Agent Based Modelling for Social Research

Time in Agent based models

Hello! I'm Oliver. In this video, I will introduce you to two different ways to advance time in an agent-based simulation. You will become aware of some of the inherent problems of time-stepped simulations, and learn how to avoid them with discrete event simulation.

To begin, let us imagine a simple agent-based model. The basic unit of agent-based models are agents, who act and interact. If you implement your model, you might want to write something like this. The agents does one thing, and then another, and so on, for example their learn something, and then move somewhere. Of course, in your model, there will be multiple agents. So you will probably write a loop so all agents do their actions. Also, the agents shall not only act once, but repeatedly. The result might look a bit like this . The agents continue to act, until the end of the simulation. I might be that not all agents perform all actions each step, so you might do something like this , where the actions happen probabilistically.

This is what we call a discrete-time (or time-stepped simulation). If you imagine how time advances, it looks somewhat like this: All agents act at once at some time, then time jumps forward a set amount and all agents act again.

Many agent-based models are designed this way.

However, this way to advance time can lead to some problems. In this situation, there are different agents moving on a grid. Each cell may only be occupied by one agent. How do we decide, which of these two agents is allowed to move to the cell X. Remember: fundamentally, all agents act at the same time, so there is no good reason to prefer one over the other.

A similar problem can appear with the different actions of the same agent. In a model of infectious disease, our loop might always process the spread infection before recovery (or the other way around). This will probably introduce a bias in our results.

The core of these issues is that the real world events we want to simulate would in fact not happen all at the same time. Most of the time, there would be a natural order of events. In reality, most events would actually happen at some points in time between our steps.

This highlights another issue: our fixed time steps introduce an approximation error to the event timing. This might be especially problematic, if we are interested in analyzing the time passing between events.

To improve the situation, we might reduce the size of the time step. Like in this picture. Here, we have reduced the size of the time step by a factor of four. If we have another look at our loop, we would, at the same time, reduce the probabilities that an agent acts each step accordingly.

This reduces the aforementioned problems. It becomes less likely, that events happen in the same time step and must be ordered, and the approximation of event times gets better. However, neither of the problems can be solved completely this way. No matter how small the time step is, there is

always a chance of multiple events in one step, and there is always an approximation error. In addition, reducing the time step by a lot will make the simulation execute very slowly. We will have to check for events a lot, and they happen very rarely. In many steps, nothing will happen at all.

There is, however, a way to resolve these issues completely: by removing the time step altogether. Instead, we view time as continuous, and always jump exactly to the time of the next event. This is called “Discrete Event Simulation”. In the rest of this video, I will show one example how discrete event simulation can be implemented. If you are interested in a broader look or more detail, I recommend the book “Simulation Modeling and Analysis” by Averill Law. Especially the fifth edition, in which a chapter on agent-based simulation was added.

Let us come to our example. We will work with a simple SI model of infectious disease. In an SI model, agents might be in one of two states: they might be healthy, but susceptible to the disease, or they might be sick and infectious. In this version, the agents form a network, and the disease can only spread through network edges. In our example, we consider just four agents as shown on the right. The red agents A and B are currently infectious, C and D are currently susceptible. We now look at the model implemented in a modeling language called ML3 that is designed for agent-based discrete event simulation. If you want to learn more about the language, consider the publication shown below.

Each agent has to possible actions, or in this case events that may happen to them. In ML3 these are implemented as two rules. First, each agent that is currently susceptible might get infected. This happens to them with a certain rate, which depends on some parameter “a”, and the number of infected network neighbors they have. Second, each infectious agent may recover and return to being susceptible with a certain rate “b”.

These “rates” govern the timing of the event. Each rule in ML3 is associated with a Poisson process. The time until the next event happens follows an exponential distribution, and the rate gives the parameter of this distribution.

How is this executed? Let’s go through our example.

We begin our simulation at time 0 on the left end of this axis, in the shown state, which we call S0. At first we must go through all possible events, and schedule their happening according to their rates. We begin with the infection events. A is already infected, so nothing can happen. The same is true for B. C is susceptible. As they have 1 infected neighbor, their rate of infection (lambda) is a times 1. We then draw the time until the event Delta t from an exponential distribution with parameter lambda. We add this to our current time, to get the time when C will become infected. Then we repeat the same with D. Lets say, we get this time While the rate for D’s infection is higher, it can still happen later, as the process is stochastic.

Now we go through the recovery events. A is infectious, and gets its recovery scheduled to this time. And for B , we might get this time. C and D are not infectious, so they cannot recover.

Now that all possible events are scheduled, we can select one to be executed. We execute the event that is scheduled to be the earliest. In this case, the recovery of A . If we execute that event, the resulting state looks as this picture. A is now susceptible, and everything else stays the same. Our timeline looks like this: we were in state S0 at time 0, and are now at this time, in this state, which we call S1.

Now, one might think we could just continue with the next event, the infection of C. However, we need to be careful. Now that A has recovered, C can no longer get infected. And D has a lower rate of infection. To account for this, we need to do some rescheduling.

Fist, let us make a copy of our timeline, so we can keep track. Again, we start with infections. For A, we have to schedule an infection event. B still can't get infected. C can no longer get infected, so we need to retract this event. For D, the infection rate gas changed as A was previously infectious. So we calculate the new rate, and draw once again from the exponential distribution. We get this as our new time. Note that we can schedule the event in exactly the same way again, because the exponential distribution is memoryless. If we use other distributions, this gets a bit more complicated. Finally, we go through the recovery events again. Nothing to do for A. Nothing changed for B. And nothing for C and D either.

With this. We are ready to execute the next event one again. Unfortunately for A, they will be re-infected immediately.

To end this, let us have a look at the simulation algorithm as a whole. We call the current state s, and the current simulation time t. In addition to these two variables we use an event queue Q to schedule our potential events. The event queue is simple a priority queue – a datastructure that allows us to enqueue objects with priorities, and to efficiently retrieve the object with the lowest priority. There should be a priority queue available in any programming language you might choose to use.

At first, we schedule all events that could happen. This means, if the event could happen, we draw an execution time, in our case by drawing from an exponential distribution and adding the result to the current time. Then we requeue the event with this time as priority. The requeue operation checks if the event is currently already in the queue. If it is, its priority gets replaced. If it is not, it gets inserted with the given priority. If the event is impossible, we remove it, if it is currently in the queue.

After this initial scheduling, we begin our simulation loop. We take the event with the lowest execution time from the queue, execute it and advance time. Then we reschedule the event we just executed (as it might have to happen again, but is no longer in the queued). And we reschedule all other events that have been affected by the executed one. The difficult part here is to determine which events are affected. If you implement this algorithm for a single simple model, this might be immediately clear from the mechanics of the model. If the model is more complex, it might not be.

You have now seen two different ways to advance time in agent based models. The first is discrete time simulation, or time-stepped simulation, where time advances in fixed increments, and all actions and events happen at the same time at these steps. You have learned of some of the problems this might cause, for example with the ordering of events that happen at the same time step. Then we had a look at a different way to advance time: discrete event simulation, where time jumps directly to the time of the next event. We looked at an example, and you have seen way to implement a discrete event simulation through scheduling and rescheduling of events in an event queue. Even if you do not use a discrete event simulation, you are still able to be more careful with a discrete time simulation. Be aware of potential problems. Handle the order of events carefully. Some ABM tools will help with that, for example by processing agents in a random order each step by default. And choose the time step carefully. You may vary its size, to see if it makes a difference for your results.

Thank you for watching. If you have any questions, please do not hesitate to contact me via email.