

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

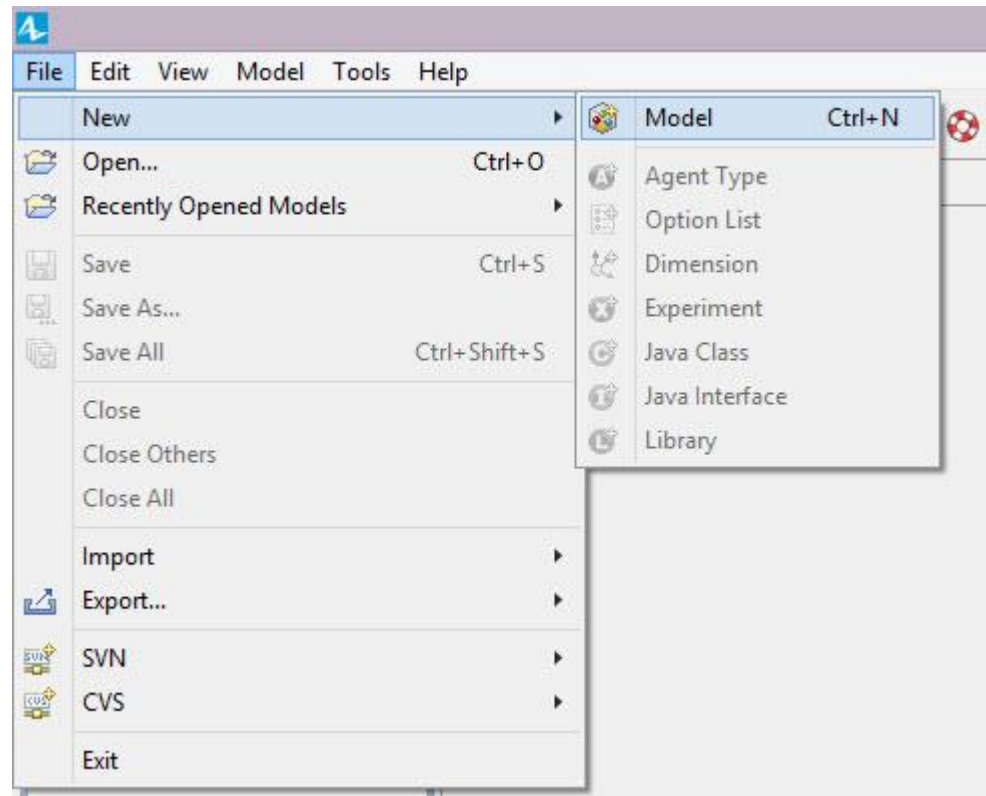
Nathaniel Osgood

Using Modeling to Prepare for Changing Healthcare Needs

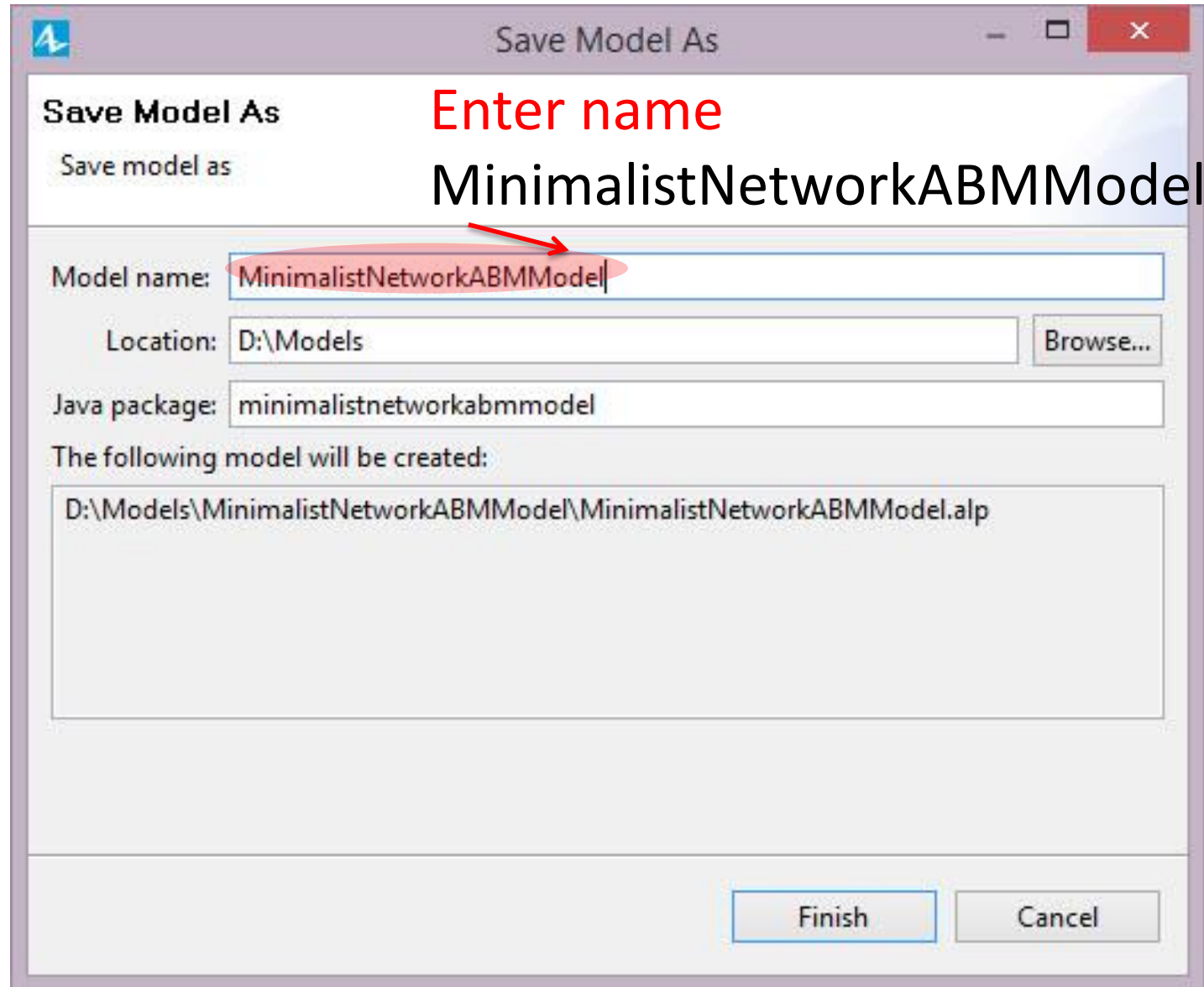
Duke-NUS

April 16, 2014

Add a New Model Project



Filling in the Model Project Details



The screenshot shows the 'Save Model As' dialog box in NetLogo. The window title is 'Save Model As'. Inside, the text 'Save Model As' and 'Save model as' are visible. The 'Model name:' field contains 'MinimalistNetworkABMModel', which is highlighted with a red oval and a red arrow pointing to it. Above this field, the text 'Enter name' is written in red, and 'MinimalistNetworkABMModel' is written in black. The 'Location:' field contains 'D:\Models' and has a 'Browse...' button next to it. The 'Java package:' field contains 'minimalistnetworkabmmmodel'. Below these fields, a text box says 'The following model will be created:' followed by the path 'D:\Models\MinimalistNetworkABMModel\MinimalistNetworkABMModel.alp'. At the bottom, there are 'Finish' and 'Cancel' buttons.

Save Model As

Save model as

Enter name

MinimalistNetworkABMModel

Model name: MinimalistNetworkABMModel

Location: D:\Models Browse...

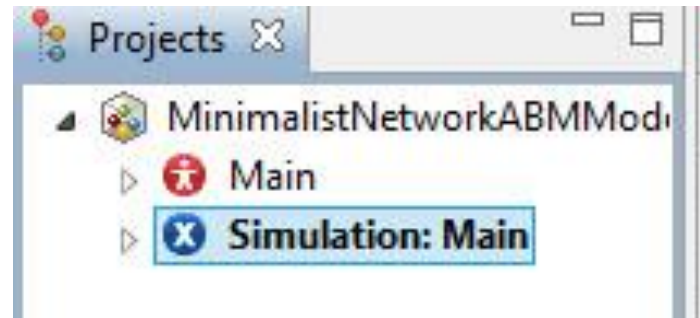
Java package: minimalistnetworkabmmmodel

The following model will be created:

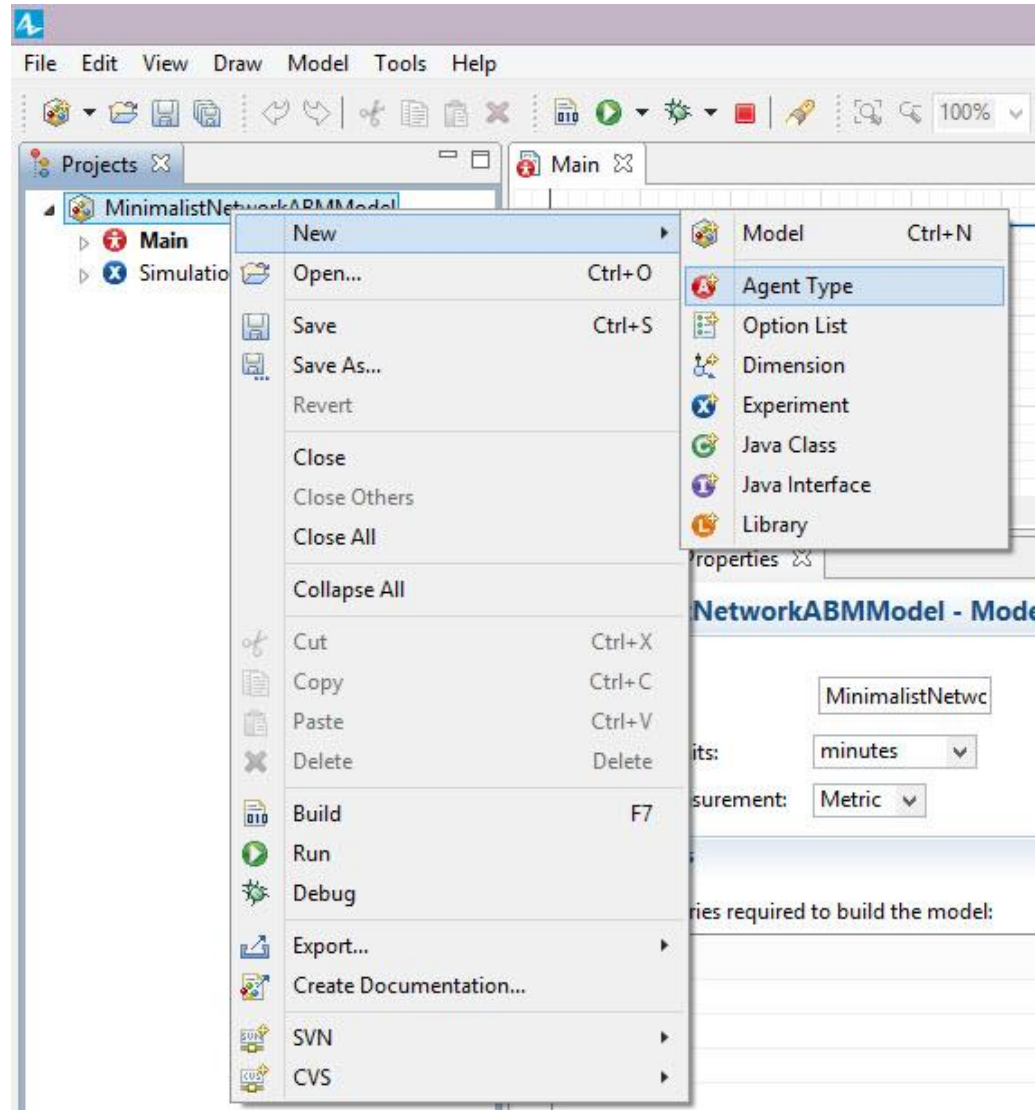
D:\Models\MinimalistNetworkABMModel\MinimalistNetworkABMModel.alp

Finish Cancel

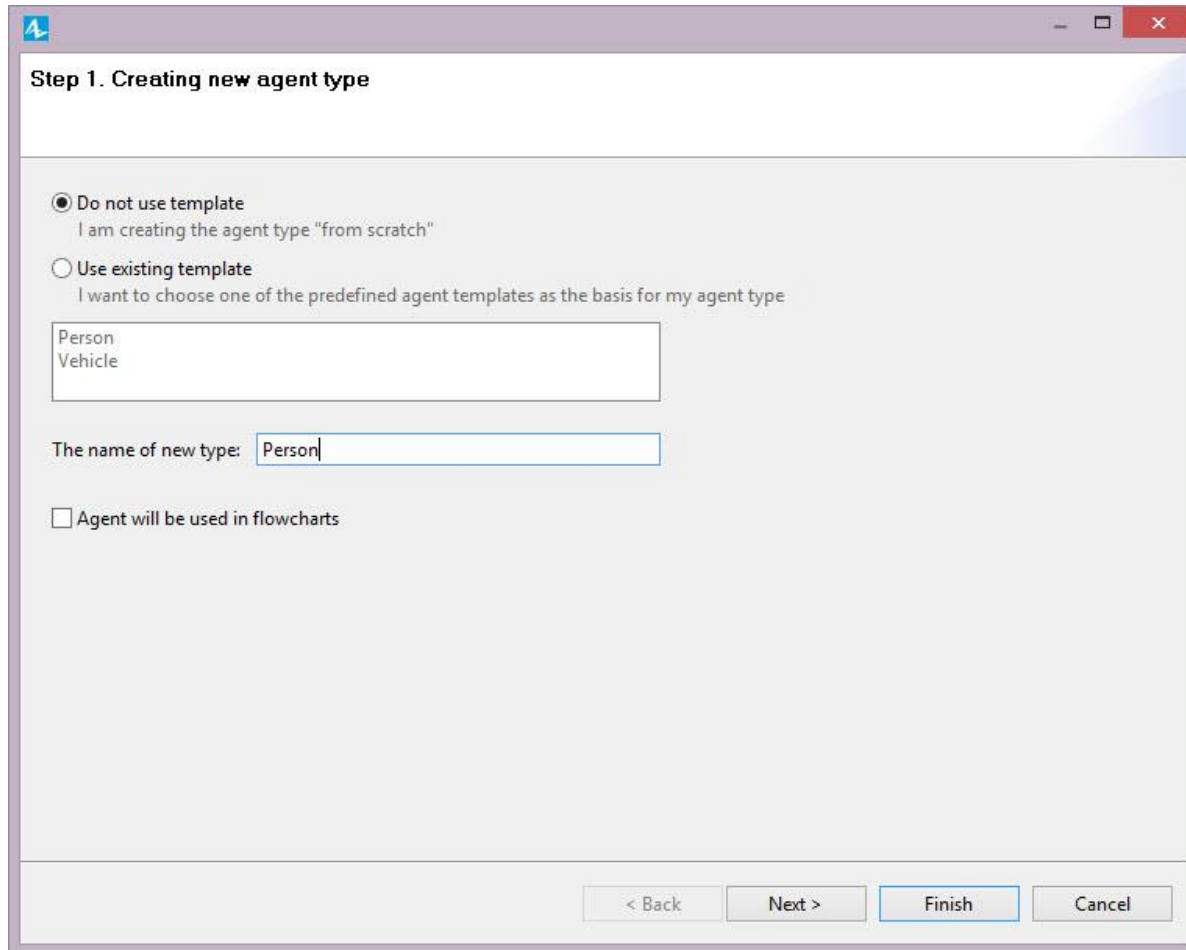
Project Window



Add an Active Object Class



Filling in the Agent Class Details



The screenshot shows a software window titled "Step 1. Creating new agent type". It contains two radio button options: "Do not use template" (selected) and "Use existing template". Below the "Use existing template" option is a list box containing "Person" and "Vehicle". Below the list box is a text field labeled "The name of new type:" with the value "Person". At the bottom left is a checkbox labeled "Agent will be used in flowcharts". At the bottom right are four buttons: "< Back", "Next >", "Finish", and "Cancel".

Step 1. Creating new agent type

☒ **Do not use template**
I am creating the agent type "from scratch"

☐ **Use existing template**
I want to choose one of the predefined agent templates as the basis for my agent type

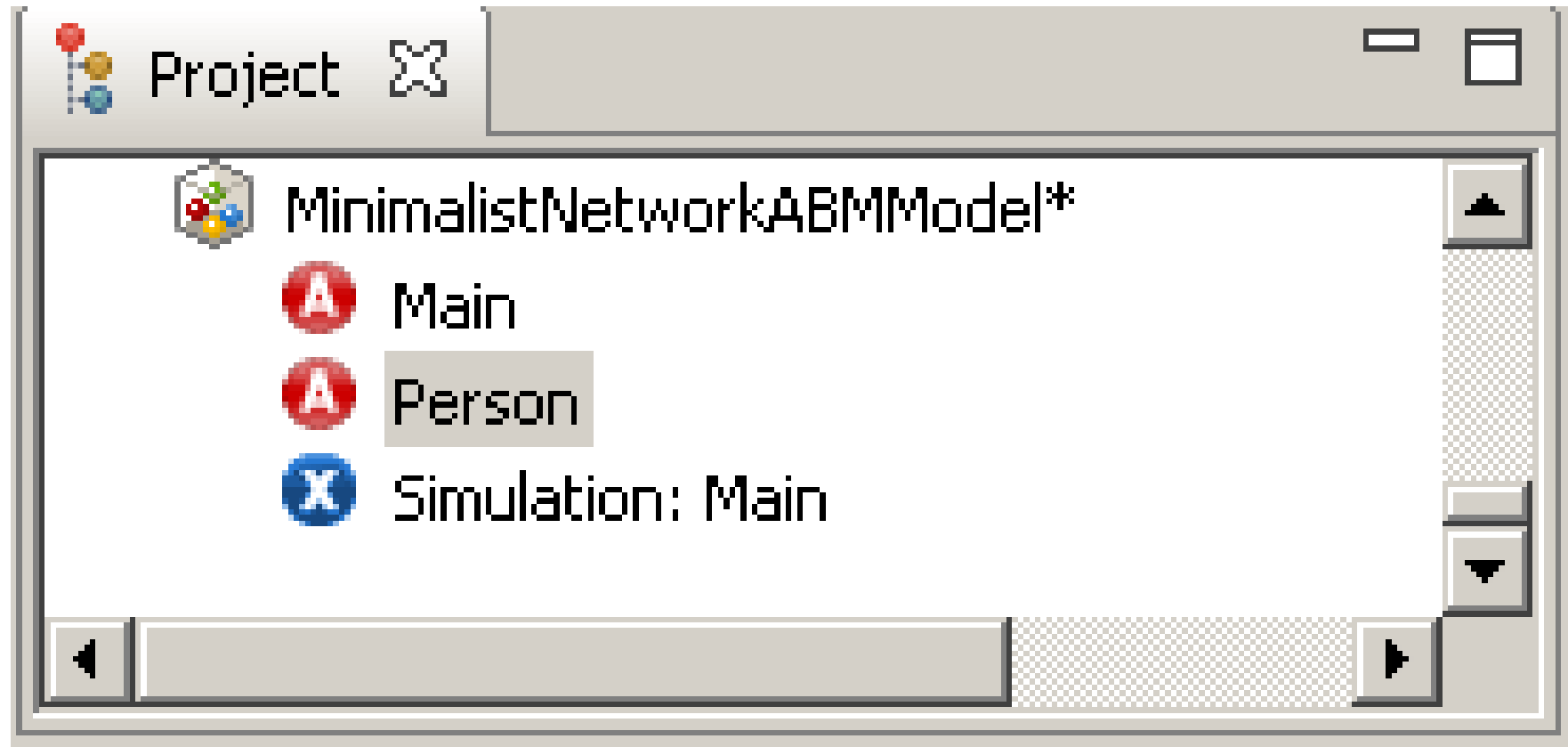
Person
Vehicle

The name of new type:

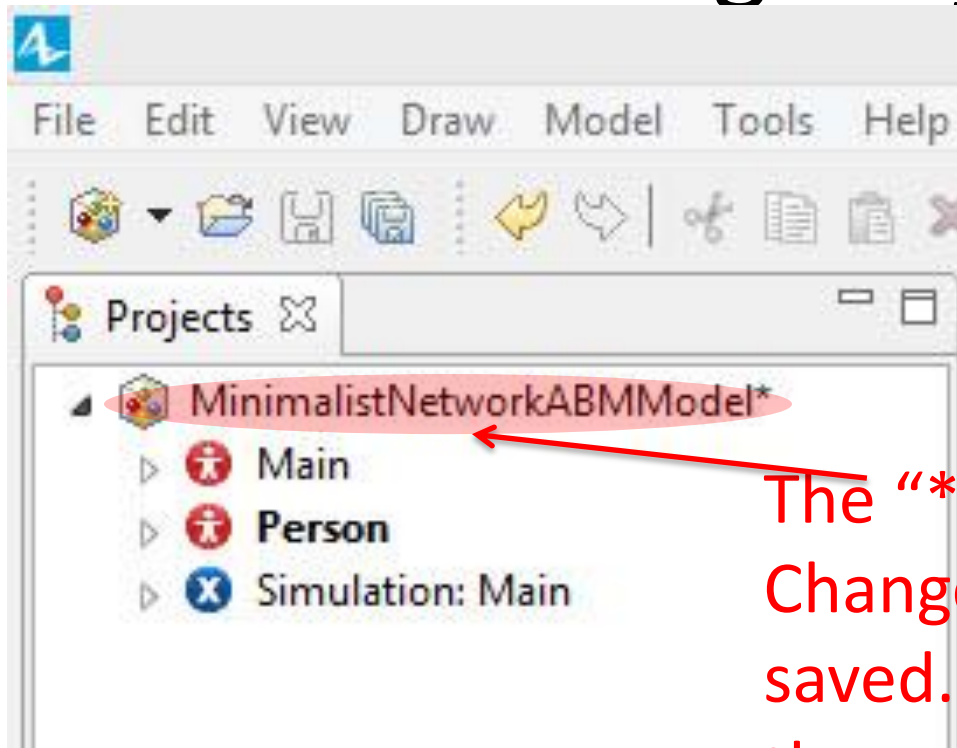
☐ Agent will be used in flowcharts

< Back Next > **Finish** Cancel

Updated Project Window



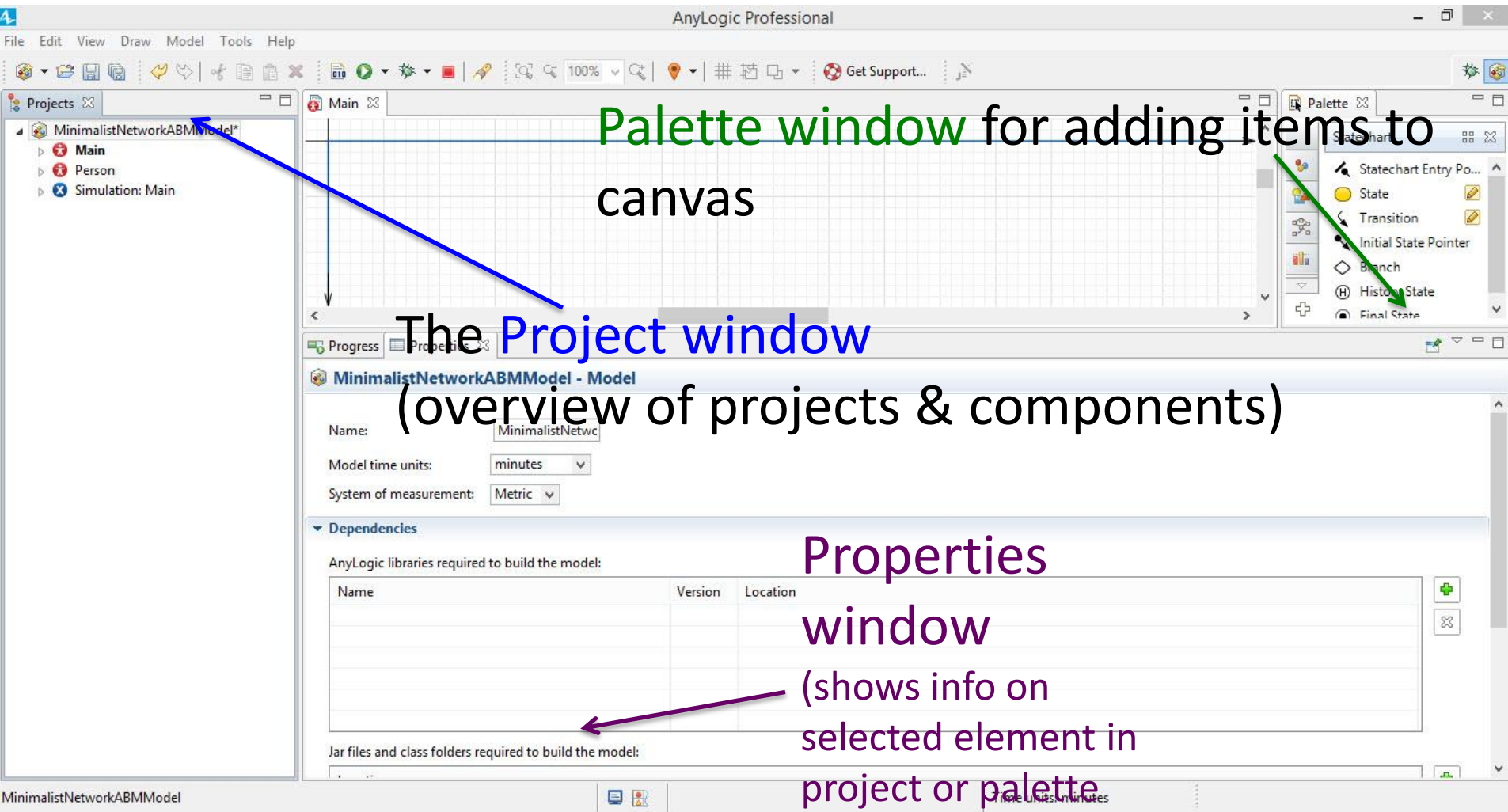
Resulting Project Window



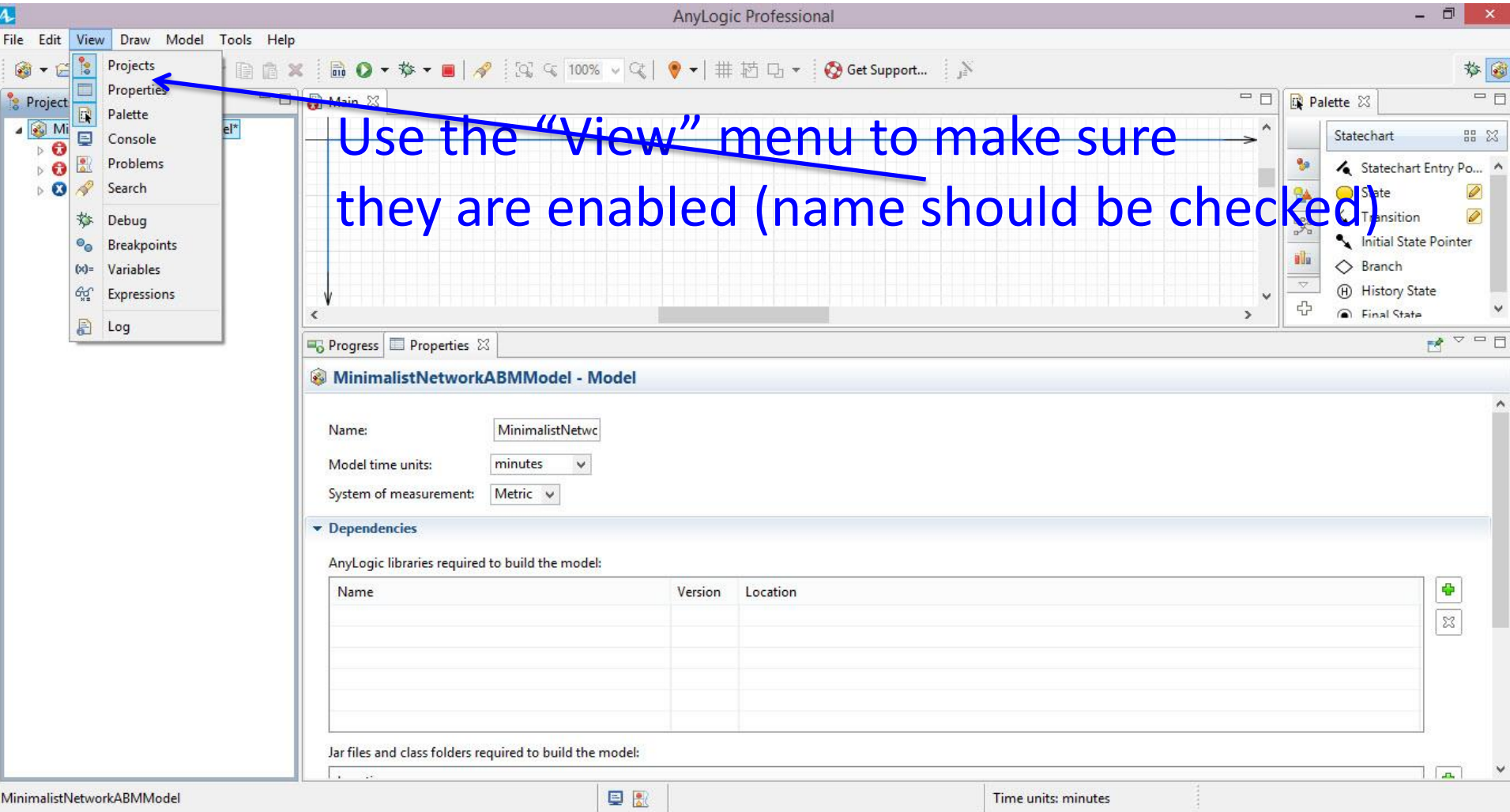
The “*” means that the model has Changed since the last time it was saved. You should consider saving the model when you see this!

AnyLogic Interface Elements

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



If Windows are Missing...

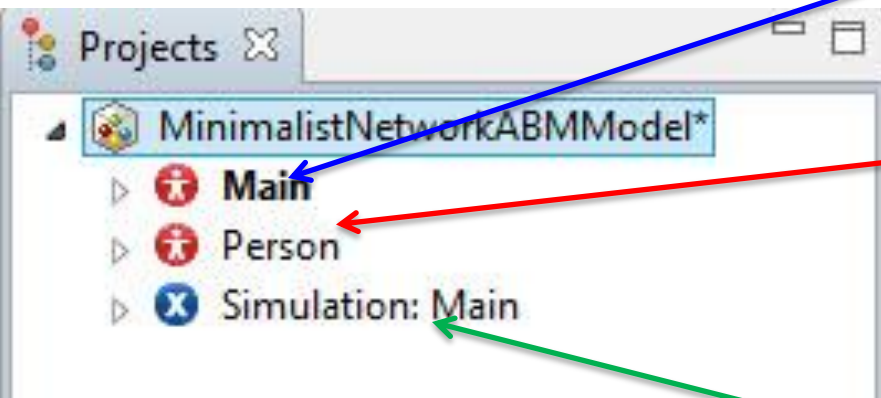


The “Project” Window

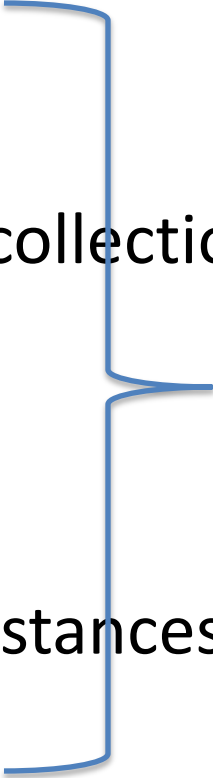
“Main” class
(Defines the “*Stage*” on
which agents circulate)

“Agent” classes
(Define the *actors*)

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



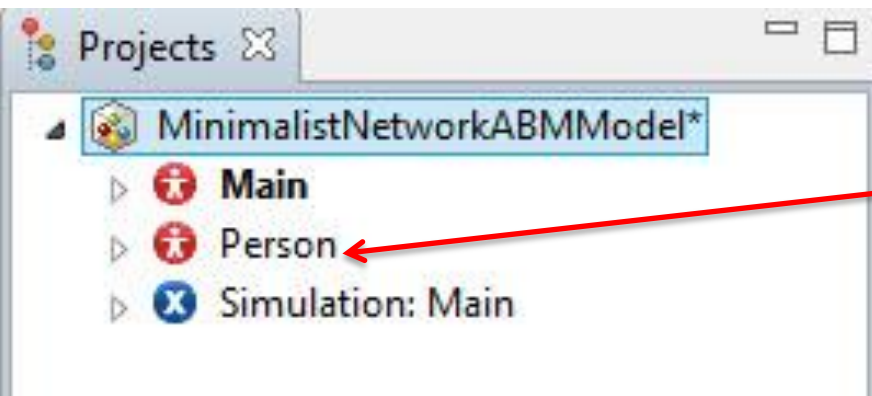
Key Customized “Classes”

- 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

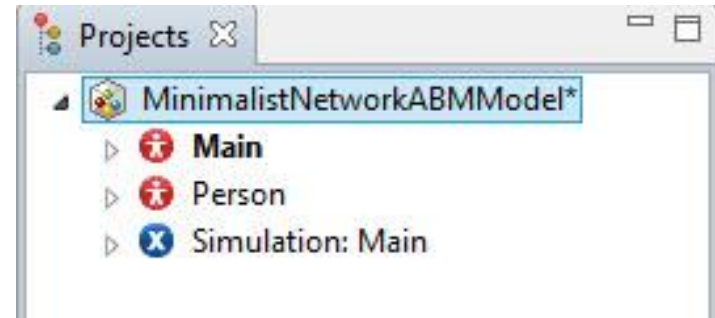
- Agents and Main classes can be associated with visual representations
- These representations can give us a clearer sense of agent behavior

Open Up Canvas for “Person”
(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

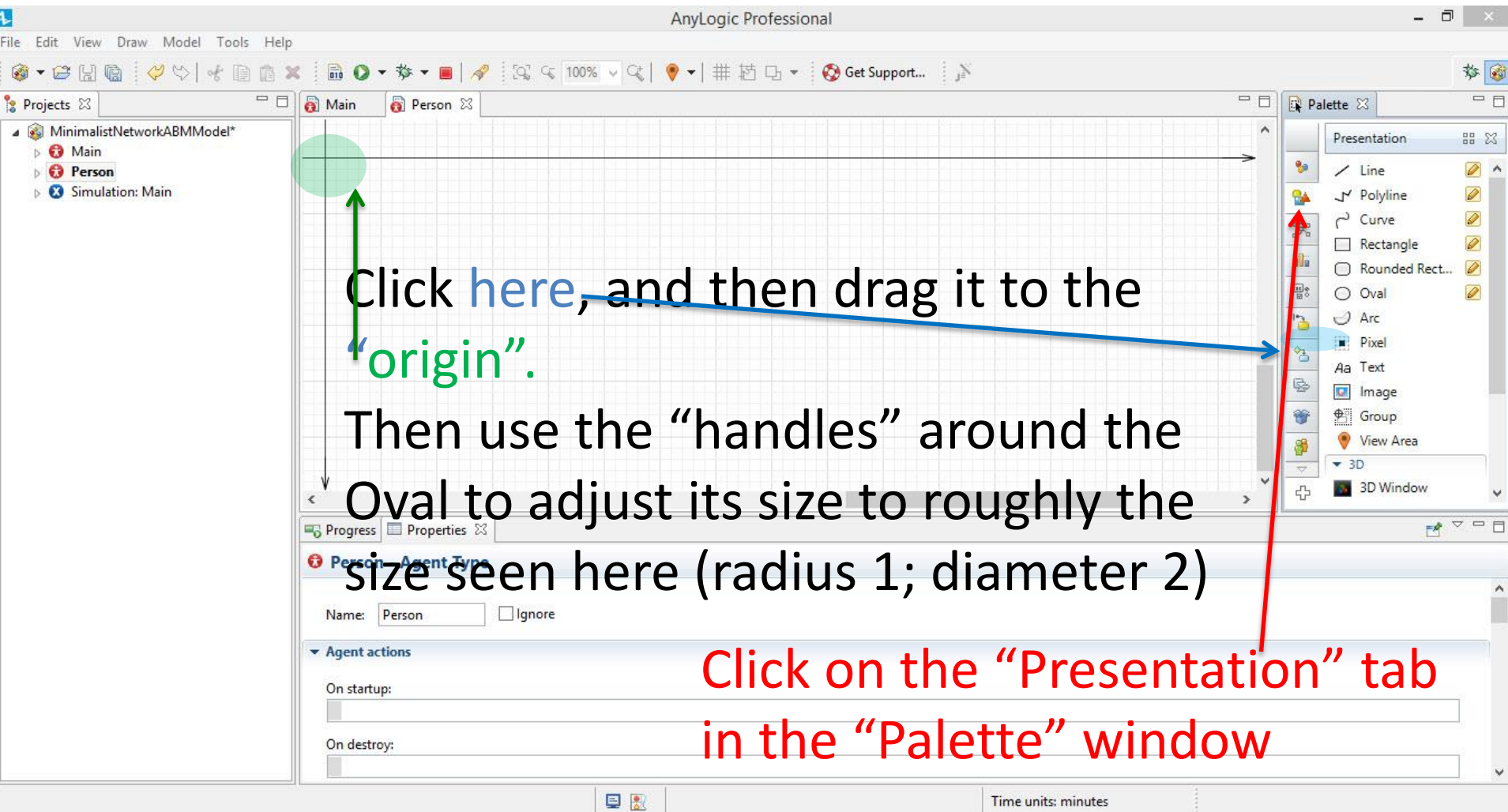
The screenshot displays the AnyLogic Professional software interface. The main workspace is a grid where a statechart is being edited. A red circle highlights the top-left corner of the grid, and a red arrow points to it. Overlaid on the grid is the text: "Scroll up and left a bit, until see a crossing of two (slightly) thicker lines".

The left sidebar shows the "Projects" panel with a tree view containing "MinimalistNetworkABMModel*", "Main", "Person", and "Simulation: Main".

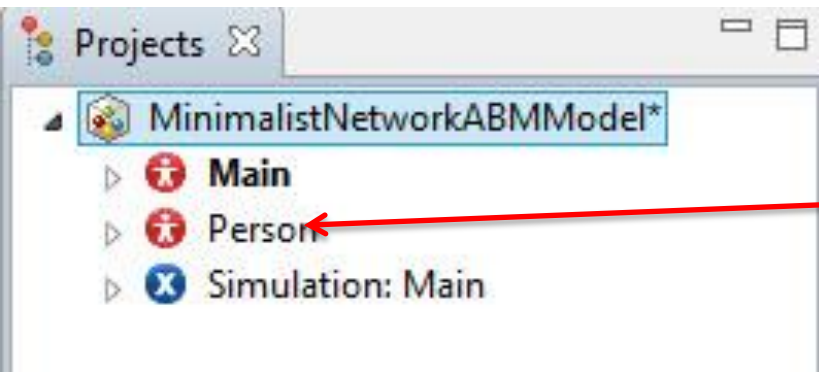
The right sidebar shows the "Palette" panel with a "Statechart" tab. It contains various statechart elements: "Statechart Entry Point", "State", "Transition", "Initial State Pointer", "Branch", "History State", and "Final State".

The bottom panel shows the "Properties" tab for the "Person - Agent Type". It includes a "Name" field set to "Person" and an "Ignore" checkbox. Below this is a section for "Agent actions" with fields for "On startup:" and "On destroy:". The status bar at the bottom indicates "Time units: minutes".

Adding an Oval to Represent Agent



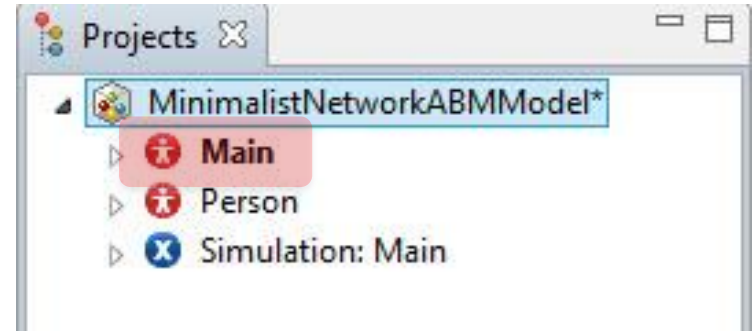
Open Up Canvas for “Main” (In case it is not already open)



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

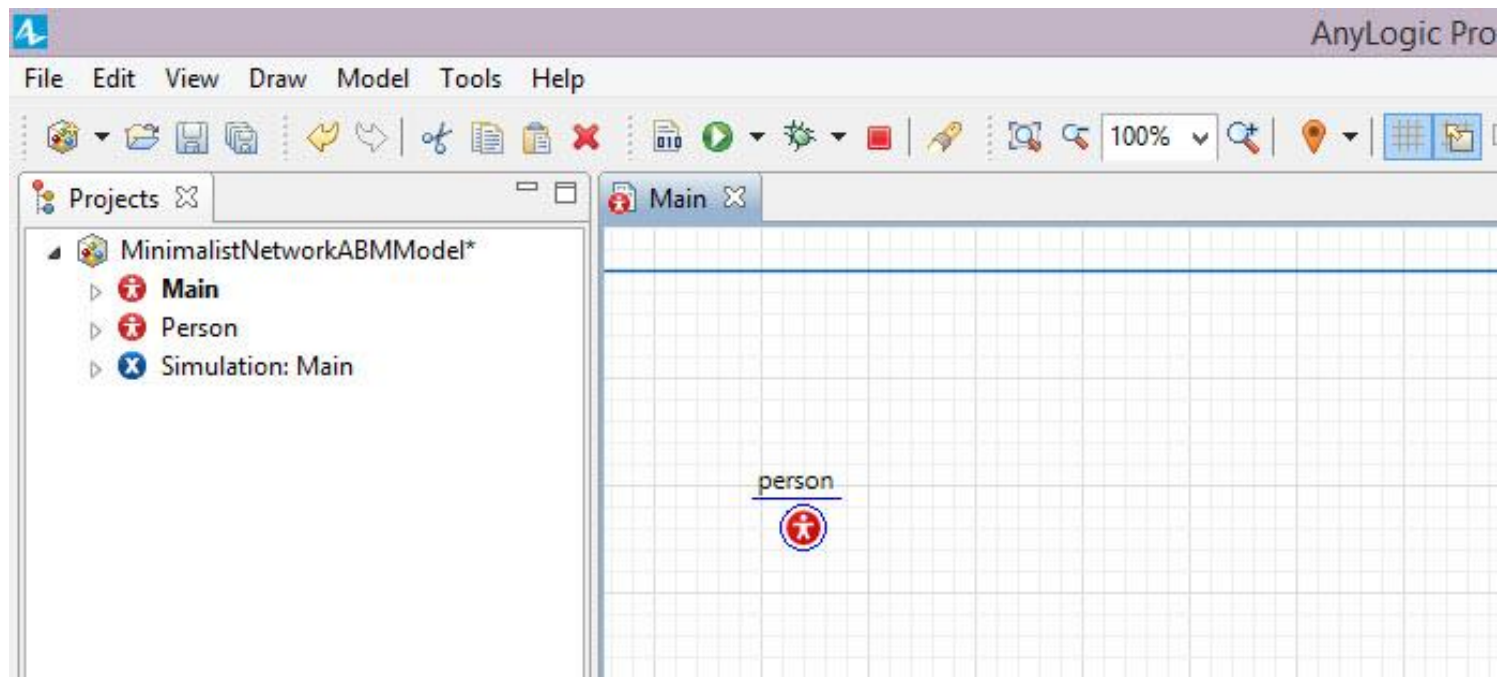


We will now add an Agent (Person) population to the “Main” Class

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

To Add an Agent (Person) Population: Drag From “Person” into the Canvas for “Main”



Specifying the Population Name & Size

The screenshot displays the AnyLogic Professional interface. On the left, the 'Projects' pane shows a hierarchy: 'MinimalistNetworkABMModel*' > 'Main' > 'Person' > 'Simulation: Main'. The central workspace shows a 'population [...]' agent icon. 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 'Initial number of agents' field, which contains the value '100'. The 'population - Person' properties window also shows 'Visible' set to 'yes', 'Single agent' unselected, and 'Population of agents' selected. The 'Initial location' section is expanded. The bottom status bar indicates 'Time units: minutes'.

Name: Enter "population" (without quotes!)

Replication (population size): Enter "100" (without quotes!)

population - Person

Name: population ☒ Show name ☐ Ignore

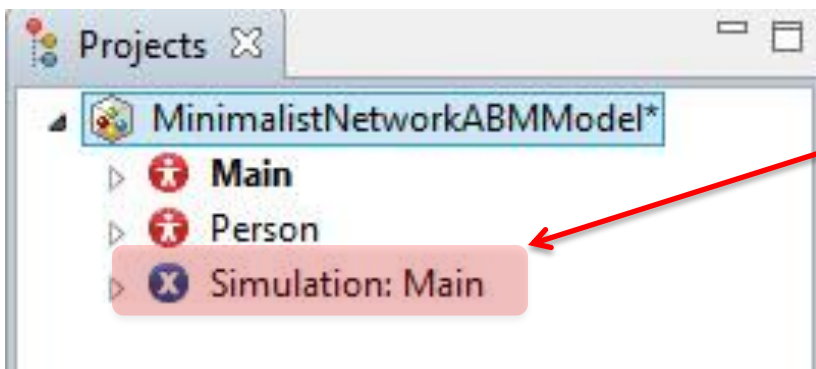
Visible: ☒ yes

☐ Single agent ☒ Population of agents

Initial number of agents: 100

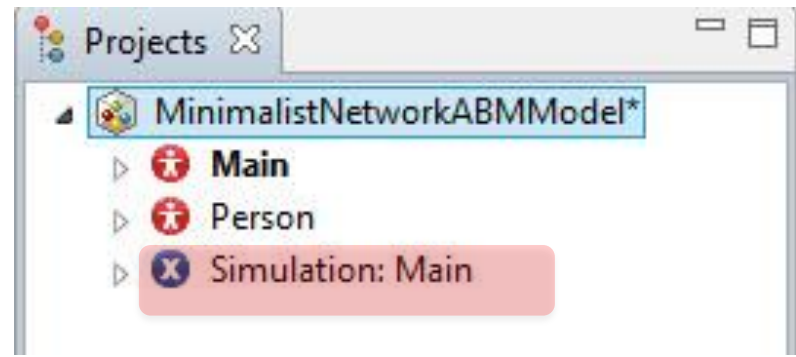
Initial location

Time units: minutes



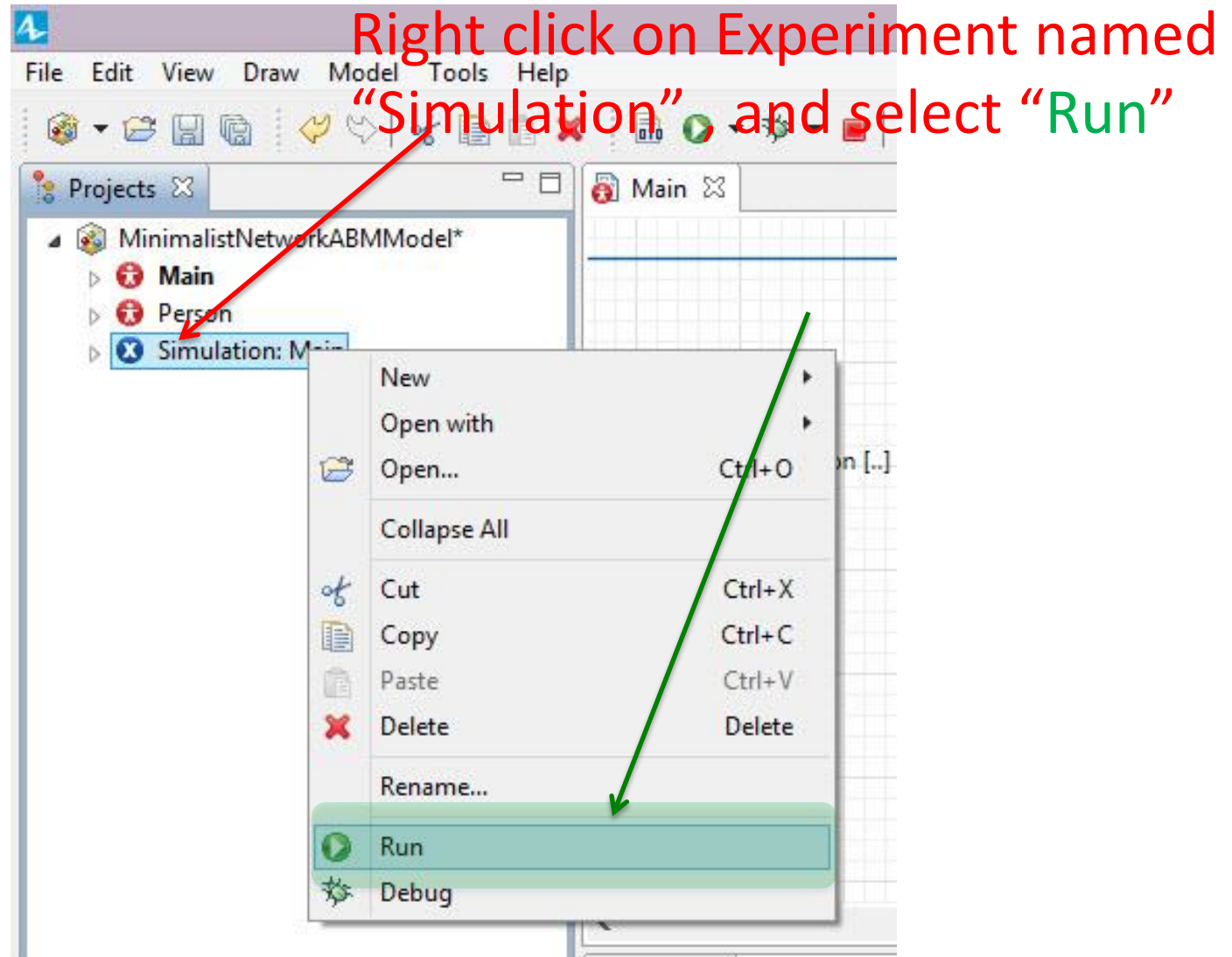
A (default) Experiment
Specifies assumptions
for a particular scenario
(e.g. population size,
pathogen contagiousness,
etc.)

Experiment Classes

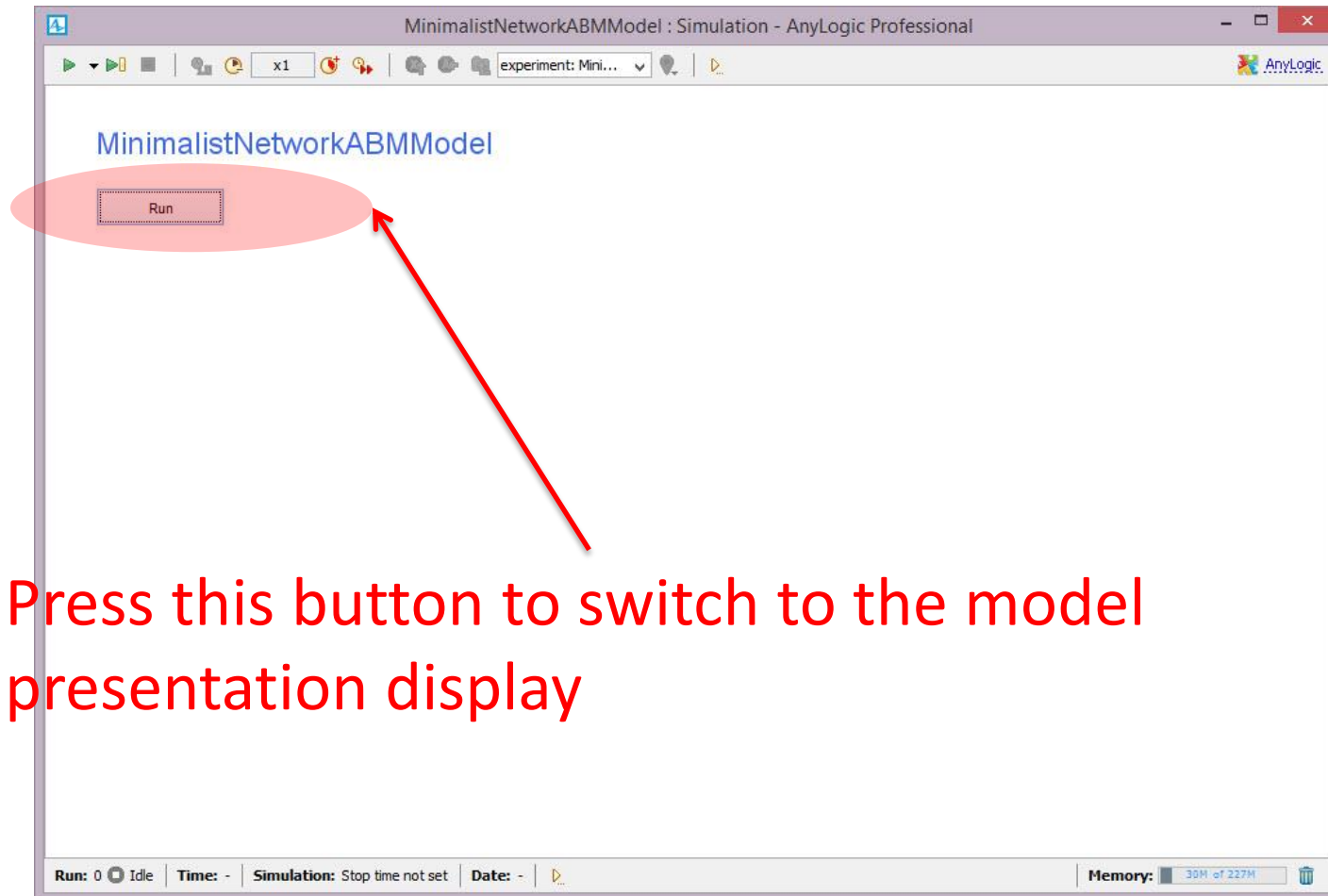


- Experiment classes allow you to define & run scenarios in which global “parameters” (i.e. assumption quantities defined in *Main*) may hold either default or alternative values
- Experiment classes are also used to set
 - The time horizon for a simulation
 - Memory limits (important for large models)
 - Details of simulation run
 - Details on random number generation
 - Virtual machine arguments
- “Parameters” 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!



Initial Simulation Screen



An Uninspiring Display

The screenshot shows the AnyLogic Professional interface for a simulation titled "MinimalistNetworkABMModel : Simulation". The top toolbar contains various icons for simulation control, including a red circle highlighting a specific icon. The main workspace displays a single green rectangular block labeled "population Person [100]" with a small red star icon. A green arrow points from the text "Our population has size 100" to this block. A red arrow points from the text "All agents (Persons) in population are identical – and are clustered up here!" to the same block. The bottom status bar shows the simulation is "Running" at "Time: 3.80", with "Simulation: Stop time not set", "Date: Apr 15, 2014 7:28:34 PM", and "Memory: 45M of 227M".

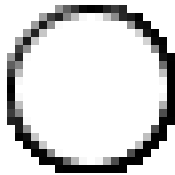
MinimalistNetworkABMModel : Simulation - AnyLogic Professional

Our population has size 100

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

Run: 0 Running | Time: 3.80 | Simulation: Stop time not set | Date: Apr 15, 2014 7:28:34 PM | Memory: 45M of 227M

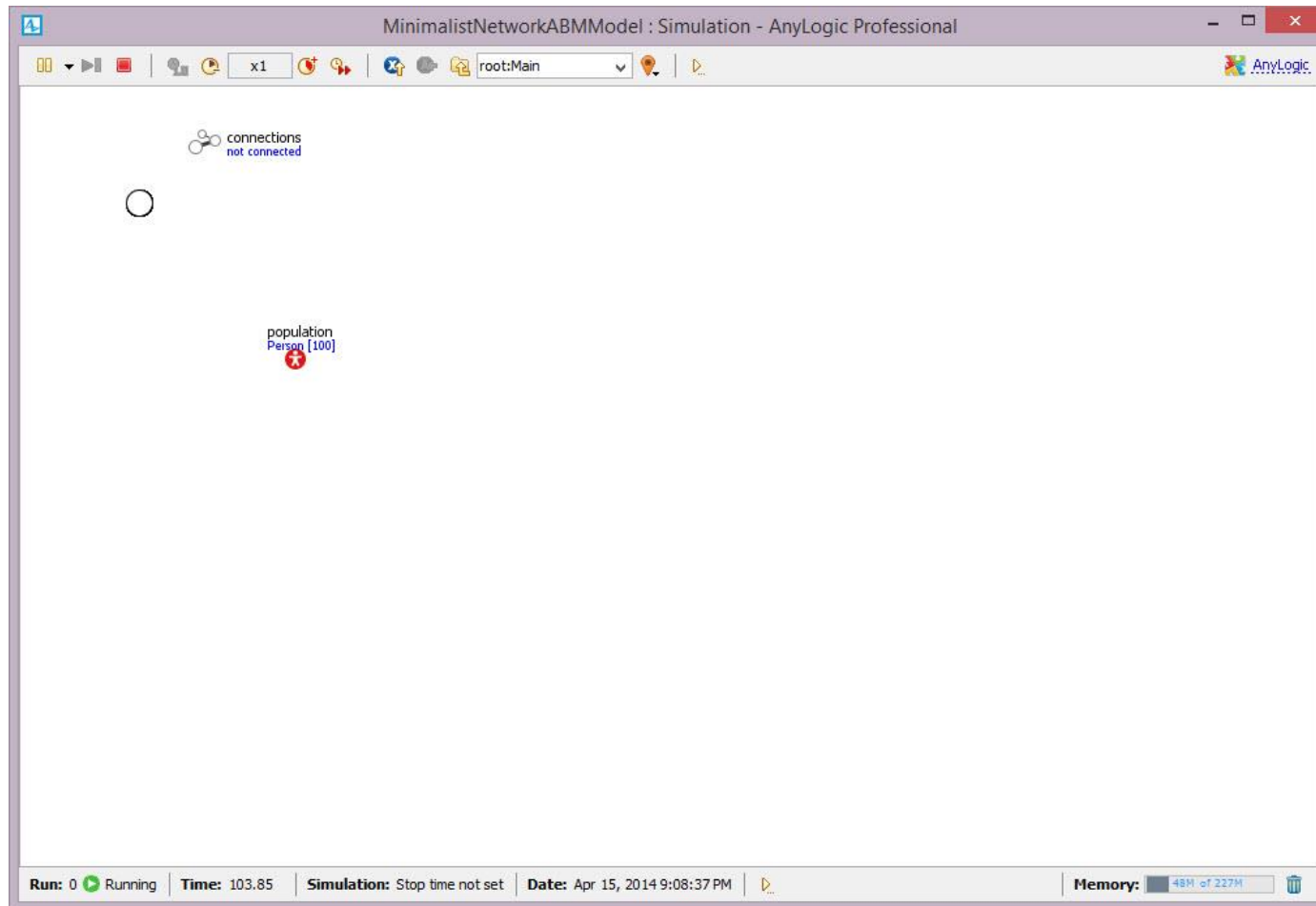
A Magnified View



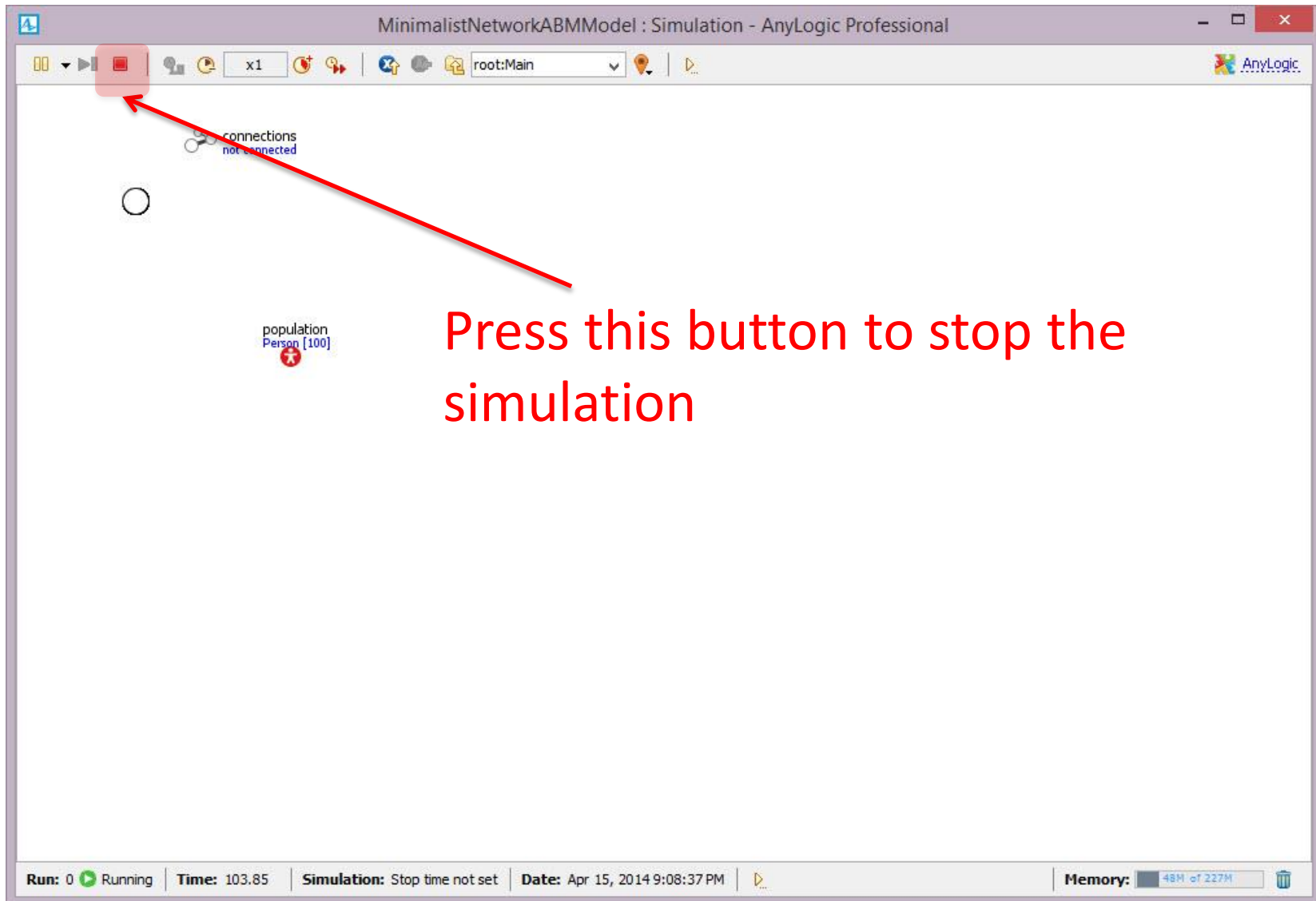
population
Person [100]



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



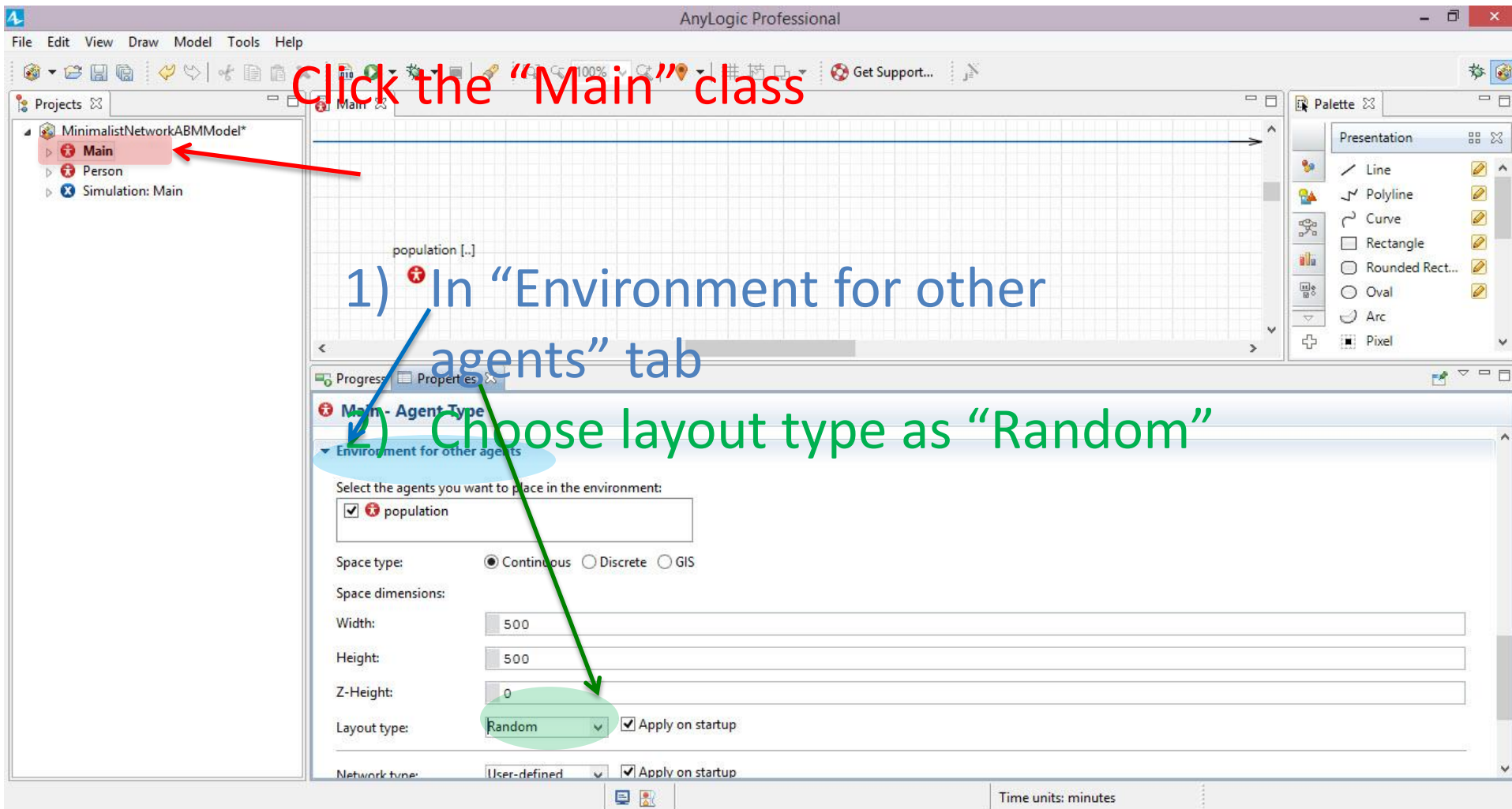
Stop Simulation



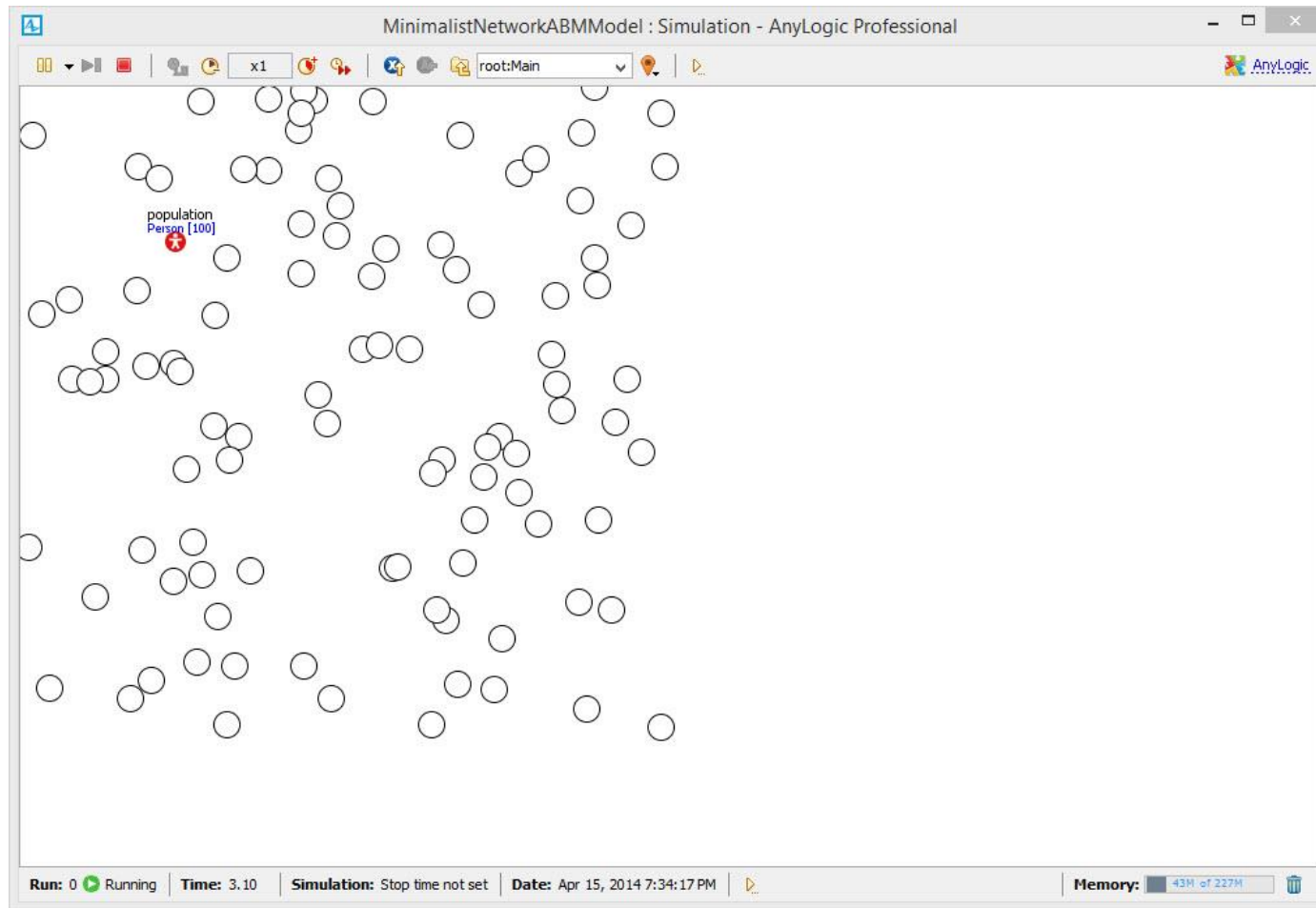
Agent Populations Live in Main Class

- 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

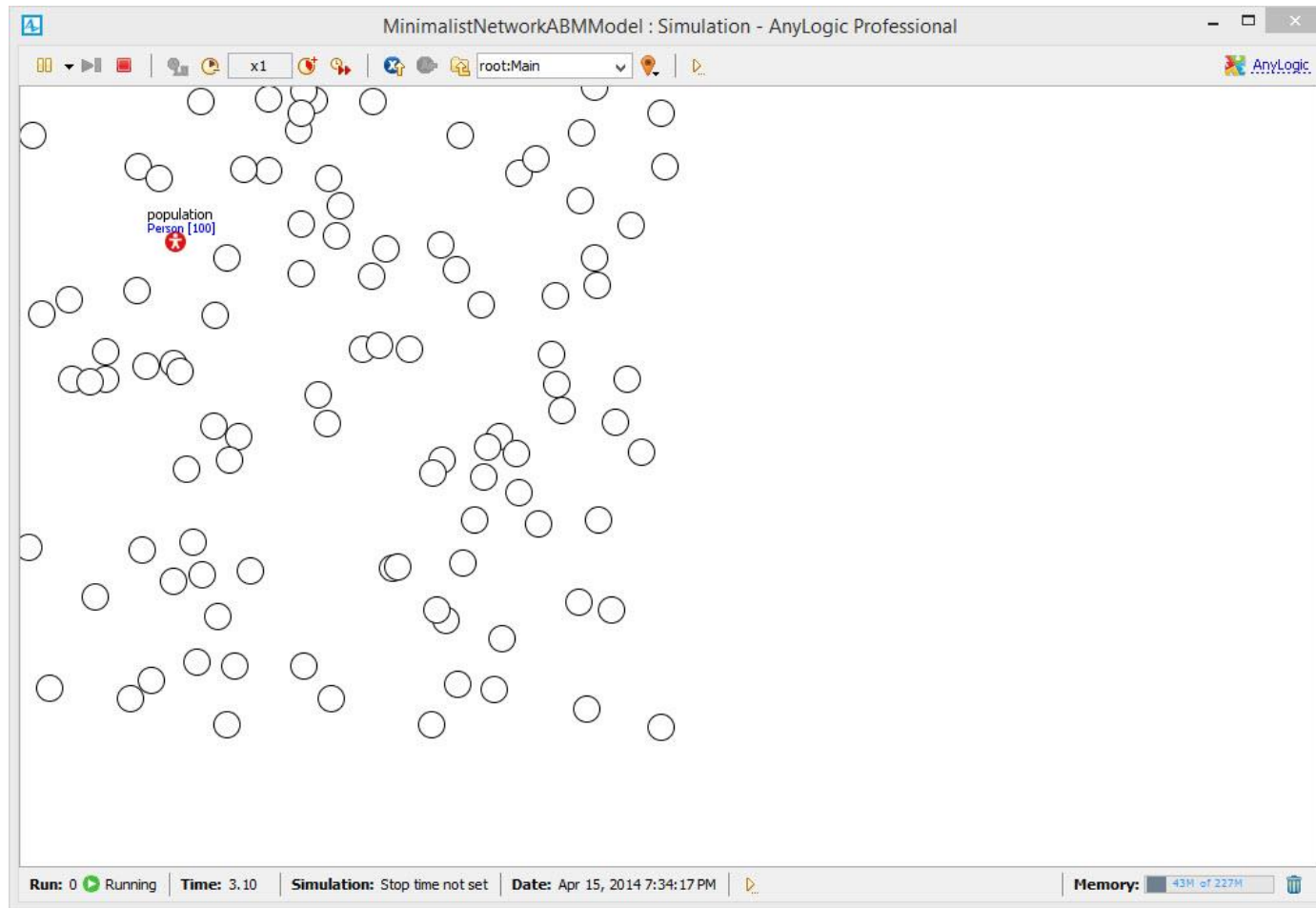
From “General” Area of “Palette” Window Add an “Environment” to the Model



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

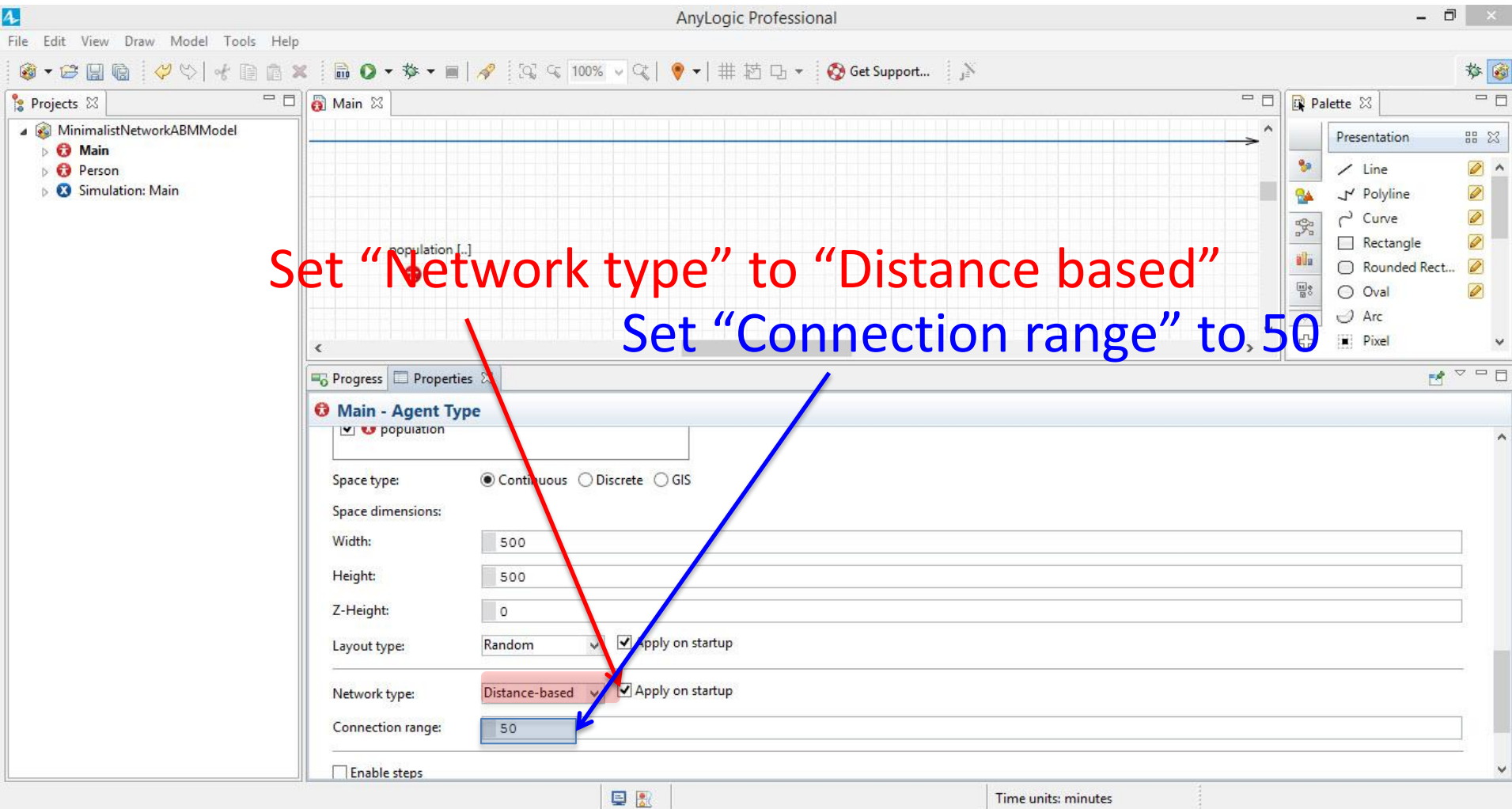
- Before we can actually realize the recipe, we need to go collect & prepare all ingredients
- We’re not yet cooking, but what we are doing makes the cooking possible
- The “cooking” here is running the model

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

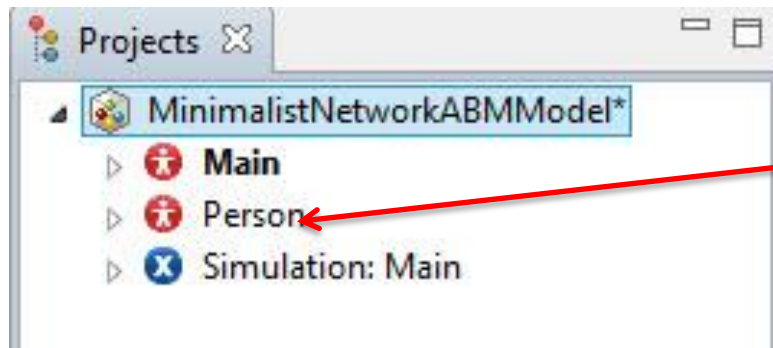
Open “Main”, Click on “environment”, and go to the “Advanced” tab in “Properties” window



Let's Place the Agents in a Network

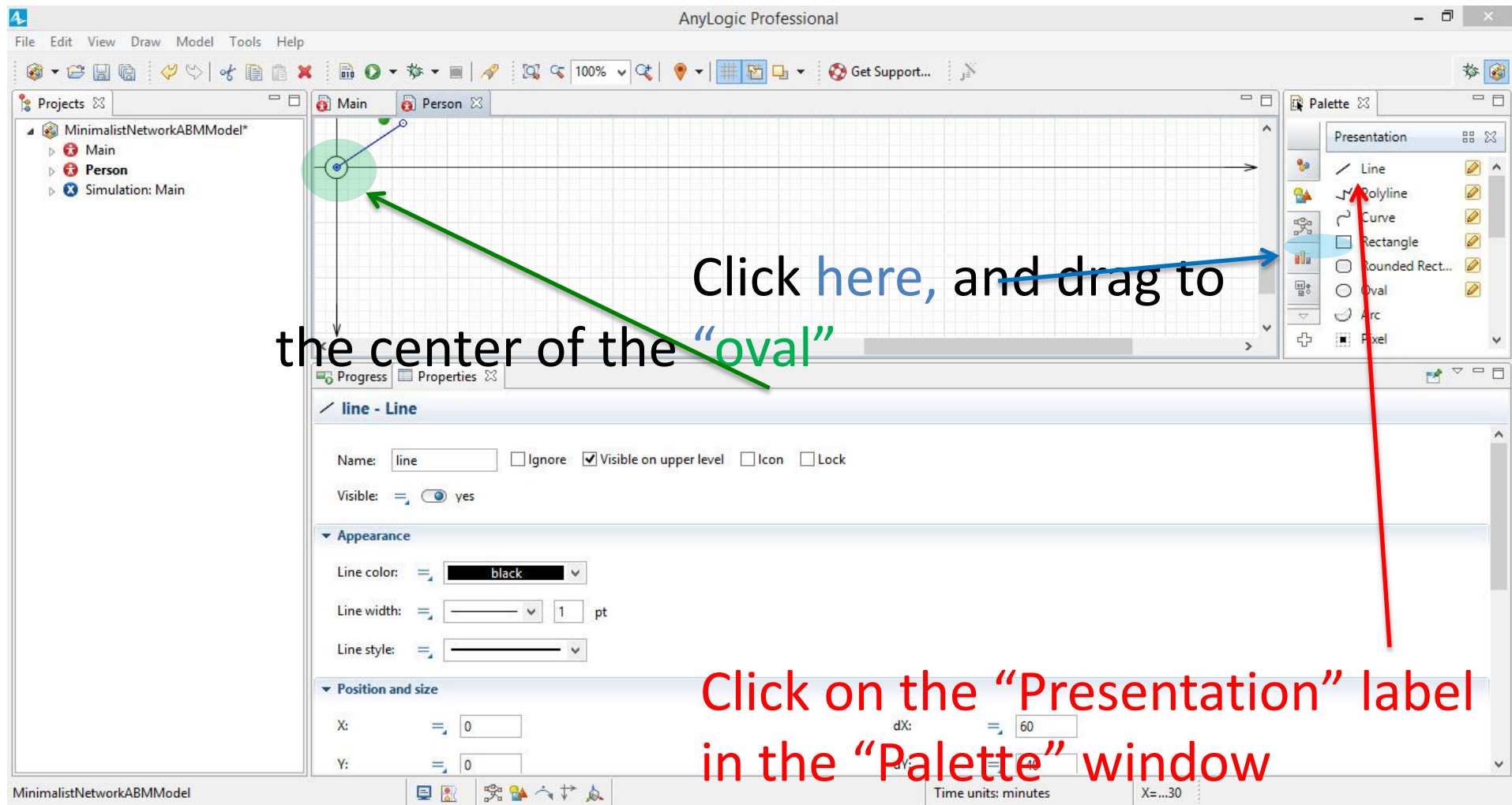
- Steps
 - ✓ Tell the Environment that we want to situate the agents in a (here, distance-based) network
 - ✓ Specify the attributes of the network (here, the distance threshold up to which agents are considered connected)
 - Give agents a way of appearing visually connected

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

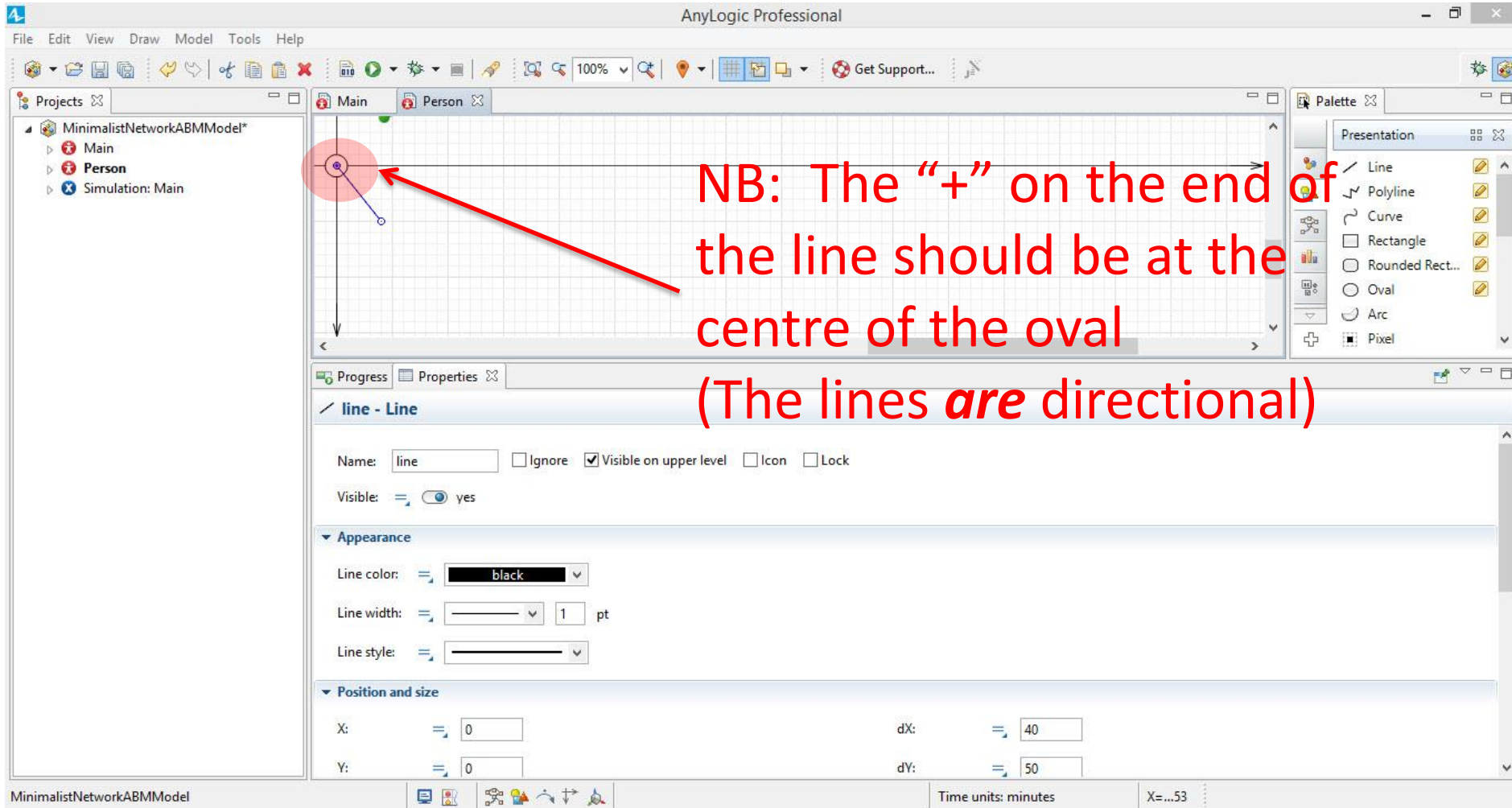


“Double Click Here

Adding a Line to Represent Connections



Adding a Line to Represent Connections



AnyLogic Professional

File Edit View Draw Model Tools Help

Projects

- MinimalistNetworkABMModel*
- Main
- Person
- Simulation: Main

Main

Person

Palette

Presentation

- Line
- Polyline
- Curve
- Rectangle
- Rounded Rect...
- Oval
- Arc
- Pixel

Progress Properties

line - Line

Name: line ☐ Ignore ☒ Visible on upper level ☐ Icon ☐ Lock

Visible: ☒ yes

Appearance

Line color: black

Line width: 1 pt

Line style:

Position and size

X: 0 dX: 40

Y: 0 dY: 50

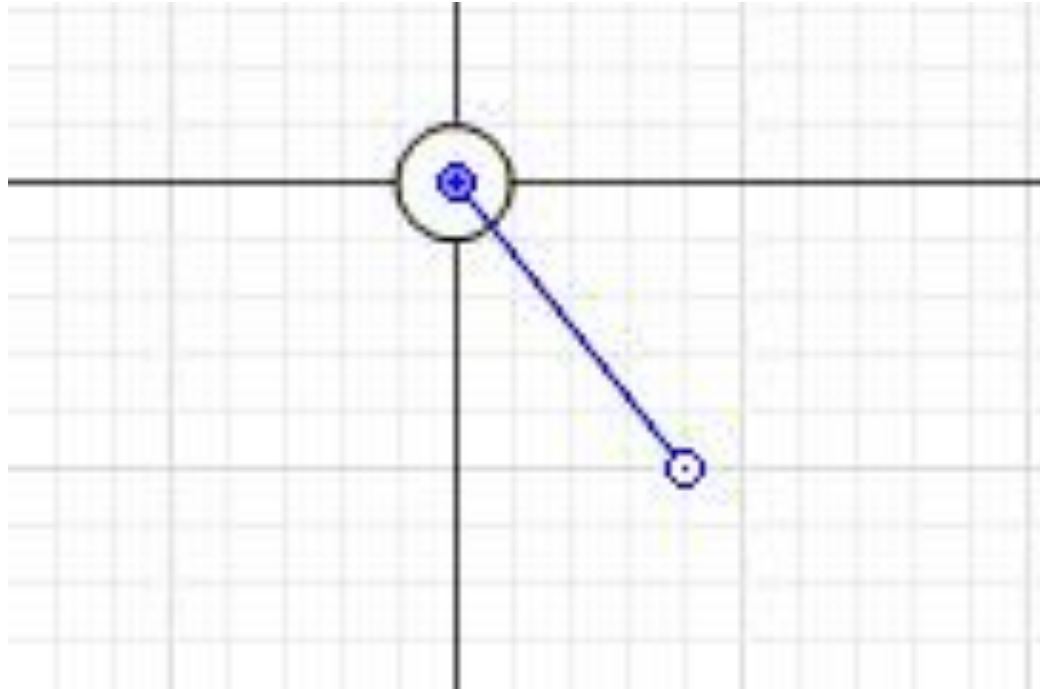
MinimalistNetworkABMModel

Time units: minutes

X=...53

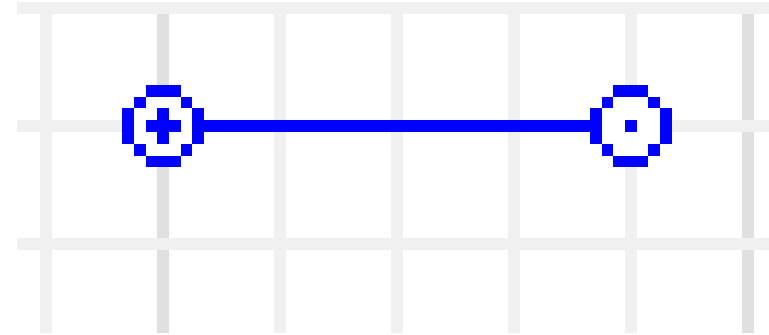
NB: The “+” on the end of the line should be at the centre of the oval
(The lines *are* directional)

Close-Up

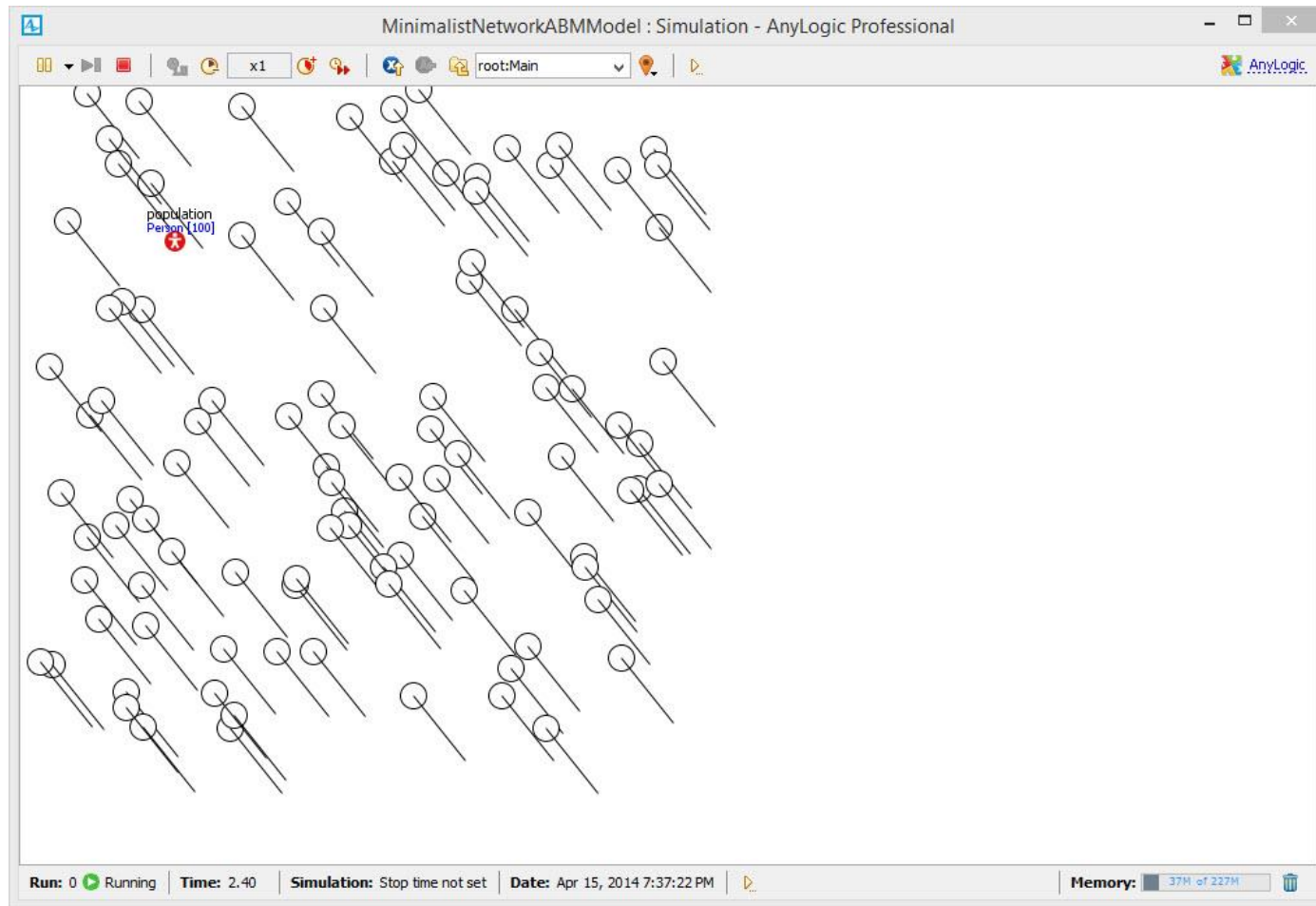


If you are Initially Unsuccessful in Placing the Line in the Circle ...

- 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 Professional interface. On the left, the 'Projects' pane shows a model named 'MinimalistNetworkABMModel' with sub-elements 'Main', 'Person', and 'Simulation: Main'. The main workspace displays a diagram with a blue oval labeled 'connections' and a line element. A blue arrow points from the text 'Make sure the line remains selected (Click on it if not!)' to the line element. The 'Properties' window for the 'line - Line' element is open, showing various settings. A green arrow points from the text 'Select the "Advanced" tab!' to the 'Advanced' tab. The 'Replication' field is highlighted in red and contains the text `this.getConnectionsNumber()`. A red arrow points from the text 'Replication should read "this.getConnectionsNumber()" (i.e. we seek 1 line per connection)' to the 'Replication' field. The 'Show in:' section has radio buttons for '2D and 3D' (selected), '2D only', and '3D only'. The 'On click' field is empty. The 'Show name' checkbox is unchecked. The status bar at the bottom indicates 'Time units: minutes'.

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

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

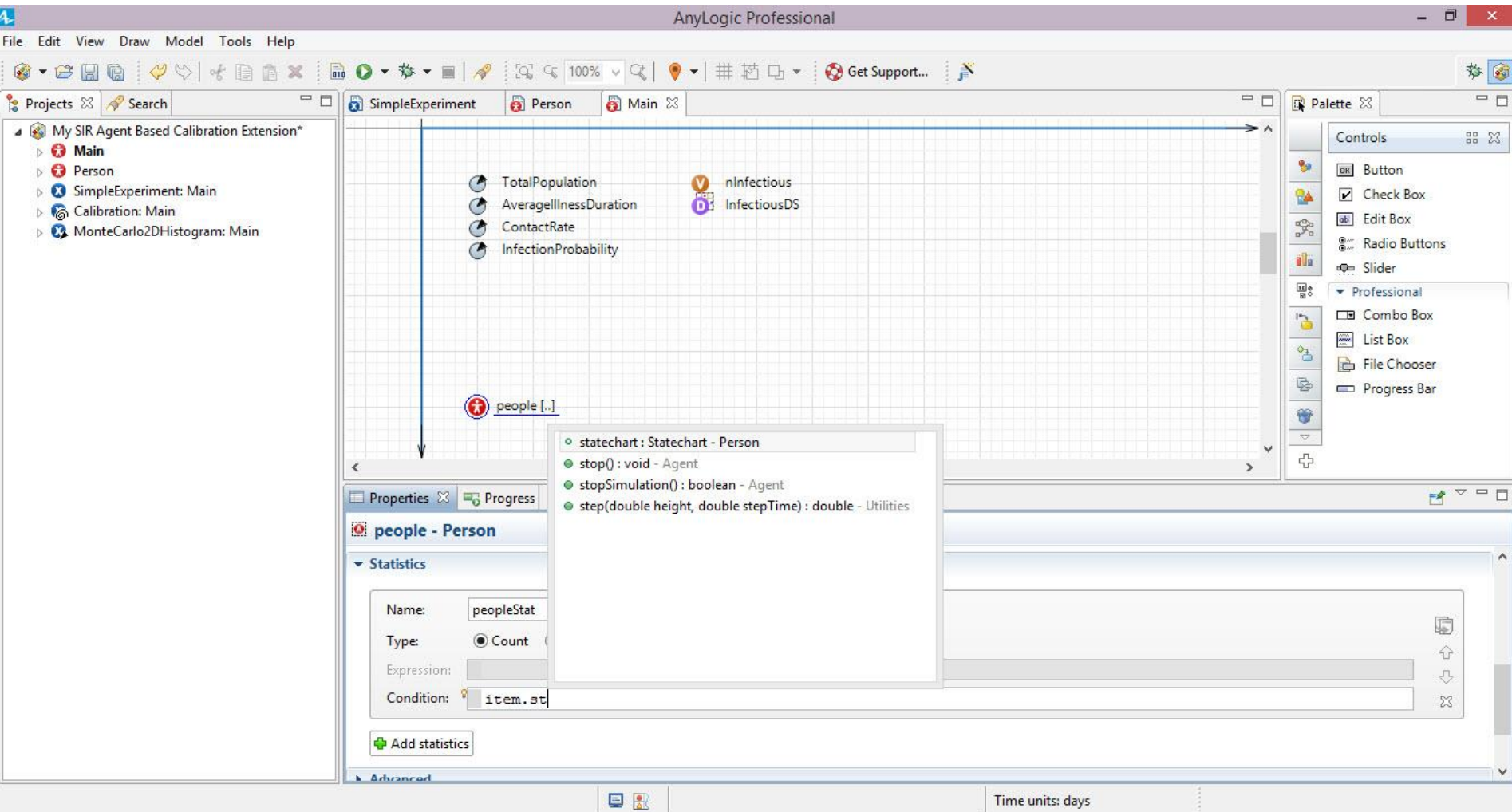
Select the "Advanced" tab!

Time units: minutes

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)

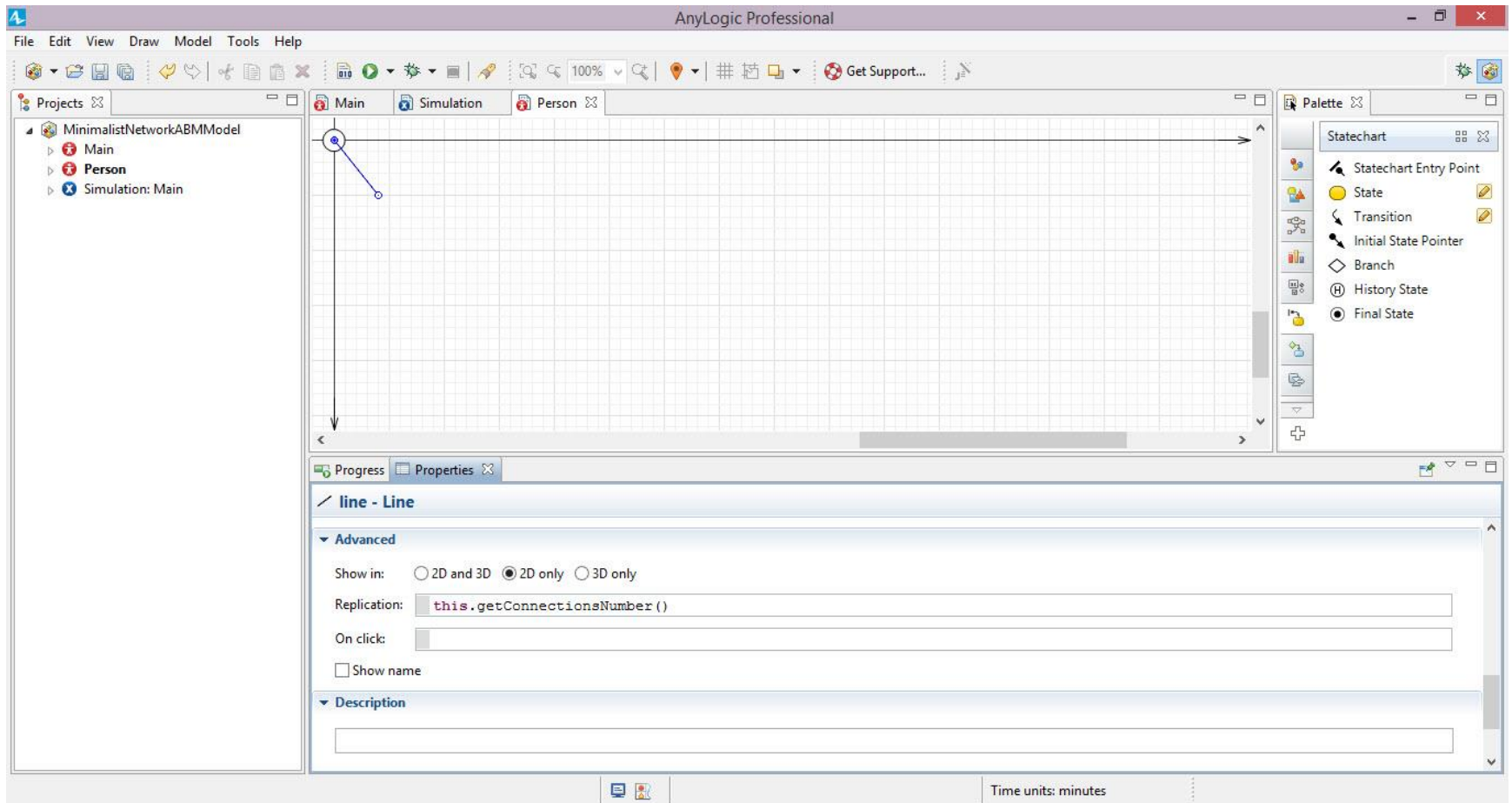
The screenshot displays the AnyLogic Professional interface. The main workspace shows a statechart diagram with a circle labeled 'main' and a state labeled 'Susceptible'. A transition labeled 'income' leads to the 'Susceptible' state. A transition labeled 'sex' leads to the 'Susceptible' state. A transition labeled 'statechart' leads to the 'Susceptible' state. The 'connections' palette is visible on the right, showing a 'Statechart Entry Point' and a 'State'.

The bottom panel, titled 'connections - Link to agents', shows the configuration for the 'connections' link. It includes a 'Message type' dropdown set to 'Other' and an 'Object' field. The 'On message received' section contains the code `statechart.receiveMessage(msg);`. The 'Forward message to' section shows a table with a checked box for 'statechart'.

Statecharts
<input checked="" type="checkbox"/> statechart

The bottom status bar shows 'Time units: minutes' and 'X=...38'.

Known AnyLogic Bug – Save, Quit & Restart AnyLogic



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 Professional software interface. The main workspace shows a statechart diagram with a 'Person' entity. A red text overlay with an arrow pointing to the 'dX' property in the 'line - Line' properties panel reads: "Clicking on 'Lightbulb' gives hint, noting that the 'index' is variable is defined as the connection number for this Person." The 'line - Line' properties panel is open, showing various parameters. The 'dX' property is highlighted with a red oval, and a tooltip indicates: "Use: index: index of replicated Line".

AnyLogic Professional

File Edit View Draw Model Tools Help

Projects

- MinimalistNetworkABMModel*
- Main
- Person
- Simulation: Main

Main Simulation Person

Statechart

- Statechart Entry Point
- State
- Transition
- Initial State Pointer
- Branch
- History State
- Final State

Progress Properties

line - Line

X: 0 dX: [lightbulb icon] [text box]

Y: 0 dY: [text box]

Z: 0 dZ: 0

Z-Height: 10

Rotation, rad: [text box]

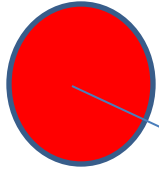
Scale X: [text box]

Scale Y: [text box]

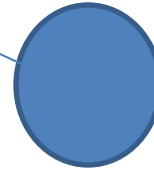
Time units: minutes

Geometry to Connect Agents

Index Agent A



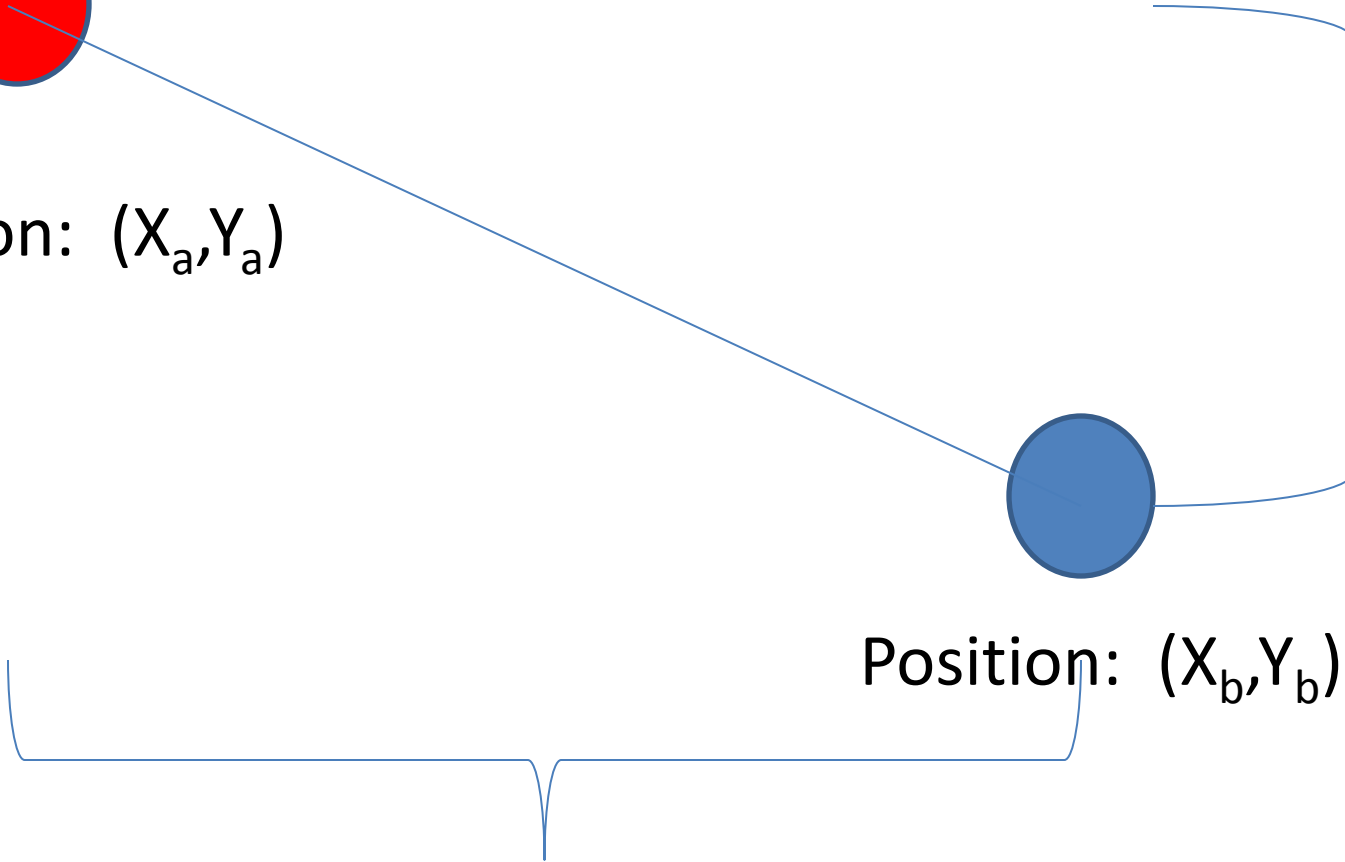
Position: (X_a, Y_a)



Position: (X_b, Y_b)

$Y_b - Y_a$

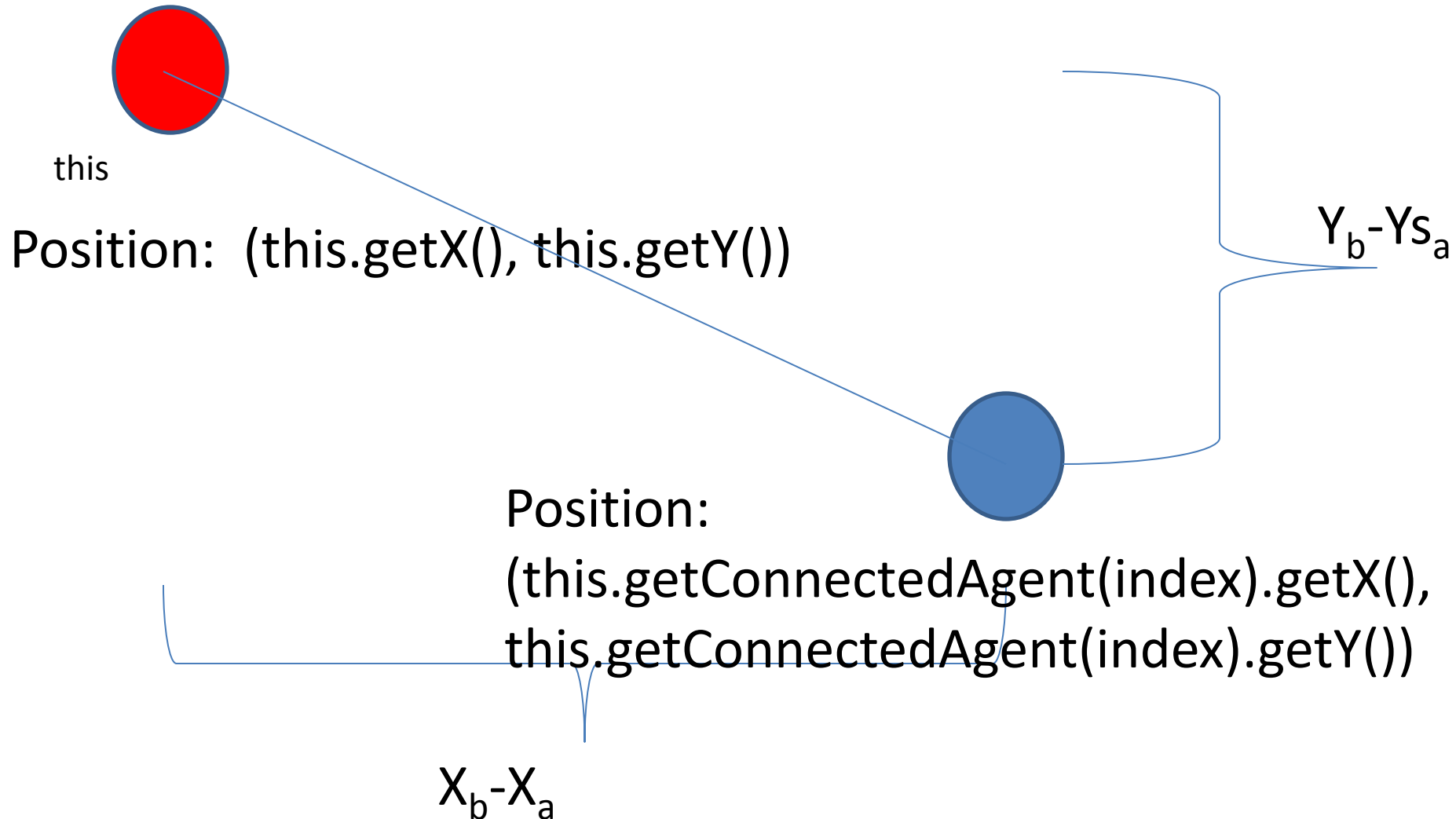
$X_b - X_a$



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 displays the AnyLogic Professional interface. On the left, the 'Projects' pane shows a hierarchy: MinimalistNetworkABMModel* > Main > Person > Presentation > line. The main workspace shows a statechart diagram with a state labeled 'line'. The 'Properties' pane on the right is open, showing the 'line - Line' object. Under the 'Position and size' section, the 'dX' and 'dY' properties are highlighted with red arrows. The 'dX' property is set to the formula `this.getConnectedAgent(index).getX() - this.getX()`, and the 'dY' property is set to the formula `this.getConnectedAgent(index).getY() - this.getY()`. The 'dZ' property is set to 0. The 'Time units: minutes' label is visible at the bottom right.

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

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

line - Line

Position and size

X: 0

dX: `this.getConnectedAgent(index).getX() - this.getX()`

Y: 0

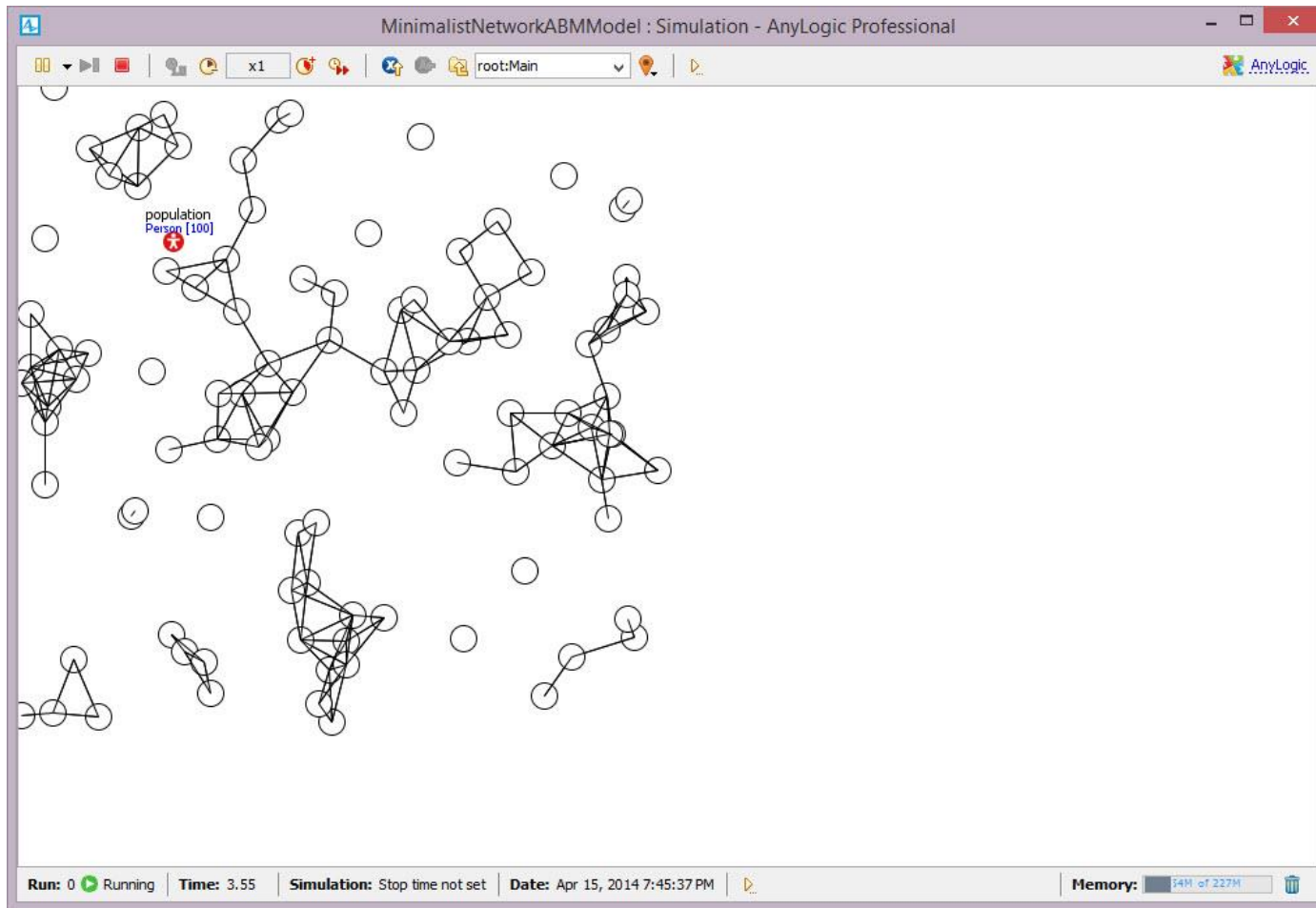
dY: `this.getConnectedAgent(index).getY() - this.getY()`

Z: 0

dZ: 0

Time units: minutes

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();
}

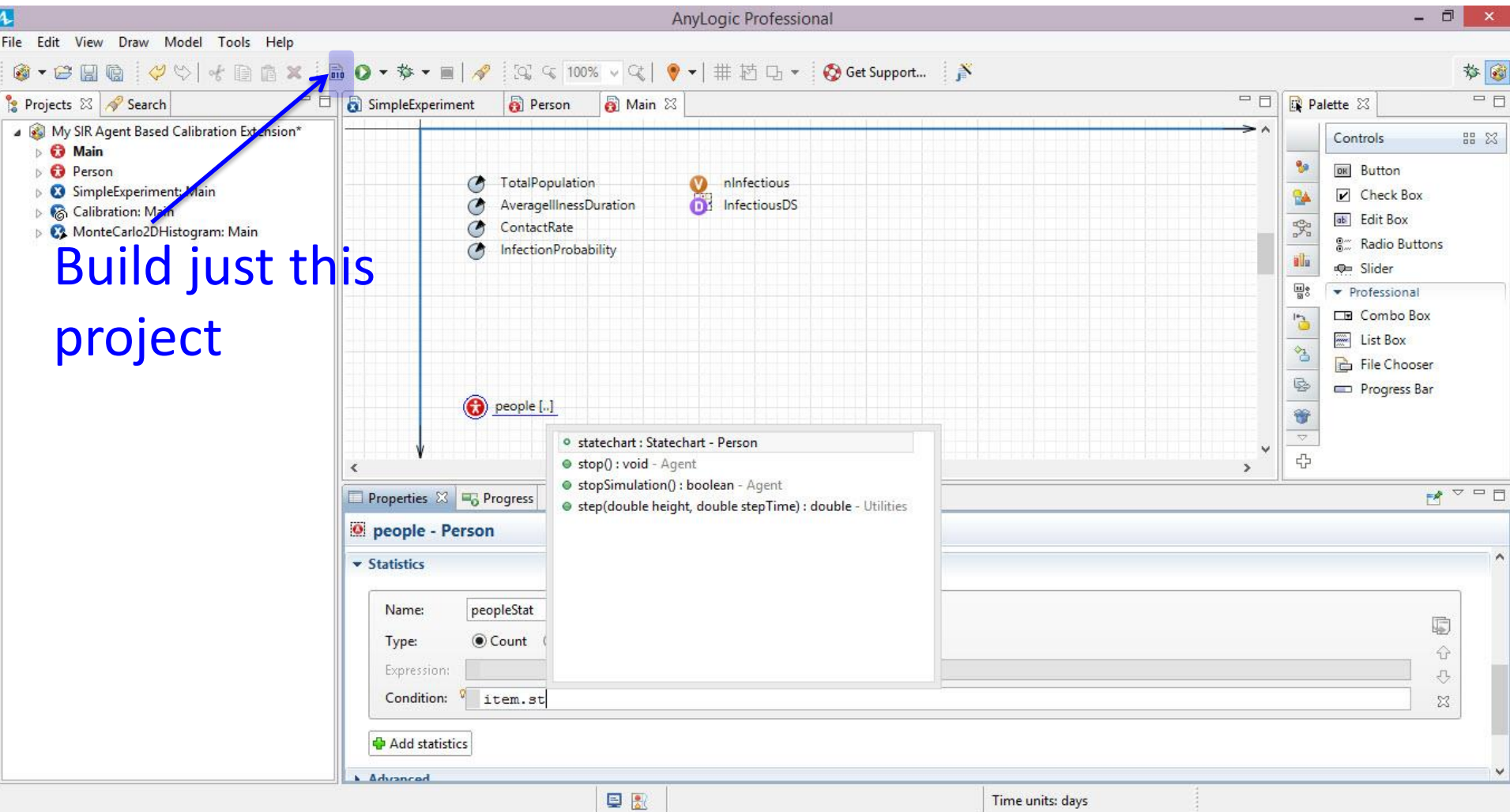
void onChange_initialPrevalenceOfInfection() {
    int index;
    index = 0;
    for ( Person object : Population ) {
        object.set_isInitiallyInfected((uniform() < initialPrevalenceOfInfection));
        index++;
    }
}
```

JVM
Byte
Code

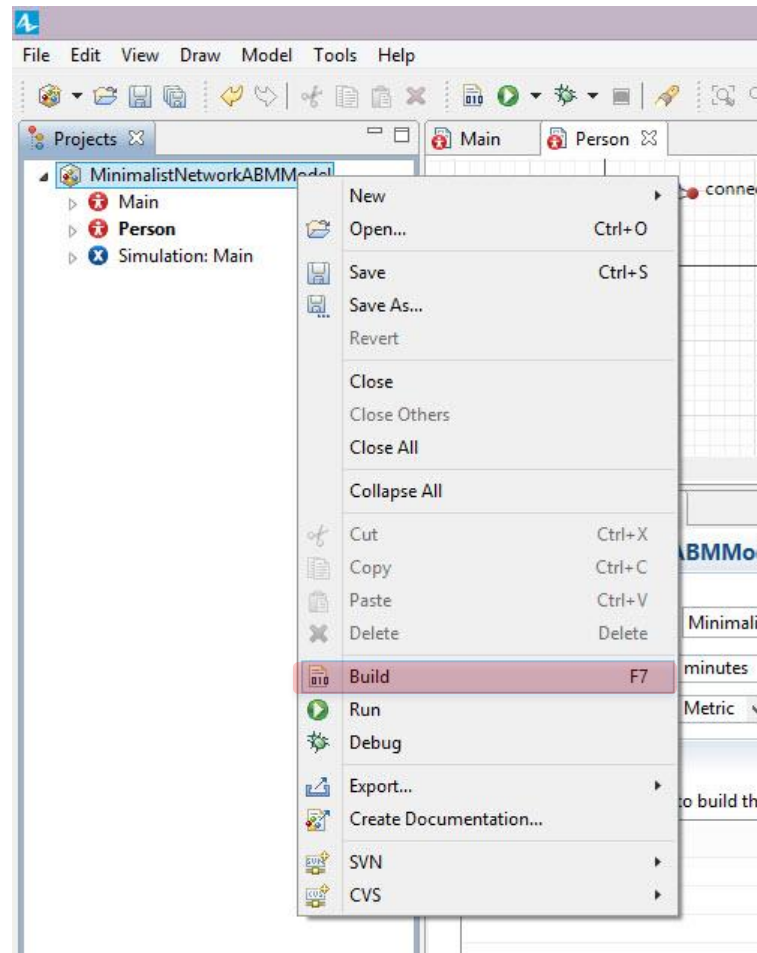
Person.class

“Build” Buttons

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



Alternative: Building via Context Menu



Builds Gone Good: Model Execution

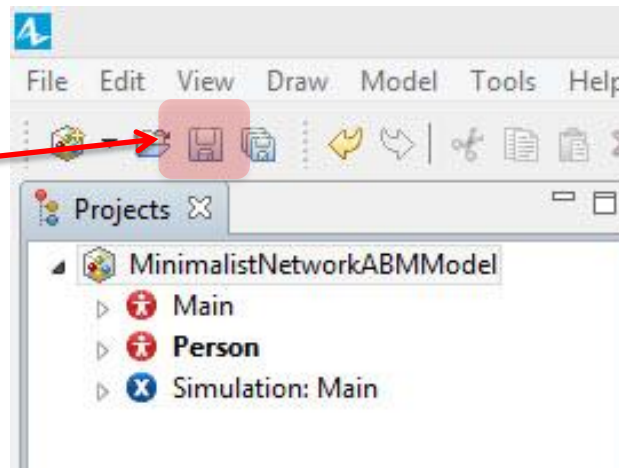
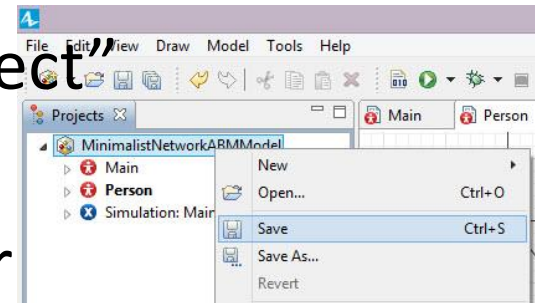
- The simulation is running
- Time is advancing in steps or as necessary to handle events
- Each agent class will typically have many particular agents in existence
 - Each agent will have a particular state
 - This population may fluctuate
- Variables will be changing value
- Presentation elements will 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





Hands on Model Use Ahead

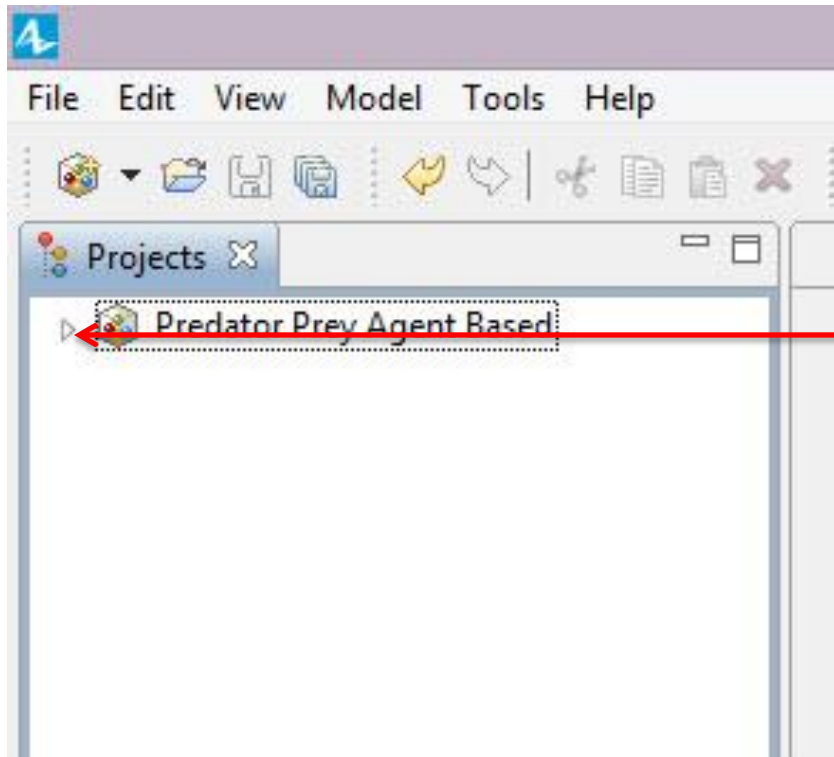


Load Sample Model:

Predator-Prey Agent Based

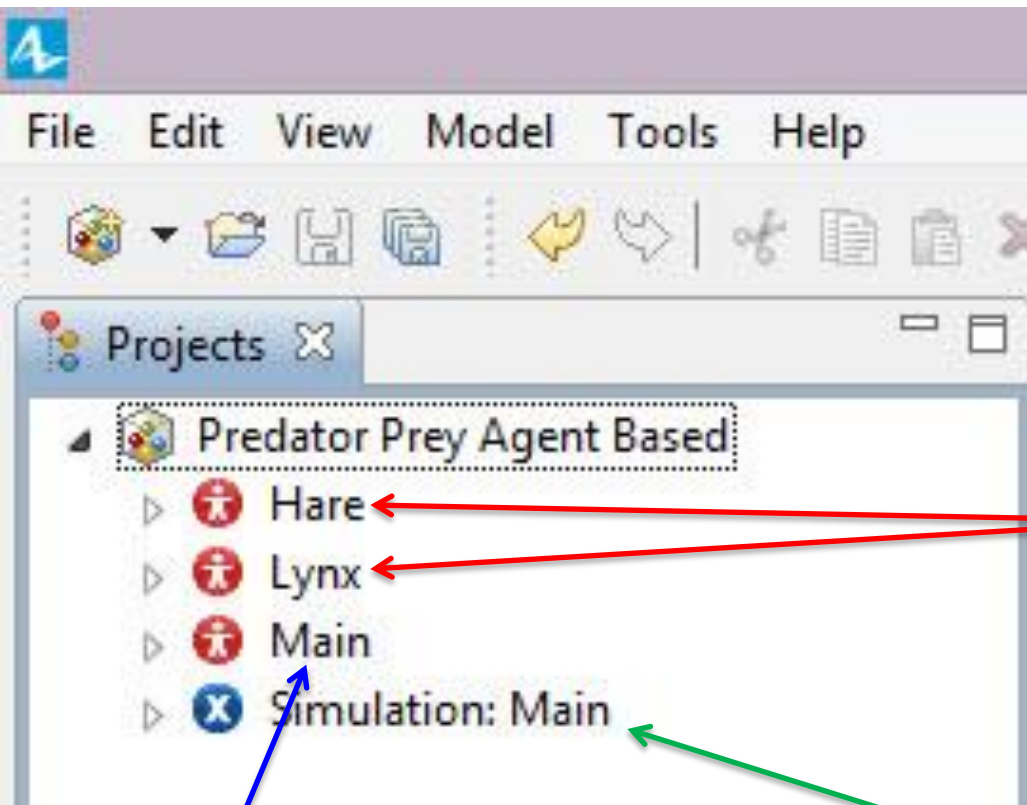
(Via “Example Models” under “Help” Menu)

After Loading in Model



Click on “+” to expand
Project details for
New model

Example “Classes”



“Agent” classes
(Define the *actors*)

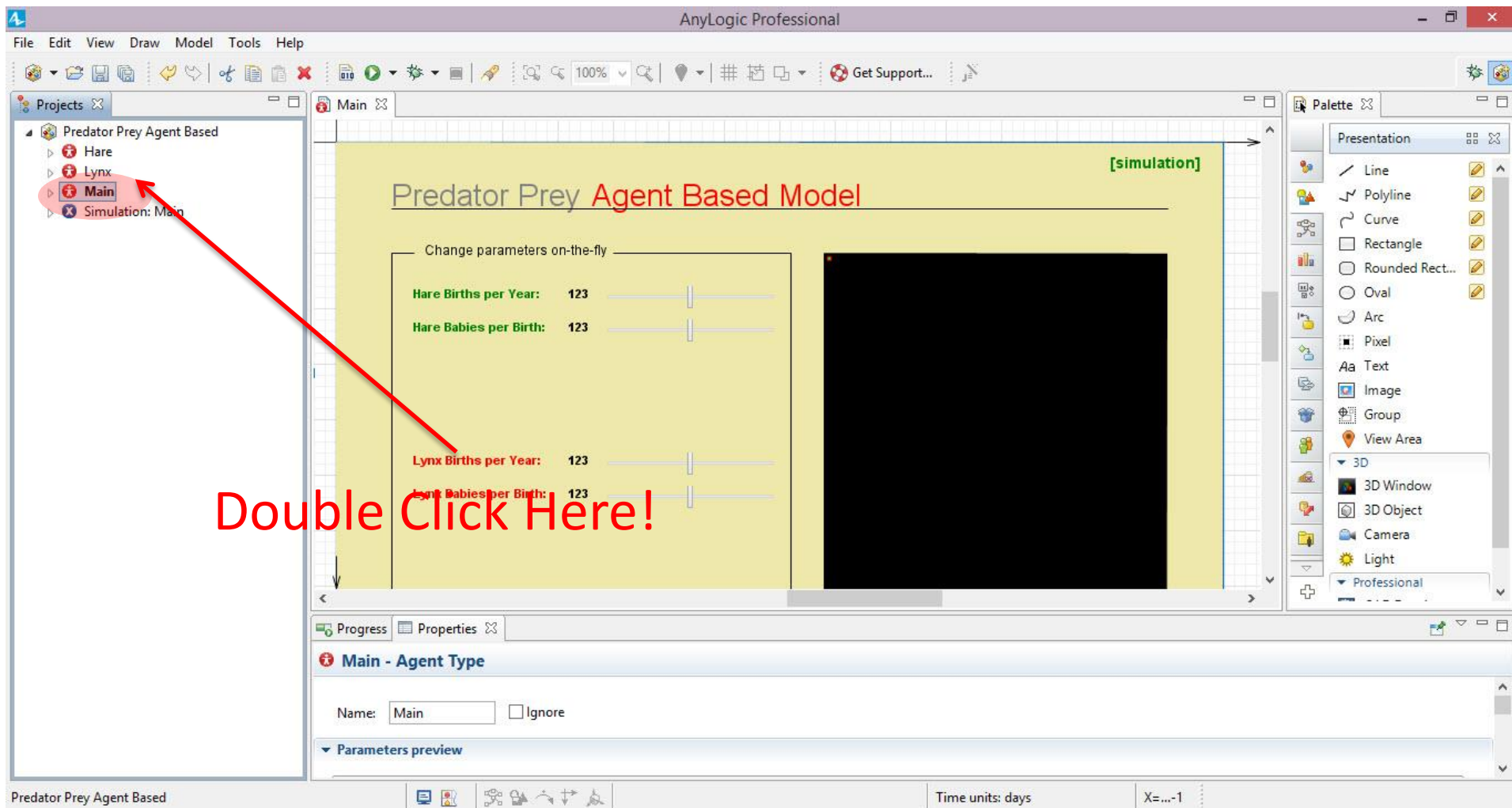
“Main” class
(Defines the “*Stage*” on
which agents circulate)

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

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

Double Click on “Main” Class Name to View this Class (Should Appear on Top Tab)



(Scroll to Left) Elements of a “Main” Class

These “parameters” specify
static model-wide characteristics

AnyLogic Professional

File Edit View Draw Model Tools Help

100%

Get Support...

Main

HaresInitial
LynxInitial
Width
CellWidth
HareNatality
HareNumberPerBirth
HareLifeExpectancy
HareMaxPerCell
LynxNatality
LynxNumberPerBirth
LynxLifeExpectancy
LynxHuntingPeriod
LynxHungerDeathThreshold

HaresInCell
auxGoodCells

NotOverpopulatedCellAround
RandomCellAround
XGlobal
YGlobal

lynx [...]
hares [...]

Predator Prey Agent Based Model

Change parameters on-the-fly

Hare Births per Year: 123
Hare Babies per Birth: 123
Lynx Births per Year: 123
Lynx Babies per Birth: 123

[simulation]

1
0.5
0

0 1 2 3 4 5 6 7 8 9 10

Time units: days

These “parameters” specify
static model-wide characteristics

Visual **input** elements used during simulation (param. setting)

These represent the agent
populations

These “functions”
Calculate things or can
change model behavior

Visual **output** elements used during simulation

Recall: “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 Class Defines the Characteristics & Behaviour of Agent Population Members

The screenshot displays the AnyLogic Professional software interface. On the left, the 'Projects' pane shows a tree structure under 'Predator Prey Agent Based' with sub-items: 'Hare', 'Lynx', 'Main', and 'Simulation: Main'. A red arrow points to the 'Lynx' item. The main workspace shows a diagram of the 'Lynx' agent, which is a yellow rounded rectangle containing a statechart. The statechart has a 'statechart' entry point, a 'NoLuck' state, a 'Hunt' state, and an 'Eat' state. A 'cell' object is visible in the workspace. The right side features a 'Palette' with various drawing tools. The bottom status bar shows 'Predator Prey Agent Based' and 'Time units: days'.

Double Click on “Lynx”!

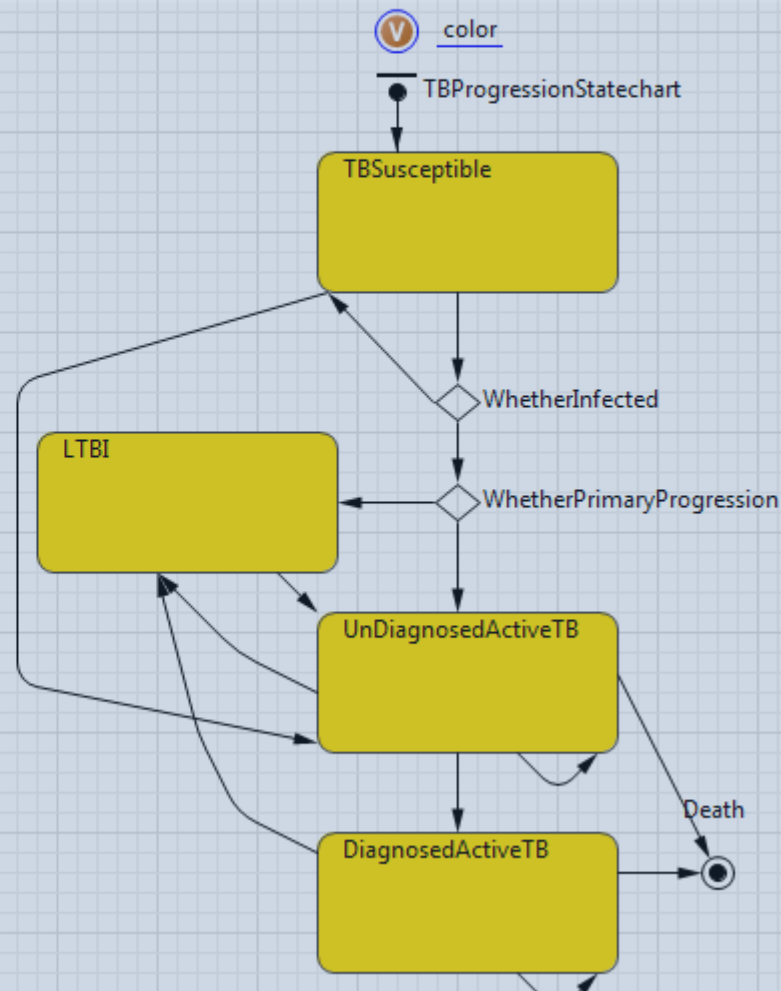
This defines the visual elements to be used for this object when it is displayed at runtime.

Common Agent-Class Elements

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

CirclePerimeterColorFromState
CirclePerimeterWidthFromState
ReactivationRateCoefficientForCKDStage
ReactivationRateForCKDStage
getDegree

Sex
Ethnicity
DaysPerTimeUnit
MeanDaysToNaturallyClearInfection



These “parameters” specify static agent characteristics

These describe the agent state & behaviour – the mechanisms that will govern agent dynamics

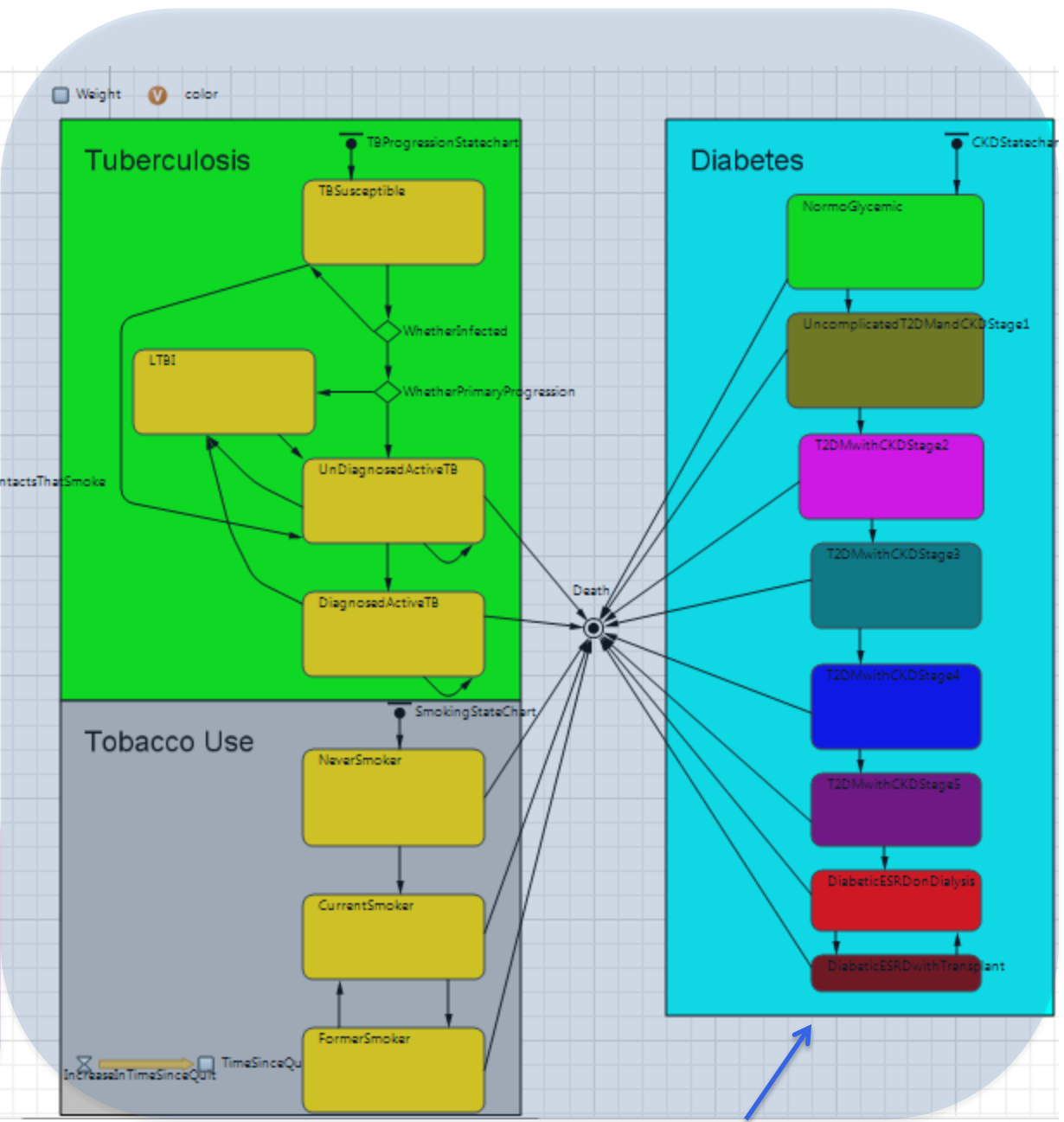
This defines the visual elements to be used for this object when it is displayed at runtime.

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

- CirclePerimeterColorFromState
- CirclePerimeterWidthFromState
- SmokingInitiationHazardCoefficientAsAFunctionOfFractionOfContactsThatSmoke
- CountSmokingContacts
- CountContacts
- FractionOfContactsThatSmoke
- SmokingInitiationHazard
- ReactivationRateCoefficientForSmokingStatus
- ReactivationRateCoefficientForCKDStage
- ReactivationRateForSmokingStatusAndCKDStage
- IsCurrentSmoker
- AgeCoefficientForSmokingInitiation
- getDegree

- Sex
- Ethnicity
- MeanDaysToNaturallyClearInfection
- ReactivationRateForNormoGlycemicPeople
- SmokingInitiationHazardLogisticSteepnessCoefficient
- SmokingInitiationHazardLogisticValueWhenNoContactsSmoke
- SmokingInitiationHazardLogisticValueWhenAllContactsSmoke
- ReactivationRateHazardForNeverSmoker
- ReactivationRateHazardForCurrentSmoker
- RapidnessOfDecreaseInReactivationRateWithTimeSinceQuit
- SmokingInitiationHazardLogisticMidpoint
- RapidnessOfDecreaseInChanceOfRelapseWithTimeSinceQuit
- DaysPerTimeUnit

These “parameters” give static characteristics of the agent



These describe the “behaviours” – the mechanisms that will govern agent dynamics

Setting Memory & Virtual Machine Arguments

The screenshot displays the AnyLogic Professional software interface. The main workspace shows a yellow background with the title "Predator Prey Agent Based Model" and a subtitle "[experiment setup]". Below the title, a text box describes the predator-prey problem. A "Step 1" section is highlighted, showing "Hares: 123" and "Lynx: 123" with corresponding sliders. The left sidebar shows the project structure with "Predator Prey Agent Based" as the root, containing "Hare", "Lynx", "Main", and "Simulation: Main". The bottom panel, titled "Simulation - Simulation Experiment", shows the "Java actions" section with the following code:

```
Initial experiment setup:  
Before each experiment run:  
Before simulation run:  
getEngine().getPresentation().setPresentable( root );  
root.HaresInitial = HaresInitial;  
root.LynxInitial = LynxInitial;  
After simulation run:
```

The bottom status bar indicates "Time units: days" and "X=...44".

Close “Predator-Prey” Model

Right Click on project name

(“Predator Prey Agent

Based”) & select “Close”

