Indra: An Agent-Based Modeling System

Gene Callahan
Cardiff University
Collingwood and British Idealism Centre
I. Introduction

Indra is the outcome of a long-standing interest in using agent-based modeling (ABM) to explore social interactions. Most specifically, I have long believed that ABM offers a way to formalize and make accessible to mainstream economists important insights from Austrian economics. (Paul Krugman once asked, “If the Austrian’s Business Cycle Theory is any good, where is their model?”) Hayek did not consider *The Pure Theory of Capital* to have come out well, as he lacked the technical tools to complete it per his vision: I suggest ABM might have filled that gap.

But why build a new ABM framework? Part of this is simply cockiness: I believe I can write a large programming framework better than have others. But it is not merely that: While there are existing ABM frameworks, this one is written with the aim of modeling Austrian capital theory at its core (although it enables the creation of many other types of models as well). As such, it ought to be easiest to create such a model using this framework.

II. What Is a Model?

Let us open with some philosophical considerations, of two varieties: In this section, we will examine the scientific status of models of this sort, as they are a fairly recent phenomena and it may not be obvious what value, if any, they have. In the next, we will take a brief look at the metaphysical considerations governing the building of the framework itself.

Based upon my recent reading of Morgan (2012) and reflection upon my own modeling efforts over many years, I offer the following précis of what models are:

1) Models are constructed.
2) They are made of distinct parts. (E.g., "a supply curve, a demand curve, an x-axis, a y-axis," or "red lines for highways, black lines for local roads, dashed lines for dirt roads.")
3) The parts are made to fit together. (The supply curve is measured in the same units as the demand curve, and crosses it somewhere. The roads are laid out on the same grid, using the same scale.)

4) We can adjust those parts, either purely mentally, or with our hands (as with an architectural model), or a pencil and eraser (a mechanical drawing), a computer (a weather model), and so on. (In using a map, we actually adjust a "part" we put in: where we are. Sometimes, this part is represented by our finger, as we trace a route, or the mark of a highlighter.)

5) Adjusting the parts produces an "answer" of some sort from the model: "Oh-oh, if we move that wall there, the stairs won't fit," or "If the supply curve shifts that far right, the new price will be $4.50."

6) The modeler hopes that the answer produced by the model says something about what will happen when changes occur in (or are deliberately made to) the thing being modeled.

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

Indra is primarily aimed at the second and third categories of ABM use.

¹ See also Downey (2012: 43-44) for another discussion of this same topic.
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.

Object-oriented programming and agent models

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.

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

**Whitehead’s process philosophy**

Although work on this aspect of the project is in a preliminary stage, some progress has been made on incorporating Alfred North Whitehead’s “process philosophy” into Indra.
For our purposes, the major difference between Whitehead’s view and that of the Porphyrian tree is that, for Whitehead, an entity does not have an “essence” existing in a vacuum. Instead, its nature is, to a great extent, determined by its relationships to other entities. In terms of idealist philosophy, rather than entering into merely external relations, entities on Whitehead’s view are constituted by internal relations (Whitehead, 2014).

For Whitehead, what we would traditionally see as an “object” is the entire universe “prehended” from a certain point of view. An electron is what it is only in terms of its relationships with other electrons, protons, neutrons, and so on. The field equations that rule the roost in modern physics describe such relationships: they tell us that, when an electron’s electromagnetic field operates with that of a proton in such-and-such a fashion, here is what will result.

Using Whitehead’s metaphysics as the basis for a programming system might be thought to be a mere curiosity, or a form of self amusement. But I am hoping that building the idea of prehensions into the Indra system can help solve a serious problem: how can a software system like this account for discovery (see Kirzner, 1973). Programming a model where agents trade known goods for which they have known utility functions is not trivial (see the EdgeworthBox section below), but it is not a mystery as to how we can build it. But how can we program what happens when an agent discovers a good previously unknown to her? How much does she fancy this good?

Prehensions provide a possible solution to this problem: if we create relationships between objects as objects in their own right (prehensions), and then create all objects so that they can accept “plug-in” relationships to other objects, neither an agent type nor even the creator of the agent type need know about another class of agents or things in advance in order to interact with those entities.

This plug-in approach to creating agent behavior promises other advantages. For instance, Epstein and Axtell (1996: 72-73) develop a very minimal model of cultural transmission: a “culture” is a sequence of ones and zeros, a sequence which doesn’t influence the agent’s behavior in any way except to distinguish those who have similar sequences (“my tribe”) from those who don’t (“their tribe”). But if culture is transmitted
by prehensions, actual behavior will be adopted in the process of acculturation, and different cultures will actually have different “customs.”

And ultimately, the goal is to allow non-programmers to create ABMs by filling in a template for some agent characteristics, and then choosing from a library of existing prehensions to create the agent’s behavior.

Our initial tests of this idea of prehensions have had positive results. For instance, in our predator-prey model, we have introduced new animals, and we only need to give the pre-existing ones a prehension of the new type in terms of a universal—“Eats” or “Flees,” for instance—and the existing creatures respond to it in the way we want.

Our Models

Let us take a brief survey of the models already constructed using the Indra system. The source code for all of these models is open source, and is available online at Indra (2015).

Forest Fire

This is a simple cellular automaton (see Glossary), where a fire starts at one side of a forest and spreads through it, based on Drossel and Schwabl (1992). Even though this model is simple, we still get interesting dynamics: if we set the density of trees high enough (above .5, it turns out), the fire spreads in an almost straight line across the forest. If we set it low enough, it will burn out. But at intermediate values of tree density, we get interesting patterns of burning:
Schelling’s Height Model

Schelling (2006) asks what will happen if the genetic engineering of humans advances to the point where we can control the height of our children, and further, that what people generally desire is merely that their own offspring not be “runts”: perhaps, no one wants their child to be in the bottom 10% of heights in the population, recalling how the “runts” got picked on in school.

Schelling notes that since a population must have a bottom 10%, it is not possible for everyone at once to ensure that their children are not in the decile. Instead, what will occur is that the effort to see that that is so will result in a increasing average height for humans, a result intended by no one.

Our modeling here shows that Schelling’s intuition can be easily formalized in an ABM.
Krugman’s Baby-Sitting Co-op

Krugman (1998) sought to explain his view of recessions by pointing back to a paper by Joan and Richard Sweeney. As he put it:

Well, it turned out that there was a small technical problem. Think about the coupon holdings of a typical couple. During periods when it had few occasions to go out, a couple would probably try to build up a reserve—then run that reserve down when the occasions arose. There would be an averaging out of these demands. One couple would be going out when another was staying at home. But since many couples would be holding reserves of coupons at any given time, the co-op needed to have a fairly large amount of scrip in circulation.

Now what happened in the Sweeneys' co-op was that, for complicated reasons involving the collection and use of dues (paid in scrip), the number of coupons in circulation became quite low. As a result, most couples were anxious to add to their reserves by baby-sitting, reluctant to run them down by going out. But one couple's decision to go out was another's chance to baby-sit; so it became difficult to earn coupons. Knowing this, couples became even more reluctant to use their reserves except on special occasions, reducing baby-sitting opportunities still further.

Our model captures this situation: we distribute coupons to our agents, assign them each a minimum coupon holding below which they will not hire a sitter, and above which they will go out or sit with equal probability. The model develops as Krugman described it: below a certain number of coupons, the “economy” grinds to a halt and we enter a “recession.”

Note: Since I am addressing an Austrian group with this paper (at least in its first incarnation), I must note that showing a model is coherent is not the same as showing it applies! The obvious rejoinder to Krugman on this point by “Austerians” is that the real problem was not limited scrip, but fixed babysitting prices.

Schelling’s Segregation Model

As Thomas Schelling famously pointed out (2006), 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. We have implemented this model, and it indeed plays out as Schelling saw: once a “tolerance” threshold is set somewhat below 50%, neighborhoods will tend to become almost completely segregated.

This model also demonstrates our ability to experiment with these constructs: one of my students has been creating groups with different tolerance levels, adding additional groups, and so forth, with interesting results.

A Predator-Prey Model
Based on the hypothesis, explored in Callahan and Hoffman (2015), that many theories of social cycles appear to have a predator-prey dynamic (Insightmaker, 2015) at their core, we sought to build a predator-prey model as the basis for further models exhibiting cyclical behavior. Subsequent models that harbor this dynamic can inherit the features of this model, demonstrating one of the benefits of object-oriented programming.

This model itself is an interesting case of the dangers of building too much into a model from the start: it was the very first model built in Indra, and the ambitions of the programmer (me) were bigger than his development resources. The model has too many parameters, and thus is extremely difficult to stabilize. But, we live and learn, and subsequent models have avoided this problem.

Adam Smith’s Fashion Model
The first successful descendant of our predator-prey model is based upon a passage of Adam Smith’s from *The Theory of Moral Sentiments*:

“Fashion is different from custom, or rather is a particular species of it. That is not the fashion which every body wears, but which those wear who are of a high rank, or character. The graceful, the easy, and commanding manners of the great, joined to the usual richness and magnificence of their dress, give a grace to the very form which they happen to bestow upon it. As long as they continue to use this form, it is connected in our imaginations with the idea of something that is genteel and magnificent, and though in itself it should be indifferent, it seems, on account of this relation, to have something about it that is genteel and magnificent too. As soon as they drop it, it loses all the grace, which it had appeared to possess before, and being now used only by the inferior ranks of people, seems to have something of their meanness and awkwardness.” (Smith, 2015)
This model performed as we expected it to, so that we see cycles of fashion occurring in our two populations:

**Edgeworth Box Model**

Following Morgan (2012), we took the Edgeworth Box to be a fundamental building block of further models of exchange, and so we began creating exchange-based models by making an Edgeworth Box. Two traders are initially endowed each with a different good, and the model is working if they exchange goods until no more exchanges are mutually beneficial.

A major concern here was to keep the model “realistic,” in the sense that we did not want our agents to be able to peer inside other agents endowments or utility functions. An agent proposing a trade had to do so blindly, without any knowledge of whether the other agent had any interest in a good at all. This was a difficult bit of coding to get correct. Here is what was ultimately implemented for this process. I don’t expect all potential audiences for this paper to be able to follow the code in detail, but, I do urge
you to look through it, even if you’re not a programmer, and see how code is framed in a familiar economic terms, such as goods, offers, rejections, acceptances, and marginal utility:

```python
1. def act(self):
2.     
3.     Act is called in an interactive loop by code
4.     in the base framework
5.     
6.     potential_traders = self.survey_env(TRADE)
7.     for t in potential_traders:
8.         if t is not self:
9.             for g in self.goods:
10.                amt = 1
11.                while self.goods[g]["endow"] >= amt:
12.                    ans = t.rec_offer(g, amt, self)
13.                    if ans == ACCEPT or ans == REJECT:
14.                        break
15.                amt += 1
16.     
17. def rec_offer(self, his_good, his_amt, counterparty):
18.     
19.     Agent has received an offer of a good,
20.     and loops over her goods to
21.     see if there is a profitable trade.
22.     If 'yes,' make a counter-offer.
23.     
24.     my_amt = 1
25.     util_gain = self.__marginal_util(his_good, his_amt)
26.     for my_good in self.goods:
27.         if((my_good != his_good)
28.                 and (self.goods[my_good]["endow"] > 0)):
29.             util_loss = -self.__marginal_util(my_good, -my_amt)
30.             if util_gain > util_loss:
31.                 if(counterparty.rec_reply(his_good,
32.                     his_amt,
33.                     my_good,
34.                     my_amt)
35.                     == ACCEPT):
36.                     self.trade(my_good, my_amt,
37.                     counterparty, his_good, his_amt)
38.                 return ACCEPT
39.         else:
40.             return INADEQ
41.     return REJECT
42. 
43. def rec_reply(self, my_good, my_amt, his_good, his_amt):
44.     
45.     This is a response to a trade offer this agent has initiated
46.     
47.     util_gain = self.__marginal_util(his_good, his_amt)
48.     util_loss = -self.__marginal_util(my_good, -my_amt)
49.     if util_gain > util_loss:
50.         return ACCEPT
51.     else:
52.         return INADEQ
53.     return REJECT
```
def trade(self, my_good, my_amt, counterparty, his_good, his_amt):
    """
    We actual swap goods, and record the trade in the environment
    """
    logging.info("%s is trading %i units of %s for %i units of %s with %s" % (self.name, my_amt, my_good, his_amt, his_good, counterparty.name))
    self.__adj_good_amt(my_good, -my_amt)
    self.__adj_good_amt(his_good, his_amt)
    counterparty.__adj_good_amt(his_good, -his_amt)
    counterparty.__adj_good_amt(my_good, my_amt)
    self.env.record_trade(self, counterparty)
    ""
    util_gain = self.__marginal_util(his_good, his_amt)
    util_loss = -self.__marginal_util(my_good, -my_amt)
    if util_gain > util_loss:
        return ACCEPT
    else:
        return INADEQ

def list_goods(self):
    """
    List the goods an agent possesses.
    """
    goods_descr = ""
    for g in self.goods:
        goods_descr += g + " : "
        + str(self.goods[g]["endow"])) + ", "
    return goods_descr.strip()
Barter Model

Next, we added multiple agents to our Edgeworth Box model. All of the “negotiating” code from the Edgeworth Box model was inherited in this model, and so we found this model relatively trivial to code. We did add a spatial component to trading in this model, so that we can increase or decrease the scope over which agents can “see” other agents and trade with them.

Menger Model

Finally, in our Menger model, we are attempting to see money arise from a good gaining increasing acceptance as a medium of exchange. The first attempt to program this model was interesting as an instance of a spectacular failure of our intentions, and a refutation of the idea that a researcher can merely set up an ABM to get whatever outcome they want. In creating the model, the intention was to have gold emerge as the medium of exchange, by first making it the most durable good, and then by adding to the utility of any good whenever someone will accept it in exchange.

What happened with our first attempt was a dramatic illustration of the fact that, when done properly, these models are vehicles for experimentation, and not simply ways to get out of a computer what you put into it: we thought we had given gold durability and utility characteristics that would make it emerge as money. Instead, what happened was that gold treated in the first round of agent interaction, and then never traded again! In fact, we had created a sort of anti-money that would never circulate.

The “interesting result” was eventually fixed so that gold did emerge as money. However, it took extensive experimentation with the model to see what had been done “wrong” in the first place. (The chief thing was the gold had been given too low a marginal utility for the first unit an agent would gain or lose.)

Conclusion

The Indra system has now been developed to the point that it is relatively easy to generate interesting models using the framework it provides. However, there are many interesting
ideas contained in the system that are merely in their embryonic stage. To advance its development, the main thing it needs now is more users and developers.
Glossary

Agent: A software “unit” that has a goal and acts independently of other units to achieve that goal.

Cellular Automaton: “A cellular automaton consists of a regular grid of cells, each in one of a finite number of states, such as on and off (in contrast to a coupled map lattice). The grid can be in any finite number of dimensions. For each cell, a set of cells called its neighborhood is defined relative to the specified cell. An initial state (time $t = 0$) is selected by assigning a state for each cell. A new generation is created (advancing $t$ by 1), according to some fixed rule (generally, a mathematical function) that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood.” (Wikipedia, 2015)

Class: A genera or species, e.g., ‘dog’ or ‘market economy.’ Classes define the behavior of the objects that are members of the class.

Object: An instance of a class. It is a chunk of memory that combines both code and data, so that the two have a tighter union than in other programming paradigms.

Object-oriented programming: A style of programming that makes extensive use of classes and objects.

Porphyrian tree: A tree showing the descent of classes from the most generic to the least, with individuals as “leaves” at the tips of the branches.
References


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.].


http://www.monadnock.net/menger/money.html.


