# Modeling Schelling: A Demonstration of the Value of Agents for Modeling Emergent Orders

By Gene Callahan

St. Joseph’s College, Brooklyn

**Abstract:** The aim of this paper is to demonstrate the usefulness of agent-based models for formalizing the notion of emergent orders. The method by which it will do so is to attempt to capture each of the models offered in Thomas Schelling’s *Micromotives and Macrobehavior* in an agent-based model. We will demonstrate how these models can serve as formal proofs (or disproofs) of the correctness of Schelling’s (mostly verbal) reasoning.

**Keywords:** Schelling, emergent order, agent-based modeling

### Introduction

The aim of this paper is to demonstrate the usefulness of agent-based models for formalizing the notion of emergent orders. The method by which it will do so is to attempt to capture each of the models offered in Thomas Schelling’s *Micromotives and Macrobehavior* in an agent-based model. We will demonstrate how these models can serve as formal proofs (or disproofs) of the correctness of Schelling’s (mostly verbal) reasoning.

### I. Agent-Based Models and Emergent Orders

Schelling:

“To make that connection [between individual’s intentions and aggregate outcomes] we usually have to look at the system of interaction between individuals in their environment, that is, between individuals and other individuals or between individuals and the collectivity. And sometimes the results are surprising. Sometimes they are not easily guessed. Sometimes the analysis is difficult. Sometimes it is inconclusive.” (2006: 14)

Axtell comments upon how agent-based models fit into the broader world of models as follows:

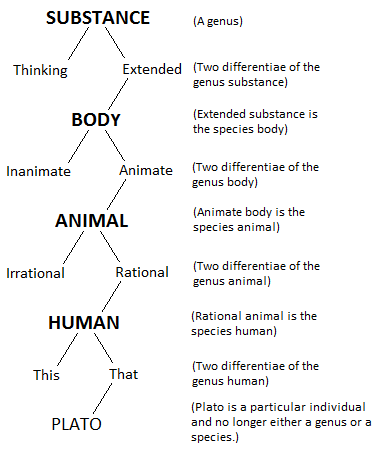
One such use — the simplest — is conceptually quite close to traditional simulation in operations research. This use arises when equations can be formulated that completely describe a social process, and these equations are explicitly soluble, either analytically or numerically. In the former case, the agent model is merely a tool for presenting results, while in the latter it is a novel kind of Monte Carlo analysis. A second, more commonplace usage of computational agent models arises when mathematical models can be written down but not completely solved. In this case the agent-based model can shed significant light on the solution structure, illustrate dynamical properties of the model, serve to test the dependence of results on parameters and assumptions, and be a source of counter-examples. Finally, there are important classes of problems for which writing down equations is not a useful activity. In such circumstances, resort to agent-based computational models may be the only way available to explore such processes systematically, and constitute a third distinct usage of such models. (Axtell, 2000)[[1]](#footnote-1)

**II. How Indra Is Constructed**

A programming paradigm reflects a view of how a computer program can best “cut reality at its joints.” Ultimately, all programs wind up being strings of zeros and ones, and the computer cares not a lick how we humans organized our code for ourselves. So what we want from a paradigm, and from a language supporting a paradigm, is that it makes it easier for us to model the problem with which we are dealing, and to survey the code once written and understand it in terms of the model world, rather than the world of the computer.

Over the course of six-plus decades of development, a number of programming paradigms have been employed, including procedural, functional, and object-oriented programming. Indra relies heavily on the latter, so let us examine the paradigm briefly.

As noted by Epstein and Axtell (1996), object-oriented programming (OOP) and ABM are a natural fit, and we have tried to exploit that dovetailing to the greatest extent possible. OOP, when done properly, presents the world of the model as a Porphyrian tree (see Figure 1), where we descend the tree from the most general categories (classes, in OOP) to the most specific. Furthermore, it enables the programmer to “pick up” the characteristics and capabilities of classes further up the tree “for free,” by *inheritance*. So, for instance, in our particular case, we can establish a class Agent that can act with a goal. Then we can move down the tree and create SpatialAgent, that inherits all of Agent’s capabilities, while also having a location in space. Next, we create a class we call MobileAgent that inherits from SpatialAgent and can also move through space. Next, we can create Creature, inheriting all capabilities of MobileAgent while also eating and reproducing. Finally we create classes representing actual critters, such as Rabbit and Fox.



*Figure 1: Public domain image downloaded from Wikipedia, http://upload.wikimedia.org/wikipedia/commons/e/ea/Porphyrian\_Tree.png.*

Later in the paper, I will offer an actual example of an inheritance hierarchy in Indra.

While OOP is a natural fit for ABM, I have sought to push beyond the OOP paradigm as well, by beginning to incorporate some concepts from Whitehead’s process philosophy.

### III. Schelling’s Segregation Model

Schelling’s model demonstrated that it is not necessary for all or even most individuals to want to live in a largely segregated neighborhood for such neighborhoods to arise: all that is needed is for most people not to want to be “too small” a minority in their neighborhood.

In keeping with good programming practice, rather than implement Schelling’s full-blown model in one shot, we began with the simplest possible version of the model, and added refinements on top of that simple framework. So to start, we simply gave agents one of two colors, and then placed them randomly on a grid. Each “unit” of time, each agent “looks around” its “neighborhood” and sees what its neighbors are like. If the agent is satisfied with its neighborhood’s racial composition, it stays put. If not, it randomly jumps to another square on the grid.

Here is the code that “looks around” the agent:

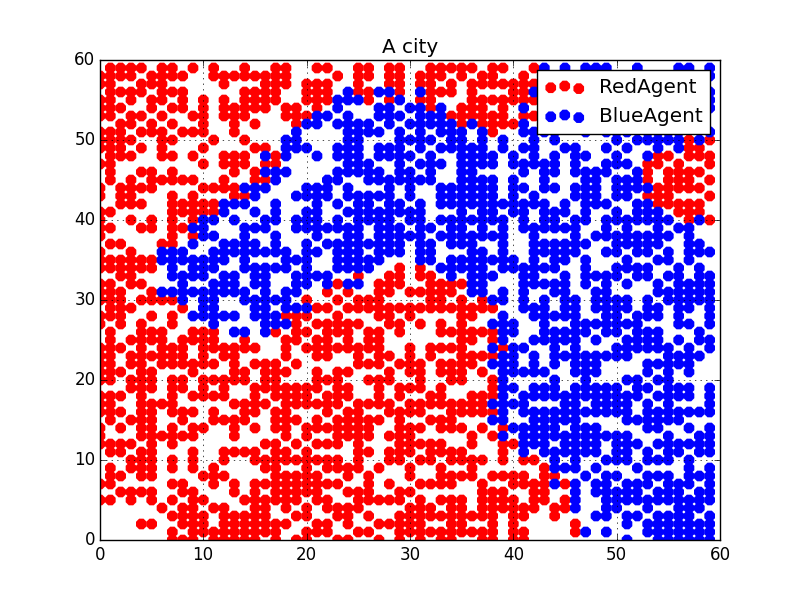
1. **def** survey\_env(self, this\_view):
2. """
3. Look around and see what our env holds for us.
4. """
5. resembles\_me = 0
6. total\_neighbors = 0
7. **for** neighbor **in** self.neighbor\_iter(view=this\_view):
8. total\_neighbors += 1
9. **if** self.get\_type() == neighbor.get\_type():
10. resembles\_me += 1
11. **return** (resembles\_me, total\_neighbors)

And here is the code that decides whether to move or not. A return of “True” means, in this context, “I’m happy where I am”:

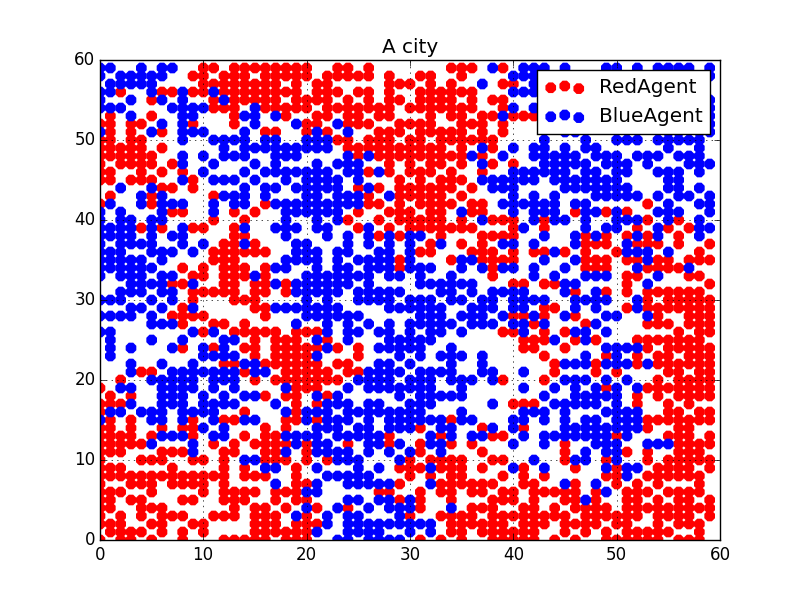
1. **def** evaluate\_env(self, resembles\_me, total\_neighbors):
2. """
3. Use the results of surveying the env to decide what to do.
4. """
5. **if** total\_neighbors > 0:
6. **return** resembles\_me / total\_neighbors >= self.tolerance
7. **else**:
8. **return** True  # everyone is OK with no neighbors

This model lacks several elements of Schelling’s full model. Most significantly, in this first cut, agents do not try choose an acceptable neighborhood when they move: they just jump to a random, empty spot. An interesting finding of our work, showing the value of formalizing verbal reasoning, merely random moves suffice to produce the phenomenon Schelling describes. This means that he added an unnecessary condition to the model. Furthermore, that unnecessary condition can cause the model to run forever: it is quite possible that there is *no* acceptable neighborhood for some agent and some combination of parameters, so that an attempt to move to one, with no check on the number of attempts, will never terminate.

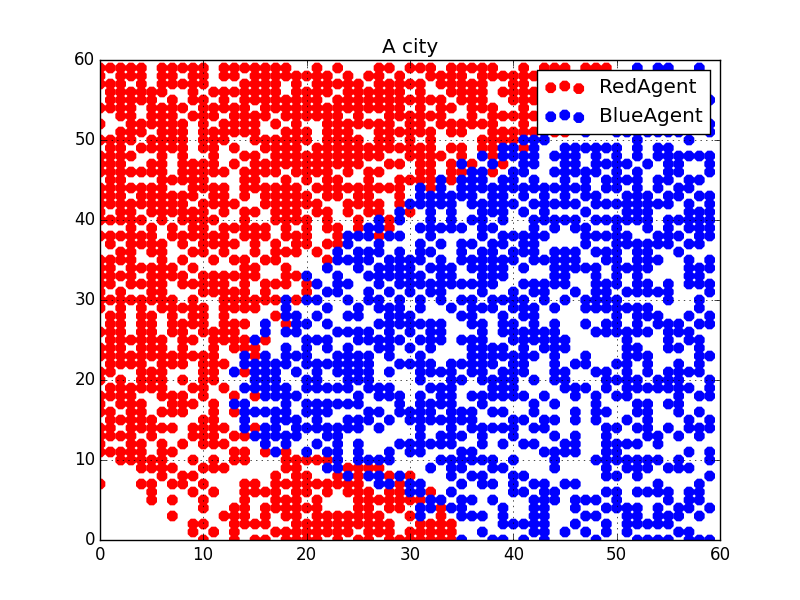
By making the tolerance level a range, rather than a single scalar, we see neighborhoods that are “ragged at the edges”: instead of clean divides when we reach equilibrium, like we got with a single tolerance number, there are a scattering of highly tolerant agents “hanging around” the edges of neighborhoods where they are a distinct minority. Here is what a run looks like where we set the minimum and maximum intolerance to the same value (in this case, .4), meaning there is no range:



We run with again with the default minimum intolerance of .1, and maximum of .7, and we see this:



Certain agents are deeper inside the other color’s territory in the second graph: those are the ones with the lowest intolerance settings. Now we run again, with no range, but the neighborhood size increased from 4 to 8:



With this larger neighborhood size, we get two fully segregated sections of our city, with only a little mingling at the edges. (A larger neighborhood size means that agents take more distant surrounding “houses” into consideration when deciding if their neighborhood is acceptable. Naturally, this produces larger segregated areas.)

### Conclusion

### Bibliography

Axtell, Robert. 2000. “Why Agents? On the Varied Motivations for Agent Computing in the Social Sciences.” *The Brookings Institution*. Accessed March 15. http://www.brookings.edu/research/reports/2000/11/technology-axtell.

Downey, Allen. 2012. *Think Complexity*. Sebastopol, Calif.: O’Reilly.

Epstein, Joshua M, Robert Axtell, and Brookings Institution. 1996. *Growing Artificial Societies: Social Science from the Bottom up ; a Product of the 2050 Project, a Collaborative Effort of the Brookings Institution, the Santa Fe Institute, and the World Resources Institute*. Washington, DC [u.a.: Brookings Inst. Press [u.a.].

Indra. 2015. *GitHub*. Accessed July 24, 2015. https://github.com/gcallah/Indra.

Schelling, Thomas C. 2006. *Micromotives and Macrobehavior*. New York: Norton.

1. See also Downey (2012: 43-44) for another discussion of this same topic. [↑](#footnote-ref-1)