

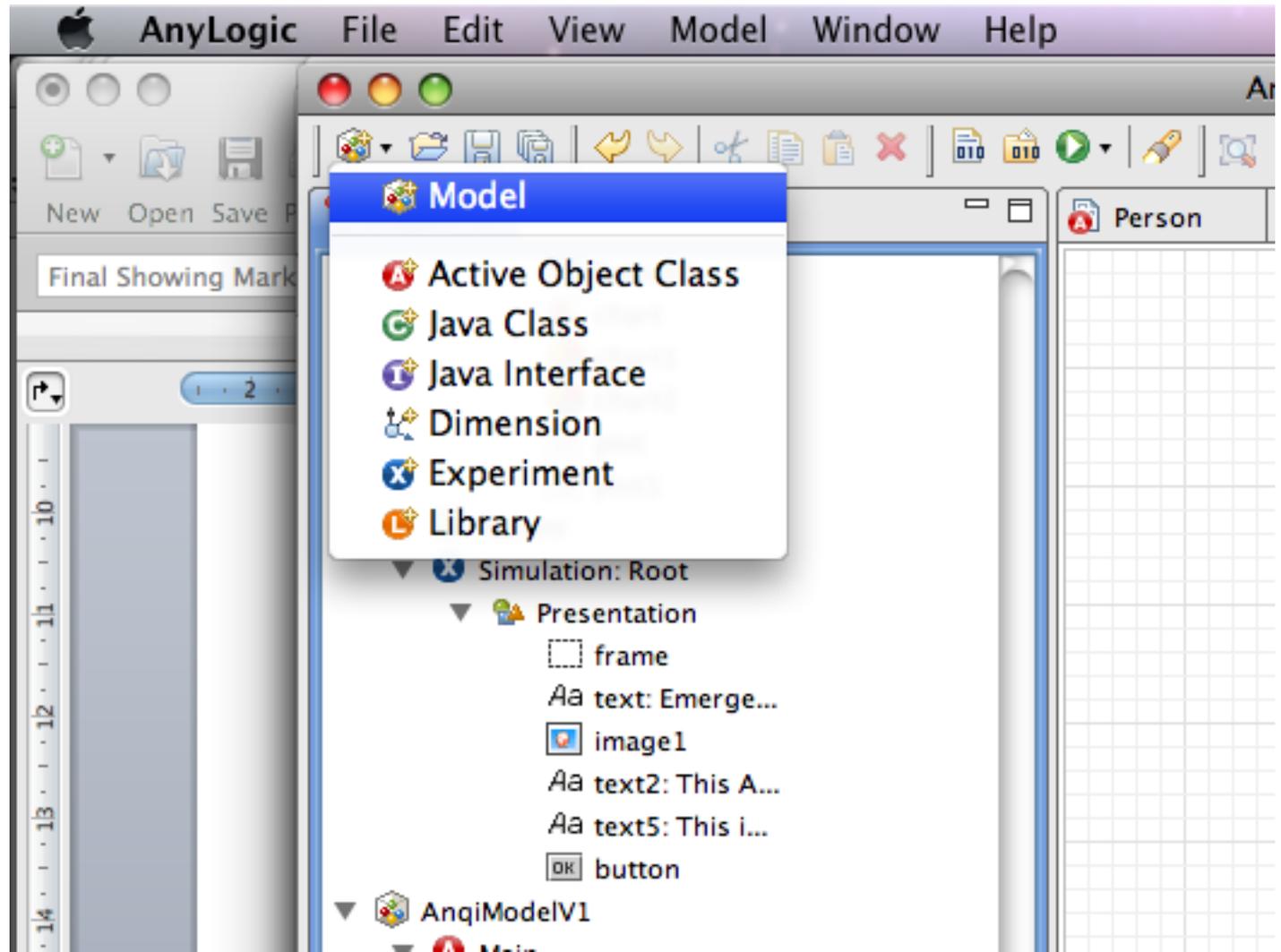
Introduction to the Anylogic Interface by Building Up a Simple Networked Model

Nathaniel Osgood

MIT 15.879

March 2, 2012

Add a New Model Project



Filling in the Model Project Details

New Model

Create a new model

Enter name
MinimalistNetworkABMModel

Model name:

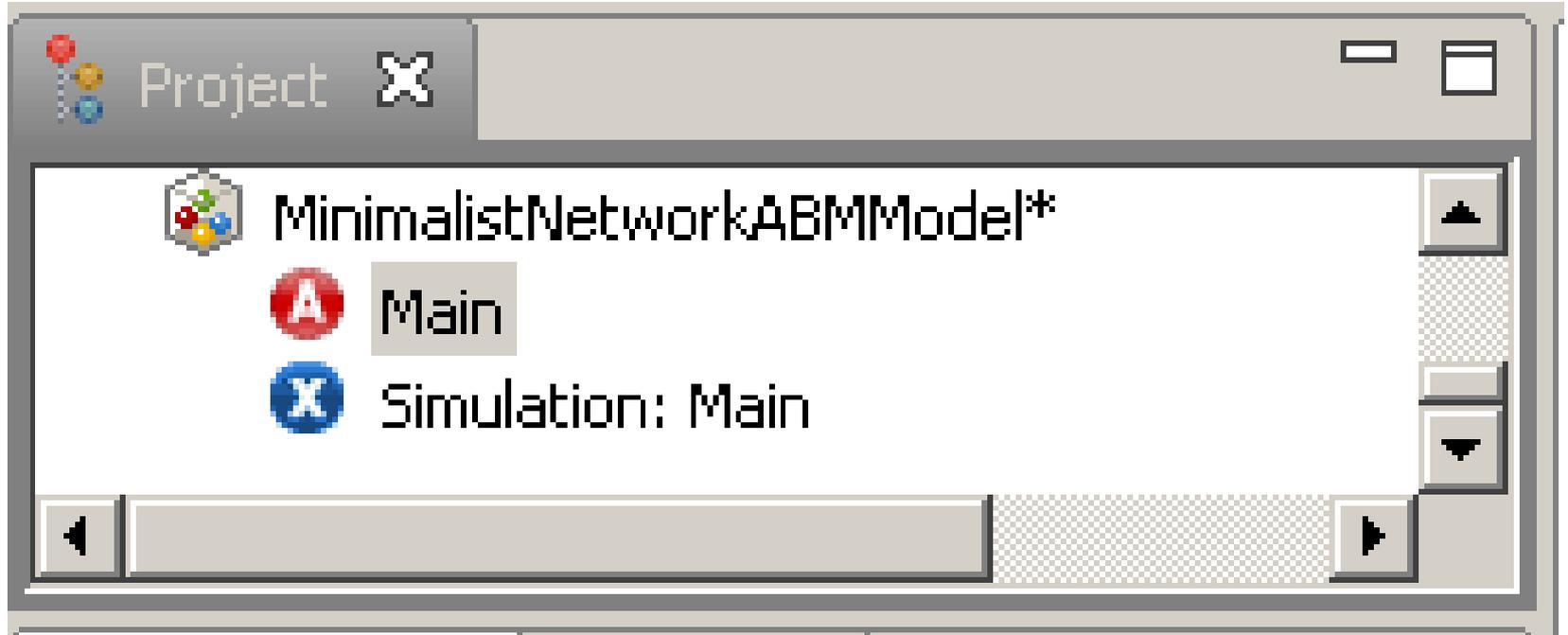
Location:

Java Package:

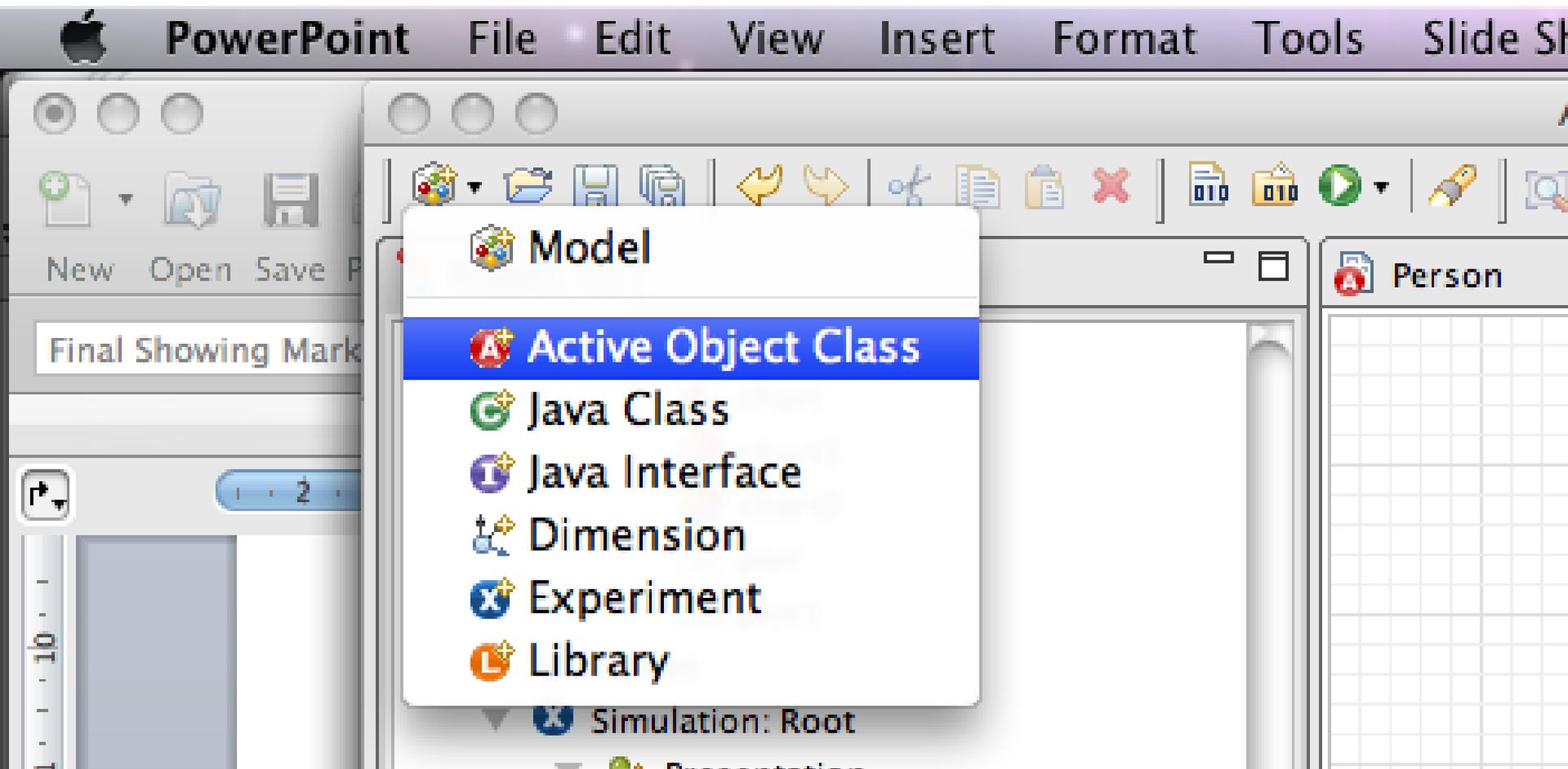
The following model will be created:

```
/Users/osgood/Models/Model/Model.alp
```

Project Window



Add an Active Object Class



Filling in the Agent Class Details

New Active Object Class

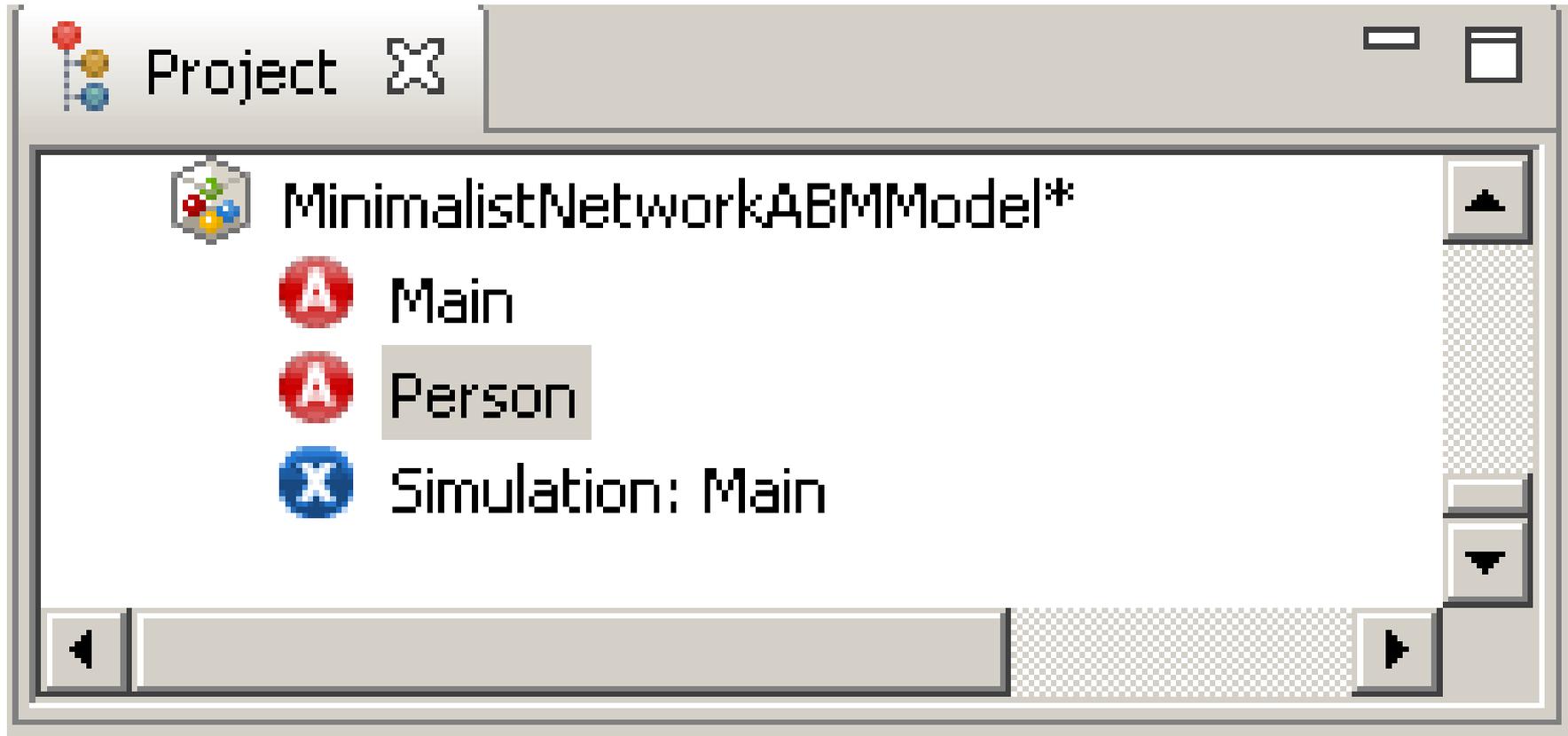
Active Object Class

Name:

Description:

Cancel Finish

Updated Project Window



Recognizing “Person” as an “Agent”

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a grid with a red text overlay: "Check this box (in the 'Properties' Window)". A red arrow points from this text to the "Agent" checkbox in the "Person - Active Object Class" properties window. The "Agent" checkbox is checked, and the "Generic" checkbox is unchecked. The "Person" class name is entered in the "Name" field. The "Ignore" checkbox is also unchecked. The "Startup Code" and "Destroy Code" fields are empty. The "Properties" window has tabs for "Properties" and "Console". The "Project" window shows a tree view with "MinimalistNetworkABMModel*" containing "Main", "Person", and "Simulation: Main". The "Problems" window shows a table with columns "Description" and "Location". The "Palette" window shows a list of model elements: Parameter, Flow Aux Variable, Stock Variable, Event, Dynamic Event, Plain Variable, Collection Variable, Function, Table Function, Port, Connector, Entry Point, State, Transition, Initial State Pointer, Branch, History State, Final State, and Environment. The "Action" window shows buttons for Action, Analysis, Presentation, Connectivity, and Enterprise Library. The status bar at the bottom shows "Selection" and "Cursor: X=0, Y=0".

Check this box
(in the “Properties” Window)

Person - Active Object Class

General Name: Person Ignore

Advanced

Agent Agent Generic

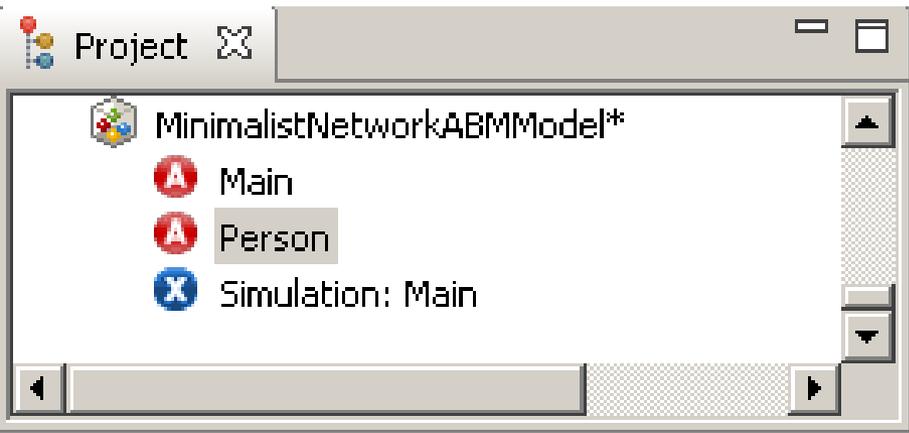
Parameters

Description

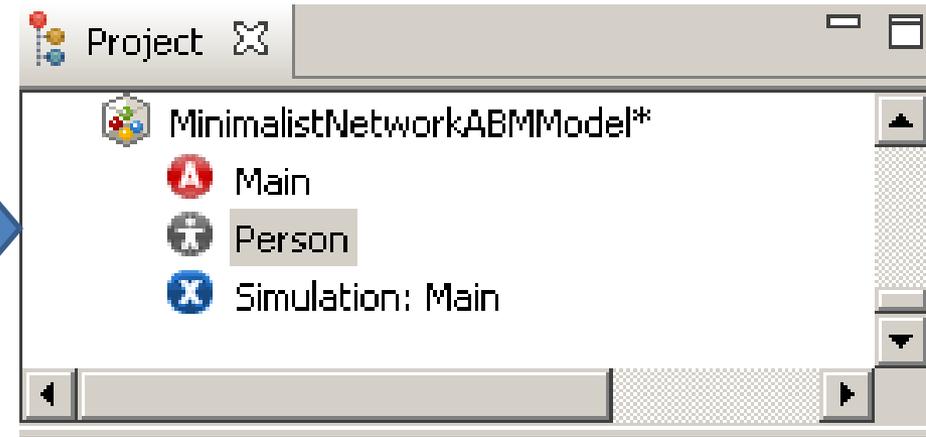
Startup Code:

Destroy Code:

Note Resulting Difference in Project Window



Person as a generic
"Active object"



Person as an agent

Resulting Project Window

The screenshot shows the AnyLogic Advanced software interface. The main window displays a project tree on the left with the following items:

- MinimalistNetworkABMModel*
- Main
- Person
- Simulation: Main

A red circle highlights the 'MinimalistNetworkABMModel*' item, and a red arrow points to it from the text overlay. The text overlay reads:

The "*" means that the model has Changed since the last time it was saved. You should consider saving the model when you see this!

The interface also shows a 'Problems' table with the following content:

Description	Location
By convention, Java type names...	NDOAd...

The 'Properties' window is open, showing the following settings for a selected element:

Multiple selection - 2 elements

General

Name: oval Show Name Ignore Public Icon

Line Color: black Line Width: 1 pt Line Style: solid

The 'Palette' window on the right shows various modeling elements such as Parameter, Flow Aux Variable, Stock Variable, Event, Dynamic Event, Plain Variable, Collection Variable, Function, Table Function, Port, Connector, Entry Point, State, Transition, Initial State Pointer, Branch, History State, Final State, and Environment.

AnyLogic Interface Elements

Note: Double-Clicking on a Tab opens view as Full-Screen

The screenshot shows the AnyLogic Advanced software interface. The main canvas is a grid with a 'Person' element placed on it. The interface is divided into several windows:

- Project window (overview of projects & components):** Located at the top left, it shows a tree view of the project structure. A blue arrow points from the 'Person' element in the tree to the 'Person' element on the canvas.
- Palette window for adding items to canvas:** Located on the right side, it contains a list of modeling elements such as 'Flow Aux Variable', 'Stock Variable', 'Event', 'Dynamic Event', 'Plain Variable', 'Collection Variable', 'Function', 'Table Function', 'Port', 'Connector', 'Entry Point', 'State', 'Transition', 'Initial State Pointer', 'Branch', 'History State', 'Final State', and 'Environment'. A green arrow points from the 'Table Function' element in the palette to the main canvas.
- Problem window (indicates problem "Building"/simulating Model):** Located at the bottom left, it displays a table with columns for 'Description' and 'Location'. A red arrow points from the text 'Building"/simulating Model' to the table.
- Properties window (shows info on selected element in project or palette window):** Located at the bottom center, it shows the properties of the selected 'oval' element, including 'Name', 'Line Color', 'Line Width', and 'Line Style'. A purple arrow points from the text 'shows info on selected element' to the 'Line Color' field.

Other visible windows include 'Problems', 'Properties', and 'Console'. The title bar indicates 'AnyLogic Advanced [EDUCATIONAL USE ONLY]'.

If Windows are Missing...

The screenshot shows the AnyLogic Advanced software interface. The 'View' menu is open, showing a list of views with checkboxes: Project View, Properties View, Problems View, Palette View, Search View, and Help View. A blue arrow points from the text overlay to the 'View' menu. The main workspace displays a grid with a 'Person' window. The bottom panel shows the 'Person - Active Object Class' properties, including fields for Name, Agent, Startup Code, and Destroy Code.

Use the "View" menu to make sure they are enabled (name should be checked)

Description	Location
By convention, Java type names...	NDOAd...

Person - Active Object Class

General Name: Ignore

Advanced

Agent Agent Generic

Parameters

Description

Startup Code:

Destroy Code:

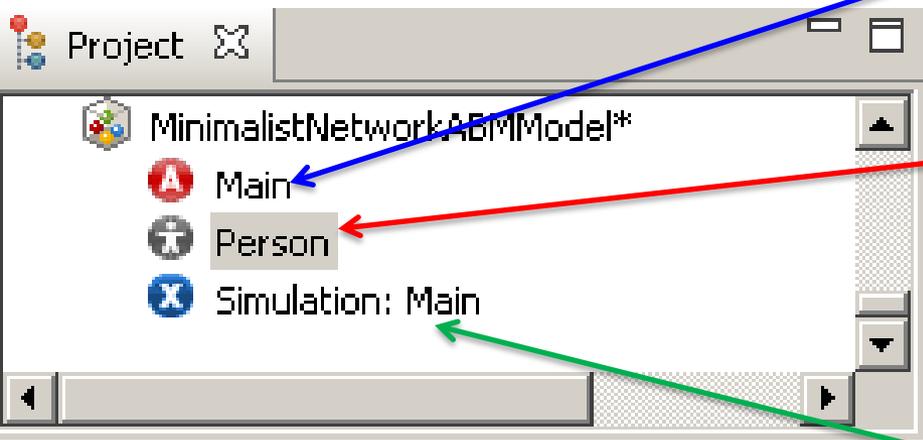
Selection

The “Project” Window

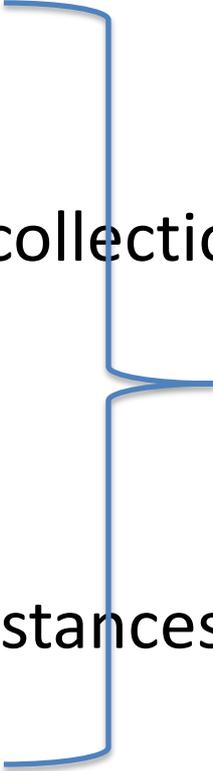
“Main” class
(Defines the “*Stage*” on
which agents circulate)

“Agent” classes
(Define the *actors*)

“Experiment” classes
(Define the
*Assumptions for
Simulation scenarios*)



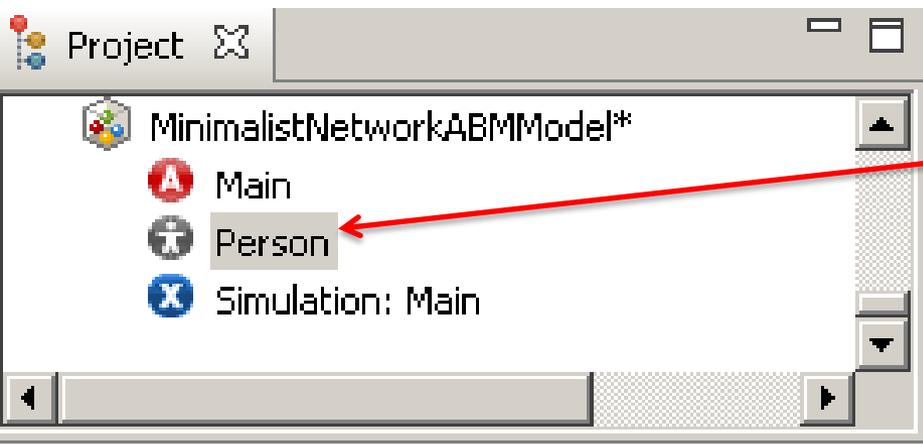
Key Customized “Classes”

- 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
- 
- Varieties of “ActiveObject”

Creating a Visual Representation

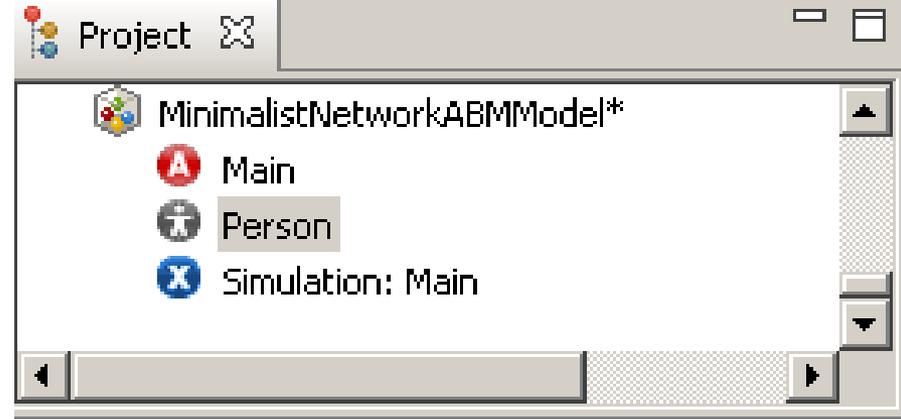
- Agents and Main classes can be associated with visual representations
- These representations can give us a clearer sense of agent behavior

(In case it is not already open)
this is an Agent class, which defines
the Characteristics & Behaviour of
Agent Population Members



“Double Click Here

Agent “Class”



- A particular agent “class” defines “what it means” to be that particular type of agent in our model with respect to characteristics (static [“parameters”], dynamic [“state”]), behaviour & appearance.
 - e.g. a “Person” class defines “Personhood” (“Personness”)
- A given agent “class” will often have many particular representatives (instances) during simulation
 - e.g. While there may be just one “Person” class, there may be many specific People circulating within a model
- Our model may have define types of agents (e.g. Persons, Doctors; Hares & Lynxes), each with one or more accompanying populations

What is a Class?

- A class is like a mold in which we can cast particular objects
 - From a single mold, we can create many “objects”
 - These objects may have some variation, but all share certain characteristics – such as their behaviour
 - This is similar to how objects cast by a mold can differ in many regards, but share the shape imposed by the mould
- In object oriented programming, we define a class at “development time”, and then often create multiple objects from it at “runtime”
 - These objects will differ in lots of (parameterized) details, but will share their fundamental behaviors
 - Only the class exists at development time
- Classes define an interface, but also provide an *implementation* of that interface (code and data fields that allow them to realized the required behaviour)

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
 - Just like the shape of one cookie cutter gets reflected in many particular cookies
 - One agent class has many particular “instances” (Persons)
 - The visual representation of that class gets spread around
 - One visual element in the design of a class can become many during simulation

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 when running the simulation)

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.
- Defining 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”)
 - The populations of agents are just “instances” of this class

Agent Class Defines the Characteristics & Behaviour of Agent Population Members

AnyLogic Advanced [EDUCATIONAL USE ONLY]

File Edit View Model Window Help

100%

Get Support

Project

- MinimalistNetworkABMModel*
- Main
- Person
- Simulation: Main

Problems Search

Description	Location
By convention, Java type names...	NDOAd...

Person

Scroll up and left a bit, until see a Crossing of two (slightly) thicker lines

Properties Console

Person - Active Object Class

General Name: Person Ignore

Advanced

Agent Agent Generic

Parameters

Description

Startup Code:

Destroy Code:

Palette

- Model
- Parameter
- Flow Aux Variable
- Stock Variable
- Event
- Dynamic Event
- Plain Variable
- Collection Variable
- Function
- Table Function
- Port
- Connector
- Entry Point
- State
- Transition
- Initial State Pointer
- Branch
- History State
- Final State
- Environment

Action

Analysis

Presentation

Connectivity

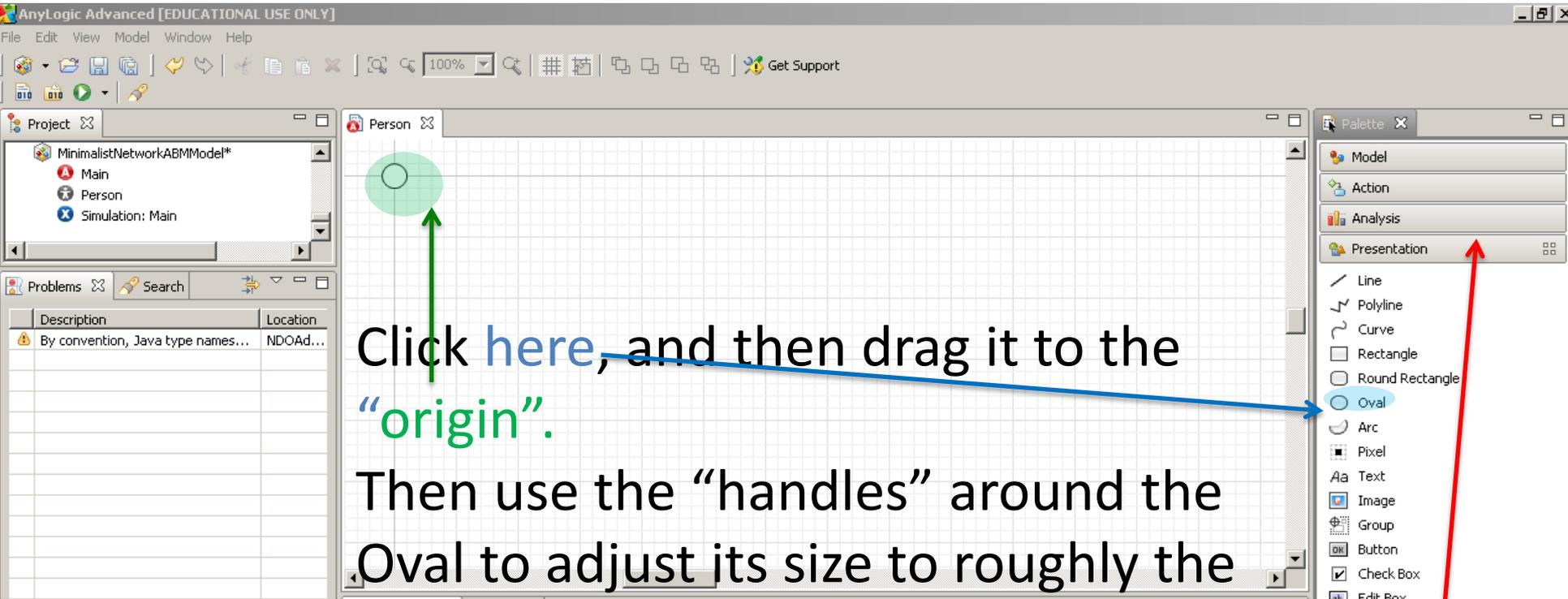
Enterprise Library

More Libraries...

Selection

Cursor: X=0, Y=0

Adding an Oval to Represent Agent

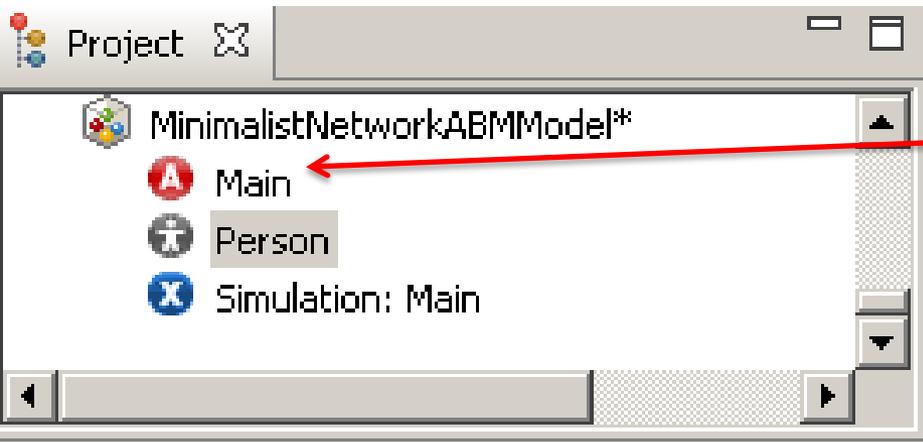


Click **here**, and then drag it to the "origin".

Then use the "handles" around the Oval to adjust its size to roughly the size seen here (radius 1; diameter 2)

Click on the "Presentation" tab in the "Palette" window

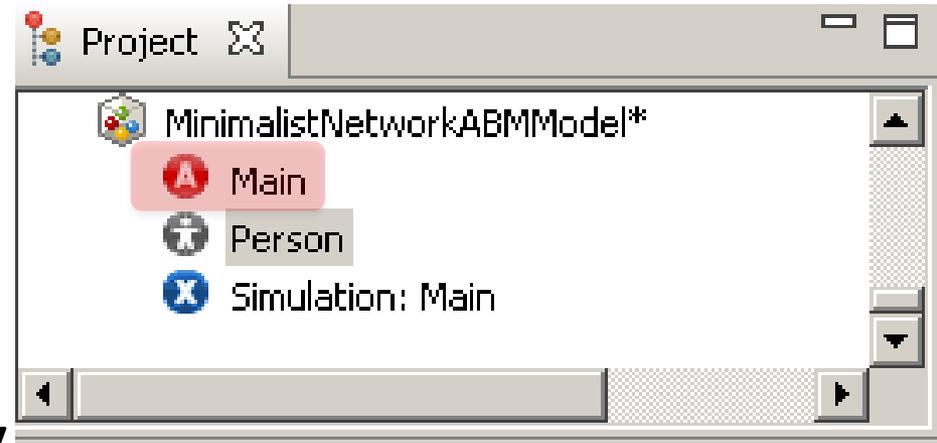
Open Up Canvas for “Main” (In case it is not already open)



Double Click Here

“Main” Class: The “Stage” for Agents

- 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
- We will now add an Agent (Person) population to the “Main” Class



Agent Populations in the Main Class

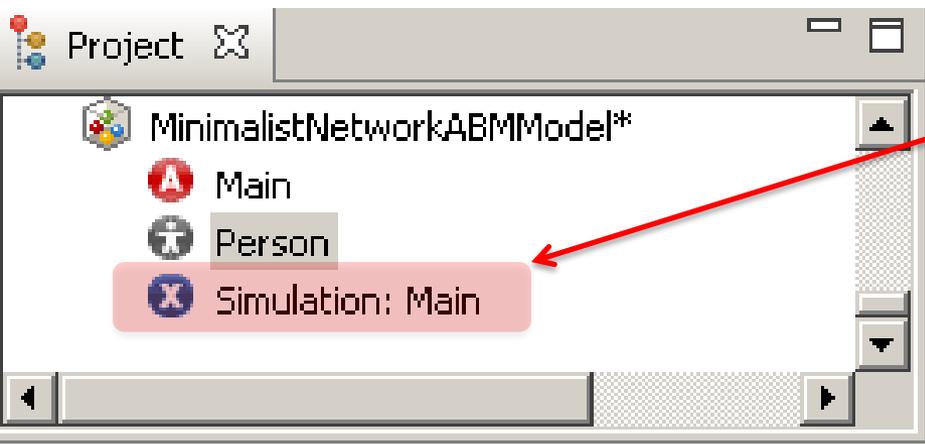
- Through the “Replication” property, the number of these agents can be set
- The “Environment” property can be used to associated the agents with some surrounding context (e.g. Network, embedding in some continuous space, with a neighborhood)
- Statistics can be computed on these agents
- Within the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

Specifying the Population Name & Size

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a model diagram with a population element labeled 'population' and an environment element labeled 'environment'. A red arrow points from the text 'Name: Enter "population" (without quotes!)' to the 'Name' field in the 'population - Person' properties window, which contains the text 'population'. A green arrow points from the text 'Replication (population size): Enter "100" (without quotes!)' to the 'Replication' field in the same properties window, which contains the number '100'. The 'population - Person' properties window is open, showing the following fields:

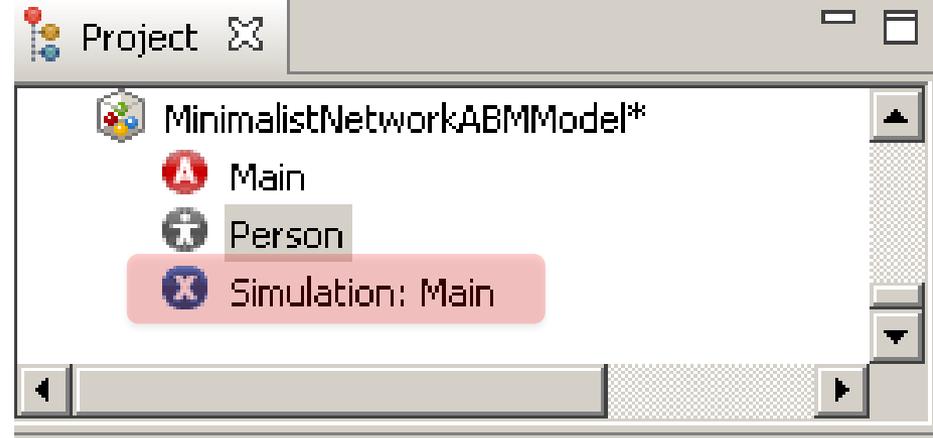
- Name: population
- Type: Person
- Package: minimalistnetworkabmmodel
- Environment: environment
- Replication: 100

Additional fields in the 'General' tab include 'Show Name' (checked), 'Ignore' (unchecked), 'Public' (unchecked), and 'Show At Runtime' (checked). A 'Create Presentation' button is also visible.



A (default) Experiment Specifies assumptions for a particular scenario (e.g. population size, pathogen contagiousness, etc.)

Experiment Classes



- Experiment classes allow you to define & run scenarios in which global “parameters” (i.e. assumption quantities 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

Let's Simulate the Model!

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a grid with a diagram element labeled "population [...]". A context menu is open over the "Simulation: Main" experiment in the Project Explorer, with the "Run" option highlighted in green. A red arrow points from the text "Right click on Experiment named 'Simulation', and select 'Run'" to the "Simulation: Main" experiment. A green arrow points from the text to the "Run" option in the context menu. The Properties panel at the bottom shows the "Simulation - Simulation Experiment" configuration, including fields for Name, Main active object class (root), and Random number generation options.

Right click on Experiment named "Simulation", and select "Run"

Simulation - Simulation Experiment

General

Name: Simulation Main active object class (root): Main Ignore

Advanced

Model Time

Presentation

Window

Parameters

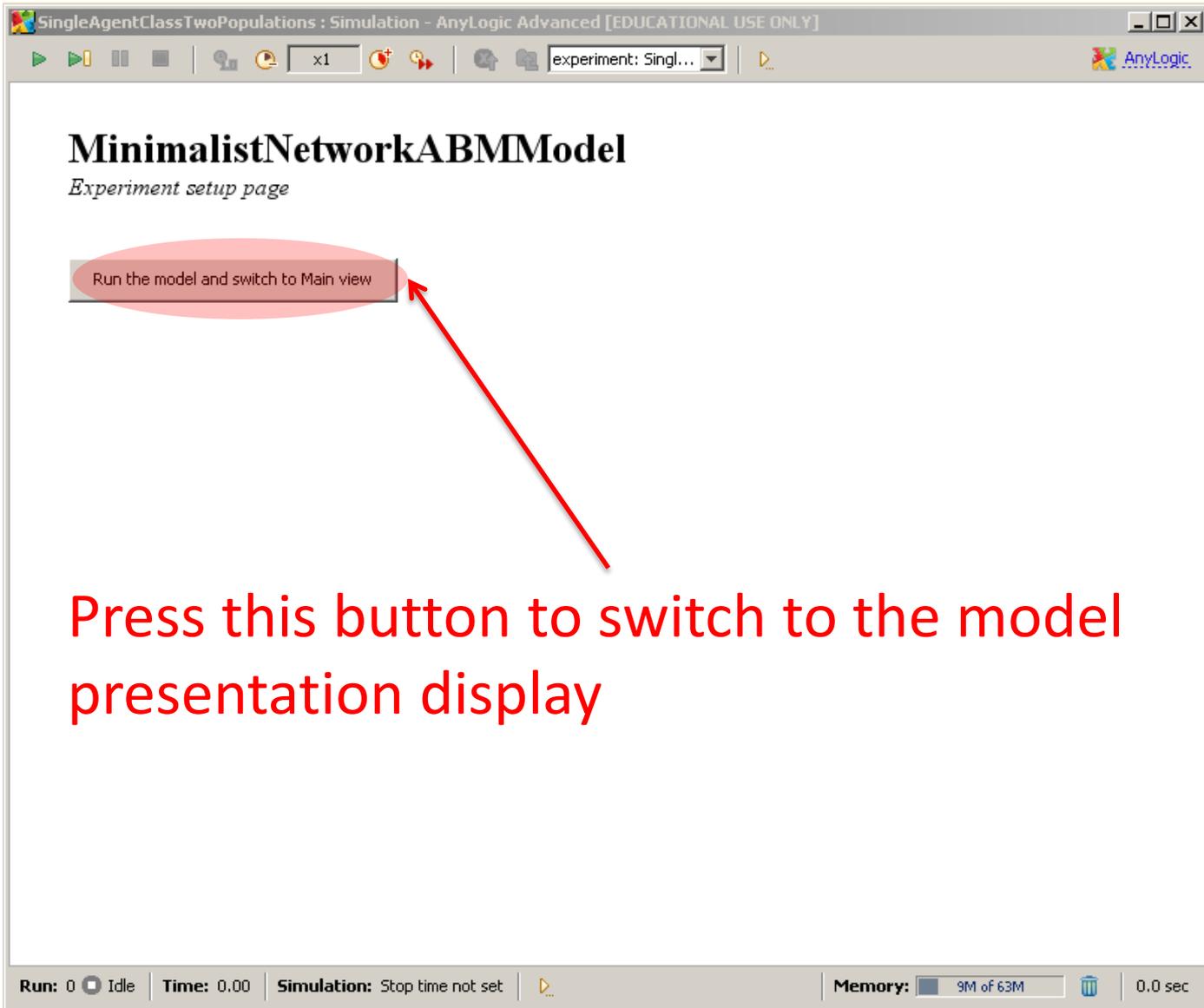
Description

Random number generation:

Random seed (unique simulation runs)

Fixed seed (reproducible simulation runs) Seed Value: 1

Initial Simulation Screen



SingleAgentClassTwoPopulations : Simulation - AnyLogic Advanced [EDUCATIONAL USE ONLY]

experiment: Singl...

MinimalistNetworkABMModel

Experiment setup page

Run the model and switch to Main view

Press this button to switch to the model presentation display

Run: 0 Idle Time: 0.00 Simulation: Stop time not set Memory: 9M of 63M 0.0 sec

An Uninspiring Display

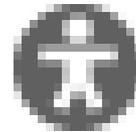
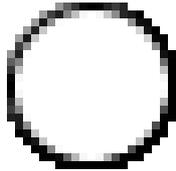
The screenshot shows the AnyLogic simulation interface. The title bar reads "SingleAgentClassTwoPopulations : Simulation - AnyLogic Advanced [EDUCATIONAL USE ONLY]". The toolbar includes play, pause, and stop buttons, a zoom level of "x1", and a location dropdown set to "root:Main". A red oval highlights the toolbar. A green box labeled "population Person [100]" is shown in the top-left corner of the simulation area, with a green arrow pointing to it from the text "Our population has size 100". A red arrow points from the text "All agents (Persons) in population are identical – and are clustered up here!" to the toolbar. The status bar at the bottom shows "Run: 0 Running", "Time: 24.20", "Simulation: 24%", "Memory: 9M of 63M", and "25.1 sec".

Our population has size 100

All agents (Persons) in population are identical – and are clustered up here!

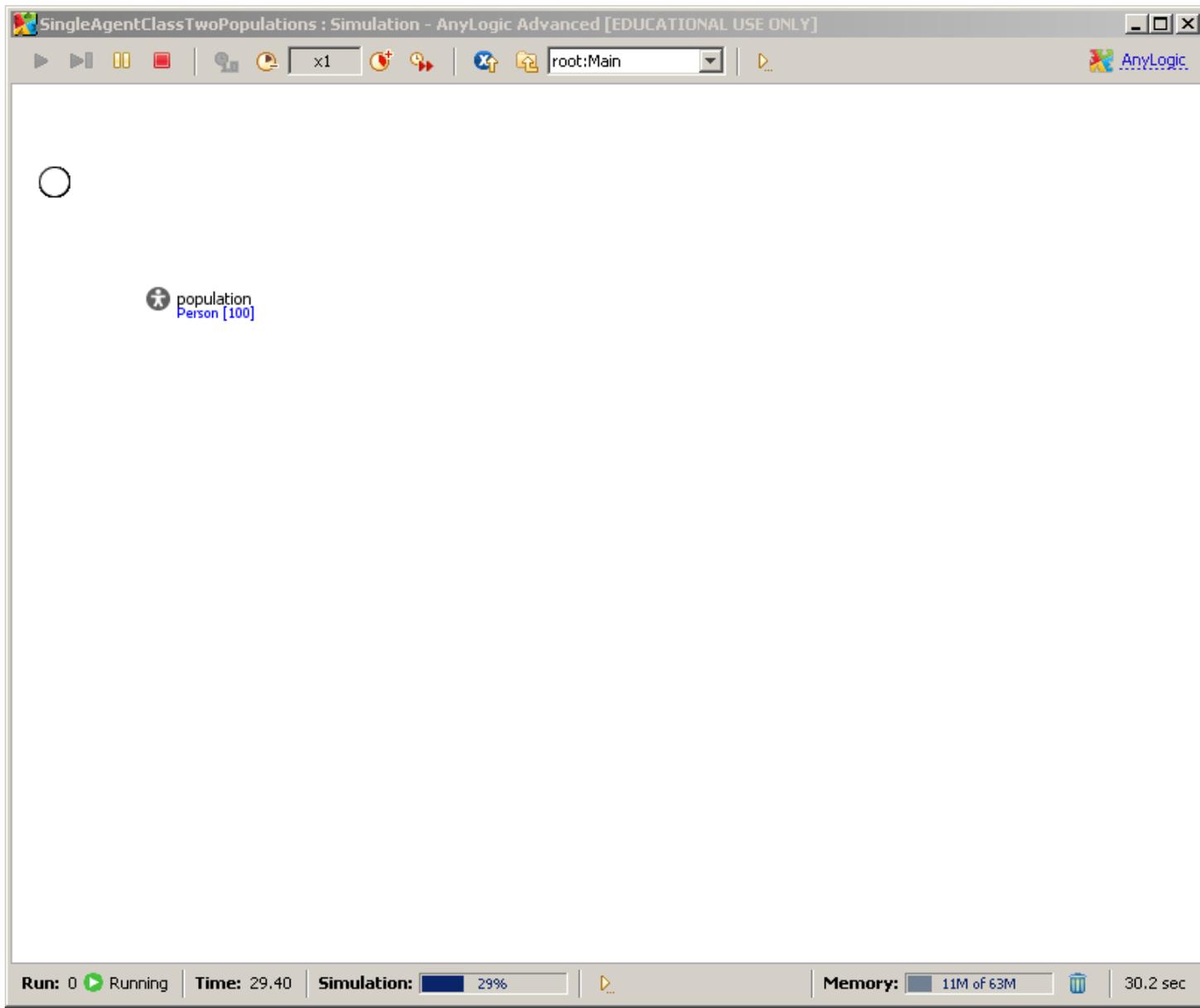
Run: 0 ▶ Running | Time: 24.20 | Simulation: ■ 24% | Memory: ■ 9M of 63M | 25.1 sec

A Magnified View

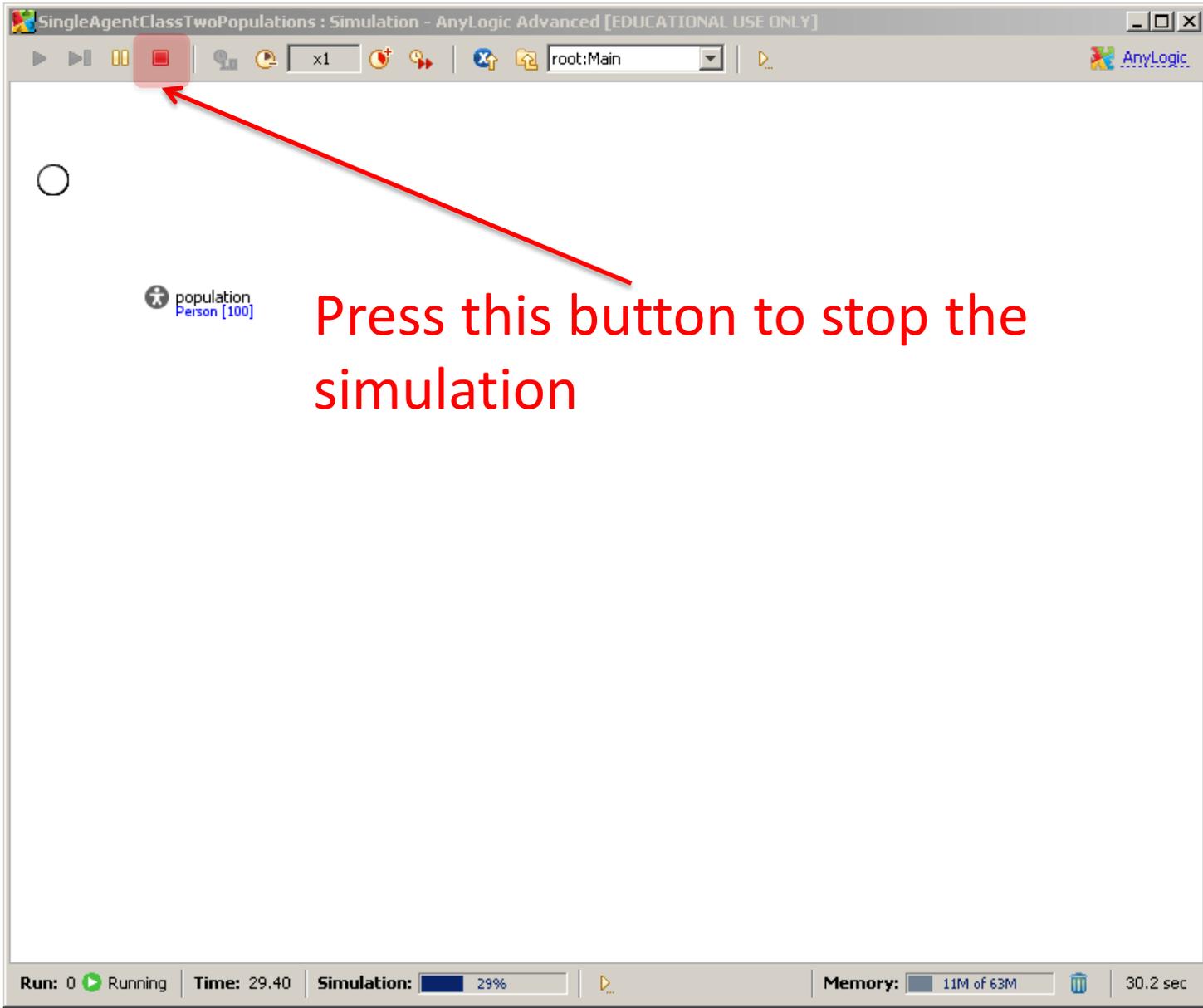


population
Person [100]

“Right Click” & Drag to “Pan” (“Pull”) viewer



Stop Simulation



The screenshot shows the AnyLogic simulation interface. The title bar reads "SingleAgentClassTwoPopulations : Simulation - AnyLogic Advanced [EDUCATIONAL USE ONLY]". The toolbar contains various control icons, with the red square stop button highlighted by a red arrow. The main workspace is mostly empty, with a small circle on the left and a "population Person [100]" label. The status bar at the bottom shows "Run: 0" with a green play icon and "Running", "Time: 29.40", "Simulation: 29%" with a blue progress bar, "Memory: 11M of 63M" with a blue progress bar, and "30.2 sec".

Press this button to stop the simulation

Agent Populations Live in Main Class

- 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
- Within the Main class, you can create representations of subpopulations by dragging from an Agent class into the Main class area

From “Model” Area of “Palette” Window Add an “Environment” to the Model

The screenshot displays the AnyLogic Advanced software interface. The main canvas shows a model with a 'population' entity and an 'environment' entity. The 'Palette' window on the right contains various model components, with the 'Environment' component highlighted in blue. A red arrow points to the 'Model' label in the Palette, and a blue arrow points to the 'Environment' component. A green arrow points to the 'environment' entity on the canvas.

Click on the “Model” label in the “Palette” window

1) Click here (“Environment”)

2) Click somewhere on the canvas

Properties Console
population - Person

General Name: population Show Name Ignore Public Show

Parameters

Statistics Type: Person

Description Package: minimalistnetworkabmodel

Environment:

Replication: 100

Action
Analysis
Presentation
Connectivity
Enterprise Library
More Libraries...

Tell the Population to let the Environment Control its Location

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a grid with two entities: 'population [...]' and 'environment'. A hand icon is positioned over the 'population' entity, indicating it is selected. The 'Properties' panel at the bottom is open for the 'population - Person' entity. The 'Environment' field is highlighted in red, showing the value 'environment'. The 'Palette' on the right lists various model components, including 'Environment'.

AnyLogic Advanced [EDUCATIONAL USE ONLY]

File Edit View Model Window Help

100%

Get Support

Project

- MinimalistNetworkABMModel*
- Main
- Person
- Simulation: Main

Problems Search

Description	Location
By convention, Java type names...	NDOAd...

Person Main

population [...]

environment

Properties Console

population - Person

General

Name: Show Name Ignore Public Show

Parameters

Type:

Statistics

Package:

Description

Environment:

Replication:

Palette

Model

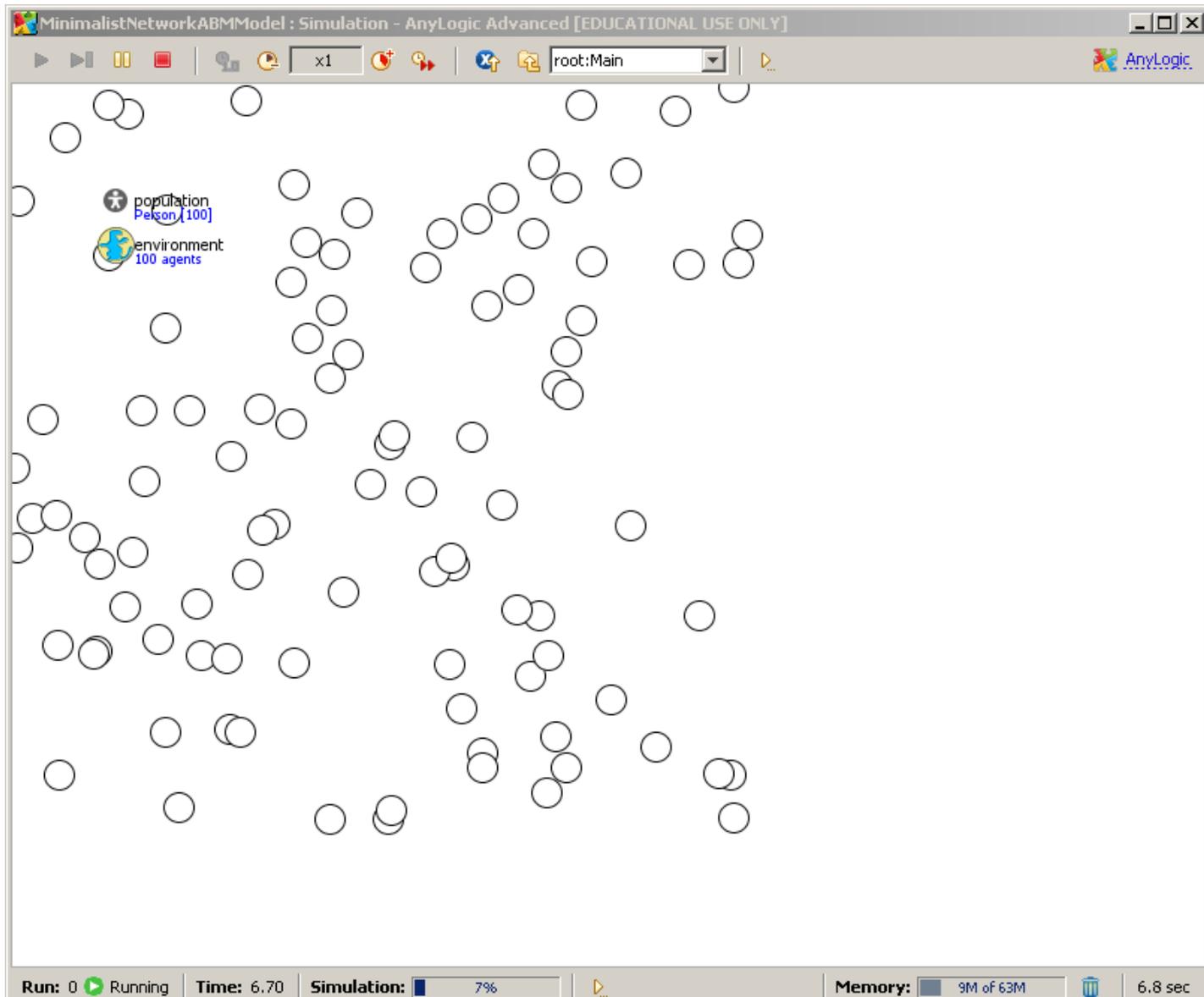
- Parameter
- Flow Aux Variable
- Stock Variable
- Event
- Dynamic Event
- Plain Variable
- Collection Variable
- Function
- Table Function
- Port
- Connector
- Entry Point
- State
- Transition
- Initial State Pointer
- Branch
- History State
- Final State
- Environment

Action

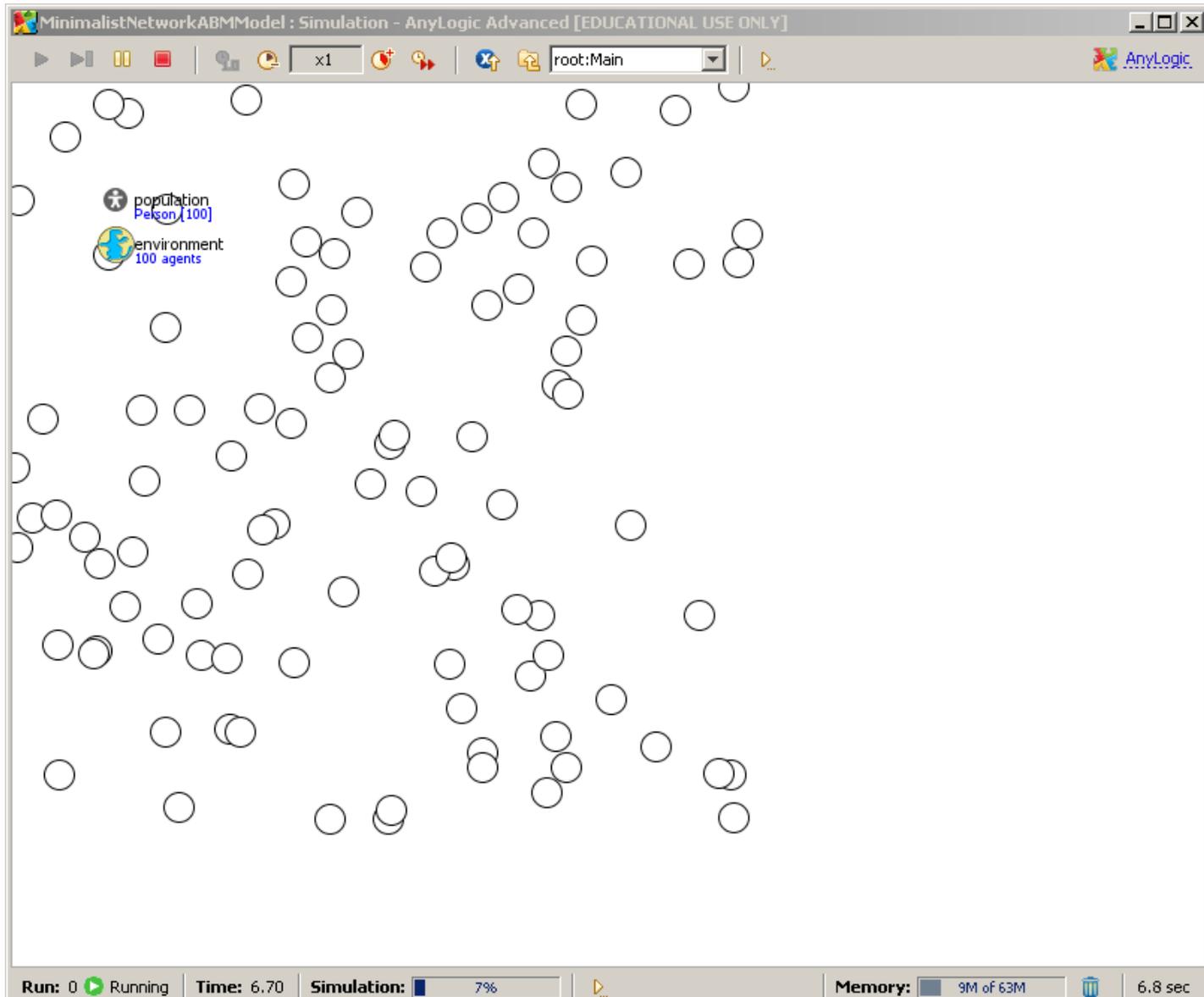
Analysis

Presentation

Run the Model: Environment Distributes Agents Around Space



Run the Model: Environment Distributes Agents Around Space



Recall: A Familiar Analogy

- The distinction between model design time & model execution time is like the distinction between
 - Time of Recipe Design: Here, we're
 - Time of Cooking: When we actually are following the recipe

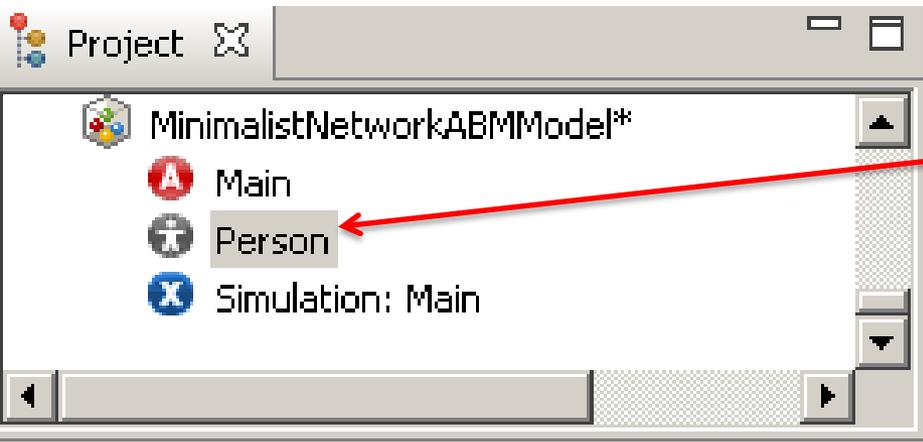
The Notion of a “Build”

- 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

Open Up Canvas for “Person” (In case it is not already open)



“Double Click Here

Let's Place the Agents in a Network

- Steps
 - Tell the Environment that we want to situate the agents in a (here, distance-based) network
 - Specify the attributes of the network (here, the distance threshold up to which agents are considered connected)
 - Give agents a way of appearing visually connected

Setting Network Type in the Environment

Open “Main”, Click on “environment”, and go to the “Advanced” tab in “Properties” window

The screenshot displays the AnyLogic Advanced interface. The main workspace shows a grid with a 'population' agent and an 'environment' agent. The 'environment' agent is selected, and the 'Properties' window is open, showing the 'Advanced' tab. The 'Network type' is set to 'Distance based' and the 'Connection range' is set to 50. A red arrow points from the text 'Set “Network type” to “Distance based”' to the 'Network type' dropdown menu. A blue arrow points from the text 'Set “Connection range” to 50' to the 'Connection range' input field.

Set “Network type” to “Distance based”

Set “Connection range” to 50

environment - Environment

General

Space type: Continuous Discrete GIS

Advanced

Description

Width: 500

Height: 500

Columns: 100

Rows: 100

Neighborhood type: Moore

Layout type: User-defined Apply on startup

Network type: Distance based Apply on startup

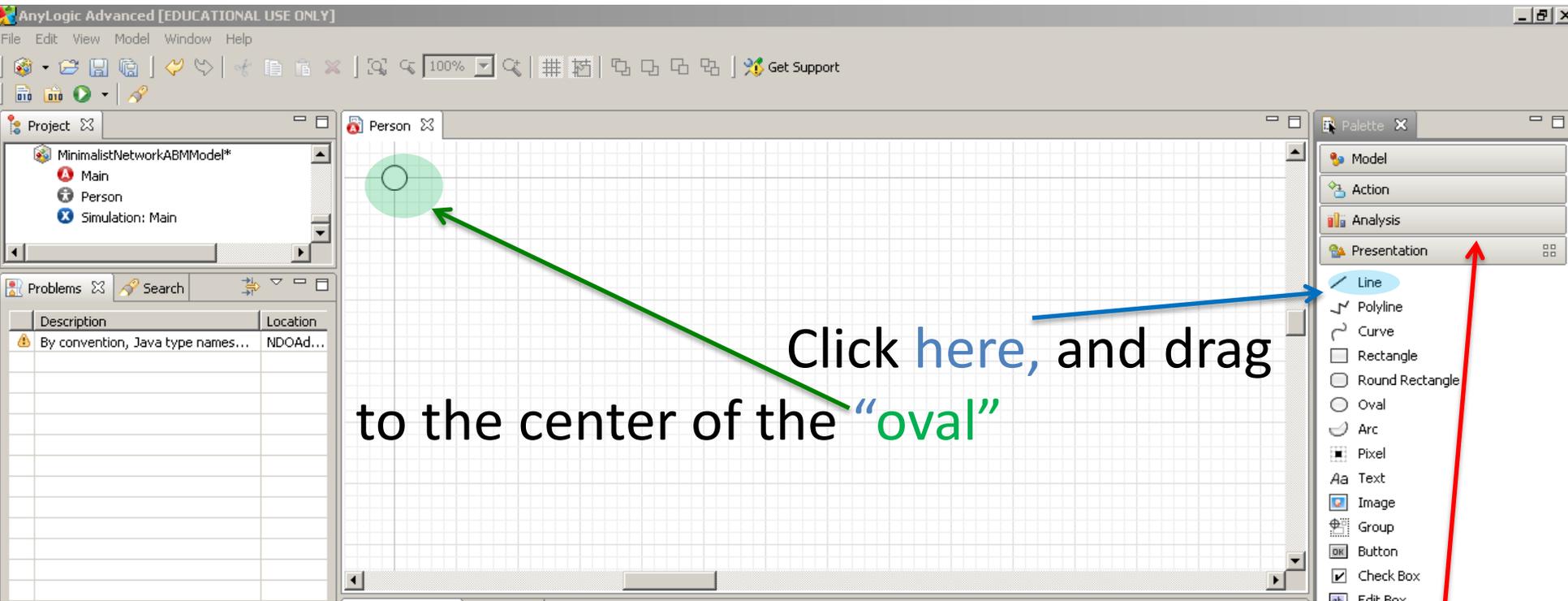
Connections per agent: 2

Connection range: 50

Let's Place the Agents in a Network

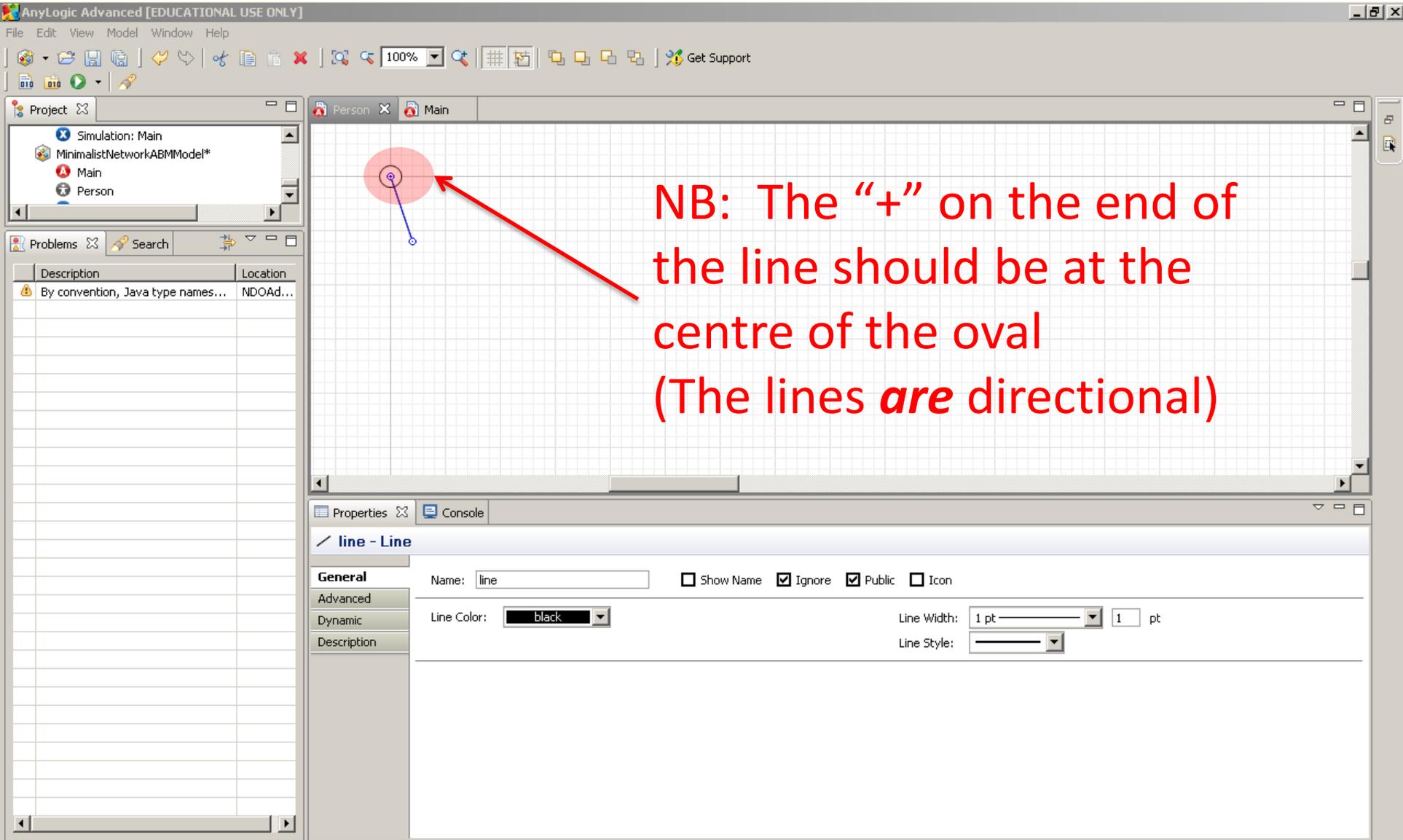
- Steps
 - √ Tell the Environment that we want to situate the agents in a (here, distance-based) network
 - √ Specify the attributes of the network (here, the distance threshold up to which agents are considered connected)
 - Give agents a way of appearing visually connected

Adding a Line to Represent Connections



Click on the "Presentation" label in the "Palette" window

Adding a Line to Represent Connections



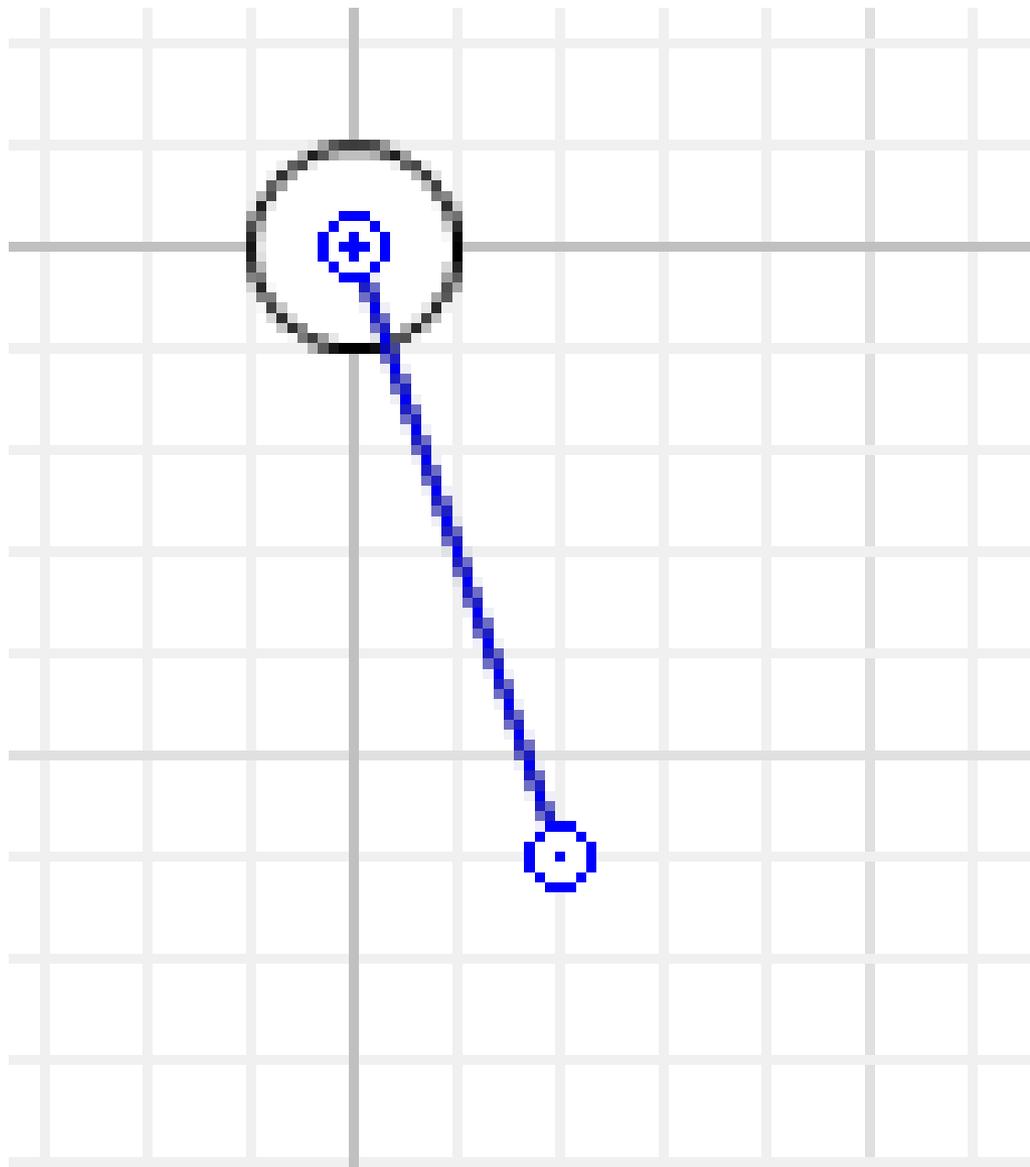
The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a diagram with a red oval and a blue line extending from its center. A red arrow points to the blue line with the following text:

NB: The “+” on the end of the line should be at the centre of the oval
(The lines *are* directional)

The interface includes a Project browser on the left showing a simulation named 'MinimalistNetworkABMModel*' with components 'Main' and 'Person'. Below it is a Problems table with one entry: 'By convention, Java type names...' located at 'NDOAd...'. The Properties panel at the bottom shows the selected 'line - Line' object with the following settings:

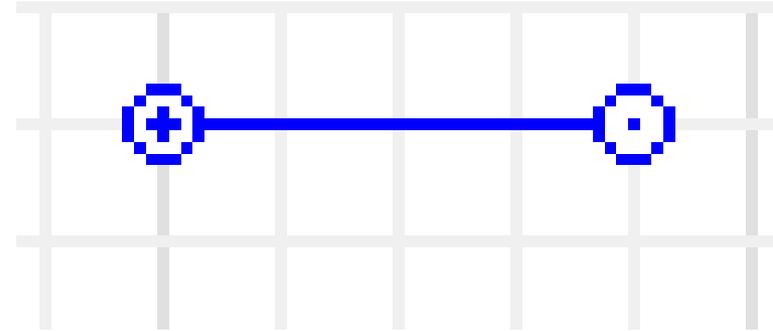
Property	Value
Name	line
Show Name	<input type="checkbox"/>
Ignore	<input checked="" type="checkbox"/>
Public	<input checked="" type="checkbox"/>
Icon	<input type="checkbox"/>
Line Color	black
Line Width	1 pt
Line Style	Solid

Close-Up

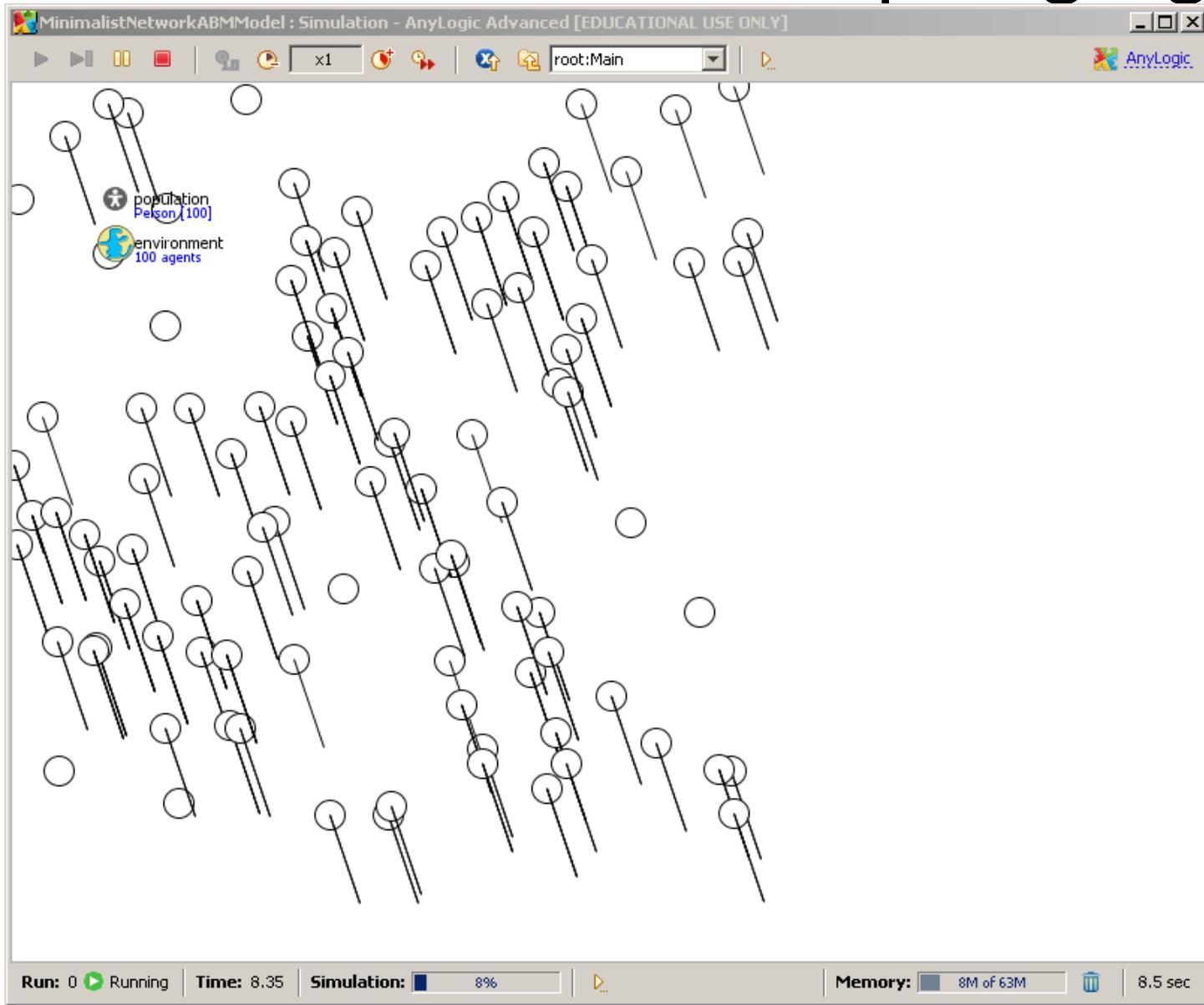


If you are Initially Unsuccessful in Placing the Line in the Circle ...

- Place the line on the canvas
- A line looks like this:
 - Pull the end with a small “+” into the very center of the circle
 - The “dotted” end can dangle



Run the Model: An Uninspiring Sight



We Need to Multiply & Adjust the Lines

- Right now, there is only 1 line per agent
- We need
 - One line per connection between one person and another
 - The lines to connect the two persons

Duplicating the Lines for Each Connection

The screenshot shows the AnyLogic Advanced software interface. The main workspace displays a simulation model with a grid background. A blue circle highlights a line object, and a blue arrow points to it from the text "Make sure the line remains selected (Click on it if not!)". The Properties window at the bottom shows the configuration for the selected line object, with the "Dynamic" tab selected. The "Replication" property is set to `this.getConnectionsNumber()`, which is highlighted with a red oval and a red arrow pointing from the text "“Replication” should read “this.getConnectionsNumber()” (i.e. we seek 1 line per connection)".

Make sure the line remains selected (Click on it if not!)

“Replication” should read **“this.getConnectionsNumber()”** (i.e. we seek 1 line per connection)

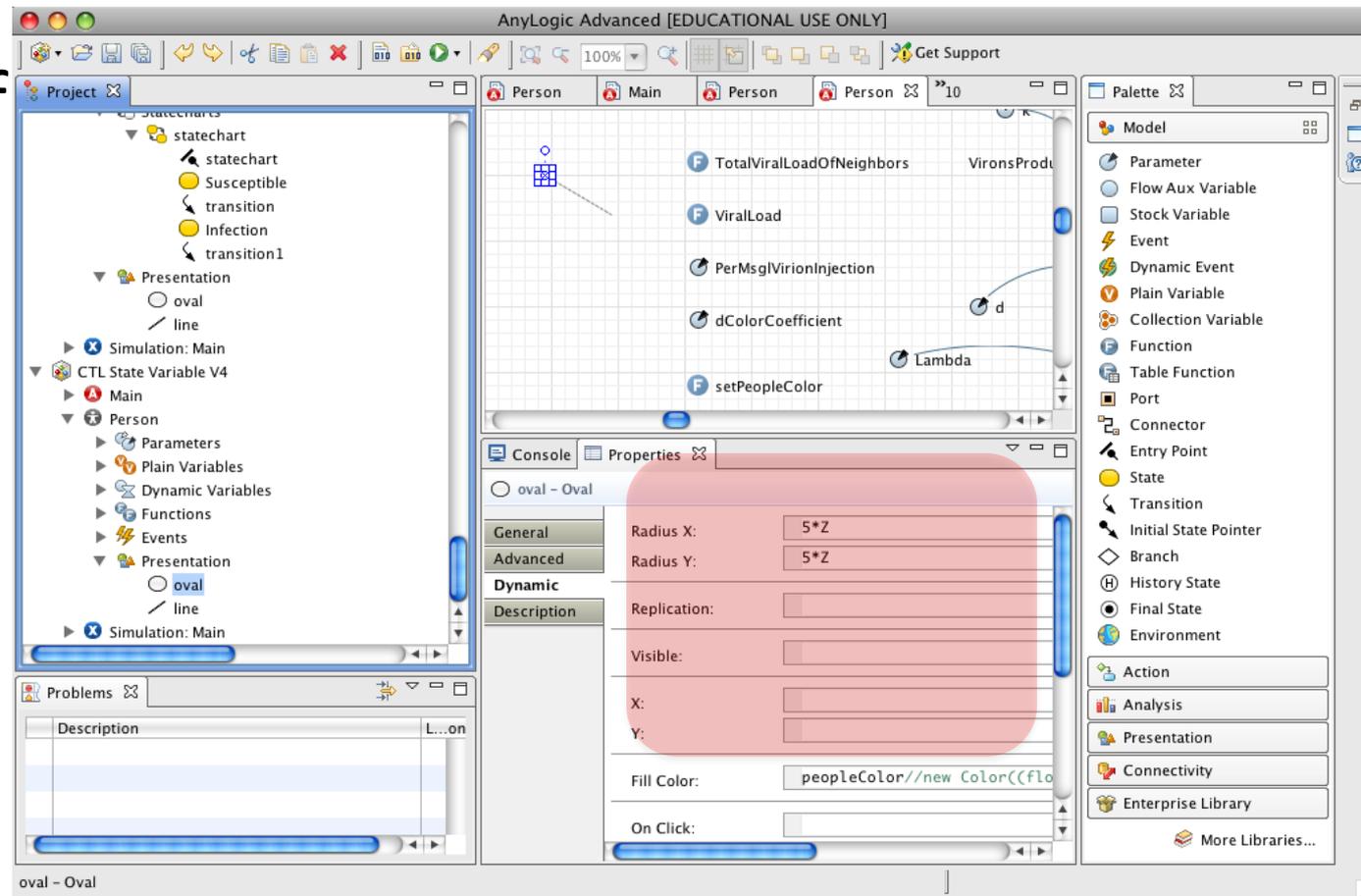
Select the “Dynamic” tab!

Description	Location
By convention, Java type names...	NDOAd...

Properties	Console
line - Line	
General	Replication: <code>this.getConnectionsNumber()</code>
Advanced	Visible: <input type="checkbox"/>
Dynamic	X: <input type="checkbox"/>
Description	Y: <input type="checkbox"/>
	On Click: <input type="checkbox"/>
	Rotation: <input type="checkbox"/>

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

The screenshot displays the AnyLogic software interface, titled "AnyLogic Advanced [EDUCATIONAL USE ONLY]". The main workspace shows a simulation model with a grid background. The model contains several elements: a "people [...]" entity, an "environment" entity, and a "CountInfectious" state variable. The "CountInfectious" state variable is highlighted, and its properties are shown in the "Properties" window below. The "Properties" window has tabs for "General", "Parameters", "Statistics", and "Description". The "Statistics" tab is active, showing the following information:

- Name: CountInfectious
- Type: Count Sum Average Min Max
- Expression: Use: item: the embedded object
- Condition: item.statechart.isStateActive(Person)

The "Console" window is also visible, showing the text "people - Person". The "Palette" window on the right lists various model elements, including Parameter, Flow Aux Variable, Stock Variable, Event, Dynamic Event, Plain Variable, Collection Variable, Function, Table Function, Port, Connector, Entry Point, State, Transition, Initial State Pointer, Branch, History State, Final State, and Environment. The "Project" window on the left shows a hierarchical structure of the simulation model, including "Simulation: Root", "Ophthalmology Department", "MainPhase1", "MainPhase2", "MainPhase3", "Simulation: MainPhase3", "NetworkSEIR", "Main", "Parameters", "Plain Variables", "Functions", "Environments", "environment", "Embedded Objects", "Analysis Data", "datasetAbsolutePrevalence", "CountInfectious", "Presentation", "Person", "Plain Variables", "Statecharts", "statechart", and "statechart".

Autocompletion Info (via Control-Space)

The screenshot displays the AnyLogic University software interface. The main workspace shows a statechart diagram with a 'statechart' node and two states: 'Susceptible' and 'Infected'. The 'Person' class is selected in the Project browser. The Properties window shows the 'Person - Active Object Class' with various parameters and methods. A tooltip is visible over the 'receiveMessage' method, providing details about its signature and parameters.

Person - Active Object Class

General: X: [], Y: []

Advanced: []

Agent

Preview: []

Description: []

Movement parameters:

Velocity: []

Rotation: []

On arrival: []

On message received: []

statechart.receive

On before step: []

On step: []

receiveMessage

public boolean receiveMessage(int msg)

Same as receiveMessage(Object msg) but with an integer as message

Parameters:

msg - the integer posted to the statechart

Press 'Tab' from proposal table or click for focus

Selection X=246, Y=275

Known AnyLogic Bug – Save, Quit & Restart AnyLogic

The screenshot displays the AnyLogic Advanced [EDUCATIONAL USE ONLY] interface. The main workspace shows a diagram with a blue line connecting two circular nodes. The interface includes a menu bar (File, Edit, View, Model, Window, Help), a toolbar with various icons, and several panels:

- Project:** MinimalistNetworkABMModel
 - Main
 - Person
 - Simulation: Main
- Problems:** Search
 - Description: By convention, Java type names... Location: NDOAd...
- Properties:** Console
 - line - Line
 - General: Replication: `this.getConnectionNumber ()`
 - Advanced: Visible: package
 - Description: X: Y: On Click:

We need to Multiply & Adjust the Lines

- Right now, there is only 1 line per agent
- We need
 - √ One line per connection between one person and another
 - The lines to connect the two persons
 - This requires *each line* (i.e. the line associated with *each connection*) to be adjusted so that it goes between the position of the current agent (Person) and the position of the other person to whom the connection relates

Scroll Down to “dX” Property

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a diagram with a blue line connecting a lightbulb icon to a 'Person' object. A red text overlay reads: "Clicking on 'Lightbulb' gives hint, noting that the 'index' is variable is defined as the connection number for this Person." A red arrow points from this text to the 'dX' property field in the 'line - Line' properties panel. The 'dX' field is highlighted with a red oval and contains the text: "Use: index; index of replicated Line".

AnyLogic Advanced [EDUCATIONAL USE ONLY]

File Edit View Model Window Help

100%

Get Support

Project

MinimalistNetworkABMModel*

- Main
- Person
- Simulation: Main

Problems Search

Description	Location
By convention, Java type names...	NDOAd...

Person Main

Palette

- Model
- Action
- Analysis
- Presentation

- Line
- Polyline
- Curve
- Rectangle
- Round Rectangle
- Oval
- Arc
- Pixel
- Text
- Image
- Group
- Button
- Check Box
- Edit Box
- Radio Buttons
- Slider
- Combo Box
- List Box
- File Chooser
- Progress Bar
- CAD Drawing
- GIS Map

Connectivity

Enterprise Library

More Libraries...

Properties Console

line - Line

General On Click:

Advanced Rotation:

Dynamic

Description

Scale X:

Scale Y:

dX:

dY:

Line Color:

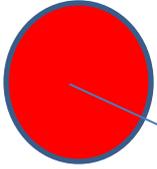
Line Width:

Line Style:

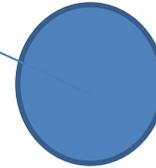
Use: index; index of replicated Line

Geometry to Connect Agents

Index Agent A



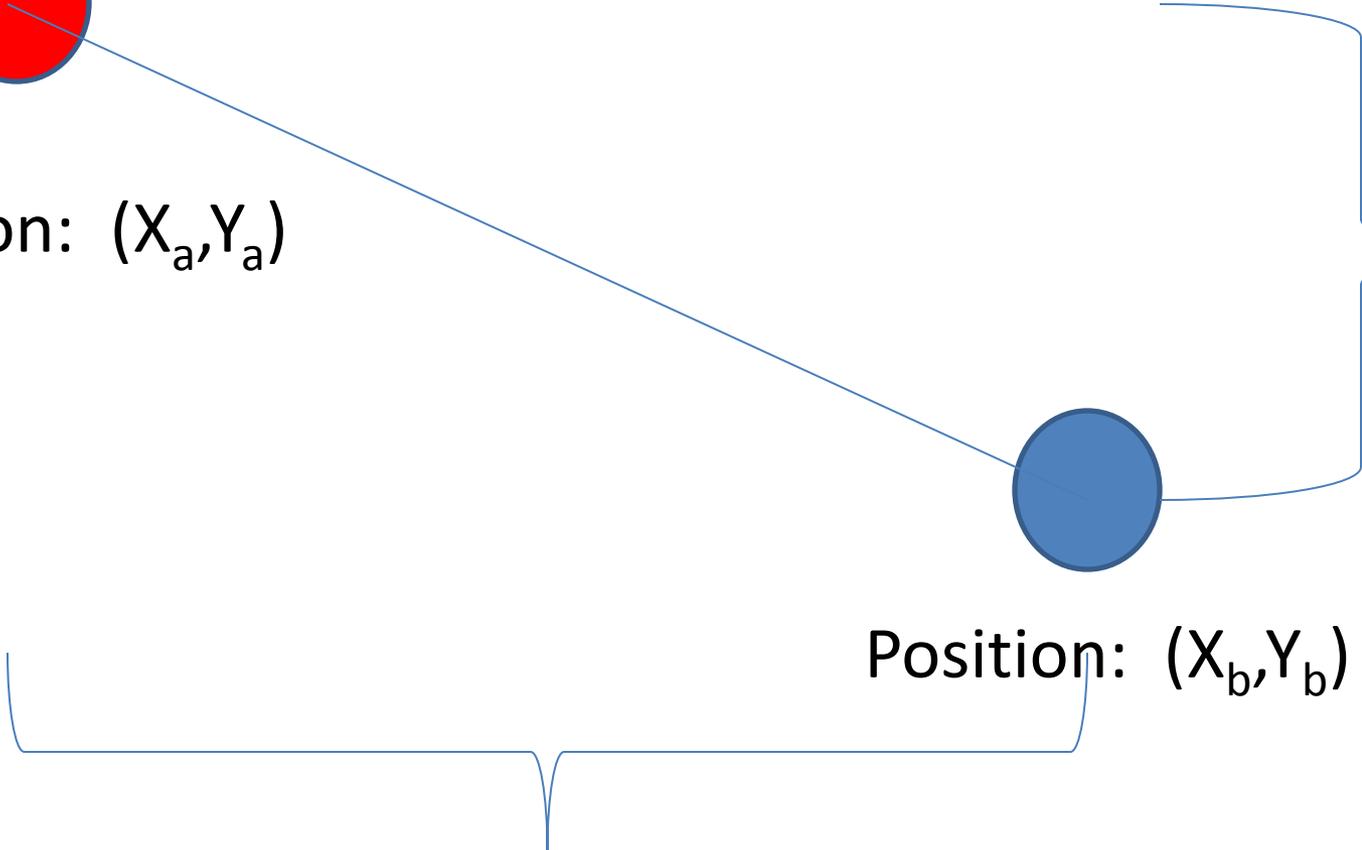
Position: (X_a, Y_a)



Position: (X_b, Y_b)

$Y_b - Y_a$

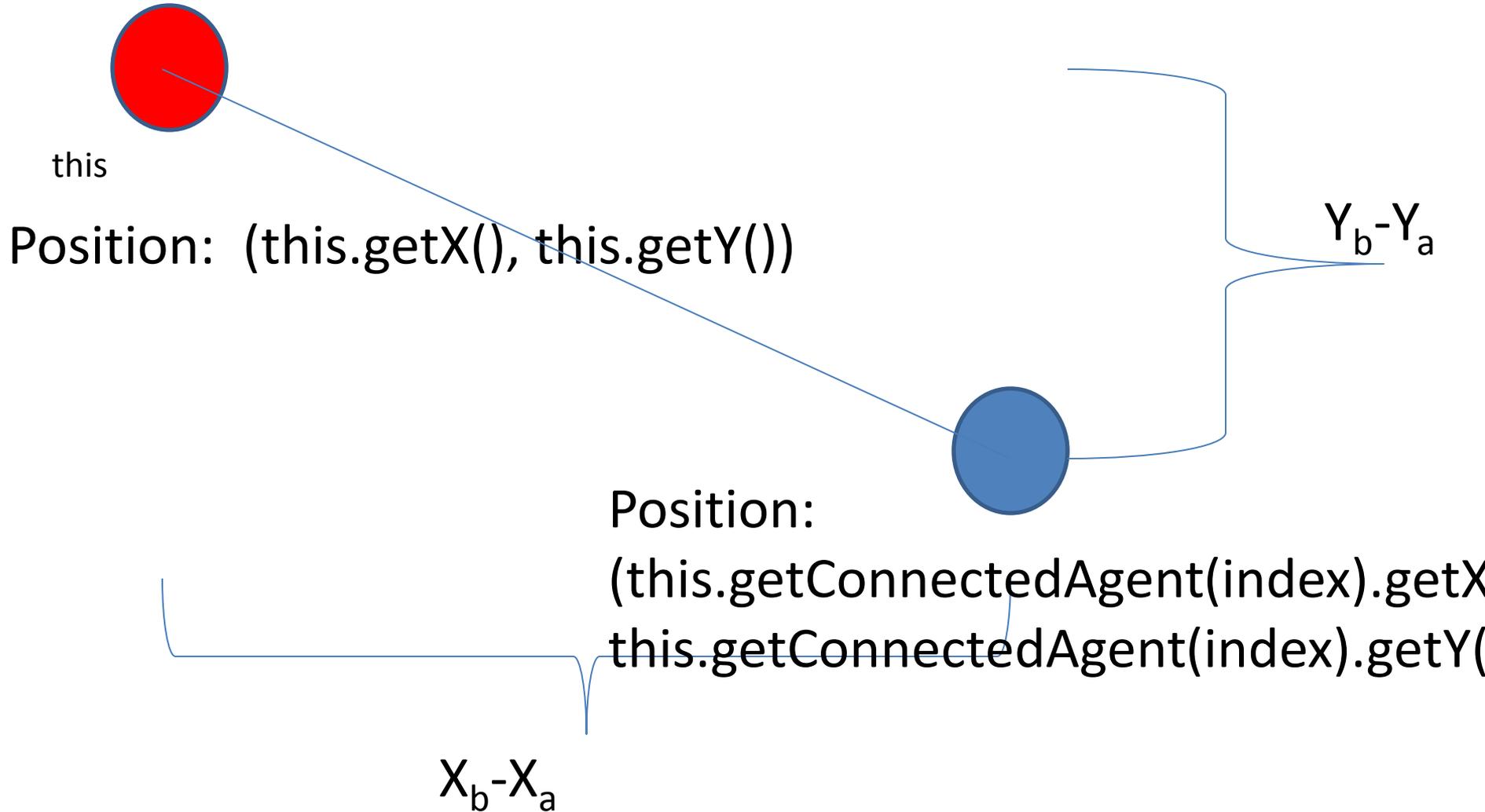
$X_b - X_a$



A Few Useful Points

- Agents are “objects” in Java (self-contained structures with state & behavior)
- The reference to the current agent is called “this”
- If we have a reference, we can request information from it by “calling” a method on it
- To get a reference to the i^{th} person connected to “this”, we call “this.getConnectionedAgent(i)”
- To get the X or Y position of “this”, we “call” “this.getX()” or “this.getY()”, respectively

Geometry to Connect Agents



Setting Per-Instance Additional Properties

The screenshot shows the AnyLogic Advanced software interface. The main workspace displays a model diagram with a 'Person' agent. A red arrow points from the text 'Formula for "dX" should be' to the 'dx:' property field in the 'line - Line' properties window. Another red arrow points from the text 'Formula for "dY" should be' to the 'dY:' property field in the same window. The properties window also shows 'Scale X:', 'Scale Y:', 'Line Color:', 'Line Width:', and 'Line Style:' fields.

Project: MinimalistNetworkABMModel
Main
Person
Simulation: Main

Problems: Search

Description: Location

Person

Formula for "dX" should be
`this.getConnectionedAgent(index).getX() - this.getX()`

Formula for "dY" should be
`this.getConnectionedAgent(index).getY() - this.getY()`

Properties: Console

line - Line

General

Advanced

Dynamic

Description

dx: `this.getConnectionedAgent(index).getX() - this.getX()`

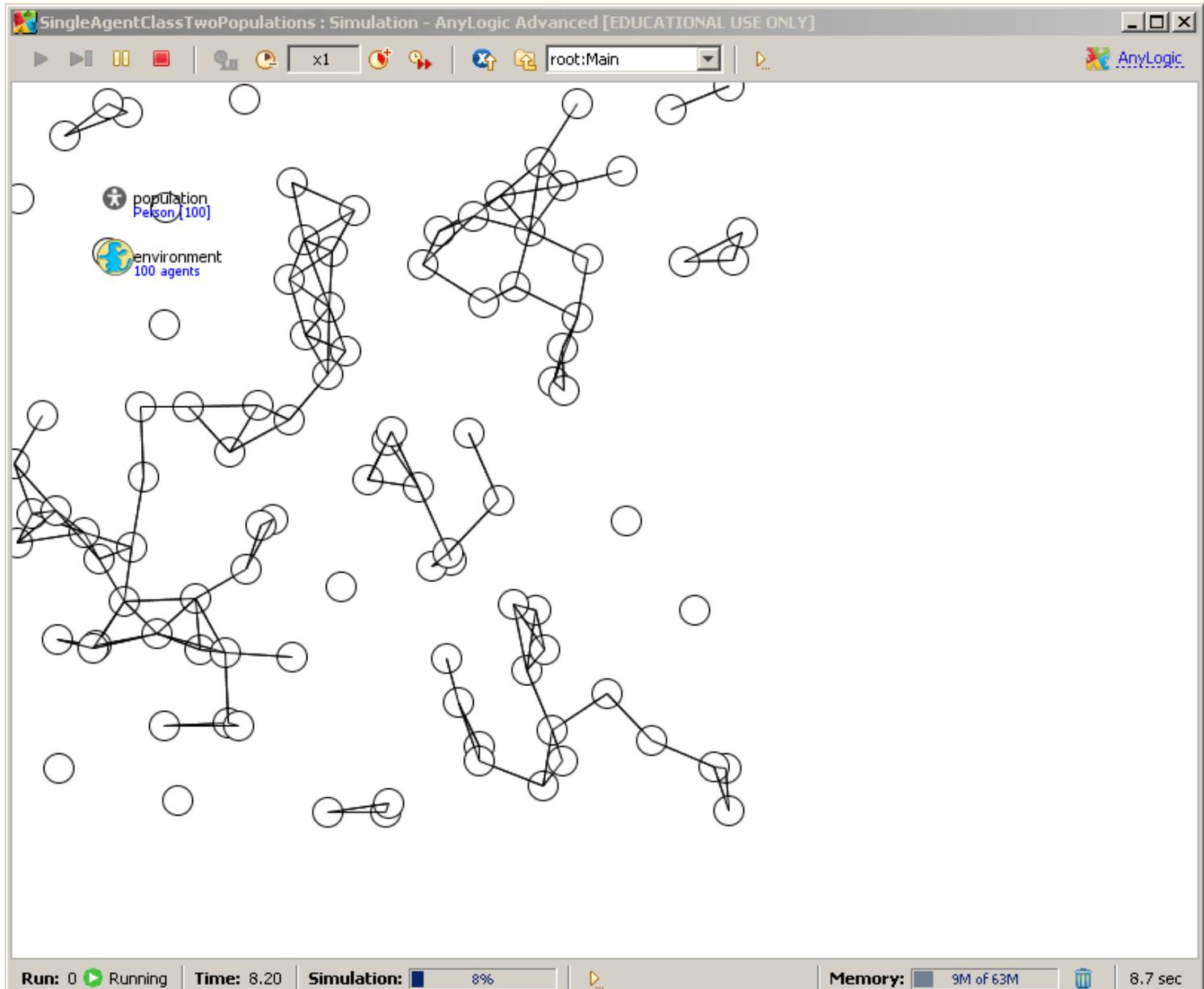
dY: `this.getConnectionedAgent(index).getY() - this.getY()`

Line Color:

Line Width:

Line Style:

Result of Running the Model



AnyLogic: Above & Below the “Hood”

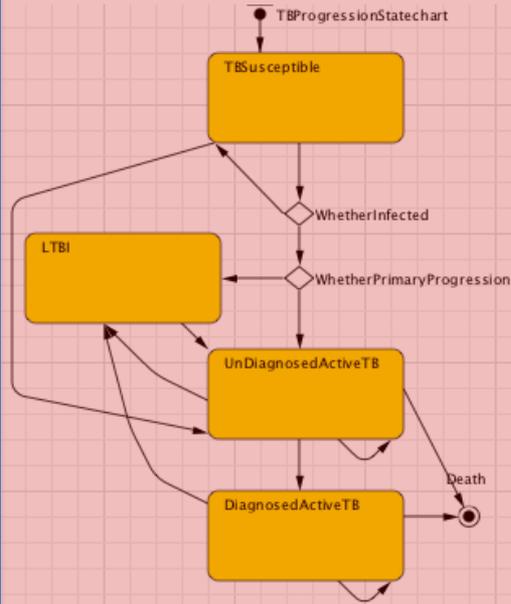
- One of AnyLogic’s greatest strengths is the presence of diverse & powerful *declarative* mechanisms for building models
 - These let you focus on the “what” you are modeling, rather than “how” it will be implemented
 - AnyLogic will take care of figuring out the “*how*”
 - This is in contrast to writing code in a general purpose computer language, which generally requires specifying more of the *how*
- For Anylogic, declarative mechanisms include statecharts, stock & flow diagrams, “action” flow charts & process maps
- Other familiar declarative mechanisms include spreadsheet formulas and stock & flow diagrams.
- For most interactions with AnyLogic, you will be able to specify your intentions using these declarative mechanisms
- On occasion, you will need to write & look at Java code

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



Modification Not Possible

Java Code

```
double initialPrevalenceOfInfection ) {
    if (initialPrevalenceOfInfection == this.initialPrevalenceOfInfection)
        return;
    this.initialPrevalenceOfInfection = initialPrevalenceOfInfection;
    onChange_initialPrevalenceOfInfection();
    onChange();
}

void onChange_initialPrevalenceOfInfection() {
    int index;
    index = 0;
    for ( Person object : Population ) {
        object.set_isInitiallyInfected((uniform() < initialPrevalenceOfInfection));
        index++;
    }
}
```

JVM
Byte
Code

Person.class

“Build” Buttons

(One just for this project, one for all projects)

The screenshot displays the AnyLogic University software interface. The top menu bar includes File, Edit, View, Model, Window, and Help. The toolbar contains various icons, with two 'Build' buttons (one with a single 'B' and one with multiple 'B's) highlighted by red arrows. A blue arrow points to the single 'B' button, and a red arrow points to the multiple 'B's button. The main workspace shows a statechart with two states: 'Susceptible' and 'Infected', connected by arrows. The left sidebar shows a project tree with 'HybridABMNetworkModeling1*' expanded to show 'Main' and 'Person'. The bottom-left pane shows a 'Problems' table with columns for 'Description' and 'Location'. The bottom-right pane shows a 'Properties' window for 'Person - Active Object Class', with fields for 'X:', 'Y:', 'Velocity:', 'Rotation:', and 'On arrival:'. The bottom status bar displays 'Build completed successfully. Time: 3.000 s.' and 'Selection X=83, Y=181'.

Build just this project

Build all projects

Build completed successfully. Time: 3.000 s.

Selection X=83, Y=181

Alternative: Building via Context Menu

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a state transition diagram with states: Susceptible, Infective, NonPregnant, and Pregnant. A context menu is open over the diagram, listing standard file operations such as New, Open, Save, and Build. The 'Build' option is highlighted in red. The 'Build' option is associated with the keyboard shortcut F7. Below the diagram, the 'Console' window shows the model name 'EclipseDebuggingExample - Model' and the file path 'U:\Research\EclipseDebuggingExample\EclipseDebuggingExample.alp'. The right-hand side of the interface features a 'Model' palette with various components like Parameter, Flow Aux, Stock Vari, Event, Dynamic, Plain Vari, Collection, Function, Table Fun, Port, Connector, Entry Point, State, Transition, Initial State, Branch, History St, Final State, and Environment. The bottom of the interface shows a 'Description' window with a table structure.

AnyLogic Advanced [EDUCATIONAL USE ONLY]

File Edit View Model Window Help

100%

Proj... Sea... Main Main Person

EclipseDebuggingExa

- New
- Open... Ctrl+O
- Save Ctrl+S
- Save As...
- Revert
- Close
- Close Others
- Close All
- Cut Ctrl+X
- Copy Ctrl+C
- Paste
- Delete Delete
- Refresh
- Build F7**
- Export...
- Team

circleSize

Susceptible

Infective

Death

appearanceTime

InitialAge

CurrentAge

FinalizeDeath

sex

ethnicity

PregnancyStatus

NonPregnant

Pregnant

FertilityRateAgeSexEthnicity

PerformBirth

EstablishOffspringConnectionsBasedOnMothersConnections

EstablishOffspringLocationBasedOnMothersLocation

Console

EclipseDebuggingExample - Model

Name: EclipseDebuggingExample

Package: abmmmodelwithbirthdeath

File: U:\Research\EclipseDebuggingExample\EclipseDebuggingExample.alp

Description

Description

Model

- Parameter
- Flow Aux
- Stock Vari
- Event
- Dynamic
- Plain Vari
- Collection
- Function
- Table Fun
- Port
- Connector
- Entry Point
- State
- Transition
- Initial State
- Branch
- History St
- Final State
- Environment
- Action
- Analysis
- Presentati..
- Connectivi.
- Enterprise..

Builds Gone Bad: The “Problems View”

The screenshot displays the AnyLogic Advanced [EDUCATIONAL USE ONLY] interface. The main workspace shows a statechart diagram for a TB progression model. The diagram includes states: **TBSusceptible**, **LTBI**, **UnDiagnosedActiveTB**, and **DiagnosedActiveTB**. Transitions are labeled with events like **WhetherInfected** and **WhetherPrimaryProgression**. A **Death** state is also present. A red arrow points from the title to the **UnDiagnosedActiveTB** state in the diagram.

The **Problems View** window is open, showing a list of errors:

- ✗ The constructor DataSet() is undefined
- ✗ Engine.log cannot be resolved
- ✗ mViewer cannot be resolved
- ✗ mViewer cannot be resolved

The **Person - Active Object Class** properties window is also visible, showing the **General** tab with the following code:

```
Imports section:  
  
Extends (single ActiveObject or Agent subclass):  
  
Implements (comma-separated list of interfaces):  
  
Additional class code:  
public static final int MsgInfectiousTBContact=1;  
public static final int MsgForceInitialTBInfection=2;
```

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

Save Away Your Model

- Multiple ways
 - Right click on project name in “Project” window, and choose “Save”
 - If you are currently working on your project, either
 - Press “disk” icon
 - Use “Save” item on “File” menu

