Chapter 10

Agent-based models

10.1 Thomas Schelling

In 1971 Thomas Schelling published "Dynamic Models of Segregation," which proposes a simple model of racial segregation. The Schelling model of the world is a grid; each cell represents a house. The houses are occupied by two kinds of "agents," labeled red and blue, in roughly equal numbers. About 10% of the houses are empty.

At any point in time, an agent might be happy or unhappy, depending on the other agents in the neighborhood. The neighborhood of each house is the set of eight adjacent cells. In one version of the model, agents are happy if they have at least two neighbors like themselves, and unhappy if they have one or zero.

The simulation proceeds by choosing an agent at random and checking to see whether it is happy. If so, then nothing happens; if not, the agent chooses one of the unoccupied cells at random and moves.

You might not be surprised to hear that this model leads to some segregation, but you might be surprised by the degree. Fairly quickly, clusters of similar agents appear. The clusters grow and coalesce over time until there are a small number of large clusters and most agents live in homogeneous neighborhoods.

If you did not know the process and only saw the result, you might assume that the agents were racist, but in fact all of them would be perfectly happy in a mixed neighborhood. Since they prefer not to be greatly outnumbered, they might be considered xenophobic at worst. Of course, these agents are a wild simplification of real people, so it may not be appropriate to apply these descriptions at all.

Racism is a complex human problem; it is hard to imagine that such a simple model could shed light on it. But in fact it provides a strong argument about the relationship between a system and its parts: if you observe segregation in a real city, you cannot conclude that individual racism is the immediate cause, or even that the people in the city are racists.

But we have to keep in mind the limitations of this argument: Schelling's model demonstrates a possible cause of segregation, but says nothing about actual causes.

Exercise 10.1. *Implement Schelling's model in a grid. From random initial conditions, how does the system evolve?*

Define a statistic that measures the degree of segregation, and plot this statistic over time.

Experiment with the parameters of the model. What happens as the agents become more tolerant? What happens if the agents are only happy in mixed neighborhoods; that is, if they are unhappy if too many of their neighbors are like themselves?

Exercise 10.2. *In a recent book,* The Big Sort*, Bill Bishop argues that American society is increasingly segregated by political opinion, as people choose to live among like-minded neighbors.*

The mechanism Bishop hypothesizes is not that people, like the agents in Schelling's model, are more likely to move if they are isolated, but that when they move for any reason, they are likely to choose a neighborhood with people like themselves.

Modify your implementation of Schelling's model to simulate this kind of behavior and see if it yields similar degrees of segregation.

10.2 Agent-based models

Schelling's model is one of the first, and one of the most famous, agent-based models. Since the 1970s, agent-based modeling has become an important tool in economics and other social sciences, and in some natural sciences.

The characteristics of agent-based models include:

- Agents that model intelligent behavior, usually with a simple set of rules.
- The agents are usually situated in space (or in a network), and interact with each other locally.
- They usually have imperfect, local information.
- Often there is variability between agents.
- Often there are random elements, either among the agents or in the world.

Agent-based models are useful for modeling the dynamics of systems that are not in equilibrium (although they are also used to study equilibrium). And they are particularly useful for understanding relationships between individual decisions and system behavior, or as in the title of Schelling's book, *Micromotives and Macrobehavior*.

For more about agent-based modeling, see `http://en.wikipedia.org/wiki/Agent-based_model`.

10.3 Traffic jams

What causes traffic jams? In some cases there is an obvious cause, like an accident, a speed trap, or something else that disturbs the flow of traffic. But other times traffic jams appear for no apparent reason.

Agent-based models can help explain spontaneous traffic jams. As an example, I implemented a simple highway simulation, based on a model in Resnick, *Turtles, Termites and Traffic Jams*.

You can download my program from `thinkcomplex.com/Highway.py`. It uses `TurtleWorld`, which is part of Swampy. See `thinkpython.com/swampy`.

This module defines two classes: `Highway`, which inherits from TurtleWorld, and `Driver`, which inherits from Turtle.

The Highway is a one-lane road that forms a circle, but it is displayed as a series of rows that spiral down the canvas.

Each driver starts with a random position and speed. At each time step, each Driver accelerates or brakes based on the distance between it and the Driver in front. Here is an example:

```
def choose_acceleration(self, dist):
    if dist < self.safe_distance:
        return -1
    else:
        return 0.3
```

If the following distance is too short, the `Driver` brakes; otherwise it accelerates. `Highway.py` enforces two other constraints: there is a speed limit for each driver, and if the current speed would cause a collision, the `Driver` comes to a complete stop.

If you run `Highway.py` you will probably see a traffic jam, and the natural question is, "Why?" There is nothing about the `Highway` or `Driver` behavior that obviously causes traffic jams.

Exercise 10.3. *Experiment with the parameters of the system to identify the factors that are necessary and sufficient to cause traffic jams. Some of the factors to explore are:*

Density: *What happens as the number of drivers increases (or the length of the highway decreases)?*

Acceleration and braking: *What happens if drivers accelerate faster or brake more gently?*

Safe distance: *What happens as the safe distance between drivers changes?*

Heterogeneity: *What if all drivers are not the same; for example, if they have different speed limits or following distances?*

10.4 Boids

In 1987 Craig Reynolds published "Flocks, herds and schools: A distributed behavioral model," which describes an agent-based model of herd behavior. You can download his paper from `http://www.red3d.com/cwr/papers/1987/boids.html`.

Agents in this models are called "boids," which is both a contraction of "bird-oid" and an accented pronunciation of "bird" (although boids are also used to model fish and herding land animals).

Each agent simulates three behaviors:

Collision avoidance: avoid obstacles, including other birds.

Flock centering: move toward the center of the flock.

Velocity matching: align velocity with neighboring birds.

Boids make decisions based on local information only; each boid only sees (or pays attention to) other boids in its field of vision and range.

The `Visual` package, also known as VPython, is well-suited for implementing boids. It provides simple 3-D graphics as well as vector objects and operations that are useful for the computations.

You can download my implementation from `thinkcomplex.com/Boids.py`. It is based in part on the description of boids in Flake, *The Computational Beauty of Nature*.

The program defines two classes: `Boid`, which implements the boid algorithm, and `World`, which contains a list of Boids and a "carrot" the Boids are attracted to.

The boid algorithm uses `get_neighbors` to find other boids in the field of view:

```
    def get_neighbors(self, others, radius, angle):
        boids = []
        for other in others:
            if other is self:
               continue

            offset = other.pos - self.pos

            # if not in range, skip it
            if offset.mag > radius:
                continue

            # if not within viewing angle, skip it
            if self.vel.diff_angle(offset) > angle:
                continue

            # otherwise add it to the list
            boids.append(other)

        return boids
```

`get_neighbors` uses vector subtraction to compute the vector from `self` to `other`. The magnitude of this vector is the distance to the other boid. `diff_angle` computes the angle between the velocity of `self`, which is also the line of sight, and the other boid.

`center` finds the center of mass of the boids in the field of view and returns a vector pointing toward it:

```
    def center(self, others):
        close = self.get_neighbors(others, r_center, a_center)
        t = [other.pos for other in close]
        if t:
            center = sum(t)/len(t)
            toward = vector(center - self.pos)
            return limit_vector(toward)
        else:
            return null_vector
```

Similarly, `avoid` finds the center of mass of any obstacles in range and returns a vector pointing away from it, `copy` returns the difference between the current heading and the average heading of the neighbors, and `love` computes the heading toward the carrot.

`set_goal` computes the weighed sum of these goals and sets the overall goal:

```
    def set_goal(self, boids, carrot):
        self.goal = (w_avoid * self.avoid(boids, carrot) +
                     w_center * self.center(boids) +
                     w_copy * self.copy(boids) +
                     w_love * self.love(carrot))
```

Finally, `move` updates the velocity, position and attitude of the boid:

```
    def move(self, mu=0.1):
        self.vel = (1-mu) * self.vel + mu * self.goal
        self.vel.mag = 1

        self.pos += dt * self.vel
        self.axis = b_length * self.vel.norm()
```

The new velocity is the weighted sum of the old velocity and the goal. The parameter `mu` determines how quickly the birds can change speed and direction. The time step, `dt` determines how far the boids move.

Many parameters influence flock behavior, including the range, angle and weight for each behavior, and the maneuverability, `mu`.

These parameters determine the ability of the boids to form and maintain a flock and the patterns of motion and organization in the flock. For some settings, the boids resemble a flock of birds; other settings resemble a school of fish or a cloud flying insects.

Exercise 10.4. *Run my implementation of the boid algorithm and experiment with different parameters. What happens if you "turn off" one of the behaviors by setting the weight to 0?*

To generate more bird-like behavior, Flake suggests adding a fourth behavior to maintain a clear line of sight; in other words, if there is another bird directly ahead, the boid should move away laterally. What effect do you expect this rule to have on the behavior of the flock? Implement it and see.

10.5 Prisoner's Dilemma

The Prisoner's Dilemma is a topic of study in game theory, but it's not the fun kind of game. Instead, it is the kind of game that sheds light on human motivation and behavior.

Here is the presentation of the dilemma from `http://en.wikipedia.org/wiki/Prisoner's_dilemma`.

> Two suspects [Alice and Bob] are arrested by the police. The police have insufficient evidence for a conviction, and, having separated the prisoners, visit each of them to offer the same deal. If one testifies against the other (defects) and the other remains silent (cooperates), the defector goes free and the silent accomplice receives the full one-year sentence. If both remain silent, both prisoners are sentenced to only one month in jail for a minor charge. If each betrays the other, each receives a three-month sentence. Each prisoner must choose to betray the other or to remain silent. Each one is assured that the other would not know about the betrayal before the end of the investigation. How should the prisoners act?

Notice that in this context, "cooperate" means to keep silent, not to cooperate with police.

It is tempting to say that the players should cooperate with each other, since they would both be better off. But neither player knows what the other will do. Looking at it from Bob's point of view:

- If Alice remains silent, Bob is better off defecting.
- If Alice defects, Bob is better off defecting.

Either way, Bob is better off defecting. And from her point of view, Alice reaches the same conclusion. So if both players do the math, and no other factors come into play, we expect them to defect and be worse off for it.

This result is saddening because it is an example of how good intentions can lead to bad outcomes, and, unfortunately, it applies to other scenarios in real life, not just hypothetical prisoners.

But in real scenarios, the game is often iterated; that is, the same players face each other over and over, so they have the opportunity to learn, react, and communicate, at least implicitly.

The iterated version of the game is not as easy to analyze; it is not obvious what the optimal strategy is or even whether one exists.

So in the late 1970s Robert Axelrod organized a tournament to compare strategies. He invited participants to submit strategies in the form of computer programs, then played the programs against each other and kept score.

I won't tell you the outcome, and if you don't know you should resist the temptation to look it up. Instead, I encourage you to run your own tournament. I'll provide the referee; you provide the players.

Exercise 10.5. *Download `thinkcomplex.com/Referee.py`, which runs the tournament, and `thinkcomplex.com/PlayerFlipper.py`, which implements a simple player strategy.*

Here is the code from `PlayerFlipper.py`*:*

```
def move(history):
    mine, theirs = history
    if len(mine) % 2 == 0:
        return 'C'
    else:
        return 'D'
```

Any file that matches the pattern `Player*.py` *is recognized as a player. The file should contain a definition for* `move`*, which takes the history of the match so far and returns a string:* `'D'` *for defect and* `'C'` *for cooperate.*

`history` *is a pair of lists: the first list contains the player's previous responses in order; the second contains the opponent's responses.*

`PlayerFlipper` *checks whether the number of previous rounds is even or odd and returns* `'C'` *or* `'D'` *respectively.*

Write a `move` *function in a file like* `PlayerFlipper.py`*, but replace "Flipper" with a name that summarizes your strategy.*

Run `Referee.py` *and see how your strategy does.*

After you run your own tournament, you can read about the results of Axelrod's tournament in his book, The Evolution of Cooperation.

10.6 Emergence

The examples in this chapter have something in common: emergence. An **emergent property** is a characteristic of a system that results from the interaction of its components, not from their properties.

To clarify what emergence is, it helps to consider what it isn't. For example, a brick wall is hard because bricks and mortar are hard, so that's not an emergent property. As another example, some rigid structures are built from flexible components, so that seems like a kind of emergence. But it is at best a weak kind, because structural properties follow from well-understood laws of mechanics.

Emergent properties are surprising: it is hard to predict the behavior of the system even if we know all the rules. That difficulty is not an accident; it may be the defining characteristic of emergence.

As Wolfram discusses in *A New Kind of Science*, conventional science is based on the axiom that if you know the rules that govern a system, you can predict its behavior. What we call "laws" are often computational shortcuts that allow us to predict the outcome of a system without building or observing it.

But many cellular automata are **computationally irreducible**, which means that there are no shortcuts. The only way to get the outcome is to implement the system.

The same may be true of complex systems in general. For physical systems with more than a few components, there is usually no model that yields an analytic solution. Numerical

methods provide a kind of computational shortcut, but there is still a qualitative difference. Analytic solutions often provide a constant-time algorithm for prediction; that is, the run time of the computation does not depend on t, the time scale of prediction. But numerical methods, simulation, analog computation, and similar methods take time proportional to t. And for many systems, there is a bound on t beyond which we can't compute reliable predictions at all.

These observations suggest that emergent properties are fundamentally unpredictable, and that for complex systems we should not expect to find natural laws in the form of computational shortcuts.

To some people, "emergence" is another name for ignorance; by this reckoning, a property is emergent if we don't have a reductionist explanation for it, but if we come to understand it better in the future, it would no longer be emergent.

The status of emergent properties is a topic of debate, so it is appropriate to be skeptical. When we see an apparently emergent property, we should not assume that there can never be a reductionist explanation. But neither should we assume that there has to be one. The examples in this book and the principle of computational equivalence give good reasons to believe that at least some emergent properties can never be "explained" by a classical reductionist model.

You can read more about emergence at `http://en.wikipedia.org/wiki/Emergence`.

10.7 Free will

Many complex systems have properties, as a whole, that their components do not:

- The Rule 30 cellular automaton is deterministic, and the rules that govern its evolution are completely known. Nevertheless, it generates a sequence that is statistically indistiguishable from random.
- The agents in Schelling's model are not racist, but the outcome of their interactions is as if they were.
- Traffic jams move backward even though the cars in them are moving forward.
- The behavior of flocks and herds emerges from local interactions between their members.
- As Axelrod says about the iterated prisoner's dilemma: "The emergence of cooperation can be explained as a consequence of individual[s] pursuing their own interests."

These examples suggest an approach to several old and challenging questions, including the problems of consciousness and free will.

Free will is the ability to make choices, but if our bodies and brains are governed by deterministic physical laws, our actions would be determined. Arguments about free will are innumerable; I will only mention two:

- William James proposed a two-stage model in which possible actions are generated by a random process and then selected by a deterministic process. In that case our actions are fundamentally unpredictable because the process that generates them includes a random element.
- David Hume suggested that our perception of making choices is an illusion; in that case, our actions are deterministic because the system that produces them is deterministic.

These arguments reconcile the conflict in opposite ways, but they agree that there is a conflict: the system cannot have free will if the parts are deterministic.

The complex systems in this book suggest the alternative that free will, at the level of options and decisions, is compatible with determinism at the level of neurons (or some lower level). In the same way that a traffic jam moves backward while the cars move forward, a person can have free will even though neurons don't.

Exercise 10.6. *Read more about free will at* `http://en.wikipedia.org/wiki/Free_will`. *The view that free will is compatible with determinism is called* **compatibilism**. *One of the strongest challenges to compatibilism is the "consequence argument." What is the consequence argument? What response can you give to the consequence argument based on what you have read in this book?*

Exercise 10.7. *In the philosophy of mind,* Strong AI *is the position that an appropriately-programmed computer could have a mind in the same sense that humans have minds.*

John Searle presented a thought experiment called "The Chinese Room," intended to show that Strong AI *is false. You can read about it at* `http://en.wikipedia.org/wiki/Chinese_room`.

What is the **system reply** *to the Chinese Room argument? How does what you have learned about complexity science influence your reaction to the system response?*

You have reached the end of the book. Congratulations! When you first read Chapter 1, some of the topics might not have made sense. You might find it helpful to read that chapter again now. Then get to work on your case study! See Appendix A.