident. For example, Ackland et al. (2007) published a study looking at cultural traits like language or neutral genetic markers, and how they may be carried along a diffusion wave with an advantageous cultural trait like farming.

Model: *Cultural Hitchhiking* by Ackland et al.

Add a small additional line to your code to introduce small mutations (random increases or decreases) in the value of these attributes. From this simple addition, evolutionary dynamics can emerge from the underlying combination of population dynamics, random inheritance, and selection:

CODE BLOCK 6.22

```
to reproduce
  hatch 1 [
    set age 0
    let mutation_direction one-of [-1 1]
    set color color + (1 * mutation_direction)
  ]
end
```

In the `hatch` code, we first randomly choose whether the mutation will increase or decrease the value, then adjust the color inherited from the parent agent by a small amount (brighter or darker color).

Fission models also usually include mobility algorithms since "child" agents move to a new cell to avoid over-exploiting the parent's patch. A

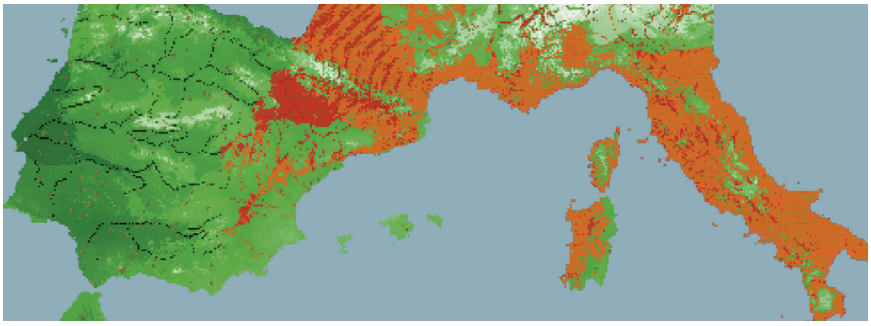

Figure 6.9. A screenshot of the *Cardial Spread* model. Detailed parameters, including birth and death rates, fissioning thresholds, mobility behaviors, and ecological conditions (lighter shades), control the spread of the Cardial Neolithic (darker shades).

good example of a model that integrates subsistence and mobility is the leap-frog model of Cardial Neolithic dispersal discussed in section 4.5 of chapter 4. Bernabeu Aubán et al. (2015) model villages of farmer agents each with their own local `village-population` size variable. The model uses a `while` loop such that each of the `people`, which individually are not agents but just numbers in the `village-population` variable, is subject to a probability of birth and death. If the village population reaches a specified population size, it marks itself as ready to fission, and the child village moves on according to one of several movement algorithms (fig. 6.9):

Model:
Cardial Spread by Bergin

CODE BLOCK 6.23

```
to farmer-subsistence
 let people village-population
 let life-cycled 0
 while [life-cycled < people] [
  if random-float 100 + 1 < farmer-birth-rate [ set
  village-population village-population + 1 ]
  if random-float 100 + 1 < farmer-death-rate [ set
  village-population village-population - 1 ]
  set life-cycled life-cycled + 1
 ]
```

CODE BLOCK 6.23 (cont.)

```
  if village-population > max-farmer-per-patch and
  village-population > farmer-fission-hh-size [
    set time-to-fission? true
  ]
  ...
end
```

As discussed in chapter 4, an early stage of model development involves selecting whether an agent is an individual, a household, or a village; that is, the scale of the model's dynamics. Some subsistence models incorporate multilevel interactions within a single model. Gauthier (2019a) uses three nested models: one for individuals, one for households, and one for settlements. Each has its own dynamics, and the outputs of each level interact with the others. For example, individuals have age-specific fertility rates, which vary depending on the amount of food available within that individual's household. The population of a household affects the food production rate of that household, and if a household's population increases above a certain threshold, one individual may leave to form their own household (i.e., fission). At the settlement level, the model is a cellular automata where neighboring cells (in this case a hexagonal lattice) may be occupied as settlement populations grow beyond their local carrying capacity (Gauthier 2019b).

Model: *Multilevel Population Dynamics* by Gauthier

This R model can be replicated in NetLogo using the LevelSpace extension.

Using symmetrical (triangular or hexagonal) lattice removes many of the distance issues discussed in sec. 4.3.

6.6 Tragedy of the Commons

So far we have focused on the interactions between the environment and agents. However, the interactions among agents in such a system can be equally rich. Let's start with introducing one of the most famous models of group interaction over limited resources: the **tragedy of the commons** (Hardin 1968).

The **tragedy of the commons** is a phenomenon that extends well beyond collective agreements in small-scale agrarian populations.

For centuries, an area of public land known as *the commons* was an emblematic feature of the British countryside. This was land that belonged to everyone. The tragedy of the commons refers to a model of how the use of such common resources can be negotiated. It is a well-known paradox that by optimizing self-interest, individuals can drive the system to a collapse

where it serves no one at all. If each farmer grazes as many cattle in the commons as they can (maximizing their own short-term returns), the result will be a barren patch of land where no grass can regenerate. This paradox occurs because the gain from using the commons goes to an individual farmer, while the cost of the damage from overgrazing is shared among all farmers. How to prevent the inevitable outcome of collapse depends on the group's ability to negotiate social contracts, thus bringing us to the vast topic of modeling human cooperation and the theoretical, mathematical framework behind it, known as game theory.

Model:
Multi-Agent System of the Tragedy Of the Commons (*MASTOC*) by Schindler

Schindler (2012) implemented the classical version of the tragedy of the commons (fig. 6.10. Here we will simplify it even further. Open a new model and code the setup:

ABMA Code Repo:
ch6_tragedy_comm

- Set up a world with two breeds: cows and herders living in the world of green patches.
- Use sliders to set the initial number of each breed (default: 10 cows and 5 herders).
- Create cow variables: `owner` and `forage` and assign one of the herders to each cow's `owner` variable.
- Create herder variables: `now_cows` and `past_cows` and set `now_cows` to each herder's cow `count`.

Use `with [ owner = myself ]` to count herders' cows.

In the `go` procedure, the cows graze, the herders can add cows after they evaluate their stock, and grass regrows. We also include a stop condition so that the model's run ends when tragedy strikes the commons (i.e., the grass is so depleted that the cows die).

It is good practice to work out the structure of key algorithms, like `graze`, in pseudocode. At this point, you should be able to come up with a working solution even if it isn't the same algorithm as our version in code block 6.25.

Before looking at the following code blocks, think about how you could implement the `graze` procedure using the variables you have created and what we have covered so far in this chapter. How will you track the cows' forage and mark patches as eaten? How will you stop cows from trying to graze in depleted patches? How will the patches recover?

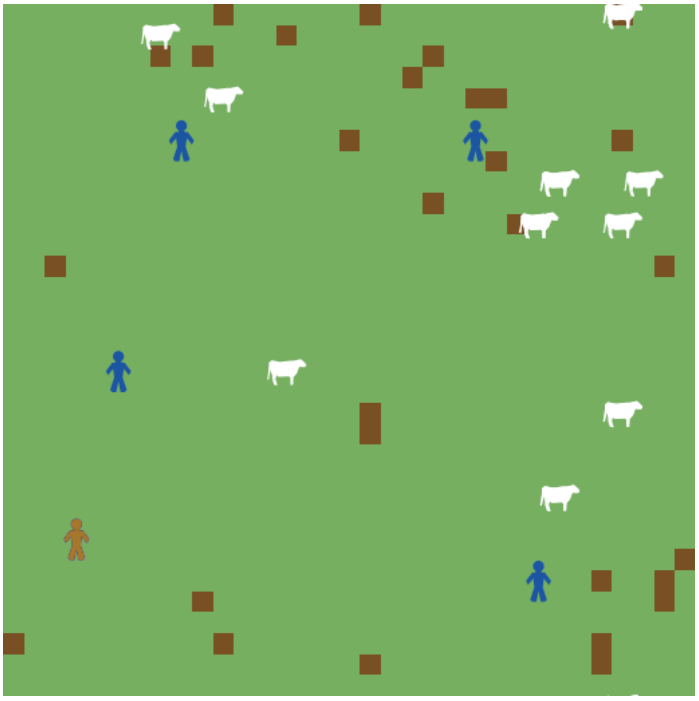

Figure 6.10. Screenshot of the *MASTOC* model. A group of herders and their too-rapidly-increasing cow population. Note that one herder's cows ran out of available grass and died.

CODE BLOCK 6.24

```
to go
  graze
  evaluate-stock
  grass-regrowth
  if not any? cows [stop]
  tick
end
```

Here, each time step represents one day and uses a while loop to have cows repeatedly move to grass patches and forage until they meet their day's food requirement (set by a slider `cow-forage-requirement` with values between 0 and 100) or until the grass runs out. If they cannot meet their energy needs, they die.

CODE BLOCK 6.25

```
to graze
  ask cows [ set forage 0 ]
  while [max [ forage ] of cows < cow-forage-requirement
  and any? patches with [ pcolor = green ]] [
    ask cows with [forage < cow-forage-requirement] [
      if any? patches with [ pcolor = green ] [
        move-to min-one-of patches with [ pcolor =
        green ] [ distance myself ]
        set pcolor black
        set forage forage + 1
      ]
    ]
  ]

  ask cows [
    if forage < cow-forage-requirement [ die ]
  ]
end
```

Note that unlike in some of the previous models, when energy was a continuous variable, patches here only have a single unit of grass and are immediately marked as depleted (through `set pcolor black`) when a cow grazes on them.

Next, we have herders evaluate their stock and add more animals to the pasture if they did not lose too many cows during the previous day. To establish this, they compare the number of cows they have now, `now_cows`, with that in the previous time step, `past_cows`. If some of their cows died in the previous step, that means that there was not enough grass for everyone. The unselfish thing to do would be not to add any more animals. Herders have a variable `selfishness` (set with a slider), which determines how many cows need to die before they become concerned enough not to add any more to the common pasture.

TIP
Don't expect to be able to write the whole model out in one go. Build up the complexity of your procedures bit by bit.

CODE BLOCK 6.26

```
to evaluate-stock
  ask herders [
    set past_cows now_cows
    set now_cows count cows with [owner = myself]
    if now_cows = 0 [set color red]
    if now_cows >= past_cows - selfishness [
      let mycows cows with [owner = myself]
      if any? mycows [
        ask one-of mycows [hatch 1]
      ]
    ]
  ]
end
```

DON'T FORGET

Add to the INTERFACE a `selfishness` slider with values 0–10 and a `grass-regrowth-rate` with values 0–1, increment 0.01.

Finally, the grass regrows at a rate set by a slider. Since the grass is a single energy unit (marked as green or black), we regrow patches using a probability of regrowth rather than increasing by even increments. Accounting is easier here, since only black patches are asked to regrow and they cannot grow past a `max-grass` limit.

CODE BLOCK 6.27

```
to grass-regrowth
  ask patches with [pcolor = black] [
    if random-float 1 < grass-regrowth-rate [
      set pcolor green
    ]
  ]
end
```

In the `evaluate-stock` procedure we have asked herders whose cows have died to turn red. If you now run the model under different combinations of `cow-forage-requirement`, `grass-regrowth-rate`, and `selfishness`, you can see how many herders can take advantage of the common land. See if you can find a parameter combination that results in a lasting stable system instead of the herders driving the cow population to extinction.

Add a few plots showing the number of grass patches, cows, and active herders to trace the evolution of the system.

GAME THEORY

The tragedy of the commons emerges from agents' interactions in which each agent tries to maximize their own utility in effect competing over a limited subsistence resource. In this case, the game is simple: the more cows, the higher the individual payoff up to a point when grass gets completely depleted and no one can take advantage of it anymore. You can easily imagine how this kind of situation will lead to a social contract in which individual payoff is constrained to a solution sustainable for the whole community over long term. This in turn creates a situation in which agents can follow two strategies: cooperate or defect. This kind of framework of interaction falls under the scope of **game theory**. Game theory is a family of mathematical models of interactions between agents trying to optimize their individual utility while outwitting their opponents (Bonanno 2018).

game theory: a mathematical framework studying models of interaction between competing agents, usually rational decision makers.

The most basic of all game-theoretical games is the *Prisoner's Dilemma* (Axelrod 1980; Poundstone 1993). Imagine two gang members caught by the police. They are separated and can either betray their colleague (defect) or stay silent to protect them (cooperate). If they both defect, their prison sentences will be ten years each; if they both stay silent, they will be sentenced to two years. Yet if only one of them defects, the betrayed partner gets a five-year sentence while their deceitful colleague goes free. The problem is set in this way to show that although the optimal solution of the system is for both to stay silent (a total of four years of incarceration), individually, betraying the colleague holds the highest potential reward.

	Cooperate	Defect
Cooperate	2 yrs/2yrs	5 yrs/free
Defect	free/5 yrs	10 yrs/10 yrs

Now, this is a single instance of the situation, but in the real world we make such decisions repeatedly, which means that the system evolves over time and reputation becomes important. Also, consider how this game would play out in a space where agents are neighbors spread over an area and free to choose a strategy: defect or cooperate. It pays to cooperate if one is among other cooperators. However, a single defector can easily take advantage of their sincere neighbors, causing them to switch and start defecting. This dynamic is captured in the *Prisoner's Dilemma Basic Evolutionary* model, which you can find in the NetLogo Models Library (Wilensky 2002).

Model:
Prisoner's Dilemma by Wilensky

We initiate all patches with either "cooperate" or "defect."

CODE BLOCK 6.28

```
patches-own [ cooperate? score ]

to setup
  clear-all
  ask patches [
    ifelse random-float 1.0 < (66.6 / 100)
      [set cooperate? true
      set pcolor yellow]
      [set cooperate? false
      set pcolor red]
  ]
  reset-ticks
end
```

The agents (i.e., patches) then interact and calculate the payoff of this interaction `score`, which they will use to determine how to behave in the next round.

CODE BLOCK 6.29

```
to go
  ask patches [interact]
  ask patches [select-strategy]
  tick
end
```

To calculate the payoff, agents simply count the number of cooperating neighbors if they are one of the cooperators, or multiply that number by a `defection-award` (set by a slider) if they are a defector. At each time step, agents assess who was the most successful agent (i.e., the neighbor with the highest score) in their neighborhood and copy that agent's strategy.

The slider `defection-award` takes values between 0 and 3, with a 0.25 increment.

CODE BLOCK 6.30

```
to interact
  let total-cooperaters count neighbors with
  [cooperate?]
  ifelse cooperate?
    [set score total-cooperaters]
  [set score defection-award * total-cooperaters]
end

to select-strategy
  set cooperate? [cooperate?] of max-one-of neighbors
  [score]
  ifelse cooperate? [set pcolor yellow][set pcolor red]
end
```

If you set the value of `defection-award` to close to 1.5, you'll find an interesting equilibrium of constantly shifting areas dominated by either defectors or cooperators.

This is just a simple application of the most basic model in game theory, a continually growing field of mathematics. There has been very limited application of it in archaeology, with only a handful of models taking advantage of this powerful framework (Kohler, Cockburn, et al. 2012; Crabtree et al. 2017).

6.7 Parameterization, Realism, Abstraction & Heterogeneity

As you were building the subsistence models, you might have noticed that in many places the code required specific numbers—parameters—rate of growth, size of the yield, number of days between harvests, etc. These are different from **variables**, i.e., individual attributes of the agents that dynamically change over the course of a run, such as age, or energy level. Parameter values must be carefully selected in the process known as **parameterization**. The task of parameterization involves selecting values and ranges for all the parameters in a model before running experiments. In practice, this involves some combination of selecting ranges derived from the archaeological or ethnographic records, making up values that seem logical, or picking arbitrary ranges like 0 to 100. While often challenging, being forced to select

variable: attribute of an agent, patch, or the world that can change over the course of a simulation run, such as age, fitness, grass, or number of agents alive.

parameterization: the process of selecting appropriate parameter values and value ranges for a model before running experiments.

values is a key advantage of the ABM approach, because it forces formalism. Terms like "rapid," "efficient," or "close" need to be replaced with values indicating, for example, how rapid the change of subsistence strategy occurs, how efficient a new subsistence strategy is, or whether "close surroundings" means a 5 or 50 km radius. Models often surprise us in terms of how responsive they are to certain parameters, regardless of our intuition regarding their importance or role in the model's dynamics (Romanowska 2015b). As an example, in the *Wolf–Sheep Predation* model the grass growth rate has an impact on the population of wolves (if it is too high, they go extinct first; if it is too low, they . . . also go extinct first), even though wolves do not eat grass themselves.

The models discussed in this chapter vary widely in their level of abstraction versus empiricism. Of course, all models are simplified representations of the real world, since by definition a model is an abstraction. However, some models attempt to be closer to our empirical understanding of a given phenomenon, while others are more abstract and focus instead on the dynamics of a modeled system. Premo (2010) referred to the different goals of models: **emulation**, where the modeler's goal was to test specific hypotheses by using empirically derived parameter values, and **exploration**, where the modeler was focused on theory building through understanding the dynamics of a simpler and more abstract model.

emulation-driven models: data-driven models encompassing a high degree of realism and precision. Commonly validated against archaeological datasets.

In emulation models, parameter values are meticulously derived from data and their ranges closely coupled with each other in respect to the scale (e.g., energy consumption is matched to the harvest period). For example, in a population dynamics model we could determine age-specific fertility rates using literature on demography and have the probability of death correspond to the Siler hazard equation (Gurven and Kaplan 2007). Importantly, though, once one of the parameters is pegged to a real-world value, others need to follow suit. Otherwise, you risk losing the realism and precision you achieved because of the arbitrariness or uncertainty of even one other parameter. For example, having exact ranges for crop yield, calorie consumption per person, and storage capacity is meaningless if the number of people per household consuming the crop is unrealistic.

exploration-driven models: usually theory-driven models, showing a high degree of abstraction and generality. Commonly validated against stylized facts or not validated at all.

In contrast, exploration-type models usually operate without reliance on exact parameter values. More often, parameters are varied along wide but arbitrary ranges to explore dynamics under a wide range of conditions.

What is usually more important is the relative relationship between the values (e.g., predators derive twice as much energy from sheep than sheep derive from grass).

Compared to the wide variety of algorithmic choices in mobility models or modes of cultural transmission, subsistence models tend to be fairly similar in algorithms: agents extract resources from patches, patches regrow, agents live, reproduce, and die depending on their success in gaining energy. However, the empirical basis for the parameterization of these models can be highly realistic and detailed. Rainfall records, soil quality, erosion factors, reproductive probabilities from ethnographically recorded fertility rates and age structures, daily grams of meat protein needed per person, and stored food rotting rates have all been intricately measured and incorporated into one subsistence model or another.

6.8 Summary

Combining all of the algorithms we have covered here—from having patches with varying productivity, degradation rates, harvesting algorithms, storage capacities, and energy accrual and costs—can create the base of a model that can be used to examine many social phenomena. Indeed, the combination of these simple algorithms creates the basis for most models that examine subsistence, with only small variations for modeling farming versus foraging societies. The simple versions we coded here can be used to examine any number of scenarios where their subsistence strategy is key to understanding the society you study.

Many of the more complex agent-based models of human societies are built on top of subsistence models. For example, the *Village Ecodynamics Project* (VEP) has developed a suite of agent-based models using sophisticated subsistence algorithms combined with other dynamics of human society (Kohler and Varien 2012). From a base of algorithms that calculate spatial and temporal variability in productivity, the VEP team has examined a wide range of topics, including:

- Modeling the domestication of turkey to examine when Ancestral Pueblo people would have switched from primarily hunting to provisioning with turkey and maize (Bocinsky 2011).

- A public goods game, where cooperative use of public goods leads to increases in agricultural productivity and the development of within-group hierarchy (similar to the tragedy of the commons; see sec. 6.6; Kohler, Cockburn, et al. 2012).
- A model of how between-group hierarchy could form in regions with unpredictable agricultural productivity (Crabtree et al. 2017).
- The development of specialization in small-scale societies, allowing for specialist farmers and specialist hunters, among others (Cockburn et al. 2013).
- How cultural learning through social networks was impacted by variability in landscape productivity over time (Mokom and Kobti 2015).
- The exchange of maize as the basis for the development of social networks, which then formed the basis for exchange in all of the above models (we will build a simplified version of these models in ch. 8; Crabtree 2015).

The array of models built from the VEP's base subsistence model demonstrates the utility of building models from the ground up. Even though all the aspects are tightly interconnected, the productivity of the landscape impacts population size, where agents are located, and all sorts of dynamics of human interaction, so it is a good idea to introduce them into the model one by one. By following this paradigm of gradually layering complexity, the *Village* models have become a cornerstone for agent-based modeling in archaeology.

Subsistence models often incorporate external variability, from stochastic rainfall to dynamically degrading soils; in section 6.2, you learned how to incorporate simplified versions of these shifting variables. These types of variables can be quite complicated, especially if you are using realistic models or weather station data. Again, keep your research questions in mind as you choose what to include and use the principles of parsimony to guide your research. The same applies to defining parameter ranges and values that will then be tested later in experiments (see ch. 9). Note how almost all of the models in this chapter examine one specific hypothesis. Even though their authors were well aware that the lives of past people had many aspects, they represented them using the simplest model possible that still

incorporated the necessary variables to examine their hypotheses in question.

Finally, we show how we can understand the resilience of past societies based on how agents respond to changing conditions in the environment. Agents bounce back from the brink of extinction thanks to planning ahead (e.g., storage), adaptability (e.g., changes in mobility), or social structures (e.g., social contracts). Subsistence algorithms provide a strong base for understanding the function of past societies; the algorithms we present you with here can be built upon to examine additional questions about resilience, vulnerability, and the evolution of socionatural systems.

End-of-Chapter Exercises

1. How would you model stochastic (as in once every ten years) productivity shortfalls that many agricultural societies prepare for?

2. Past peoples engaged in multiple different types of sustenance. Model farmer–hunter agents who require not just farmed calories but hunted protein as well. How would that impact the model dynamics?

3. Should other types of necessities be included as well? How would water or fuel needs be included in a model of subsistence for an agent? How can you balance all of these different needs within the code?

4. In the *Patch-Choice* model, we only included a random-walk. Add an option for a target-walk, and compare the average return and the frequency of movement for the random-walk versus the target-walk.

Subsistence Model Zoo

Many of the models below are also available in our code repository.

▷ **Ache Hunting**

M. A. Janssen and K. Hill. 2014. "Benefits of Grouping and Cooperative Hunting Among Ache Hunter–Gatherers: Insights from an Agent-Based Foraging Model." *Human Ecology* 42, no. 6 (August): 823–835. doi:10.1007/s10745-014-9693-1

Code: https://doi.org/10.25937/66d6-kz70

▷ **AgModel**

L. Barton and I. I. T. Ullah. 2016. "Computer Simulation and the Origins of Agriculture in East Asia." Presented at the Seventh Worldwide Conference of the SEAA June 8–12, 2016, Cambridge/Boston, MA, USA. Boston, MA.

Code: I. I. T. Ullah. 2015. *Agmodel: Version 0.3.* doi:10.5281/ZENODO.17551

▷ **AmphorABM**

S. A. Crabtree. 2016. "Simulating Littoral Trade: Modeling the Trade of Wine in the Bronze to Iron Age Transition in Southern France." *Land* 5, no. 1 (February): 5. doi:10.3390/land5010005

Code: https://comses.net/codebases/c310d351-b629-46ec-a29e-eb365aaa08b4/

▷ **Cardial Spread**

J. Bernabeu Aubán et al. 2015. "Modeling Initial Neolithic Dispersal. The First Agricultural Groups in West Mediterranean." *Ecological Modelling* 307 (July): 22–31. doi:10.1016/j.ecolmodel.2015.03.015

Code: S. Bergin. 2019. *The Cardial Spread Model.* CoMSES Computational Model Library. https://www.comses.net/codebases/5278/releases/1.1.0/

▷ **Cultural Hitchhiking**

G. J. Ackland et al. 2007. "Cultural Hitchhiking on the Wave of Advance of Beneficial Technologies." *Proceedings of the National Academy of Sciences* 104, no. 21 (May): 8714–8719. doi:10.1073/pnas.0702469104

Code: https://www.pnas.org/content/suppl/2007/05/29/0702469104.DC1

▷ **Diet-Breadth**

C. M. Barton. 2015. *Diet-Breadth Model from Optimal Foraging Theory (Human Behavioral Ecology).* https://www.comses.net/codebases/2225/releases/1.1.0/

▷ **Fission–Fusion**

E. R. Crema. 2014. "A Simulation Model of Fission-Fusion Dynamics and Long-Term Settlement Change." *Journal of Archaeological Method and Theory* 21 (2): 385–404. doi:10.1007/s10816-013-9185-4

Code, original in R: https://github.com/ercrema/fissionfusion2014

▷ **Ger Grouper**

J. K. Clark and S. A. Crabtree. 2015. "Examining Social Adaptations in a Volatile Landscape in Northern Mongolia via the Agent-Based Model Ger Grouper." *Land* 4, no. 1 (March): 157–181. doi:10.3390/land4010157

Code:
https://comses.net/codebases/27f01923-3884-48ca-81ca-55739f976dc0/

▷ **LGM Ecodynamics**

C. D. Wren and A. Burke. 2019. "Habitat Suitability and the Genetic Structure of Human Populations during the Last Glacial Maximum (LGM) in Western Europe." *PLOS ONE* 14, no. 6 (June): e0217996. doi:10.1371/journal.pone.0217996

Code: https://doi.org/10.25937/na38-tj46

▷ **MASTOC - A Multi-Agent System of the Tragedy of the Commons**

J. Schindler. 2012. "A Simple Agent-Based Model of the Tragedy of the Commons." In *ECMS 2012 Proceedings,* edited by K. G. Troitzsch, M. Moehring, and U. Lotzmann, 44–50. ECMS, May. doi:10.7148/2012-0044-0050

Code: https://www.comses.net/codebases/2283/

▷ **MedLanD**

C. M. Barton. 2014. "Complexity, Social Complexity, and Modeling." *Journal of Archaeological Method and Theory* 21, no. 2 (June): 306–324. doi:10.1007/s10816-013-9187-2

Code: NetLogo 4.2 version, https://doi.org/2286.0/oabm.3826

▷ **Multilevel Model of Population Dynamics**

N. Gauthier. 2019a. *Multilevel Simulation of Demography and Food Production in Ancient Agrarian Societies: A Case Study from Roman North Africa.* Preprint. SocArXiv, August. doi:10.31235/osf.io/5be6a https://osf.io/5be6a

Code: https://github.com/nick-gauthier/Silvanus

▷ **Piaroa Swidden**

P. Riris. 2018. "Assessing the Impact and Legacy of Swidden Farming in Neotropical Interfluvial Environments through Exploratory Modelling of Post-Contact Piaroa Land Use (Upper Orinoco, Venezuela)." *The Holocene* 28, no. 6 (June): 945–954. doi:10.1177/0959683617752857

Code available in article's supplementary material.

▷ **Patch-Choice**

C. M. Barton. 2013. *Patch-Choice Model from Optimal Foraging Theory (Human Behavioral Ecology).* https://www.comses.net/codebases/2224/releases/1.0.0/

▷ **Swidden Farming**

C. M. Barton. 2014. "Complexity, Social Complexity, and Modeling." *Journal of Archaeological Method and Theory* 21, no. 2 (June): 306–324. doi:10.1007/s10816-013-9187-2

Code: https://doi.org/2286.0/oabm.3826

▷ **Wolf–Sheep Predation**

U. Wilensky and W. Rand. 2015. *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo.* Cambridge, MA: MIT Press, April.

U. Wilensky. 1997. *NetLogo Wolf-Sheep Predation Model.* Evanston, IL.

Code:
https://ccl.northwestern.edu/netlogo/models/WolfSheepPredation

NOTES