Fundamental Anylogic Classes

Nathaniel Osgood

10-24-2009

Object-Oriented Programming Lingo…

- A software “object” is an entity that is associated with
  - Some State
  - Some behaviour
- A software “class” describes a whole category of behaviourally similar objects
  - This is like the “mould” that is used to make the objects
  - While objects associated with this class may differ in the details of their state, they have behavioural similarities
  - We say that objects represented by this class are “instances of the class”

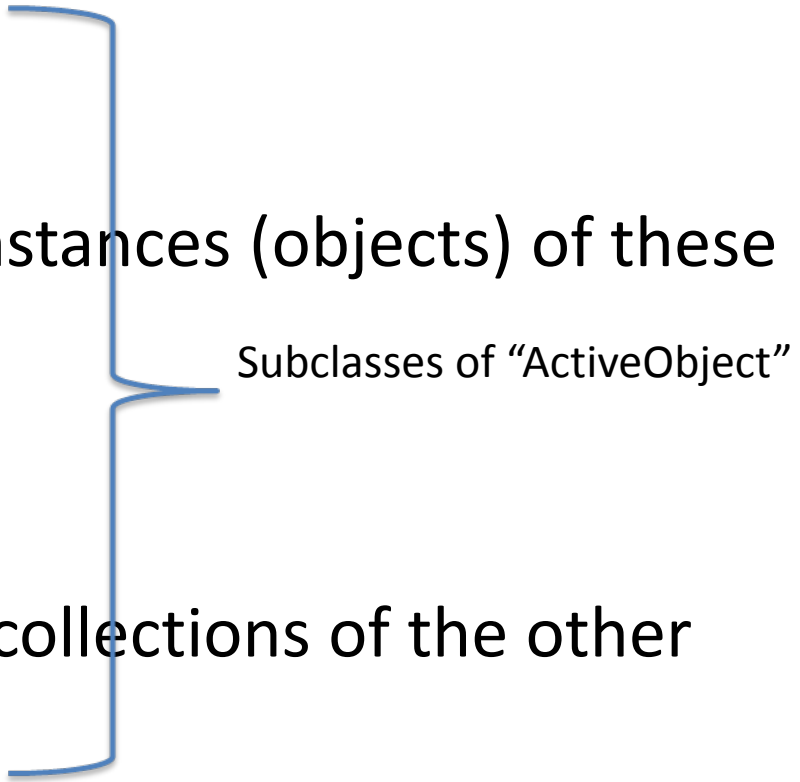
Classes: Design & Run Time Elements

- The AnyLogic interface makes critical use of a hierarchy of *classes (*e.g. *Main, Agent* classes*, Experiment* classes*)*
  - These classes each represent the properties &behaviour of one or more particular objects at runtime
  - We will be discussing this hierarchy more in a later session
- Each of these classes is associated with both
  - Design time interface (appearance at design time)
  - Run time elements (presence of the class object and instances of the class at runtime)

Key Customized Classes

- The structure of the model is composed of certain key user-customized "classes"
- "Agent" classes
  - Your agent classes
  - There are typically many instances (objects) of these classes
- "Main" class
  - Normally just one instance
  - This will generally contain collections of the other classes
- "Experiment" classes

  These describe assumptions to use when running the model



Relationship Between Key Classes

- The Main object normally contains one or more populations of “replicated” agents
  - Each population consists of agents of a certain class (or a subclass therefore) (e.g. “Hares”)
  - The Main object might contain more than one population (e.g. “Hares”, “Lynxes”)
- Agent objects are normally embedded within the (single) Main object
  - Need to mark these as Agents by checking the “Agent” checkbox in their properties

Agent Populations

- Within the Main class, you can create representations of subpopulations by dragging from an Agent class into the Main class area
- Through the “Replication” property, the number of these agents can be set
- The “Environment” property can be used to associated the agents with some surrounding context (e.g. Network, embedding in some continuous space, with a neighborhood)
- Statistics can be computed on these agents

Multiple Agent Classes

- Frequently we will seek to have multiple types of agents, each with differing types of behavior
- Sometimes these agents – while interacting – will have radically different factors that affect them
  - Cf "PredatorPrey" model, with Lynx & Hare
- Sometimes these agents – while distinctive –will be closely related in many ways
  - Here, we may wish to accomplish this through *subclasses of some common custom agent "superclass"*
  - The common features of the agents would be captured in the superclass

Capturing Agent Heterogeneity:
When To Parameterize vs. Use Distinct Classes

- We can capture heterogeneity in agent populations both via using distinct classes (e.g. via subclassing) and via parameterization
- Distinct classes are advisable when there are fundamental behavioural differences
  - The roles that govern the changes in behavior are different
  - There are differences in the types of *behaviour* that the agents can take on
- Use differences in parameterization if the agents are governed by similar rules, but different in their situation/details of context within the rules

Embedded Objects

- The primary AnyLogic customized classes (Main & Agent classes) contain certain elements
  - Parameters
  - Variables
  - “Actions”
  - Elements of presentations

Design Time Components

- Properties for entities
  - Values to use at runtime/Bits of code/Data types/Initial values of state variables/parameter values
- Declaring & manipulating variables, parameters, functions, etc.
- Prepare for runtime using “build”
  - If all goes well, this translates project to executable Java
  - This may alert you to errors in the project
- Define the visual elements to use for each agent
- In an agent-based model, we have only one class for each *type* of object (e.g. “Person”, “Doctor”)

Parameters: Static Quantities

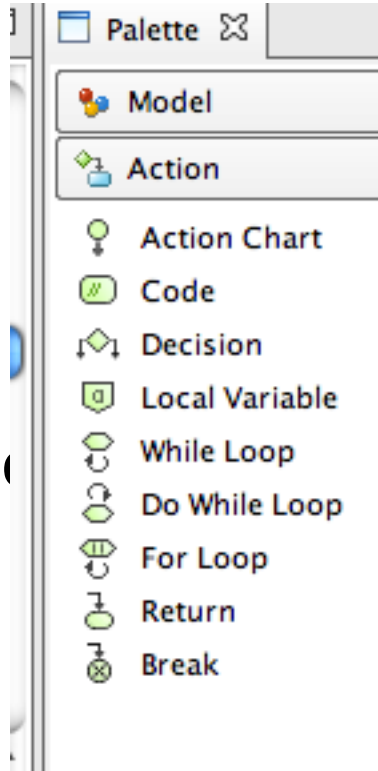
- Parameters normally define constants that represent assumptions
- In Java, such parameters can have many types
  - Integer, Double precision value, boolean, etc.
- For parameters in the *Main* class, we can override the value of the parameters in an experiment
- Presentation elements associated with an Agent have special “Presentation” tab for their parameters

Variables: Dynamic Quantities

- Variables are used for time-varying quantities
- Note that some variables (e.g. stocks) are defined using other “primitive” objects directly supported by AnyLogic
- As for parameters, variables come with many types
- If we want to create an instance variable with a particular class, we should do it with a variable
  - Declaring things using variables (rather than in code) gives us the option of browsing these things at runtime

Expressing Algorithms

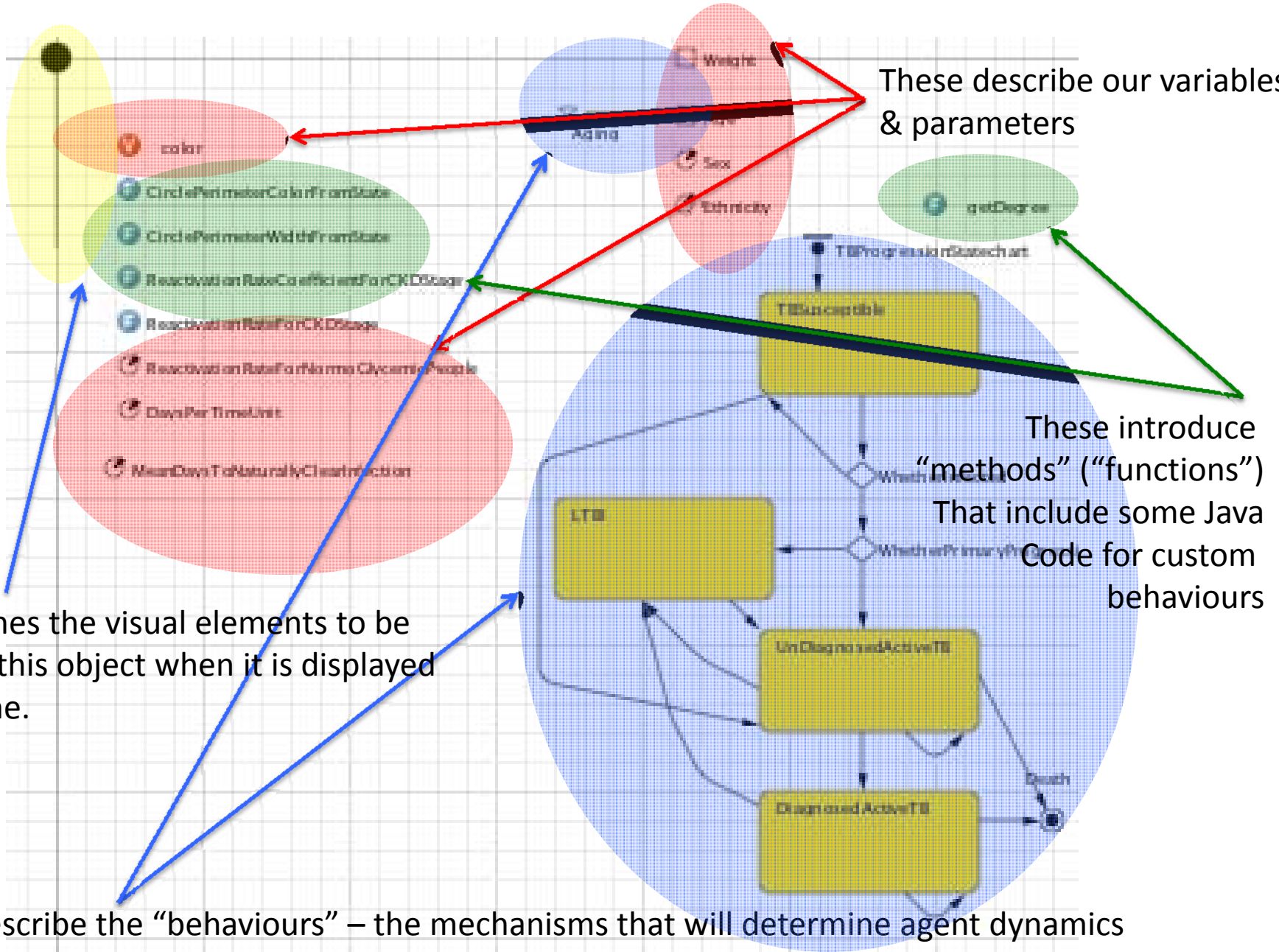
- Algorithms in AnyLogic may be expressed in two ways
  - Defining functions (here, the modeler is responsible for writing the Java code for the function)
- Using the "Action" elements
  - This defines a function primarily graphically
  - Element require filling in pieces (e.g. the expression by which to decide the condition, the variables over which to loop)
  - Custom code can be inserted where desired



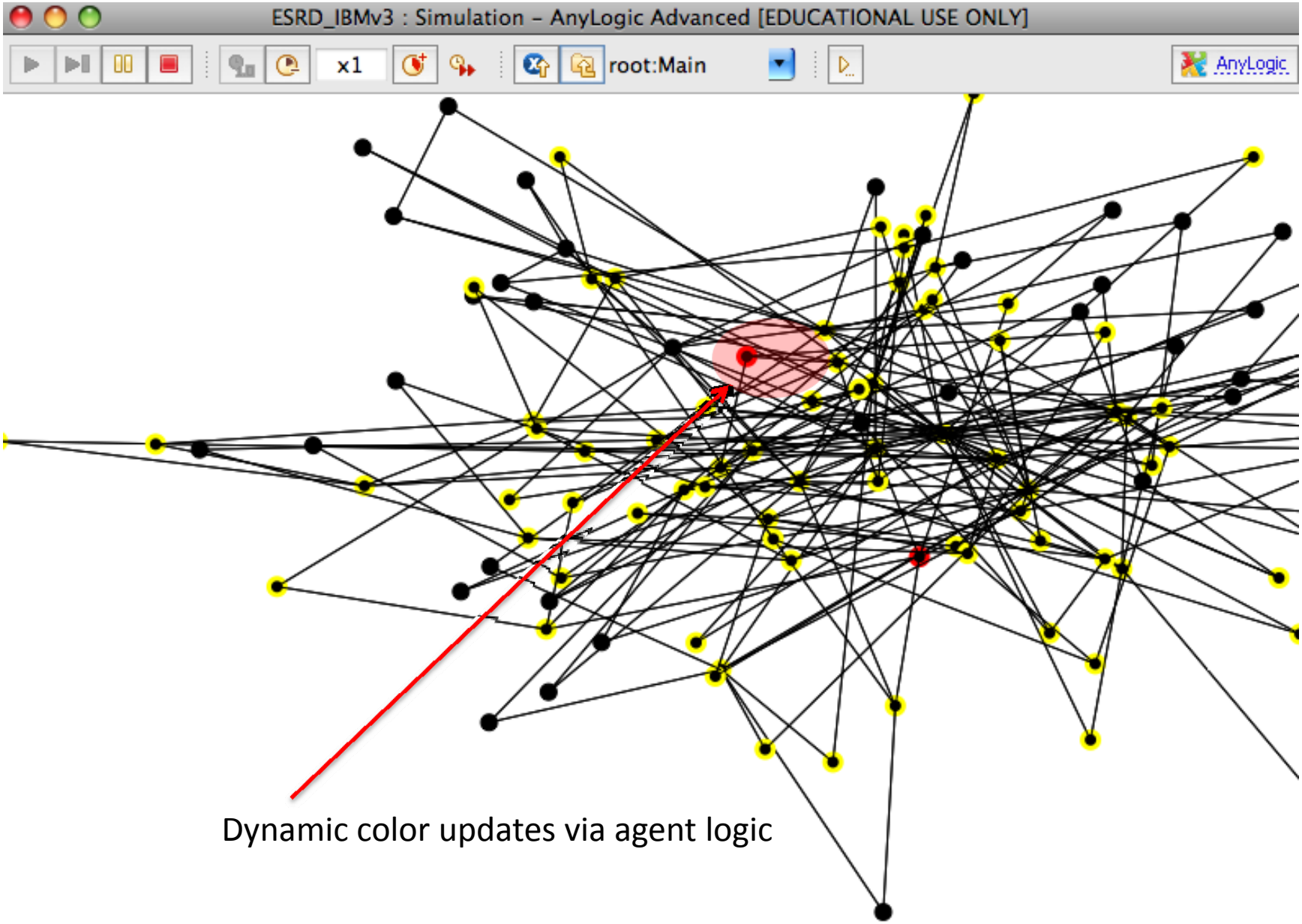
Execution Time

- Here, the simulation is running
- Time is running along
- Each agent class will typically have many particular agents in existence
  - Each agent will have a particular state
  - This population may fluctuate
- Variables will be changing value
- Presentation elements will be knit together into a dynamic presentation

Example Design Time View

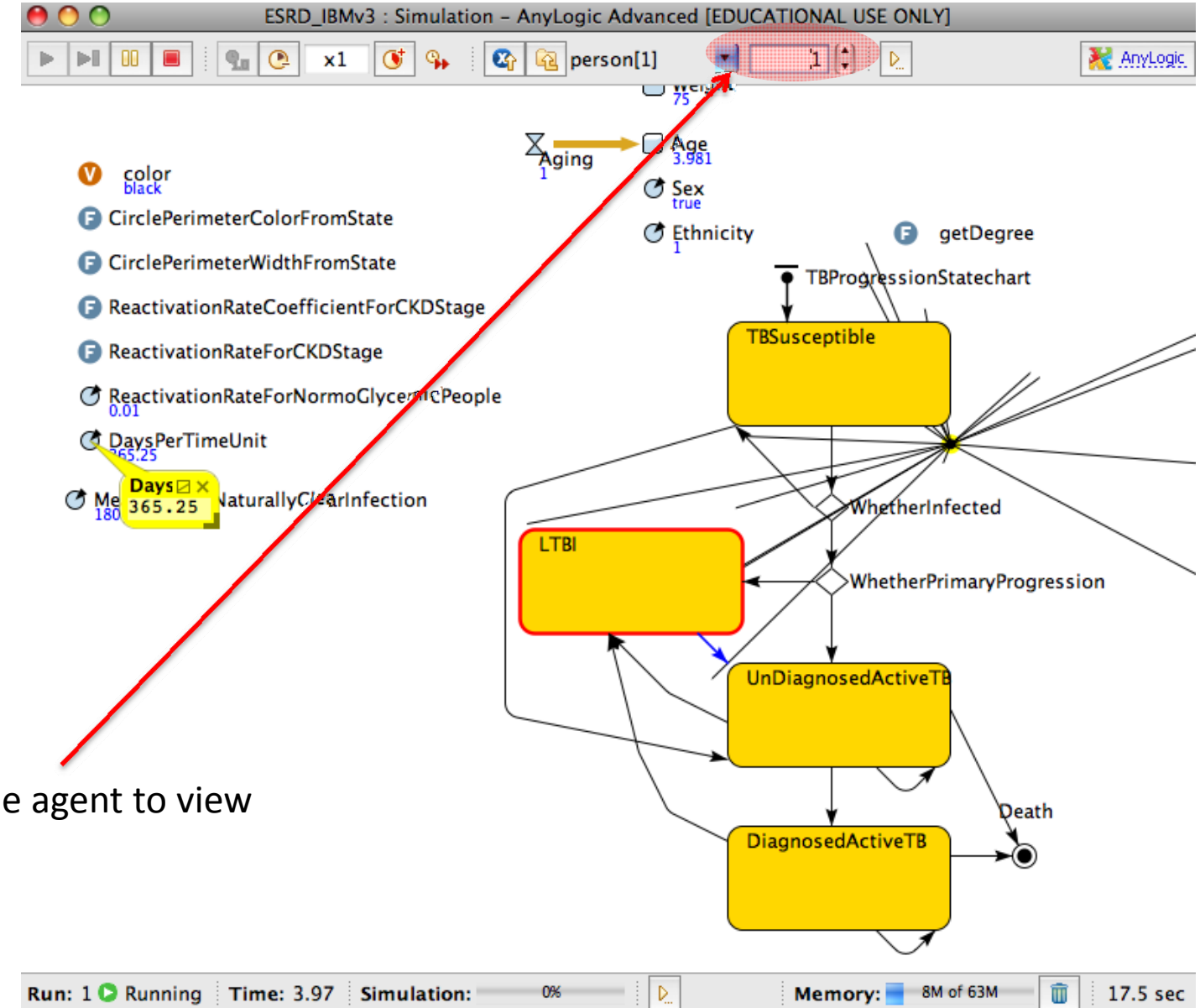


Network Embedding of Agents



Dynamic color updates via agent logic

Runtime View of Particular Agent (Drill Down from Previous View)



Selects the agent to view

Experiment Classes

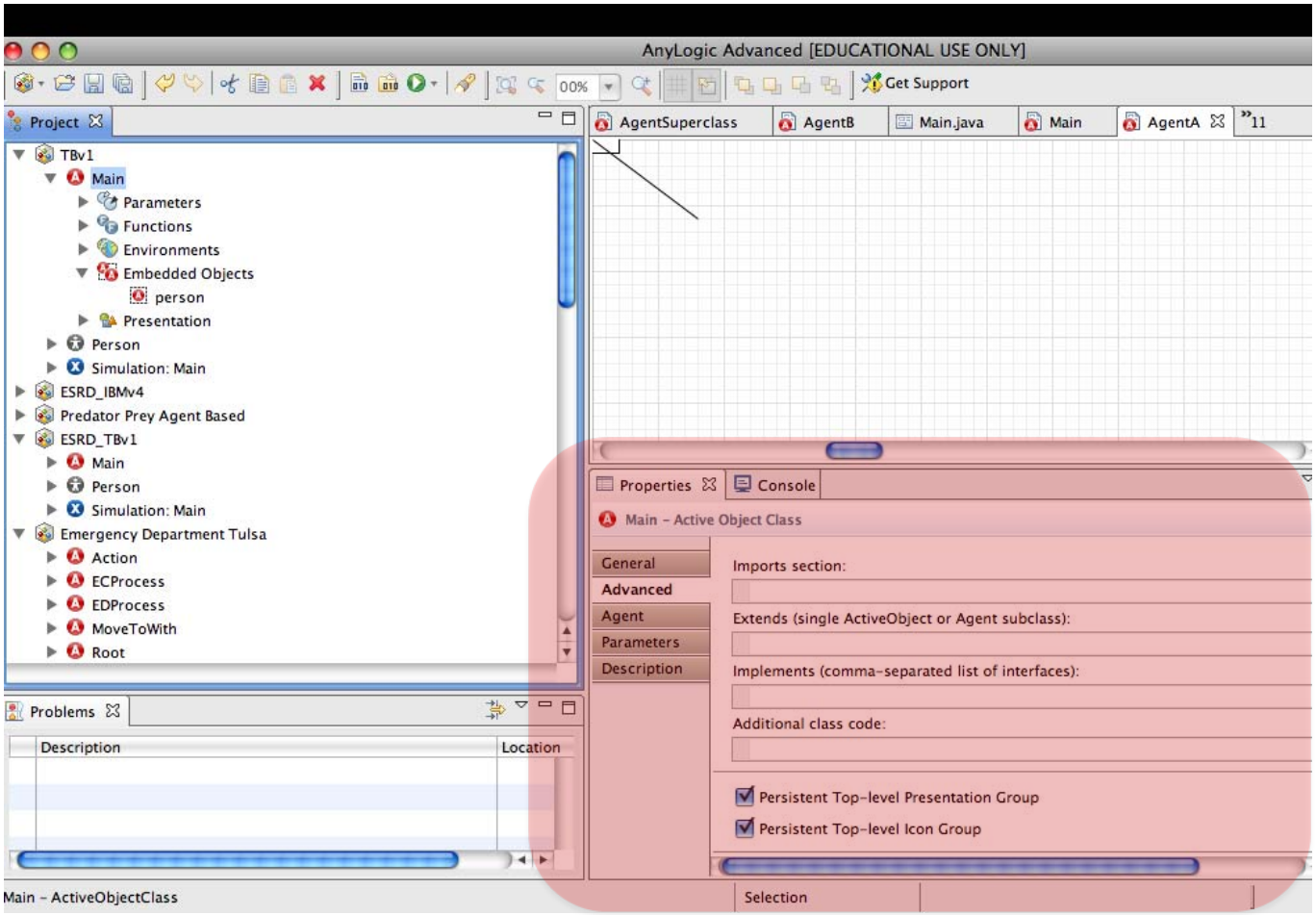
- Experiment classes allow you to define & run scenarios in which global parameters (i.e. parameters defined in *Main*) may hold either default or alternative values
- Experiment classes are also used to set
  - The time horizon for a simulation
  - Memory limits (important for large models)
  - Details of simulation run
  - Details on random number generation
- "Properties" allow one to set the values for each parameter
- Right click on these & choose "Run" to run such a scenario

Java Code: When & How Much?

- AnyLogic offers lots of ways to insert snippets ("hooks") of Java code
- You will need these if you want to e.g.
  - Push AnyLogic outside the envelop of its typical support
    - e.g. Enabling a network with diverse Agent types
  - Send messages
  - Put into place particular initialization mechanisms
  - Collect custom statistics over the population

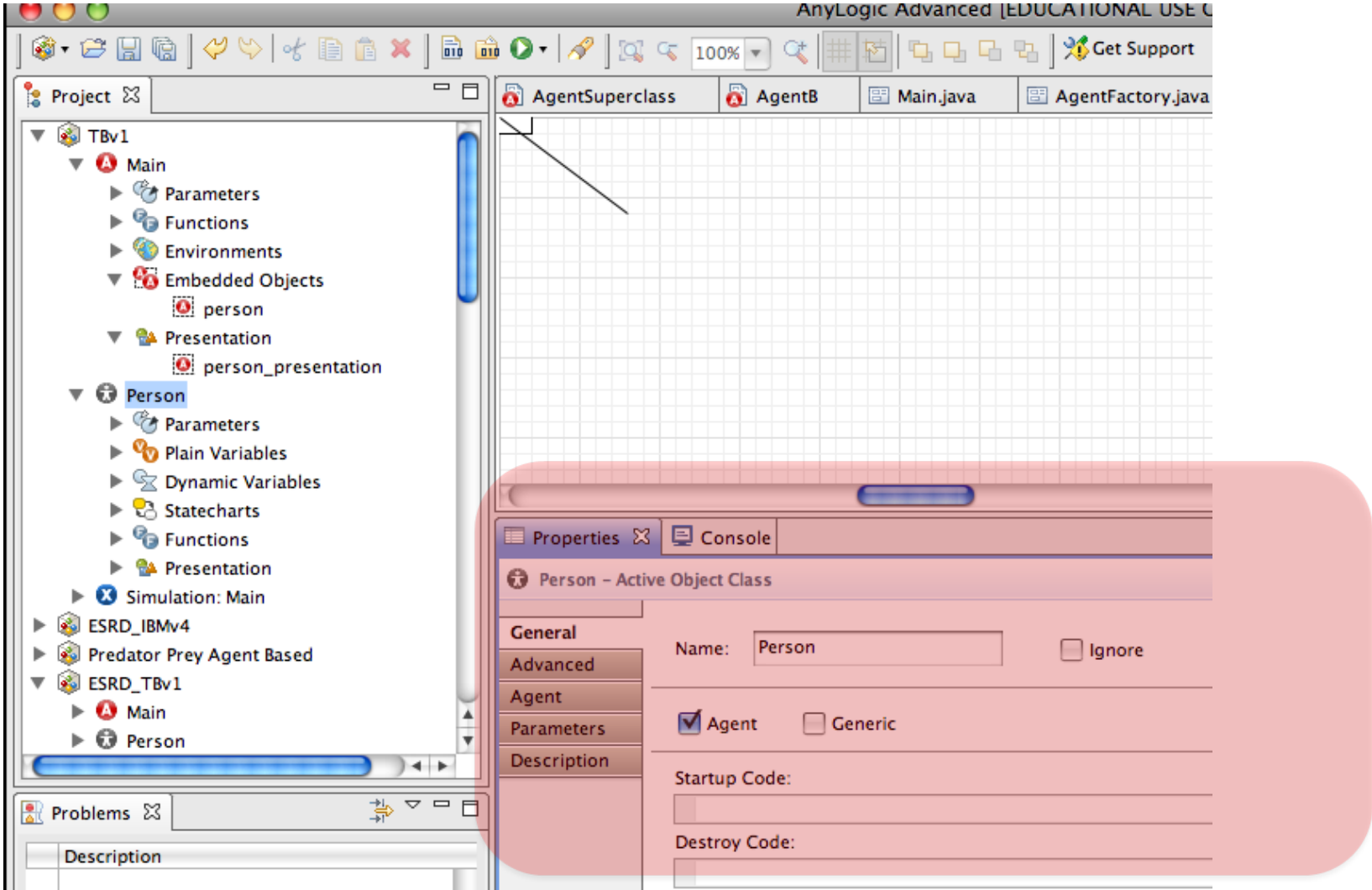
Examples of Where to Insert Code Object Properties

- “Advanced”



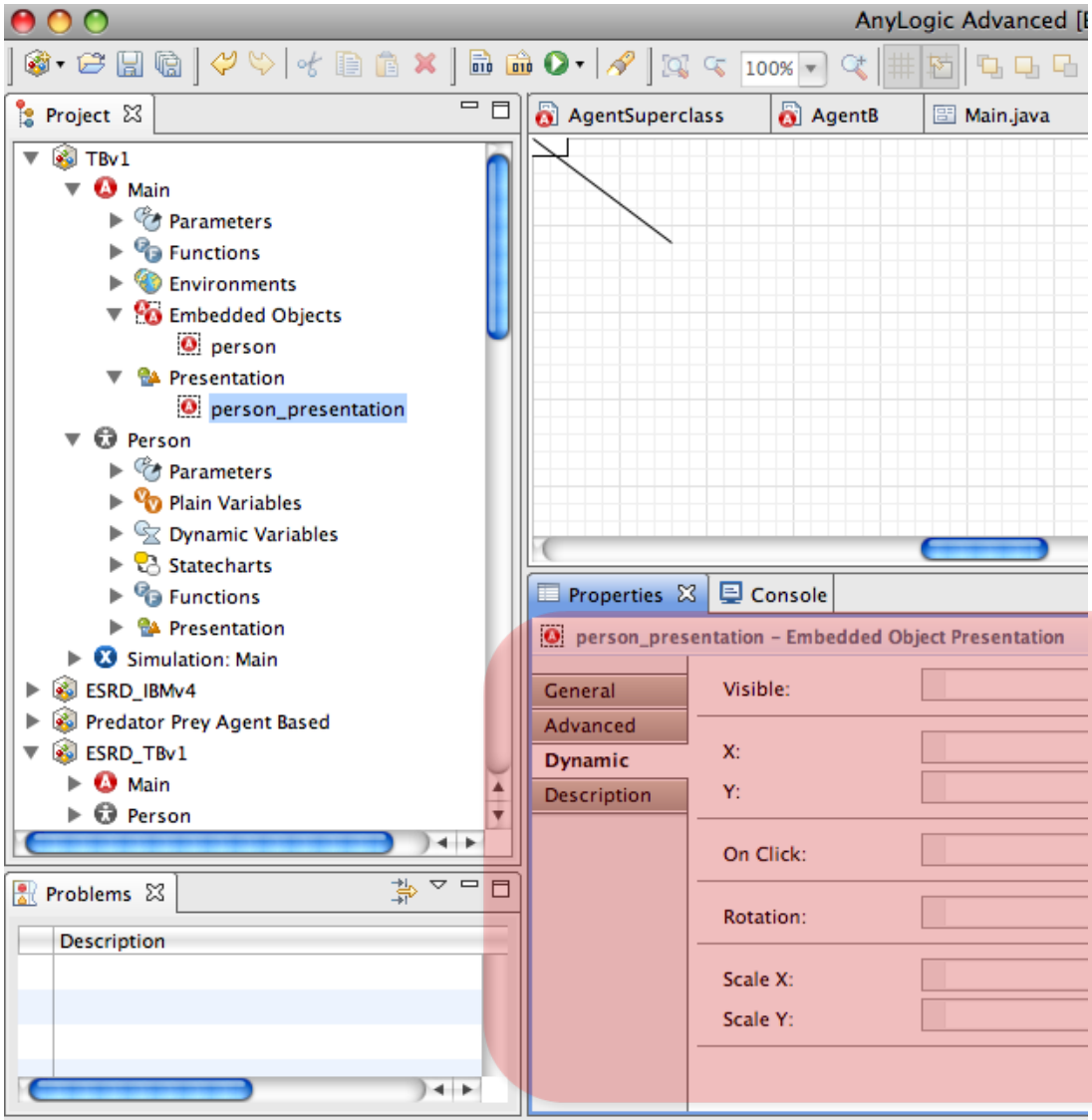
Examples of Where to Insert Code Object Properties

- "General"



Examples of Where to Insert Code Presentations Properties

- “Dynamic”



Finding the Enclosing “Main” class from an Embedded Agent

- From within an embedded Agent, one can find the enclosing “Main” class by calling get_Main()
  - This will give a reference to the single instance (object) of the Main class in which the agent is embedded
  - An alternative approach is to call ((Main) getOwner)

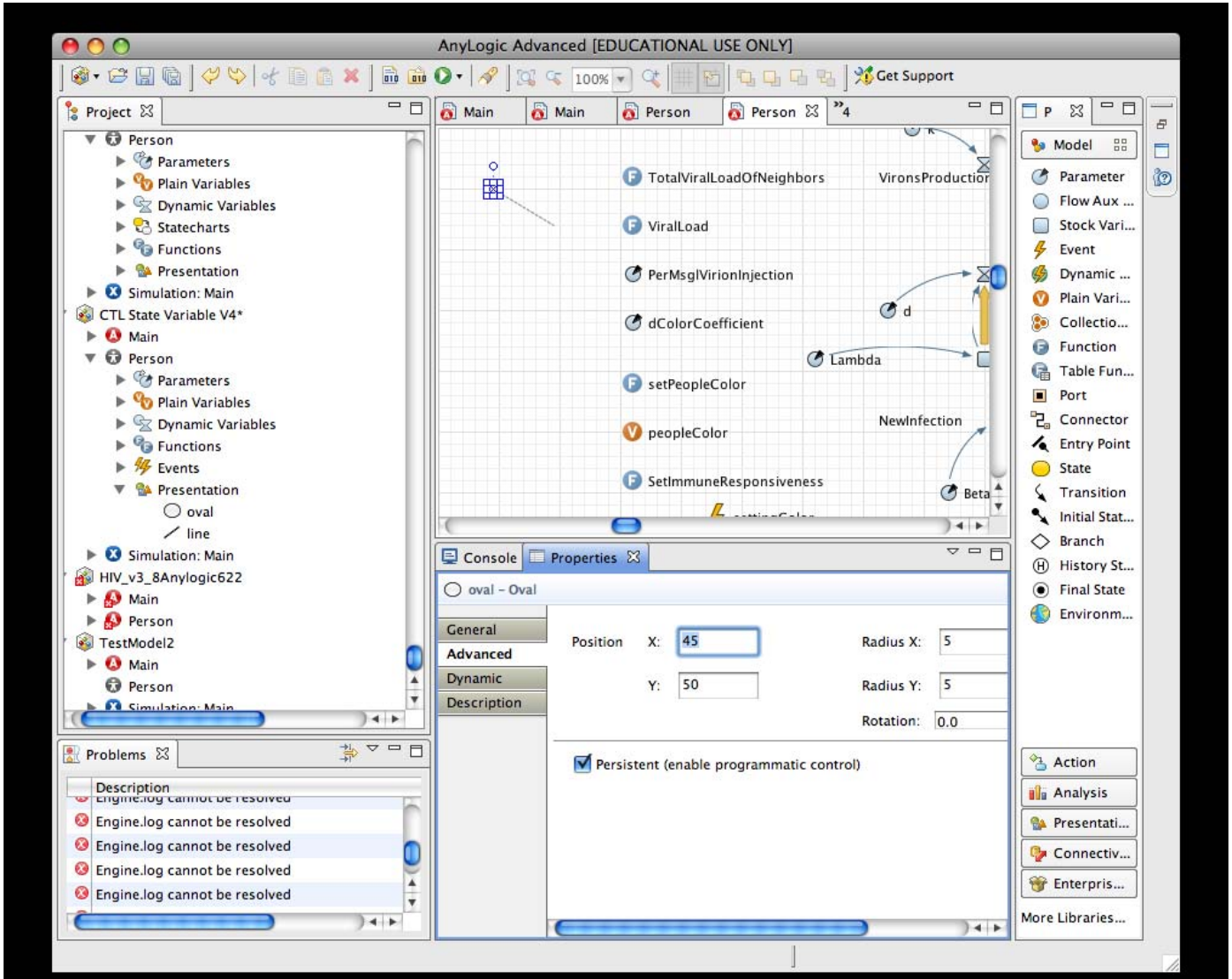
Useful Bits of Java Code

- get_Main() gets reference to Main object
- ActiveObject.trace(str) outputs string to log
- Engine.getTime() gets the current time
- agents.size() gets number of objects in collection agents
- agents.item(i) gets item i from agent collection
- uniform() generates a random number from 0..1

Presentation Properties

- Both key customizable classes (“Main”, various Agent classes) can be associated with “Presentation” elements
- These elements are assembled during execution into animations & presentations of the agents
- Many of these presentation elements have properties that can be set to Java expressions

Enabling Programmatic Control



Example of Dynamic Expressions for an Agent's Presentation Properties