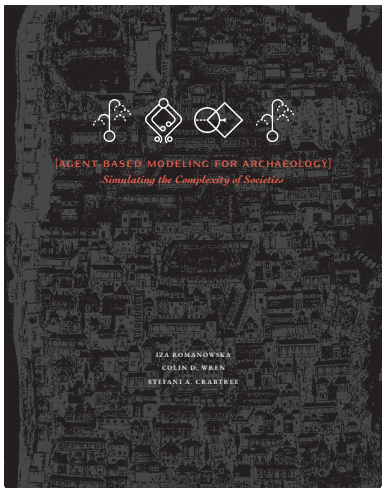
PLEASE NOTE:

The contents of this open-access PDF are excerpted from the following textbook, which is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License:

Romanowska, I., C. D. Wren, and S. A. Crabtree. 2021. *Agent-Based Modeling for Archaeology: Simulating the Complexity of Societies.* Santa Fe, NM: SFI Press.

This and other components, as well as a complete electronic copy of the book, can be freely downloaded at https://santafeinstitute.github.io/ABMA

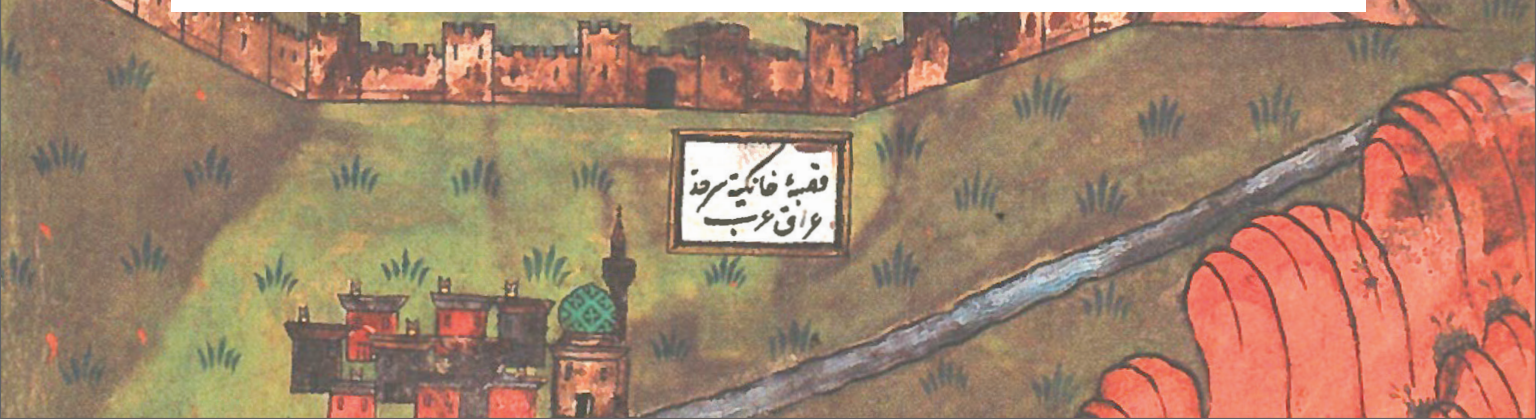


REGARDING COLOR:

The color figures in this open-access version of *Agent-Based Modeling for Archaeology* have been adapted to improve the accessibility of the book for readers with different types of color-blindness. This often results in more complex color-related aspects of the code than are outlined within the code blocks of the chapters. As such, the colors that appear on your screen will differ from those of our included figures. See the "Making Colorblind-Friendly ABMs" section of the Appendix to learn more about improving model accessibility.

SFI PRESS

THE SANTA FE INSTITUTE PRESS 1399 Hyde Park Road, Santa Fe, New Mexico 87501 | sfipress@santafe.edu



MOBILITY ALGORITHMS: HOW DOES MOVEMENT LEAVE ITS MARK?

4.0 Introduction

OVERVIEW
▷ Algorithm zoo: pedestrian movement, group migration, and population dispersals
▷ Simulation scales
▷ Types of validation
▷ What is equifinality?

The previous part, **Learning to Walk**, was designed to establish a baseline of skills necessary for any modeler. We used practical tutorials replicating published models to introduce NetLogo, its syntax, and its most common primitives. The second objective was to develop a steadfast understanding of the modeling process, its many uses, and its role in the scientific practice. You may find yourself revisiting the first chapters to look up a term or specific NetLogo primitive, but you are now well equipped to read and understand any existing models. As usual, the NetLogo Dictionary and Programming Guide are the ultimate arbiters when in doubt. We will return to the tutorial-style chapters in Part III.

Part II, **Learning to Run**, builds on the foundation of the first section. Where **Learning to Walk** was a whirlwind tour of the NetLogo language, this part aims to consolidate what was introduced before. Here you will gain steadiness in your modeling steps, in terms of both coding and model building. Thus, this and the following two chapters will differ in structure from part I. Instead of working through a published case study, we will do what you probably have been eager to do all along: give you the tools to create your own models. As in the previous part, we will use the three topics constituting the building blocks of most archaeological models: human movement (ch. 4), exchange (ch. 5), and subsistence (ch. 6). We will review the algorithms used to model these phenomena and their applications, including published examples and potential applications in archaeology. The aims of this part are:

- To offer a comprehensive overview of the modeling approaches and building blocks of most archaeological simulations.

- To look in more detail at the modeling process, from research questions, formalization, and implementation to calibration and validation.
- To gain the necessary proficiency in coding to write your own models independently.

This part showcases how modeling even seemingly basic processes can be done in multiple ways and how decisions made at each modeling step can impact the overall results of your model in occasionally unpredictable ways. One of the topics may be closer to your research than others—we nevertheless recommend you at least skim through all three chapters. You never know what the modeling journey may throw at you, so having an idea of what exists is useful, even when it is far from your current interests. While we will not produce fully worked models in this section, we nevertheless recommend that you keep NetLogo open to try things out as we go along.

TIP
We recommend you open a new model, set the world's screen to 100 × 100 patches (patch size 4, or adjust to suit your screen) with all edges wrapped.

The final section of this chapter contains a full bibliography of the models included in this chapter, along with links to their original code if available. You can also refer to the ABMA Code Repo for working versions of all models.[1]

4.1 Modeling Human Movement

Human movement is one of the prime topics in archaeology and, coincidentally, a good example of a research area where simulation techniques are particularly easy to apply. Human movement is inherently about individual decision-making in response to external factors such as the environment or other agents that, when aggregated, can produce unexpected population-level patterns. These patterns are easily recognizable in the archaeological record. For example, we can detect the earliest (known) traces of human presence in a previously uninhabited territory (see ch. 1, Mithen and Reed 2002; Romanowska et al. 2017), changes in the spatial distribution of a specific cultural tradition (Crema and Lake 2015; Gravel-Miguel and Wren 2018; Wren and Burke 2019), or the arrival of a previously unknown subsistence strategy (Lemmen, Gronenborn, and Wirtz 2011; Fort et al. 2016). We will illustrate the mechanisms that lead to these kinds of population-level pat-

The classic ABM-in-geography textbook is (O'Sullivan and Perry 2013); for a more recent one, see (Crooks et al. 2019).

[1] You can find all code written in this chapter in the ABMA Code Repo: https://github.com/SantaFeInstitute/ABMA/tree/master/ch4

terns. We will also discuss **validation**, the phase of the modeling process that addresses one of the most fundamental questions in modeling: How do we test whether our model is a correct representation of the past system?

DON'T FORGET
Remember the difference between **verification** and **validation**. Verification means we ensure there are no errors in the code; in validation we try to assess whether the model is a *valid* representation of the real system we're modeling.

It is relatively simple to model human movement, as it essentially depends on one decision event. The agent has to update its location—that is, choose where in space (including its current position) it wants to be. How this decision is made, what will be weighed to assess options, or how much information the agent possesses will depend on the studied context, and thus differs from one model to another. In many models, the ultimate goal is to maximize one's fitness by foraging for food and other resources, or by avoiding overcrowded areas. Such models are often accompanied by a sensing submodel that enables the agent to assess each option and choose accordingly. Other models focus on individual micromovements; for example, pedestrians walking through the streets of an ancient city or traveling the roads of an empire. In these cases their decisions will depend on the individual's particular goals (follow a procession, find a workshop, reach another city, etc.). Finally, in some cases the type of movement is unknown, or not relevant, so the agent's decision of where to move is modeled as a random or semirandom process, as in the *Y&B* model from chapter 1.

Not all agents have the same sensing ability. They may also differ in what they want to optimize depending on their particular circumstances.

In this chapter we will explore the question of **scale**, one of the crucial challenges of initial phases in the modeling process. The scale chosen for the model depends on the research questions as well as the archaeological data available for validation. For example, most human movement models fall into one of the following three categories:

scale (in models): the size of spatial units and the duration of temporal units within the modeled system.

- individual/pedestrian movement;
- group/household movement; and
- medium- and long-range population dispersal.

Pedestrian movement (sec. 4.3) concerns an individual; in this instance one agent equals a person, and one time step is usually measured in seconds, minutes, hours, or days. The topics that fall within this category often have very fine temporal resolution, for example, battleground simulations (Rubio-Campillo, Matías, and Ble 2015; Rubio-Campillo, Cardona, and Yubero-Gómez 2015), and be precise in terms of the topog-

raphy, for example, people finding their seats in the Colosseum (Gutierrez et al. 2007) or taking part in a procession (Crawford 2019b), but may also concern much more abstract systems. For example, in foraging models, an agent may look for food, water, or other resources within an abstract landscape (Brantingham 2003).

To model something on the scale of hours and meters, we need matching data. Thus, case studies usually come from the more recent past.

Group movement (sec. 4.4) refers to the seasonal or annual changes in the main location of a small group or family unit. The agents represent small groups of people, and time generally passes in increments of weeks, months, seasons, or years. This kind of movement may be associated with the creation of a new household (e.g., when "children" move out), a household move due to environmental or social changes, or the spreading of a city's urban area (Batty 2005).

The third type, medium- and long-range population dispersal (sec. 4.5), is modeled using whole populations of people as agents and years, decades, or generations as time steps. We often differentiate between two main types: the **cultural diffusion** that concerns the spread of an idea, such as an innovation, a decoration pattern, or a belief, and the **demic diffusion** that concerns physical movement of people, such as in the out-of-Africa dispersal. Although these processes are fundamentally different, they often leave behind similar data patterns, for example, a new artifact type being found farther and farther from a point of origin, making it difficult to distinguish them in the archaeological record without the use of mathematical tools (Fort 2015). We will cover demic diffusion in this chapter and cultural transmission in chapter 5.

We often quip that in demic diffusion it's people that move, while in cultural diffusion it's pots.

In this chapter we will use these three scale categories to structure our discussion, reviewing the algorithms used to model these types of processes and their application in archaeology and other disciplines. Since movement is not inherent to humans alone, many of our methods and algorithms originated with chemistry or ecology but have been shown to mimic human behavior in surprisingly sharp detail.

4.2 An Introduction to Algorithms

In chapter 1, we gave a brief definition of an algorithm as a set of instructions to be followed in a specific order by the computer. But algorithms do not exist only in the computing world; think of a cooking recipe.

When you open Julia Child's *Mastering the Art of French Cooking*, you follow her steps to go from raw eggs to mayonnaise. For example: "Place the egg yolks in a bowl, add vinegar, mustard, and oil, beat until the mix thickens." You may notice how the ingredients need to be specified (egg yolks and not whole eggs) and the order is important (you cannot beat the ingredients before you put them in a bowl). By employing this same clearly defined sequence of actions, we can ensure the final output can be replicated by others (e.g., not Julia Child herself) and in other contexts (e.g., using a different bowl).

Think of software as a cookbook for the computer. Data are the ingredients that go into the recipe, and the output is the prepared dish.

Computer algorithms are, in principle, the same as cooking recipes but described with much less ambiguity. The entities involved, exact order of actions, and modification to variables are all highly specified. An algorithm, then, turns the current state of a model into the state of the model at the next time step. It's a good idea to have a variety of algorithms in your modeler cookbook because, just like any methodology, each comes with its own set of assumptions that may or may not be appropriate depending on the particular research questions. With this in mind, let's look at algorithms used to model human movement.

As with different recipes for the same dish, it is possible to achieve similar results via different algorithms.

4.3 Pedestrian Movement

RANDOM WALKS

The most basic method for modeling individual movement is a **random walk**. Here, the agent has an equal probability of moving onto any location within its movement radius. The movement does not have a predominate goal or direction. We often use a random walk when we do not know what drove mobility decisions, when we think that there are many independent types of movements overlapping, or as the null-hypothesis pattern of movement to establish a benchmark other types of mobility could be compared to. The most distinctive feature in a random walk is that it involves a lot of turning back on itself and that the agent takes a while to move away from the starting point (fig. 4.0). With a large number of steps, the average behavior should approximate the central limit theorem, meaning that the agents' positions around the origin point will

Model: *Random Walk*

ABMA Code Repo: *ch4_pedestrian_movement*

be normally distributed (Pearson 1905; Viswanathan et al. 2011).[2] Random walks have been applied in many archaeological simulations, for instance, Brantingham (2003, see ch. 7 in this book) created a behaviorally neutral model of stone raw material procurement using a random walk.

During writing, the three authors of this textbook discovered that we each coded random walks in a different way.

The two implementations below illustrate the point that even something as simple and common as a random walk can be coded in different ways, with different implications. First, we make a simple `setup` to create one agent and use the `pen-down` primitive to draw the path taken by the agent. While this last step is not necessary, it helps us to see how the agent moves around on the landscape. Then we provide two different implementations of choosing the new location at random: `random-walk` and `random-walk-patches`.

CODE BLOCK 4.0

You can make buttons in the INTERFACE tab for each type of walk, or add them to the `go` procedure and uncomment as required.

```
to setup
  ca
  crt 1 [
    pen-down
  ]
end

to random-walk
  ask turtle 0 [
    if random 2 = 1 [
      ; rt random 360 ; alternative implementation
      set heading random 360
            fd 1
    ]
  ]
end
```

[2] In thanking the reader who wrote in with a solution to his random walk query, Pearson (1905) wrote: "The lesson of Lord Rayleigh's solution is that in open country the most probable place to find a drunken man who is at all capable of keeping on his feet is somewhere near his starting point!"

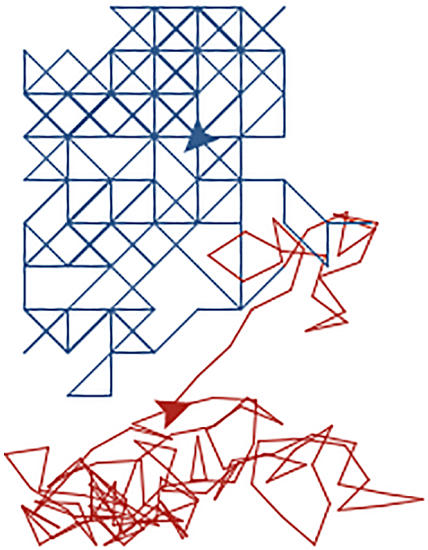


Figure 4.0. One random-walk agent and one random-walk-patches agent with "pen-down" to display their movement paths. The random-walk agent has more variation in turn angles and moves less often. Note that the agents both recross their paths frequently.

```
to random-walk-patches
  ask turtle 0 [
    move-to one-of patches in-radius 1.5
  ]
end
```

CODE BLOCK 4.0 (cont.)

One- and two-dimensional random walks are recurrent (they come to the starting point infinitely often) and unbounded (they will, eventually, reach any value).

`move-to` transports the turtle to the center of a patch. Radius 1.5 will include the centers of all eight neighboring patches and the currently occupied patch.

In the first implementation, `random-walk`, we flip a virtual coin (`random 2`) which returns a value of 0 or 1. If it is 1, the agent moves; if it is 0, they stay put. Then we ask the agent to face a random direction (using either `rt random 360` or `set heading random 360`) and take one step forward. In the second version, `random-walk-patches`, we ask the agents to move to a random cell within a radius of 1.5 cells. In effect, this is the agent's current cell plus each of the eight cells around it.

At first glance, these two ways of coding a random walk seem to be equivalent. In fact, they are not, for two reasons. First, in the `random-walk` version, the agent always moves a distance of 1.0 map unit (the width of a patch) at any direction of 360 degrees of heading. Thus, an

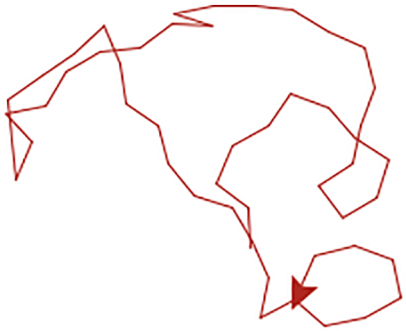


Figure 4.1. Correlated random walk agents recross their own path less often, as their turns are limited.

agent traveling along the diagonal could step on a single patch twice, then step over another without touching it at all. In the `random-walk-patches` version, the agent always moves from a patch center to another patch center, thus ensuring that interaction with all neighbors of the current patch occurs at the same rate. However, it also means that they move farther on the diagonals (~1.41 map units) versus the straight compass directions (1.0 map units). Second, in the first version the probability of staying on the current cell is 50% (one in two), whereas in the second version it is ~11% (one in nine).

TIP

If `neighbors` was used instead of `in-radius 1.5`, the agent would always move. See these and other `random-walk` versions in the ABMA Code Repo.

Neither of these implementations is more correct, but, if deployed in the *Y&B* model from chapter 1, they would yield different dispersal rates. If used, the second algorithm (`random-walk-patches`) would have moved the agents away from their origin faster because just under half of the agents' steps are 41% longer.

CORRELATED RANDOM WALKS

Model: *Correlated Random Walk*

Moving beyond the basic random walk, a more realistic approximation of an individual may be for the agent to mostly follow the same direction. Here we are taking the view that the directions chosen per step are correlated, that is, not independent of one another even if the agent is not heading anywhere specific. Correlated walks are also known as persistent random walks, because the general directionality persists as the walker continues (Patlak 1953; Codling, Plank, and Benhamou 2008). To

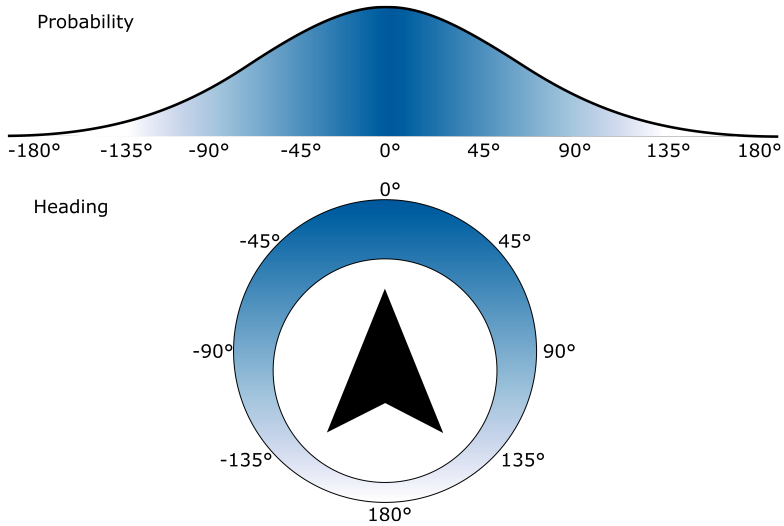


Figure 4.2. Correlated random walk. Agents have the highest probability of moving in the direction they are facing.

implement a correlated random walk, we change the base algorithm only slightly.

CODE BLOCK 4.1

```
to random-correlated-walk
    ask turtle 0 [
            rt random-normal 0 45
            fd random 2
  ]
end
```

Here, we limited the options of the agent's initial heading. Instead of turning on their heel and going in any random direction, the agent has the highest probability of moving in the general direction they are already facing (fig. 4.1). The `random-normal 0 45` primitive draws a random number from a normal distribution centered on the mean (here: 0) and the standard deviation (here: 45 degrees). See figure 4.2).

LÉVY FLIGHTS

Model: *Lévy Flights* by O'Sullivan & Perry

So far, the agent has moved only up to one step at a time. However, some research (e.g., Raichlen et al. 2014; Viswanathan et al. 1996) indicates that many animals and humans follow a completely different pat-

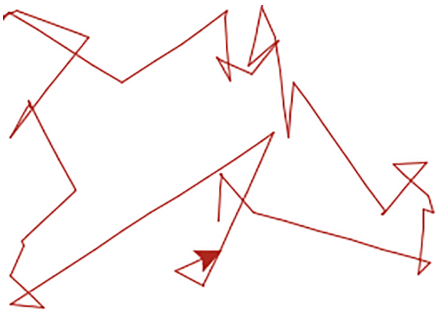


Figure 4.3. In a Lévy flight, the heading is completely random, but the step length is taken from a Cauchy probability distribution. This means that most steps are short, with only occasional longer-distance steps.

tern of movement, called *Lévy Flights*. The distribution of step lengths in *Lévy Flights* forms a fat-tailed distribution called a Cauchy distribution. In practice it means that, although most of the steps are short, from time to time the agent covers a particularly long distance in one step (fig. 4.3).

TIP
If your agent's long jumps wrap the map several times, increase the world-size.

CODE BLOCK 4.2

```
to levy-walk
  ask turtle 0 [
    set heading random 360
    let step-length abs r-cauchy 0 1
    fd step-length
  ]
end

to-report r-cauchy [loc scl]
  ; NetLogo tan takes degrees not radians
  let X (pi * (random-float 1))
  report loc + scl * tan(X * (180 / pi))
end
```

TARGETED WALK

Model: *Targeted Walk*

In many models, the agent engages in a search for resources, other agents, or particular locations. This means that they need a form of sensing—an

ability to gather information about their surroundings. Add some target patches in the `setup` that the agent aims to find.

CODE BLOCK 4.3

```
to setup
...
ask patches [set pcolor white]
ask n-of 100 patches [set pcolor black]
…
end

to target-walk
  ask turtle 0 [
    let target min-one-of patches with
      [pcolor = black] in-radius 10 [distance myself]

    ifelse target != nobody [
      move-to target
      ask patch-here [set pcolor grey]
    ][
      set heading random 360
      fd 1
    ]
  ]
end
```

The `in-radius` primitive is particularly useful here, as it can define the agent's sensing distance. The `min-one-of` primitive, followed by `[distance myself]`, specifies that the agent should move to the target *closest* to the agent if there is more than one within the sensing radius. We can include multiple criteria, here the target type and distance, in one local variable target, and then choose the one that best fits these conditions. The main decision-making algorithm here is based on an **ifelse loop**: if there is a sought-after patch within the sensing radius (`if target != nobody`), the agent moves to it; otherwise it engages in a random walk (fig. 4.4). This is a universal algorithm that can be easily adjusted in terms of the sensing radius, the characteristics of the target

TIP
It's important to account for unlikely but possible situations. Although the agent will rarely be at exactly equal distance from two targets, it can occur. In this case `min-one-of` chooses randomly between them.

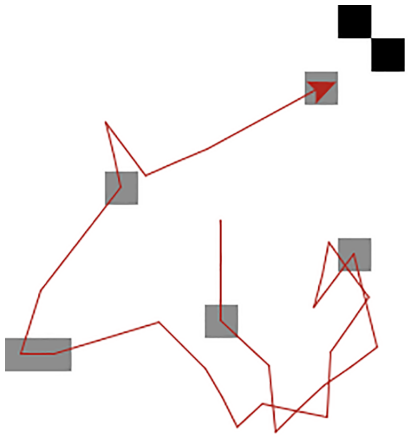


Figure 4.4. Agents find and move toward resource patches if they are within a specified radius, or random-walk if not. Lighter patches have already been visited.

(e.g., instead of the distance, it may be its caloric return), or the choice among multiple targets.

We can use this adjustable algorithm to incorporate a more goal-directed approach to searching a landscape than a simple random walk (also see sec. 6.4). In a model by Wren et al. (2018), a series of ecological habitat types were assigned kcal/hour return values to all patches in a landscape, and then a version of the code above was used to model forager agents searching out the most profitable food resources within the landscape. Rather than simply minimizing distance with `min-one-of... [distance myself]`, the model accounts for certain resources that are only available for parts of a day, calories not acquired while walking to a distant resource, and the capacity for anticipating resources that are not yet available.

Model: *PaleoscapeABM* by Wren et al.

TIP
If the friction surfaces are isotropic—not direction dependent—it is much simpler and more computationally efficient to calculate a cost map in GIS software and then import it into NetLogo.

If the landscape is particularly rugged, the algorithm may also be adjusted to account for the differences in travel cost incurred on shallow versus steep slopes or from other friction surfaces, such as differences in vegetation (e.g., traveling through short grasses versus thick jungle). It takes some knowledge of trigonometry, but this would allow you to incorporate the least-cost path methodology used in standard GIS software into your agent-based model (e.g., Gravel-Miguel and Wren 2018, see ch. 7 for details on using GIS data in NetLogo).

WEIGHTED RANDOM WALK

Model: *Weighted Random Walk* by Wren

A targeted walk assumes that the agent has perfect knowledge of the criteria being used for choosing the target and that they will always pick the most desirable cell. However, if there is a set of acceptable cells with similar values, it may not be realistic to assume a human agent could or would always choose the single best cell. We can model this by weighting a random walk by a continuous variable so that many cells have the potential to be picked randomly, but some have more chances than others. Think of it as rolling a die, but instead of having one each of numbers one to six, the die has two ones (33%) and four twos (67%).

NetLogo has a built-in extension to help us with randomness. Extensions are like an extra library with additional primitives. Some extensions are bundled with NetLogo; others you need to install.[3] Add `extensions [ Rnd ]` to the top of the code, and the primitive `rnd:weighted-one-of` is available for use (Payette 2013). Again, we can combine a number of conditions to decide on the weighting of target patches, for example, the quality of the habitat:

CODE BLOCK 4.4

```
let target rnd:weighted-one-of patches
[habitat_quality]
```

Or the quality of the habitat within five map units:

CODE BLOCK 4.5

```
let target rnd:weighted-one-of patches in-radius 5
[habitat_quality]
```

Or both of these conditions combined with another criterion, the patch being empty:

CODE BLOCK 4.6

```
let target rnd:weighted-one-of patches in-radius 5
with [not any? turtles-here] [habitat_quality]
```

[3]For more information on extensions, see the NetLogo Extensions Guide: https://ccl.northwestern.edu/netlogo/docs/extensions.html.

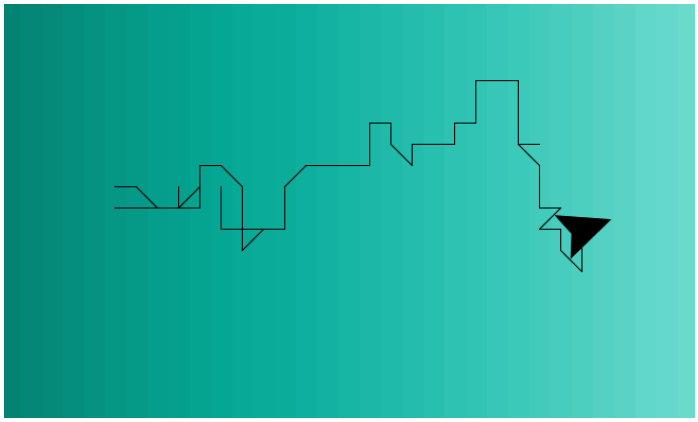


Figure 4.5. The agent's random walk is weighted toward patches with a higher value (lighter shade toward the right). There is still randomness in the movement, but over time the agent shifts to the richer side of the landscape.

Putting it all together:

CODE BLOCK 4.7

There is no limit to the number of criteria you can add, but be aware that each one will complicate the analysis of final results.

```
extensions [ Rnd ]
patches-own [ habitat_quality ]
...
to setup
...
 ask patches [
      set habitat_quality pxcor
      set pcolor scale-color green
        habitat_quality 0 100
    ]
...
to weighted-random-walk
  ask turtle 0 [
    let target rnd:weighted-one-of patches
    in-radius 5 with [not any? turtles-here]
      [habitat_quality]
    if target != nobody [face target move-to target]
  ]
end
```

Some of the criteria are strict; for example, patches must be within a radius of five cells and must be unoccupied. However, the use of `rnd:weighted-one-of patches...[patch_variable]` means that the algorithm is more flexible. The agent is merely *more likely* to choose a cell with a higher value of the patch's `habitat_quality` variable rather than forced to do so. In figure 4.5, our agent is sitting along a gradient of `habitat_quality` values. At any given point, the cells to the east have a higher value, say, a value of ten, and cells to the west have a lower `habitat_quality` of, say, five. In the strict target-walk algorithm above, the agent will always go east. With a `weighted-random-walk`, the agent is twice as likely to go east, but there is still a chance that they will occasionally go west. Adding this kind of "fuzziness" to an agent's decision-making process is a key strength of ABM. Here we can easily incorporate the erratic nature of human (or animal) behavior. Humans do not always choose the most optimal solution. Sometimes they do not know or have no time to correctly assess which option is better; sometimes it just does not matter because the difference is small or unimportant. Contrary to widely held opinion that computational modeling always implicates reducing humans to utility maximizers devoid of agency or imperfections, we can and should include these aspects in our models if the research questions or the context demands to do so.

Herbert Simon received a Nobel Prize for the "discovery" that humans tend to be *satisficers*—they often settle for a *good enough* solution rather than the optimal one.

RESTRICTED WALKS

Model: *Restricted Walk* (Look Ahead Example in NetLogo Library)

In some simulations, in particular those concerning pedestrian movement within the built environment, agent movement is restricted by barriers, such as walls and rivers, or limited to specific transport channels, such as roads (e.g., Graham 2006). In those cases, a function limiting the agents' movement is required. Here you have two options: either eliminate cells of a given color from potential targets (see code block 1.11) or change the heading every time a barrier is found at the cell ahead (code block 4.8).

CODE BLOCK 4.8

```
to setup
  ...
 ask patches with [pxcor = 50] [set pcolor white]
  ...
end

to walk-restricted
  ask turtle 0 [
    ifelse [pcolor] of patch-ahead 1 = white
      [ lt random-float 360 ]
      [ fd 1 ]
  ]
end
```

We have not unclicked WORLD WRAPS HORIZONTALLY, so agents can still access it from the other side.

Here we have built a wall the agents cannot cross. Instead, they turn until they no longer face the wall and can move forward. You can restrict them to a more limited range of turning angles to produce realistic-looking movement patterns.

REMEMBERED LANDSCAPES

Model: *MAGICAL* by Lake

ABMA Code Repo: *ch4_remembered_landscape*

In all of the above algorithms, the agents have no memory of their prior experiences; they merely keep moving forward. Imagine a situation where you would want agents to be able to remember the places they have been or the routes that they previously traveled. In an early application of agent-based modeling to human foraging called *MAGICAL* (multi-agent geographically informed computer analysis), Lake (2000, 2001) allowed each agent to slowly build up their own mental map of a landscape of foraging locations as they encountered them. The agent then made mobility decisions based on this subset of the larger landscape. In the previous versions of the walk algorithm, we restricted mobility options to a subset of the landscape closest to the agent by using the `in-radius` primitive. Here the goal is to store the location of previously visited patches and direct movement toward remembered patches.

NetLogo makes this relatively easy for us through the primitive `patch-set`, which is a list of patches that can either be defined based

on some criterion (e.g., `set patch-set patches with [pcolor green]`) or can be used as a dynamically evolving list, which is what we will do here. As with the `target-walk` above (code block 4.3), agents can hone in on a small number of white target cells on the landscape.

CODE BLOCK 4.9

```
turtles-own [memory]
...
to setup
  ...
  ask n-of 30 patches [ set pcolor white ]

  crt 2 [
    set memory patch-set patch-here
    set color 15 + (10 * random 12)
  ]
  ...
end

to memory-walk
  ask turtles [
    ifelse any? memory with [pcolor = white] [
      let p one-of memory with [pcolor = white]
      face p
      fd 1
    ][
      set heading random 360
      fd 1
    ]
    set memory (patch-set memory patch-here)
    ask memory with [pcolor != white and
      pcolor != grey]
```

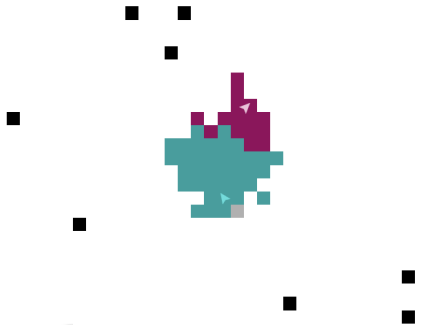


Figure 4.6. Two agents explore a landscape with isolated targets (black). The darker areas show which patches are "remembered" by each agent.

CODE BLOCK 4.9 (cont.)

```
    [ set pcolor [color] of myself - 2 ]
    if [pcolor] of patch-here = white [
      ask patch-here [ set pcolor grey ]
    ]
  ]
end
```

The result will be a colorful mess of agents taking turns to move, and then recoloring their remembered landscape according to their personal color. In figure 4.6, you can see that one agent has just discovered a new patch (now turned gray). Since multiple agents may have the same cell in their memory, when the run stops, whichever agent went last will have their color on top of the others, which is why it looks like one agent has explored less. While this color coding may not be perfect, underneath the turtles store their mental landscape regardless of the displayed color. Similarly, keeping the resource patches (marked, for example, by their white color) in memory can be important in simulations where resources regularly "regrow" and there is a benefit to returning to them periodically. We will return to patch use and regrowth in chapter 6. For now you can just implement a simple probability of gray patches changing back to white using a slider and the code that follows:

```
  ask patches with [pcolor = grey] [
    if random-float 1 < regrow-rate [
      set pcolor white
    ]
  ]
```

CODE BLOCK 4.10

DON'T FORGET
Add a `regrowth-rate` slider with values between 0 and 1 (increment 0.01).

Beyond mental maps, Lake was also interested in the idea of cultural learning. He imagined a group of foragers sharing resource information with each other back at camp at the end of the day. We can add cultural learning to our `memory-walk` by having agents merge their respective mental maps, or `patch-sets`, when they stumble upon one another (see ch. 5 for more examples of learning).

```
ask turtles [
   ...
  if any? other turtles-here [
    set memory (patch-set memory [memory]
      of one-of other turtles-here)
    ask one-of other turtles-here [
      set memory (patch-set memory [memory] of myself)
    ]
  ]
]
```

CODE BLOCK 4.11

A patch-set made up of two previous patch-sets equals a larger patch-set (i.e., the union of both). This doesn't work when combining lists; instead you end up with a list of lists.

Here, each individual agent "memorizes" a new patch-set for themselves out of the combination on their own patch-set and the turtle they currently share a patch with. Add a `learning?` switch to the INTERFACE tab and make it a second conditional after `if any? other turtles-here` if you'd like to be able to toggle this behavior on and off.

In a similar approach to this agent-learning algorithm, Costopoulos (2001) gave agents different memory capacities with respect to remembering their own past resource extraction activities. He then experimented with more or less abundant landscapes and found that the behavior of agents with different memory capacities (i.e., number of patches remembered) diverged.

Model: *Foraging Memory Capacity* by Costopoulos

Figure 4.7. *Flocking/Boids* model. Small "flocks" of agents spontaneously emerge as they adjust their headings and distances to coordinate with their nearest neighbors.

Model: *Trail Evolution* by Helbing et al.

The recognition of signs of human activity offers a less explicit model of shared past experience impacting movement. Helbing, Keltsch, and Molnar (1997) imagined the repeated footprints of many walking people flattening down grass and their accumulated tracks slowly merging into established trail systems as people found it more comfortable to walk on existing paths than on untrodden ground. They modeled the individual's walking direction as an average of movement toward a specific destination and movement toward previously flattened ground. One can imagine such a process leading to the development of a transport system. This is similar to the alignment of directions used in the flocking model (see below), though it focuses on the directions to specific targets (`face target-patch`) rather than the directions of other agents. A similar model, called *Walk This Way*, was implemented in NetLogo by Crooks et al. (2019) .

Model: *Walk This Way* by Crooks

FLOCKING

Flocking, discussed briefly in the introduction to this book, is a type of movement algorithm that is often applied to groups of agents. We are still modeling the behavior of individuals, but this time in coordination with other individuals. In some of the algorithms above, agents could "sense" specific locations or characteristics of patches (like their color), but to coordinate with other agents this sensing must expand to include other agents' locations and characteristics. Originally developed by Reynolds (1987) to mimic the flocking of birds, the *Boids* model's algorithm consists of three simple rules:

Model: *Boids* by Reynolds (*Flocking* in NetLogo Library)

1. Separation: Do not fly too close to neighboring birds (i.e., sensing their direction).
2. Alignment: Adjust the direction of your flight to an average of your neighbors' headings (i.e., sensing their headings).
3. Cohesion: Do not fly too far from the center of gravity of the nearest birds (i.e., sensing their positions).

Surprisingly, this simple algorithm replicates complex patterns of bird flocking and pedestrian movement (fig. 4.7).

Janssen and Hill (2014, 2016) used a modified flocking algorithm to model the movement of human hunting parties spreading out to search for game while staying within auditory range of each other. In contrast, Crawford (2019a) used the flocking algorithm to model the passage of ritual processions through the streets of ancient Ostia.

Model: *Procession Movement* by Crawford

4.4 Group/Household Movement

For research questions that involve a wider area and longer timescale, it often makes more sense to model movement with an agent that represents a household, family unit, or other small group of people. The decision-making algorithm may be quite similar to that of the pedestrian movement already discussed, but is designed to represent wider-scale movements. For example, the residential mobility of a forager band could be modeled by the household agent moving once a month to a neighboring patch to represent a move of a couple of kilometers. While this now represents the collective decision-making of many people, in-

stead of a single person, from a coding perspective it makes little difference. Random, targeted, restricted, and memory walks may all be applied to group movement with minor adjustments for the different spatial and temporal scales. However, when modeling once-a-season types of movement decisions, some algorithms like random walks or Lévy flights make little sense, since the modeled groups would be relatively knowledgeable about their surrounding landscape. With the longer timescale, social dynamics—household population growth, children splitting off to form new households, or interactions with other households, for example—may come into play as well.

Model: *Ger Grouper* by Clark & Crabtree

ABMA Code Repo: *ch4_ger_grouper*

We will examine the questions of household-level seasonal migration using the *Ger Grouper* model (Clark and Crabtree 2015). This model sought to understand how households can fission and fusion in response to environmental stress, as well as how seasonal migration impacts levels of storage and subsistence. We will explore this model further in chapter 6. For now, though, we will examine the movement algorithm. In this model, agents migrate between summer and winter patches and exploit productive (i.e., `pcolor = green`) patches for grazing. There are two types of movement: first, a "teleporting" type between summer and winter areas (`move_spring` and `move_autumn`), and a more local movement aiming to land precisely on a productive patch if one is available. Agents first move to a random patch in their summer grazing area (`move-to one-of summer-patches`). They do not need precise knowledge of the summer lands, so their movement is only vaguely targeted within the summer region, which mimics pastoralist practices. Next, agents move in their eight-cell Moore neighborhood (fig. 1.4 in ch. 1) to a green patch (`ifelse any? neighbors with [ pcolor = green ] [ move-to one-of neighbors with [ pcolor = green]]`) using the `ifelse` primitive to assess the productivity of the neighborhood, and moving to a better patch. Note that in the following code block we only include the movement algorithm of *Ger Grouper*.

In Mongolia, *ger* refers to the nomadic yurt-style house.

For *Ger Grouper*'s patch grazing, energy, and population dynamics, see chapter 6.

CODE BLOCK 4.12

```
to setup
  ...
  set summer-patches patches with
    [pxcor < (max-pxcor / 2)]
  set winter-patches patches with
    [pxcor >= (max-pxcor / 2)]
  ask summer-patches [set pcolor green]
  ask winter-patches [set pcolor one-of [green brown
  brown gray gray gray]]
  ...
to move-season
  ifelse ticks mod 2 = 0 [set current-season
  "winter"][set current-season "summer"]
  ask turtles [
    ifelse current-season = "winter" [
      move-to one-of summer-patches with [count
      turtles-here = 0]
    ][
      move-to one-of winter-patches with [count
      turtles-here = 0]
      move-local
    ]
  ]
end

to move-local
  if pcolor != green [
    if any? winter-patches in-radius 1.5 with [
    pcolor = green ][
      move-to one-of winter-patches in-radius 1.5
      with [ pcolor = green ]
    ]
  ]
end
```

In this algorithm, agents do not store their prior locations but merely operate on the chance that they will move near a productive location. We could update this algorithm to permit agents a memory of the productive patches they camped on during the previous winter, which is the more environmentally restrictive region. The code above would need to be adjusted to `move-to one-of visited-patches` and include a contingency (i.e., using `ifelse`) to continue to search for new patches if no remembered patches were currently productive. This is a similar approach to `memory-walk` (see code block 4.9).

When we return to this model in chapter 6, we'll see how previously productive patches need time to regenerate but that it is likely that a patch will be within the radius of other productive patches and that `local-movement` will therefore enable the household to move somewhere productive. Also, these patches are certain not to be gray patches (half of the winter area is permanently barren), meaning that the household can count on previously visited patches being productive at some point (see ch. 6 for patch-regrowth algorithms). This follows the logic Clark and Crabtree (2015) gathered from ethnographic interviews where households in northern Mongolia responded to a combination of local environmental cues (dry and short grasses that season, local moisture, perception of weather patterns based on past weather patterns, etc.), curated knowledge of past winter camps, and the avoidance of known unproductive areas when deciding where to move (see also Moritz et al. 2018).

MULTILEVEL MOBILITY

Model: *Ache Hunting* by Janssen

ABMA Code Repo: *ch4_ache_hunting*

In certain cases, it might be appropriate to have both household and individual scales of mobility operating within the same model. In their model of Ache hunting patterns in Paraguay, Janssen and Hill (2014, 2016) have both camp agents and hunter agents who operate at slightly different timescales. At the end of the day, agents select a destination for the hunters' camps that represents the collective interests of the group. This location is chosen based on a random-walk (shown in code block 4.13) or by targeting patches of preferred habitat types within a specified radius (see sec. 4.3). The agents then spread out into the forest, keeping track of their distance from camp and their remaining

foraging time to ensure they will end up at their camp agent at the end of the day. While each tick of the model represents a full day (i.e., the camp's timescale), the hunter agents use a `while` loop to repeat their movement commands many times within that tick to represent their subhourly mobility and hunting activities (i.e., the hunter's timescale).

CODE BLOCK 4.13

This code block includes a lot of variables not defined here. Refer to the ABMA Code Repo for a fully worked version.

```
breed [ camps camp ]
breed [ hunters hunter ]

to forage
  ask camps [
    let found 0
    while [found = 0]
    [
      lt random 360
      if patch-ahead 20 != nobody [
        if [vegetation_type] of
        patch-ahead 20 > 0 [
          move-to patch-ahead 20
          set found 1
        ]
      ]
    ]
  ]

  ask hunters [set time-left 12 set done 0]
  let allhuntersdone 0
  while [allhuntersdone = 0]
  [
    ask hunters with [done = 0][
      let walk-time-to-camp
      (distance my_camp * time-walk-cell)
```

CODE BLOCK 4.13 (cont.)

```
      ifelse time-left > walk-time-to-camp [
        lt random 360
        if patch-ahead 1 != nobody [ fd 1 ]
        set time-left time-left - time-walk-cell
      ][
        face my_camp
        if distance my_camp > 1 [ fd 1 ]
        if time-left <= 0 OR
        distance my_camp < 1 [
          move-to my_camp
          set done 1
        ]
      ]
    ]

    if sum [done] of hunters = nhunters [
      set allhuntersdone 1
    ]
  ]
end
```

DON'T FORGET
Make sure to code a way to break out of the `while` loop. For example, if you kill any of your agents during the hunt, make sure the loop is updated.

In this simplified version of their code, the camp agents first random-walk to a location 2 km away from their current location. Next, all hunters that still need to do their hunting estimate how long it will take to walk to their camp (the camp's ID is remembered in the hunter's `my_camp` variable). If they have time to spare, they hunt for game (not coded above) using a random-walk algorithm. If they do not have time, they move directly toward camp and `set done = 1` once they arrive. After remaining agents have had a turn, the model checks to see if the sum of all the agents' done variables equals the number of agents in the model. If so, it breaks out of the `while` loop; if not, the agents who have not finished get another turn (`hunters with [done = 0]`). In figure 4.8, you can see the hunters initially spread out from the previous day's camp location (near bottom left), but as their day goes on they move directly toward camp with increasing frequency. Since moving closer to

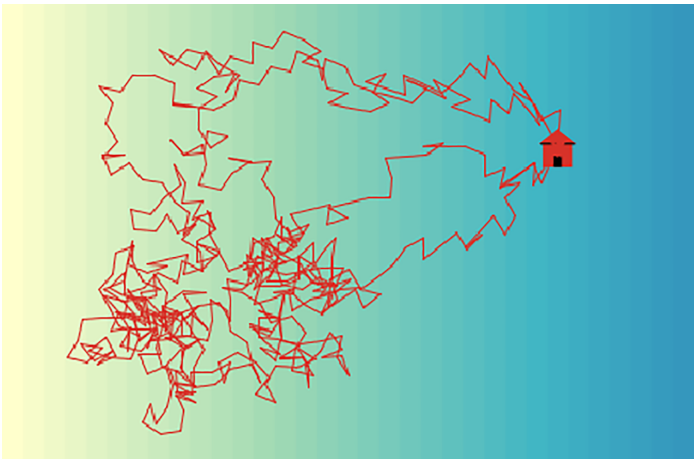


Figure 4.8. The camp agent (house icon) acts as a final destination for the hunter agents, whose paths are shown by the lines.

camp means their `walk-time-to-camp` value decreases relative to their time left, there is still some time for some random-walk hunting along the way.

The camp agent is not just a target for mobility but also a place for agents to share food. In the evaluation of the caloric returns of the model, Janssen and Hill look at the average caloric intake of both the camps and the individual hunters to evaluate hypotheses related to the evolutionary benefits (or not) of food sharing in different group sizes (see also Wren et al. 2020, which borrowed logic and model code from the *Ache Hunting* model).

We could also build a multilevel model like this using NetLogo's LevelSpace extension (Hjorth, Head, and Wilensky 2015; Hjorth et al. 2020). LevelSpace allows you to make an overarching model that can call specific "child" models. For the model we just described, we could write two separate NetLogo code files: a "parent" model that operates at the daily camp level, and a "child" model that loops through the hunting part of the model instead of the while loop. The parent model initiates one or multiple child models using `ls:create-interactive-models`. It can ask the child models to report values using `ls:of`, as in `[count turtles] ls:of child-id-number` or even `ls:ask` the child model do to things, much like the observer can ask agents for things in a single-level model. See

The LevelSpace extension is demonstrated in the NetLogo models library under `Code Examples -> Extensions Examples -> ls`.

the LevelSpace manual for more details on these primitives and the NetLogo Models Library for worked examples.[4]

4.5 Population Dispersal

Large-scale dispersals encapsulate movements that are the most visible in the archaeological record: the first out-of-Africa dispersal, the peopling of the Americas, or the spread of Neolithic farming across Europe. Although, like all dispersal events, large-scale movement is the result of an aggregation of individual behaviors and decisions, at such a large scale the population is considered largely homogeneous. We become less interested in modeling the movements and reproduction choices of individual people or households, and instead model large-scale average behaviors as probabilities. Here, an agent will represent a group rather than an individual, and one time step will represent a generation rather than a second, day, or season.

THE BASIC MODEL: FISHER–SKELLAM

The Fisher–Skellam–KPP equation was first developed in chemistry to model the spread of substances.

In a review paper on migration in archaeology, Anthony (1990) describes a number of different ways to conceptualize long-distance migrations. However, the most popular way to model large-scale human movement remains the Fisher–Skellam–KPP equation and its various extensions (Fort 2015; Steele 2009).

Fort (2015) gives a more accurate version of the equations that more fully account for human population dynamics.

If the name rings a bell, this is because you implemented it in your very first simulation in chapter 1. The Fisher–Skellam–KPP equation is commonly used across disciplines to describe the process of spatial spread. The equation describes two processes: population growth fueling dispersal, and diffusion from more to less densely occupied regions. The direction of that diffusion is often additionally shaped by exogenous factors such as topography and/or biotic zones (see equation).

See Steele (2009) for discussion on derivations of the basic Fisher–Skellam–KPP model, including effects of spatial and temporal heterogeneity, multiple population interactions, and evolutionary dynamics.

$$\frac{\delta n}{\delta t} = \alpha n \left(1 - \frac{n}{k}\right) + D\nabla^2 n \tag{4.0}$$

The change in population over time ($\frac{\delta n}{\delta t}$) is determined by the size of the population, n, and the growth rate, α, constrained by the carrying capacity, k. This change occurs along a gradient from more to less densely

[4] https://ccl.northwestern.edu/netlogo/docs/ls.html

populated regions, $D\nabla^2 n$, with D representing the distance moved with each generation.

Using sustained population growth to drive population expansion means that spread is inevitable. That is, it is assumed that the population will spread unless the population growth is negative or static. What may change is the speed of the dispersal and the shape of the dispersal front when it comes across difficult or impassable areas such as mountain chains or bodies of water. This basic algorithm was applied by Romanowska et al. (2017) , in which the authors modeled the first out-of-Africa dispersal and the relationship between the demographics at the front of the spreading population and patterns in the archaeological record. Similarly, Fort, Pujol, and Linden (2012) used this equation to model the spread of Neolithic farming and estimate its rate of dispersal in different regions in Europe.

Model: *Out-of-Africa Dispersal* by Romanowska et al.

Model: *Neolithic Spread* by Fort et al.

PROBABILISTIC CELLULAR AUTOMATA ABM-STYLE

A cellular automaton (CA) is another type of simulation in which individual cells impact other cells in specific ways. Cellular automata use cells' variable states (e.g., occupied vs. empty) so that the emergent behavior occurs through cells probabilistically changing their state because of the state of other cells, usually their neighbors. This is how John Conway's famous *Game of Life* model works (Wolfram 2002). A well-known early simulation of hominin dispersal, the *Stepping Out* model, used this cellular automata approach (Hughes et al. 2007; Mithen and Reed 2002).

TIP
Think of cellular automata as a world of just patches, with no turtles.

Model: *Stepping Out* by Mithen and Reed

ABMA Code Repo: *ch4_steppingOut*

We only give you part of the code here, so you have to work out the missing parts. Head to the ABMA Code Repo for the full code.

In their simulation, a patch marked as occupied by a hominin group had a certain probability of colonizing a random neighboring patch (P_{cr}) or of going extinct (P_{ext}). Subsequently, Hughes et al. (2007) modified *Stepping Out* to vary the P_{ext} value depending on the vegetation class (e.g., temperate forest vs. desert). We will replicate *Stepping Out* here as an agent-based model instead of true cellular automata, meaning we will make agents that move from patch to patch instead of having the patches affect each other directly.

Open NetLogo and create a new model with a world size of 360×180 patches. In the `setup` below, we first import a PNG map[5] as in chapter 1. We then must carefully assign P_{ext} values using the `pcolor` of the imported image so that some vegetation classes, like boreal forest and tundra, are excluded ($P_{ext} = 1$); the desert is a bit harsh ($P_{ext} = 0.07$), but the rest are easy ($P_{ext} = 0.01$). In the original paper, the authors experimented with making the coasts more favorable as well ($P_{ext} = 0.01$ for patches within 4 cells of ocean), though we will ignore that for now. We have listed each biome as a note in the code block below. We will also assign a new color scheme as we go to make it easier to refer to them (fig. 4.9).

We will reuse the code you wrote back in chapter 1, so have those exercises at hand.

CODE BLOCK 4.14

Here, we worked out what vegetation class each of the imported `pcolors` represented by comparing it to the figures in Hughes et al. (2007) and its data supplement.

```
...
  import-pcolors "ch4_veg.png"

  ask patches [
    ifelse pcolor = white [
      set p_ext 0
    ]
    [
      ; 1) tropical forest = 64.1
      ; 2) warm-temperate forest = 54.9
      ; 3) temperate forest = 43.9
      ; 4) boreal forest = 44.7
      ; 5) savanna and dry woodland = 44.4
      ; 6) grassland and dry shrubland = 27
      ; 7) desert = 27.5
      ; 8) tundra = 18.6
      ; 9) land ice = 9.4
```

[5] We created this PNG using layer 8 of the data supplied as a supplement with Hughes et al. (2007). It can be found here: https://archaeologydataservice.ac.uk/archives/view/valdes_nerc_2006. See the ABMA Code Repo for the PNG map and the R script to extract other time slices from this dataset and to export PNGs for them. Thanks to Matt Harris for help with the R code.

CODE BLOCK 4.14 (cont.)

```
      if pcolor = 44.7 or pcolor = 18.6 or pcolor =
      9.4 [
        set p_ext 1
        set pcolor brown
      ]

      if pcolor = 27.5 [
        set p_ext 0.07
        set pcolor yellow
      ]

      if pcolor != brown and pcolor != yellow [
        set p_ext 0.01
        set pcolor green
      ]
    ]
  ]
...
```

Create a few turtles in `setup` and set their coordinates to eastern Africa (inspect a patch to get the `pxcor pycor`). Create a slider in the INTERFACE tab for `p_cr` (which is the same as the old `pop_growth` parameter), and set it to *Stepping Out*'s default value of 0.04. The rest of the code below is slightly modified from chapter 1, but you should see that the basic structure is still the same.

DON'T FORGET
If you get a warning that `p_ext` is not defined, add `patches-own [p_ext]` at the beginning of the code.

CODE BLOCK 4.15

```
to go
  ask turtles [
    if random-float 1 <= p_ext
        [die]
    if random-float 1 <= p_cr
        [reproduce]
  ]
  tick
end
```

CODE BLOCK 4.15 (cont.)

```
to reproduce
  if any? neighbors with [count turtles-here = 0
    AND pcolor != white] [
    let empty-patch one-of neighbors with
    [count turtles-here = 0 AND pcolor != white]
    hatch 1 [
      set color color + 0.05
      move-to empty-patch
    ]
  ]
end
```

Like the original *SteppingOut* model, the probability of random colonization (`p_cr`) does not vary with vegetation class, so we have used an INTERFACE slider (i.e., a global variable). Probability of extinction (`p_ext`) does vary by vegetation class, so we use a patch variable. Note that the logic of this is the same as the Fisher–Skellam–KPP equation-based models.

When `min-one-of` finds the same minimum value on multiple neighboring patches, it chooses randomly among them.

You should also notice that we are combining some of the algorithms from section 4.3 above. The hatched turtles random walk to a select set of targeted patches (i.e., empty and land). To modify this to a fully targeted walk that assumes agents are aware of the landscape's levels of extinction risk, change `one-of` to `min-one-of` and add `[p_ext]` to the end of that line. Since the map is in large blocks of vegetation, this will probably have very little effect on the model except to direct the expanding wave front very subtly (fig. 4.9).

Model: *SteppingIn* by Scherjon

Scherjon (2013) uses a similar approach to model the expansion of *Homo sapiens* into Europe by various routes. In this model, called *SteppingIn*, the agents may consider several different criteria, including proximity to rivers, lakes, and coastlines, whether the patch is occupied, and whether a patch is along the same heading the agent is already traveling. The various criteria are then summed to determine which patch, within an iteratively increasing perception radius, has the highest value. We should note, though, that whenever you use multiple criteria, you

The process of establishing the impact of each parameter on the final results is called a **sensitivity analysis** (see ch. 9).

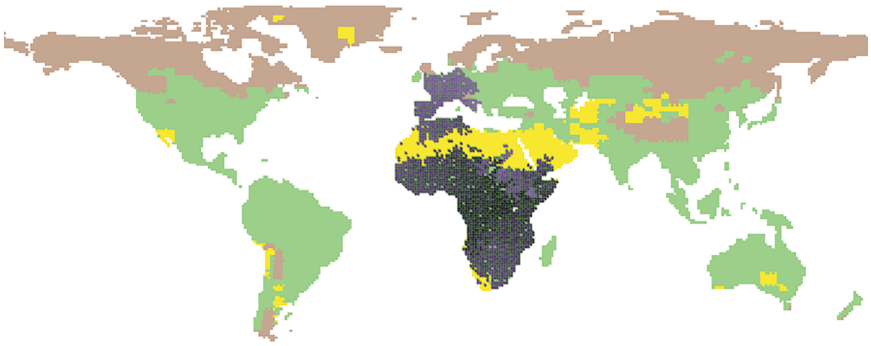


Figure 4.9. Replication of *Stepping Out* using an imported global patch landscape. Agents spread out of eastern Africa into surrounding regions.

also need to decide how they are weighted against each other. Is proximity to rivers as important as being along the agent's heading? If not, should the river patches have twice the value? Ten times? Each coding decision should be carefully justified and even more thoroughly investigated for its effects.

In another hominin-dispersal model, Wren et al. (2014) use a smoothly varying landscape to represent small changes in habitat quality rather than large blocks of specific habitat classes (see ch. 7 for more details on toy landscapes). The authors modeled a range of scenarios where the probability that agents will either choose the best-quality habitat in the local area or move randomly represents accurate perception of the environment or an error in their knowledge. In a continuously varying landscape with small ups and downs (also known as noise), an agent with no probability of error does not move very far as they get stuck on the ups in the noise (i.e., local optima) rather than finding the best part of the landscape (i.e., global optimum). However, the optimal degree of error varies with the spatial characteristics of the landscape and may substantially increase or decrease the speed of a dispersal. In figure 4.10, the agents begin in the top-right corner but split into two populations as they move toward a more suitable habitat (lighter cells).

Model: *Spatial Foresight* by Wren et al.

ABMA Code Repo: *ch4_population_movement*

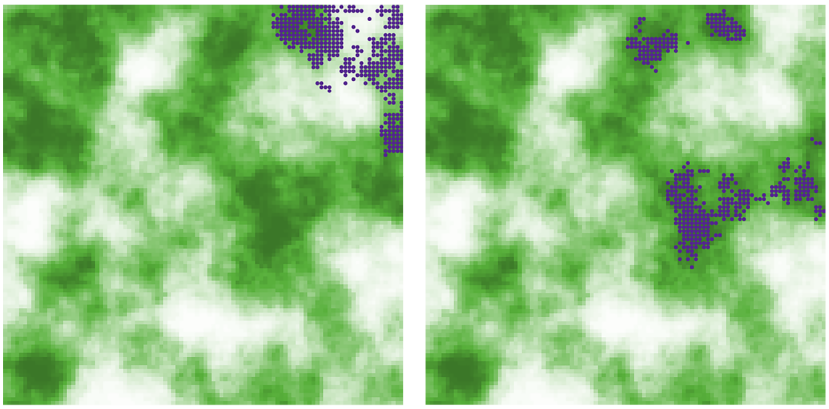


Figure 4.10. Two screenshots from the *Spatial Foresight* model at time steps 50 and 150. Agents use a version of a weighted-random-walk to navigate a heterogeneous resource landscape. They diverge into two groups around a low-resource area (lighter shade).

CODE BLOCK 4.16

```
ifelse random-float 1 < spatial-foresight [
  let p max-one-of patches in-radius
    1.5 with [not any? turtles-here]
    [habitat_quality]
  if p != nobody [move-to p]
][
  let p one-of patches in-radius
  1.5 with [not any? turtles-here]
  if p != nobody [move-to p]
]
```

This code snippet adapted from the Wren et al. (2014) model could be integrated with our *SteppingOut* replication using an INTERFACE slider (0 to 1) for `spatial_foresight`. We could insert it either as a replacement movement line in the `reproduce` procedure or as an additional step in the `go` procedure. In this case, the model's map has to be set to some kind of continuously varying `habitat_quality` variable, like net primary productivity or effective temperature, rather than vegetation classes used in our implementation (code block 4.14).

LEAP-FROGGING & DESPOTS

A notable exception to the standard practice of applying the Fisher–Skellam–KPP model, and its various derivations, to large-scale dispersal can be found in the *MedLanD* research group's model. The authors compare alternative models to represent Neolithic dispersal, and in particular the Cardial wave, which is thought to have spread along the Mediterranean coast and into Iberia following a different trajectory than the land-based dispersal north of the Alps (Bernabeu Aubán et al. 2015; Pardo-Gordó et al. 2017).

Model: *Cardial Spread* by Bergin et al.

ABMA Code Repo: *ch4_population_movement*

The model compares four approaches: simple spreading to neighbors (Fisher–Skellam–KPP), spreading to a subset of suitable neighbors (a targeted walk), leap-frog, and ideal-despotic spreading. In a leap-frog model, agents move farther per step, often well into a previously unoccupied area. Bernabeu Aubán et al. (2015) tested leap distances from 5 to 250 km per move. The modelers implemented this by randomizing the step distance as well as the direction. The version we reference here was implemented in NetLogo by Bergin (2016):

CODE BLOCK 4.17

```
to leap-frog
  let leap-distance 10
  ask turtles [
    let dir random 360
    let dist random leap-distance
    let target patch-at-heading-and-distance
    dir dist
    if target != nobody [
      move-to target
    ]
  ]
end
```

You may notice that this resembles a Lévy flight. However, here the distribution of distance lengths is uniform, that is, all distances are equally likely. In a Lévy flight the probability of a distance length decreases as the distance increases; in other words, most moves will be of a shorter distance but very occasionally a large jump happens.

See code block 4.2 for the code for a Lévy flight.

The authors also experimented with an ideal-despotic distribution spread model whereby the spreading farmers decide where to move based on the location of the best potential farmland, deducting a cost per farm that already occupies that cell. This creates a pattern where the first agent in a highly valued cell, the "despot," has an advantage over any other agents joining it, thereby providing an incentive for others to move even farther (fig. 4.11).

CODE BLOCK 4.18

```
to ideal-despotic
  let leap-distance 10
  let occupied-cost 0.05
  ask turtles [
    let target max-one-of patches in-radius
    leap-distance [soil-quality]
    move-to target
    set soil-quality (soil-quality * (1 -
    occupied-cost))
    set pcolor scale-color green soil-quality 0 100
  ]
end
```

4.6 Validation

Having seen so many different ways of coding even the simplest of processes, such as the random-walk, you may start to wonder: Can I be sure that my way of representing the past is correct?

Let's pause for a minute and consider whether it is ever possible to establish that a model is definitely *true*. "Surely, if my model produces similar data patterns to the real system, I have managed to capture its true dynamics?" you may ask. Right? We already discussed (ch. 3) that this is not actually the case because of **equifinality**.

Equifinality means that the model's fit with the data is never sufficient evidence for model truthfulness; it is possible, and in fact rather likely, that another model will eventually be devised that fits the data even better. The main caveat of this is that equifinality does not only affect computational models such as archaeological ABMs. In fact, it affects all types of models—including theories, hypotheses, explanations, statistical models, and spatial

Figure 4.11. A large population of agents spreads from the bottom left using an ideal-despotic algorithm. Some agents leap ahead of the dispersal wave to gain access to unoccupied patches.

GIS models—and all disciplines of science. As such, it is known as the *underdetermination of scientific theory* (Stanford 2017).

Although there is no way to ensure that your model is true, there are many ways to check that it is *possible* and *plausible*. In fact, you can quantify how well it performs relative to other models. The process of ensuring that your model is an appropriate representation of the real-world system is called **validation**.

Validation is such an important part of the modeling process it constitutes a whole subdiscipline within modeling studies.

In short, validation is a process of comparing the model predictions to the data to establish whether the model is *one* correct representation of the system that produced the data. The three most common validation techniques used in archaeological computational modeling are data validation, stylized facts, and model selection.

DATA VALIDATION

The simplest way to check the validity of a model is to compare its prediction with available data. If the model produces results that are in line

with patterns in the data, then we can state that the model is a plausible representation of the real-world system. There are no hard-and-fast rules regarding how similar these two sets of data need to be to establish equivalence between them, since it often depends on the quality of the actual data. In archaeology, we are acutely aware of the many biases that our data have undergone before ending up in the file on our computer: Was the pottery sample collected representative of the local pottery use? Are we missing any time periods due to erosion? Was the excavation systematic? Was the lithic analyst tired that day?

It may surprise new modelers to learn that once the code is finished, you're only about halfway through the project. Data analysis takes a long time.

Because of these caveats, capturing general trends is usually considered sufficient. However, spending some time to examine your model outputs against the data may help you find where the patterns match, where they deviate, and what the reason behind it might be. At this point, it often makes sense to perform model selection—that is, introduce changes to the model or even build an entirely new simulation from scratch, and then compare how well it matches the data relative to the original.

Archaeologists commonly use data to validate simulations. For example, the *SteppingOut* model used the timings of hominin presence at six Lower Paleolithic sites with well-established chronologies. Simulation runs that matched these dates best—that is, those where agents arrived at the localities closest to the dates—are deemed the most likely to best represent the out-of-Africa dynamics. The number of validation points can easily be scaled up. Scherjon (2019) in his *HominінSpace* model employed a large dataset of archaeological sites related to the Neanderthal occupation of Western Europe. He used an artificial intelligence technique called a genetic algorithm to search through the parameter space for scenarios that best fit the data.

TIP
If the level of uncertainty of data points has been recorded, you can create a confidence map (see, e.g., Holmes 2007).

Model: *HominінSpace* by Scherjon

Data validation is the most common method of validating simulations across all disciplines. However, it comes with its own suite of problems and research decisions, the most fundamental of which is: To which data should the simulation be compared (Ahrweiler and Gilbert 2005)? In most cases there are multiple types of data pertinent to the studied phenomena. For example, if we study the spread of the Neolithic in Western Central Asia, should we look at the changing distribution of a particular pottery type or perhaps follow the trails of animal remains showing signs of domestication? Does a simulation that matches the exact locations of known sites fit "bet-

ter" than one that fails to predict the geographical coordinates but matches the distribution of sites in regard to the underlying soil types?

One can question the validity of a model if it matches one particular data pattern. The researcher might have tinkered with the model, adding new parameters and slightly adjusting algorithms so that in the end it does match (so-called **overfitting**), or the data pattern is quite weak and easy to mimic, or the match is sheer coincidence. We learn relatively little from these kinds of models. However, if two or more independent data patterns are matched, the probability of this happening by chance or by unintended design bias becomes vanishingly small.

overfitting: the model is overfitted when it fits the data too closely, thus reflecting noise rather than significant trends.

Remember the validation against two independent data patterns in the *Artificial Anasazi* model in chapter 3?

STYLIZED FACTS

Sometimes there are not enough data points to truly validate a model against archaeological data, or the model's goal is to elucidate a particular phenomenon usually described qualitatively. In these situations, we talk about matching the model to **stylized facts** (Troitzsch 2004). Originating in economics, the term describes broad, generalized patterns in data that, although true in general, may not be entirely correct on a more detailed scale or in particular cases.

stylized facts: broad, generalized patterns in data that, although true in general, may not be entirely correct in detail.

We came across one such stylized fact in chapter 1: the "delayed" arrival of humans in Europe compared to East Asia during the out-of-Africa dispersal. Although the archaeological data provide some dates of hominins in Eastern Asia long before evidence suggests they were in Western Europe, matching them would not be the goal of a generalized model examining under what conditions Eastern Asia would be colonized prior to Europe. Instead, the modeler looks at the broad simulation outcomes to determine which parameters lead agents to arrive in East Asia before Western Europe. We may say in such cases that "it is only under conditions X and Y that hominins arrive in Asia first, supporting hypothesis Z." While we have neither a hard-and-fast rule nor a particular data distribution to which we can compare our model, it may be considered a plausible explanation.

For example, Romanowska (2015a) showed that the distribution of the earliest Lower Paleolithic sites is independent of any specific route leading into Europe (e.g., via Gibraltar or the Balkans). The agents reached Western Europe at a similar rate regardless of which route they took, leading to the conclusions that the current archaeological record may not have preserved

Model: *SHEEP* by Romanowska

information regarding the entry point into Europe of the first hominins out of Africa.

MODEL SELECTION

Model selection is an idea borrowed from statistics whereby one looks to an assemblage of potential models and identifies the one that best fits the data. Often, there are several candidate mechanisms that impact the behavior of the model in different ways, resulting in different output.

The *MedLanD* model is a great example of model selection.

Then, as with data validation, you can compare the models to the archaeological data. This can be either by switching from one scenario to another (*scenario testing*) or by varying parameters smoothly (*parameter sweep*) (see ch. 9). It is important to develop a meaningful measure of the *closeness* between the model output (artificial data) and the archaeological record (real data). That distance then determines which scenario is most plausible and provides a relative measure of their plausibility.

Statistical techniques such as Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), and the more computationally demanding Approximate Bayesian Computation (ABC) can be used to attach a level of likelihood to competing hypotheses (Marin and Robert 2014; Carrignon, Brughmans, and Romanowska 2020). The likelihood is understood here as a measure of the model's fit to the data. The model producing outputs resembling the archaeological record most closely indicates that the hypothesis it represents is more likely than alternatives. Similarly, machine-learning techniques can be used to trawl through the large space of parameter values looking for a set that matched the data best. Thus, model selection enables us to determine whether one candidate hypothesis better fits the data than others, and provides a measure of how much better it explains the data. There are no strict boundaries among these three types of validation, and often multiple techniques are used together.

TIP
You're likely to require access to a high-performance computing environment, such as a supercomputer, to run ABC.

4.7 Summary

To begin coding movement into a model, you first need to establish the spatial and temporal scale of the human activity you are trying to understand. Do you need to model individual-, group-, or population-level movements? Once that is established, you should decide how much knowledge your agents will have—of their environment and of other agents—and what fac-

tors affect their mobility and in which ways. Throughout this chapter, we have highlighted several published models that have taken subtly different approaches to modeling movement due to these aspects of scale, knowledge, and social dynamics.

A large part of the justification process is the validation of the model's results against some form of empirical data. In archaeology, we are used to dealing with uncertainty. This does not mean having your ABM spit out a multilevel stratigraphic record of tool-type frequencies that precisely matches your excavation report. Rather, we look to see if the model output achieves the following:

- is consistent with the *general trends* observed in the archaeological record.
- matches known *stylized facts* about the time period.
- is a *better fit* than other competing models by you or by others.

At this point you have begun to understand that coding a model is not just about learning how to code, but also about understanding the implications of your coding choices. Implementing a specific algorithm in your model depends not only on your coding skill level but also on the research questions you are asking of your model. It is important to remember that using a formal or computational method does not protect you from poor scholarship or incorrect scientific practice. After working through the algorithm examples in this chapter, you should now be equipped to make informed algorithmic choices in your own models—and to understand the implications of those choices.

In the next chapter, we will look more closely at how to make these modeling choices: deciding on the entities, rules of behavior, and scales for your model. We will consider parsimony and other general guidelines that ensure you do not overfit your model when validating it.

End-of-Chapter Exercises

1. Could you conceptualize a model that incorporates both individual- and group-level movement? Write some pseudocode of how that might work.

2. Reimplement the *Y&B* model from chapter 1 using different movement algorithms. Do they produce different rates of dispersal?

3. How could incorporating subsistence, as in chapter 3, impact movement? Manipulate your model to create need for calories, as in the *SugarScape* model, but with one of the movement algorithms here. How does combining algorithms impact your models?

4. Think of your area of research. What are the most robust data records that could be used to validate your model? What is the most persistent type of archaeological data in your area of study? Would you need to look at long-term patterns or local site- or region-specific changes? Could you incorporate nonsystematic data, for example, information from written sources?

Movement Model Zoo

▷ **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

▷ **Cardial Spread**

S. M. Bergin. 2016. "Mechanisms and Models of Agropastoral Spread During the Neolithic in the West Mediterranean: The Cardial Spread Model." PhD, Arizona State University. https://search.proquest.com/docview/1845308964/abstract/D9BE13322D14FB5PQ/1

Code: https://www.comses.net/codebases/4447

▷ **Correlated Random-Walk**

C. S. Patlak. 1953. "Random Walk with Persistence and External Bias." *The Bulletin of Mathematical Biophysics* 15, no. 3 (September): 311–338. doi:10.1007/BF02476407

▷ **Flocking / Boids**

C. W. Reynolds. 1987. "Flocks, Herds, and Schools: A Distributed Be-

havioral Model." In *SIGGRAPH '87 Conference Proceedings,* 21:25–34. 4. Anaheim, CA: Computer Graphics. doi:10.1145/37402.37406

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

▷ **Foraging Memory Capacity**
A. Costopoulos. 2001. "Evaluating the Impact of Increasing Memory on Agent Behaviour: Adaptive Patterns in an Agent-Based Simulation of Subsistence." *Journal of Artificial Societies and Social Simulation* 4, no. 4 (October). https://jasss.soc.surrey.ac.uk/4/4/7.html

▷ **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/

▷ **HominінSpace**
F. Scherjon. 2019. "Virtual Neanderthals: A Study in Agent-Based Modelling Late Pleistocene Hominins in Western Europe." PhD diss. https://openaccess.leidenuniv.nl/handle/1887/73639

Code: https://www.comses.net/codebases/5294/

▷ **Lévy Flights**
D. O'Sullivan and G. L. W. Perry. 2013. *Spatial Simulation: Exploring Pattern and Process.* Chichester, UK: John Wiley & Sons, September. https://patternandprocess.org/

Code: https://patternandprocess.org/category/chapter-4/

▷ **Multi-Agent Geographically Informed Computer AnaLysis (MAGICAL)**
M. W. Lake. 2000. "MAGICAL Computer Simulation of Mesolithic Foraging." In *Dynamics in Human and Primate Societies: Agent-Based Modeling of Social and Spatial Processes,* edited by T. A. Kohler and G. J. Gumerman, 107–143. Oxford, UK: Oxford University Press.

▷ **Neolithic Spread**

J. Fort, T. Pujol, and M. V. Linden. 2012. "Modelling the Neolithic Transition in the Near East and Europe." *American Antiquity* 77 (2): 203–219. doi:10.7183/0002-7316.77.2.203

▷ **Out-of-Africa Dispersal**

I. Romanowska et al. 2017. "Dispersal and the Movius Line: Testing the Effect of Dispersal on Population Density through Simulation." *Quaternary International* 431 (Part B): 53–63. doi:10.1016/j.quaint.2016.01.016

Code:
https://github.com/izaromanowska/Modelling-Hominin-Dispersals-Using-Agent-based-Modelling

▷ **PaleoscapeABM**

C. D. Wren et al. 2018. "An Agent-Based Approach to Weighted Decision Making in the Spatially and Temporally Variable South African Palaeoscape." In *Proceedings of the 44th Computer Applications and Quantitative Methods in Archaeology Conference (CAA 2016),* 507–522. Oslo, Norway: Archeopress

Code: https://doi.org/10.25937/r2qq-fn02

▷ **Procession Movement:**

K. A. Crawford. 2019a. "Rethinking Approaches for the Study of Urban Movement at Ostia." In *Finding the Limits of the Limes: Modelling Demography, Economy and Transport on the Edge of the Roman Empire,* edited by P. Verhagen, J. Joyce, and M. R. Groenhuijzen, 313–327. Cham, Switzerland: Springer International Publishing. doi:10.1007/978-3-030-04576-0_15

https://katherineacrawford.com/index.php/agent-based-modelling/

▷ **Random-Walk**

K. Pearson. 1905. "The Problem of the Random Walk." *Nature* 72, no. 1867 (August): 342–342. doi:10.1038/072342a0

▷ **Simulating Hominin Expansion in the Early Pleistocene (SHEEP)**

I. Romanowska. 2015a. "Agent-Based Modelling and Archaeological

Hypothesis Testing: The Case Study of the European Lower Palaeolithic." In *Across Space and Time. Papers from the 41st Conference on Computer Applications and Quantitative Methods in Archaeology, Perth, 25–28 March 2013,* 203–214. Amsterdam, Netherlands: Amsterdam University Press.

Code: https://github.com/izaromanowska/SHEEP

▷ **Spatial Foresight**

C. D. Wren et al. 2014. "The Role of Spatial Foresight in Models of Hominin Dispersal." *Journal of Human Evolution* 69:70–78. doi:10.1016/j.jhevol.2014.02.004

Code: https://www.comses.net/codebases/3846/

▷ **SteppingIn**

F. Scherjon. 2013. "SteppingIn—Modern Humans Moving into Europe—Implementation." In *Proceedings of the 40th Conference on Computer Applications and Quantitative Methods in Archaeology Southampton, 26–30 March 2012,* 105–117. Amsterdam, Netherlands: Amsterdam University Press.

▷ **Stepping Out**

S. J. Mithen and M. Reed. 2002. "Stepping Out: A Computer Simulation of Hominid Dispersal from Africa." *Journal of Human Evolution* 43 (4): 433–462. doi:10.1006/jhev.2002.0584

https://www.reading.ac.uk/steppingout/

▷ **Trail Evolution**

D. Helbing, J. Keltsch, and P. Molnar. 1997. "Modelling the Evolution of Human Trail Systems." *Nature* 388 (6637): 47–50. doi:10.1038/40353

▷ **Walk This Way**

A. Crooks et al. 2015. "Walk This Way: Improving Pedestrian Agent-Based Models through Scene Activity Analysis." *ISPRS International Journal of Geo-Information* 4, no. 3 (September): 1627–1656. doi:10.3390/ijgi4031627

Code, NetLogo:
https://github.com/abmgis/abmgis/tree/master/Chapter10-EvaluatingModels/Models/WalkThisWay

▷ **Weighted Random-Walk**

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

NOTES