Introduction to the Anylogic Interface & Supporting Concepts
Announcements

• Lecture recording links posted

• Tutorial time: Extended class hours on Tuesday or Thursday
  – Choice will depend on other classes following ours
  – Thursday is likely
The AnyLogic User Interface

Common Configuration

The Project View (overview of projects & components)

Problem area (indicates problem Building/running Model)

Properties area (shows info on selected element in project or palette window)

Palette for adding items to canvas

Note: Double-Clicking on a Tab opens view as Full-Screen
The “Project View” – Hierarchically Shows the Project Components

The “*” means that the model has changed since the last time it was saved. You should consider saving the model when you see this!
Hands on Model Use Ahead

Load Sample Model:
**Predator-Prey Agent Based**
(Via “Sample Models” under “Help” Menu)
Example “Classes”

“Main” classes (Define the “Stage”)

“Agent” classes (Define the actors)

“Experiment” classes (Define the scenarios)
Key Customized “Classes”

• The structure of the model is composed of certain key user-customized “classes”

• “Main” class
  – Normally just one instance
  – This will generally contain collections of the other classes

• “Agent” classes
  – Your agent classes
  – There are typically many instances (objects) of these classes at runtime

• “Experiment” classes
  These describe assumptions to use when running the model
Double Click on “Main” Class Name to View this Class (Should Appear on Top Tab)
“Main” Class

• Defines the environment where agents interact
• Defines interface & cross-model mechanisms
• 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”
    • “Lynxes”
  – The agent classes are defined separately from the Main class
Agent Populations in the Main Class

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

These “parameters” specify static model-wide characteristics

Visual input elements used during simulation (param. setting)

These “functions” calculate things or can change model behavior

These represent the agent populations

Visual output elements used during simulation
Agent Class Defines the Characteristics & Behaviour of Agent Population Members

Double Click on “Lynx”!
A Critical Distinction: Design (Specification) vs. Execution (Run) times

• The computational elements of Anylogic support both design & execution time presence & behaviour
  – Design time: Specifying the model
  – Execution time (“Runtime”): Simulating the model
• It is important to be clear on what behavior & information is associated with which times
• Generally speaking, design-time elements (e.g. in the palettes) are created to support certain runtime behaviors
A Familiar Analogy

• The distinction between model design time & model execution time is like the distinction between

  – Time of Recipe Design: Here, we’re
    • Deciding what exact set of steps we’ll be following
    • Picking our ingredients
    • Deciding our preparation techniques
    • Choosing/making our cooking utensils (e.g. a cookie cutter)

  – Time of Cooking: When we actually are following the recipe
    • A given element of the recipe may be enacted many times
      – One step may be repeated many times
      – One cookie cutter may make many particular cookies
Cooking Analogy to an Agent Class: A Cookie Cutter

- We only need one cookie cutter to bake many cookies
- By carefully designing the cookie cutter, we can shape the character of many particular cookies
- By describing an Agent class at model design time, we are defining the cookie cutter we want to use
This defines the visual elements to be used for this object when it is displayed at runtime.

**Common Agent-Class Elements**

These introduce “methods” (“functions”) that include some Java code.

- CirclePerimeterColorFromState
- CirclePerimeterWidthFromState
- ReactivationRateCoefficientForCKDStage
- ReactivationRateForCKDStage
- getDegree

These “parameters” specify static agent characteristics:
- Sex
- Ethnicity
- DaysPerTimeUnit
- MeanDaysToNaturallyClearInfection

These describe the agent state & behaviour – the mechanisms that will govern agent dynamics.
This defines the visual elements to be used for this object when it is displayed at runtime.

These introduce “methods” (“functions”) that include some Java code for custom behaviours.

These “parameters” give static characteristics of the agent.

These describe the “behaviours” – the mechanisms that will govern agent dynamics.
Experiments
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
  – Virtual machine arguments

• “Properties” allow one to set the values for each parameter.

• Right click on these & choose “Run” to run such a scenario.
Setting Memory & Virtual Machine Arguments
The Notion of a “Build”

• We prepare a fully specified model to run a simulation using a “build”
  – If all goes well, this translates project to executable Java
  – This may alert you to errors in the project

• A “Compiler” is a tool to convert from a program’s specification (e.g. state charts, Action diagrams, etc.) to a representation that can be executed
  – Normally a compiler is applied to each of several components of a program (e.g. classes)
  – AnyLogic’s “build” process applies a compiler to the components of the AnyLogic model
Cooking Analogy to “Build”ing: Obtaining & Preparing the Ingredients

• Before we can actually realize the recipe, we need to go collect & prepare all ingredients
• We’re not yet cooking, but what we are doing makes the cooking possible
• The “cooking” here is running the model
A Bit on “Java”...

• “Java” is a popular cross-platform “object oriented” programming language introduced by Sun Microsystems

• Anylogic is written in Java and turns models into Java

• 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
  – Exchange messages between Agents
  – Put into place particular initialization mechanisms
  – Collect custom statistics over the population
Stages of the Anylogic Build

Modification Possible

Java Code

Modification Not Possible

JVM Byte Code

Person.class
“Build” Buttons
(One just for this project, one for all projects)
Alternative: Building via Context Menu
Builds Gone Bad: The “Problems View”
Builds Gone Good: Model Execution

• The simulation is running
• Time is advancing in steps or as necessary to handle events
• 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
Press this button to run an experiment (a simulation)

You can pull down the menu to choose which experiment to simulate
Initial Screen: Experiment Set up
(Use to set speed, parameters via UI)

Press this button to switch to the model presentation display
Presentation of the Model “Main” Object in Operation
Network Embedding of Agents

Dynamic color updates via agent logic
Pausing the Model

![SIR Agent Based Simulation](image_url)
Drill Down from the Model to Particular Agents
Runtime View of Particular Agent
(Drill Down from Previous View)

Use this selection to switch between viewing the State of different agents
Customizing the Model Running User Interface
Switching Back to View the Main Object
Controlling Simulation Speed (Speeding up)
Controlling Simulation Speed (Slowing Down)
Toggling between Maximum and a Throttled Speed
Terminating Model Execution
Another Way to Terminate a Simulation

Use this Console “stop” button to terminate the simulation.
Examples of Where to Insert Code

Object Properties

• “Advanced”
Examples of Where to Insert Code

Object Properties

• “General”
Example of Where to Insert Code

Presentations Properties

• “Dynamic” properties of presentation elements (especially of Agents)
Tips to Bear in Mind While Writing Code

• Click on the “light bulb” next to fields to get contextual advice (e.g. on the variables that are available from context

• While typing code, can hold down the Control key and press the “Space” key to request autocompletion
  – This can help know what parameters are required for a method, etc.

• Java is case sensitive!

• Can press “Control-J” to go to the point in Java code associated with the current code snippet

• Can press “build” button after writing snippet to increase confidence that code is understood
Example of Contextual Information
Autocompletion Info (via Control-Space)