Performance & Computational Resource Considerations

Nathaniel Osgood

Using Modeling to Prepare for Changing Healthcare Needs

Duke-NUS

April 16, 2014

The Computational Burden of ABM

- Agent-Based models impose large computational burden
- Key factors:
  - “More moving parts”: Lots of values to calculate and manipulate
  - Requirement of running multiple realizations

ABM and Computational Resource Use

- The computational burden of Agent-Based Models limits value delivered
  - Opportunity cost: Reduces time spent in exploration of model results & insights gained
  - Limited time => Less thorough exploration of parameter space => Reduced quality of calibration
  - Inhibits adoption

Significant Computational Factors

- Event-limited performance
- Statistics
- Visualization
- Network Construction
- Output of data

Significant Computational Factors

- Event-limited performance
- Statistics
- Visualization
- Network Construction
- Output of data

Event-Limited Performance

- AnyLogic utilizes an “event driven” scheduler
  - The more events, the more the scheduler has to “wake up” to do things
  - In addition to the work to be done, there is some bookkeeping involved in the occurrence of an event
- One important performance saving: When sending messages, Replacing `a.send` by `a.deliver`
  - Note that this is not possible in all cases, and can lead to infinite loops when agent A `deliver`s to B, and B `deliver`s to A
- If there is greater occurrence of events (either explicit or implicit in e.g. transitions or messages), this will generally adversely affect performance

Hands on Model Use Ahead

Load Sample Model:

SIR Agent Based

(Via “Example Models” and then “How To Models” under “Help” Menu)

Suggestions

- Permit disabling of visual elements
  - Use parameter variation experiment or set update freq
- Lower event frequency
  - Use dynamic events
  - Do more on firing of each event
  - Disable when not required
- Where possible, use “bookkeeping” on transitions (increase/decrease counts) rather than statistics
- Use a profiler to find where spending time
- Send events only for “infecting dose” (rather than exposure), where possible
- Use deliver rather than send

Example of Reducing Events: Messages

- Sometimes there are simple ways to reduce event occurrence
- Example replacement
  - Worse performance: Sending “exposure” messages with rate α, each having a likelihood β of infection upon receipt
  - Better performance: Send “infect” messages with rate $\alpha\beta$
- Such simplifications are context-specific
  - For example, this transformation is much harder if the likelihood of infection given exposure varies by individual

Significant Computational Factors

- Event-limited performance
- Statistics
- Visualization
- Network Construction
- Output of data

Statistics

- AnyLogic’s capacity to define “Statistics” over a population provides an easy way to compute population statistics
- Downside: Each computed statistic requires a full iteration through each member of the population
- Example: Classifying people into each of 17 age categories using “Statistics” requires 17 passes through the population!
  - This could in principle be done in a single pass – with each individual just incrementing different bins of a histogram

Significant Computational Factors

- Event-limited performance
- Statistics
- Visualization
- Network Construction
- Output of data

Visualization

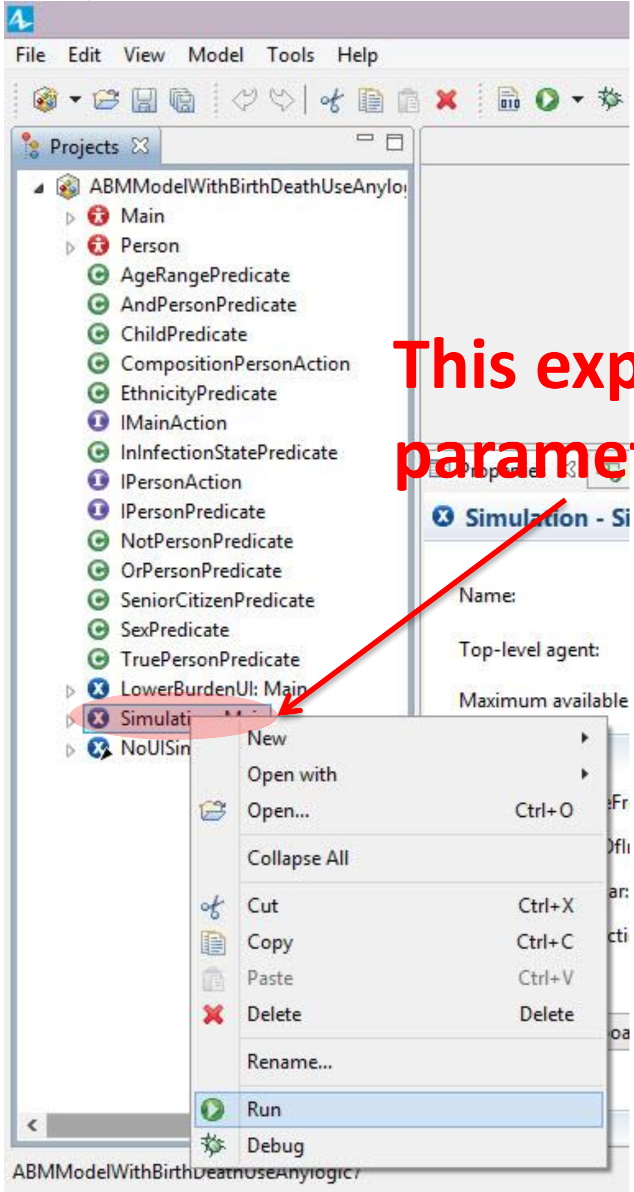
- The visual presentations of elements of a model takes considerable
  - Time
  - Memory
- Example: Dynamic properties
- Disabling visualization can lead to much faster operation
- Options
  - Creating a model without “presentation” properties
  - Set running settings so that infrequent updates
  - Running the model using “runFast()”

⚠

Hands on Model Use Ahead

Load ***Agent-Based Model with Birth Death***

Run the Experiment "Simulation" & Time Elapsed Duration until Completes



This experiment uses the default parameter values

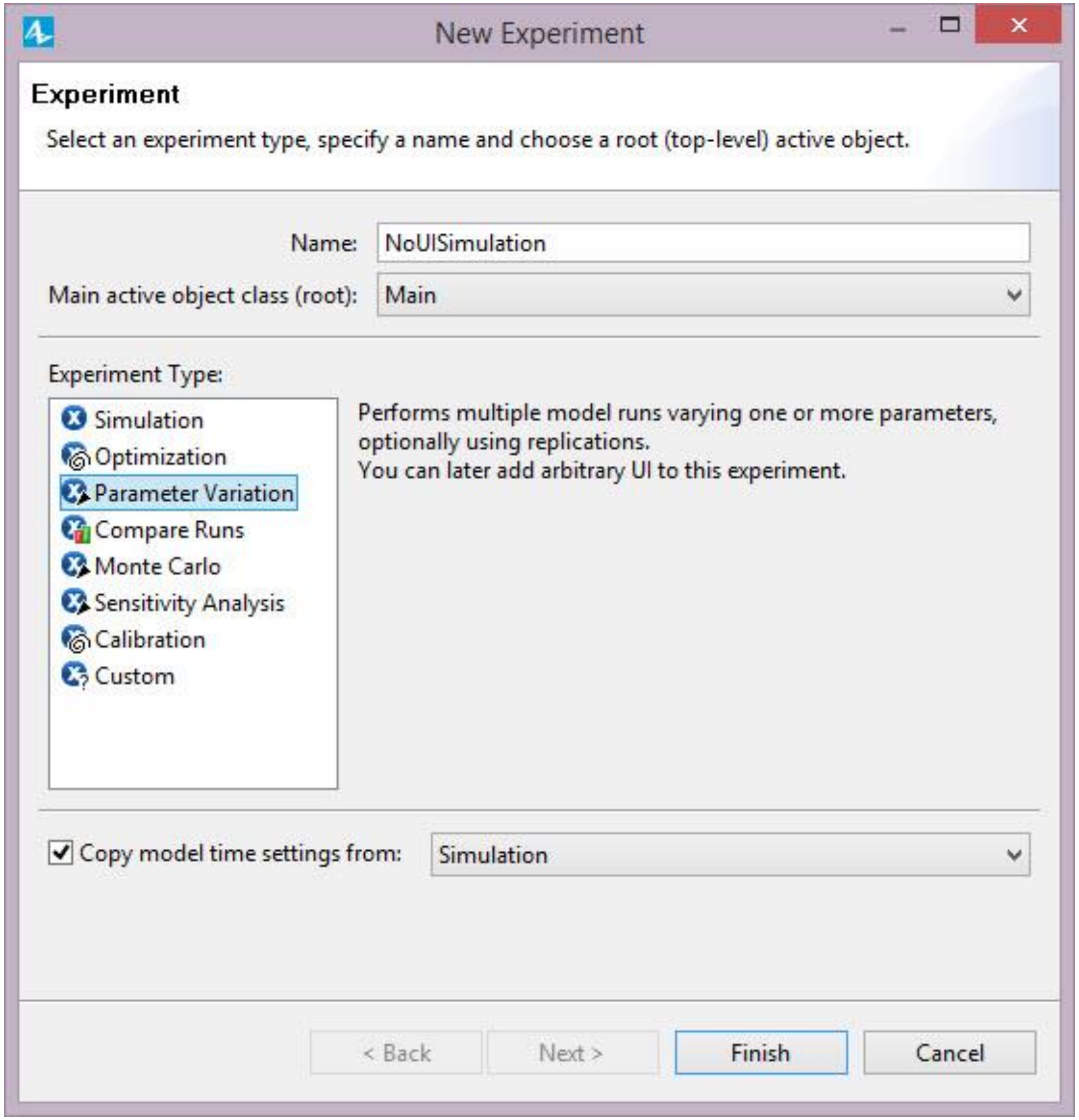
Running the Model using RunFast

- Create a Parameter variation experiment with 1 realization (no exploration of parameter space)
- Either
  - Run from the “Run” Menu item
  - Create a button to run
- Advantages
  - Simple (no custom coding)
  - Fastest
  - Model can still contain dynamic presentation properties & display visualizations in other experiments
- Disadvantages: No option to visualize

Enabling Faster Running by Settings & Displaying only when Required

- Create a simulation with slow update & don’t display by default
- Advantages: Retain option of visualization where desired
- Disadvantages:
  - A bit of custom coding required
  - Slower
  - Memory is still allocated for presentation elements

Method 1: Adding an Experiment without Visualization via Param Variation



Experiment

Select an experiment type, specify a name and choose a root (top-level) active object.

Name: NoUISimulation

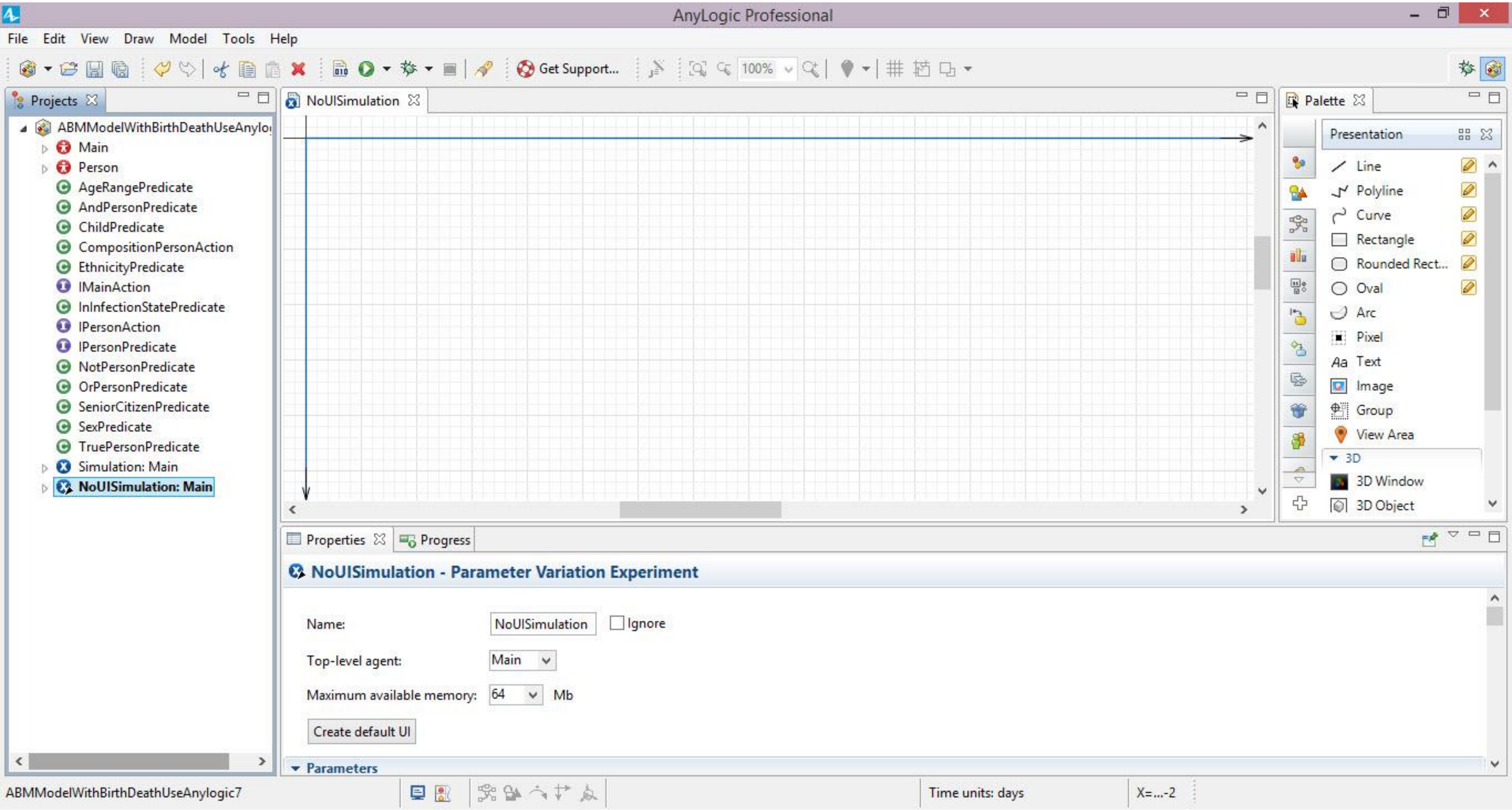
Main active object class (root): Main

Experiment Type:

Performs multiple model runs varying one or more parameters, optionally using replications.
You can later add arbitrary UI to this experiment.

☑ Copy model time settings from: Simulation

New Experiment



NoUISimulation - Parameter Variation Experiment

Name: NoUISimulation ☐ Ignore

Top-level agent: Main

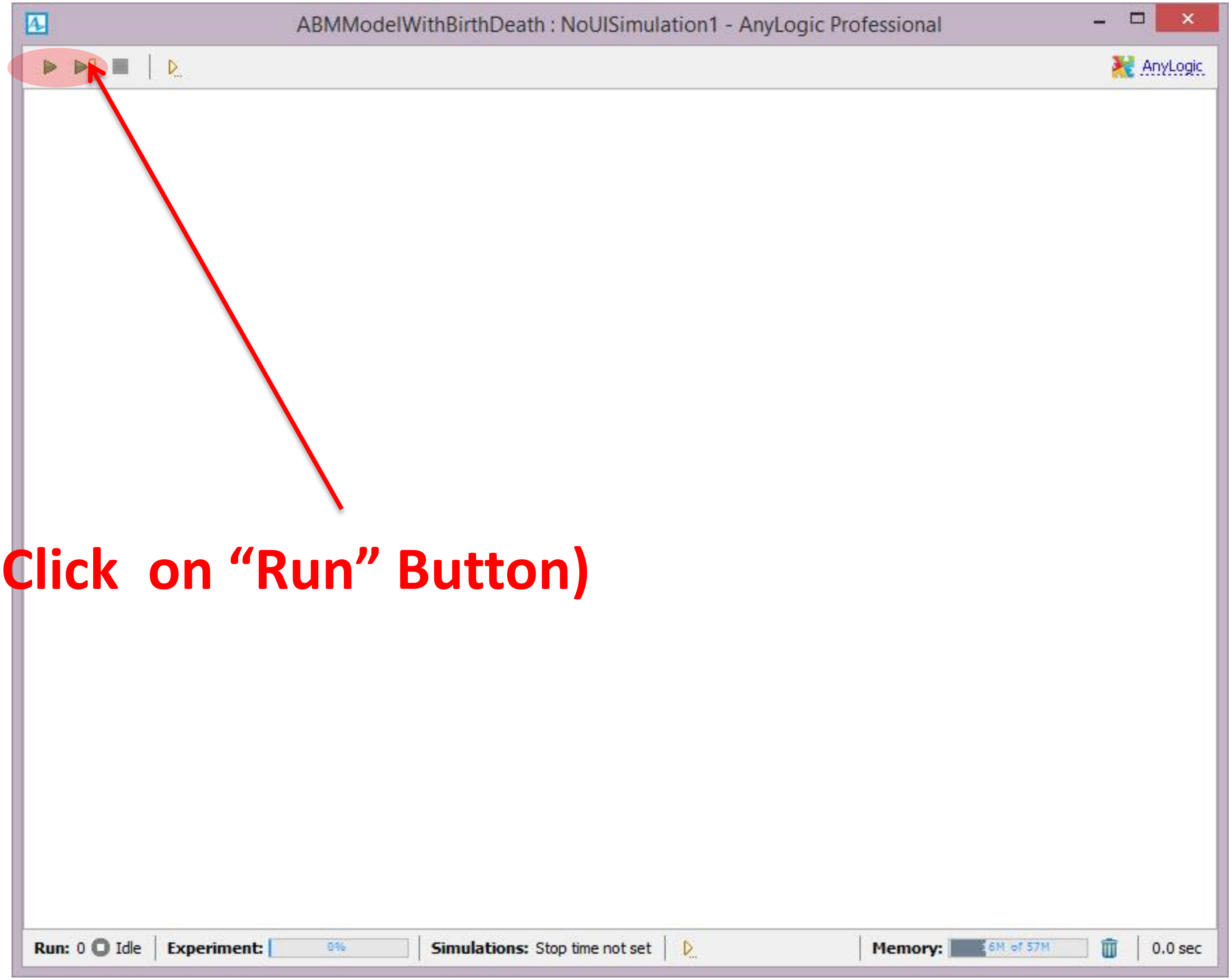
Maximum available memory: 64 Mb

Create default UI

Parameters

Running & Timing the Experiment

(b/c parameter variation exp., uses runFast())


ABMModelWithBirthDeath : NoUISimulation1 - AnyLogic Professional
AnyLogic

Run: 0 Idle

Experiment: 0%

Simulations: Stop time not set

Memory: 6M of 57M

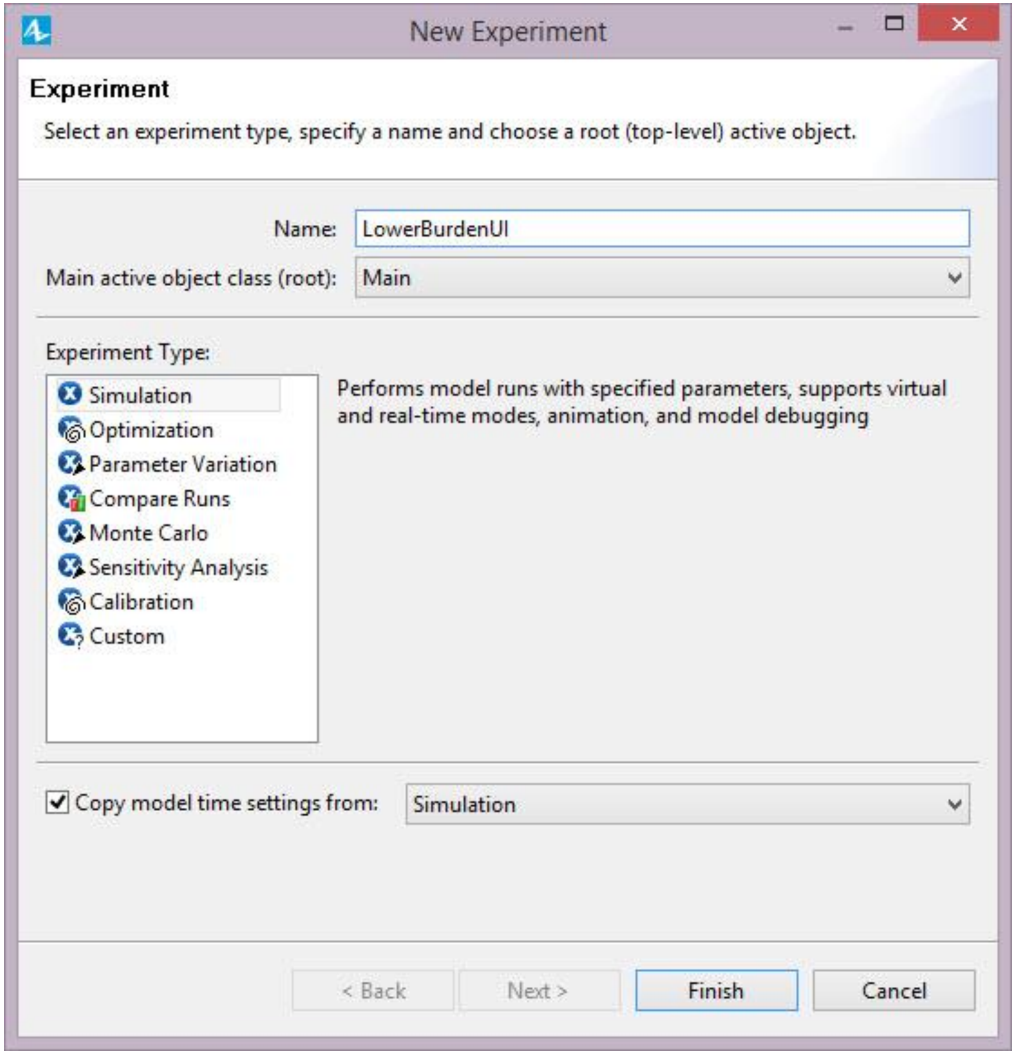
0.0 sec

Click on “Run” Button)

Comparing Timings

- The time required to run the model in the experiment that avoids a UI should be a fraction of that required with the UI

A New Experiment



Experiment

Select an experiment type, specify a name and choose a root (top-level) active object.

Name: LowerBurdenUI

Main active object class (root): Main

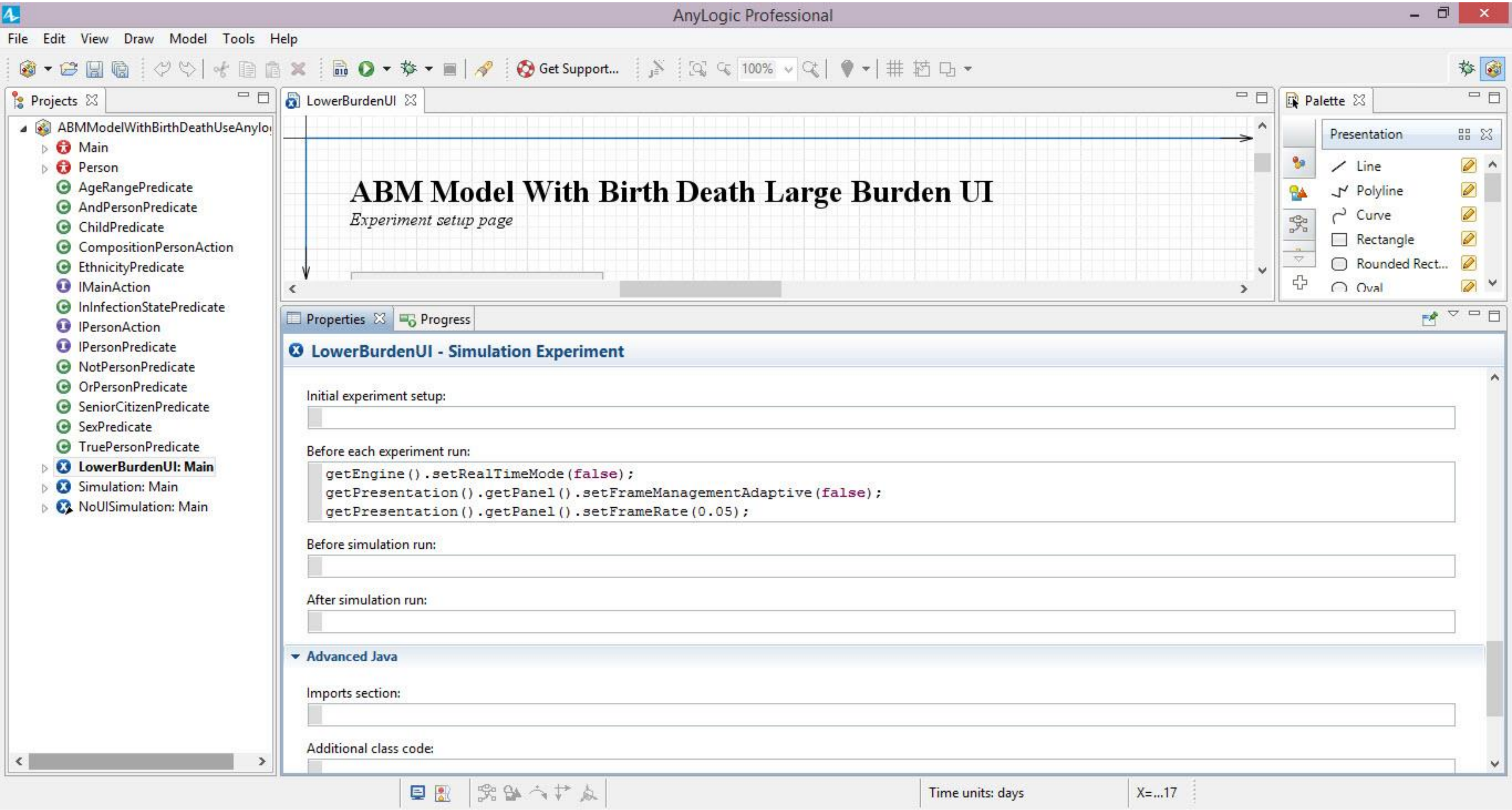
Experiment Type:

- Simulation
- Optimization
- Parameter Variation
- Compare Runs
- Monte Carlo
- Sensitivity Analysis
- Calibration
- Custom

Performs model runs with specified parameters, supports virtual and real-time modes, animation, and model debugging

☑ Copy model time settings from: Simulation

Insert Code Into "Before Each Experiment Run" in "Advanced" tab for Experiment



LowerBurdenUI - Simulation Experiment

Initial experiment setup:

Before each experiment run:

```
getEngine().setRealTimeMode(false);
getPresentation().getPanel().setFrameManagementAdaptive(false);
getPresentation().getPanel().setFrameRate(0.05);
```

Before simulation run:

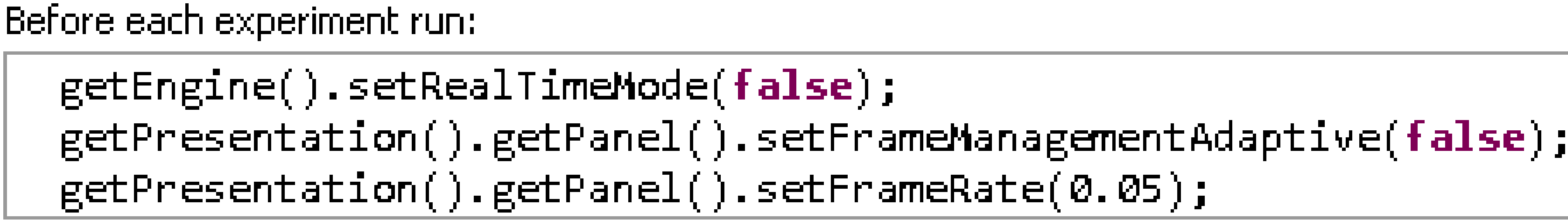
After simulation run:

Advanced Java

Imports section:

Additional class code:

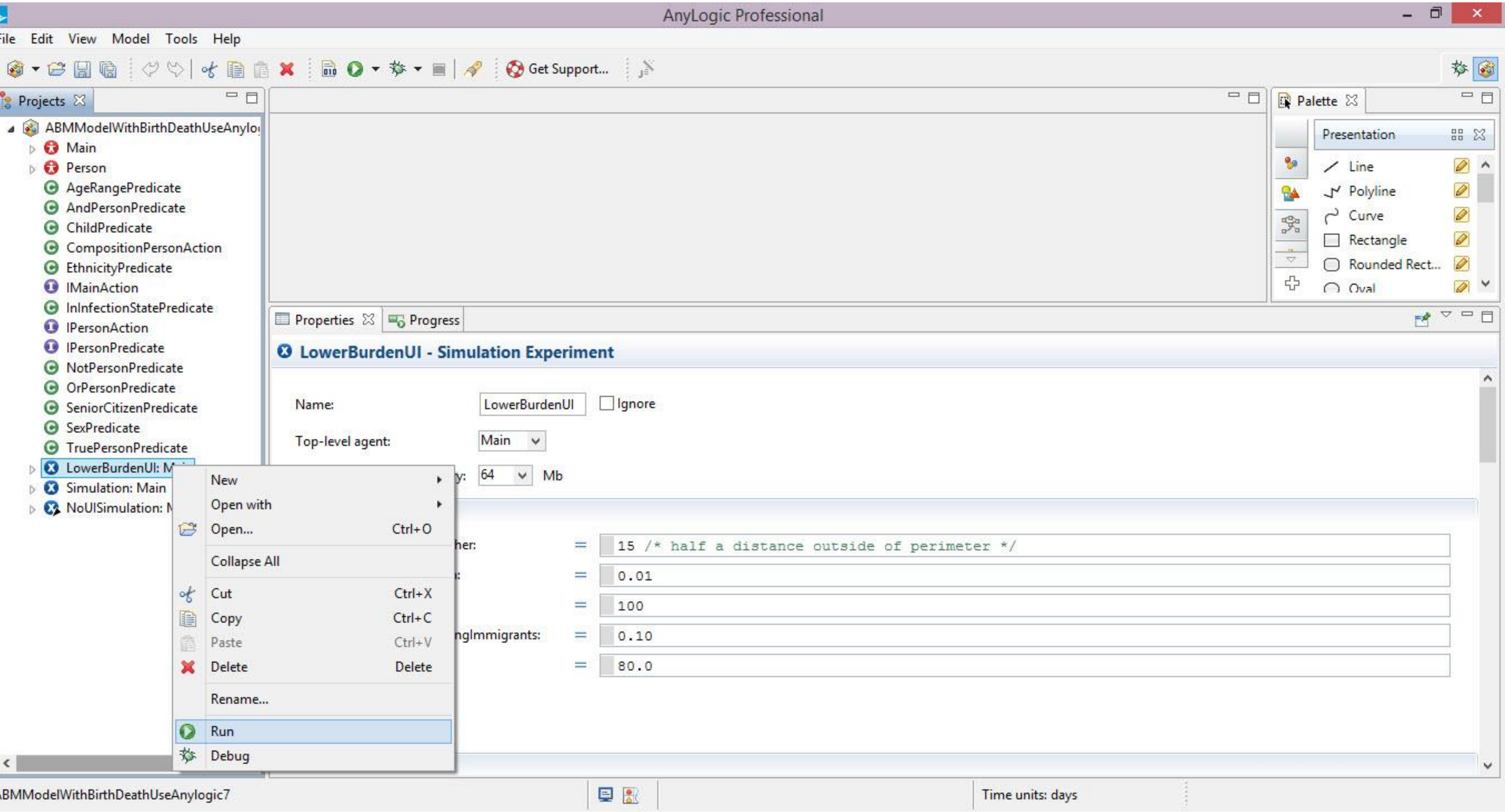
Code to Insert



Before each experiment run:

```
  getEngine().setRealTimeMode(false);
  getPresentation().getPanel().setFrameManagementAdaptive(false);
  getPresentation().getPanel().setFrameRate(0.05);
```

Run the Experiment



LowerBurdenUI - Simulation Experiment

Name: LowerBurdenUI ☐ Ignore

Top-level agent: Main

y: 64 Mb

her: = 15 /* half a distance outside of perimeter */

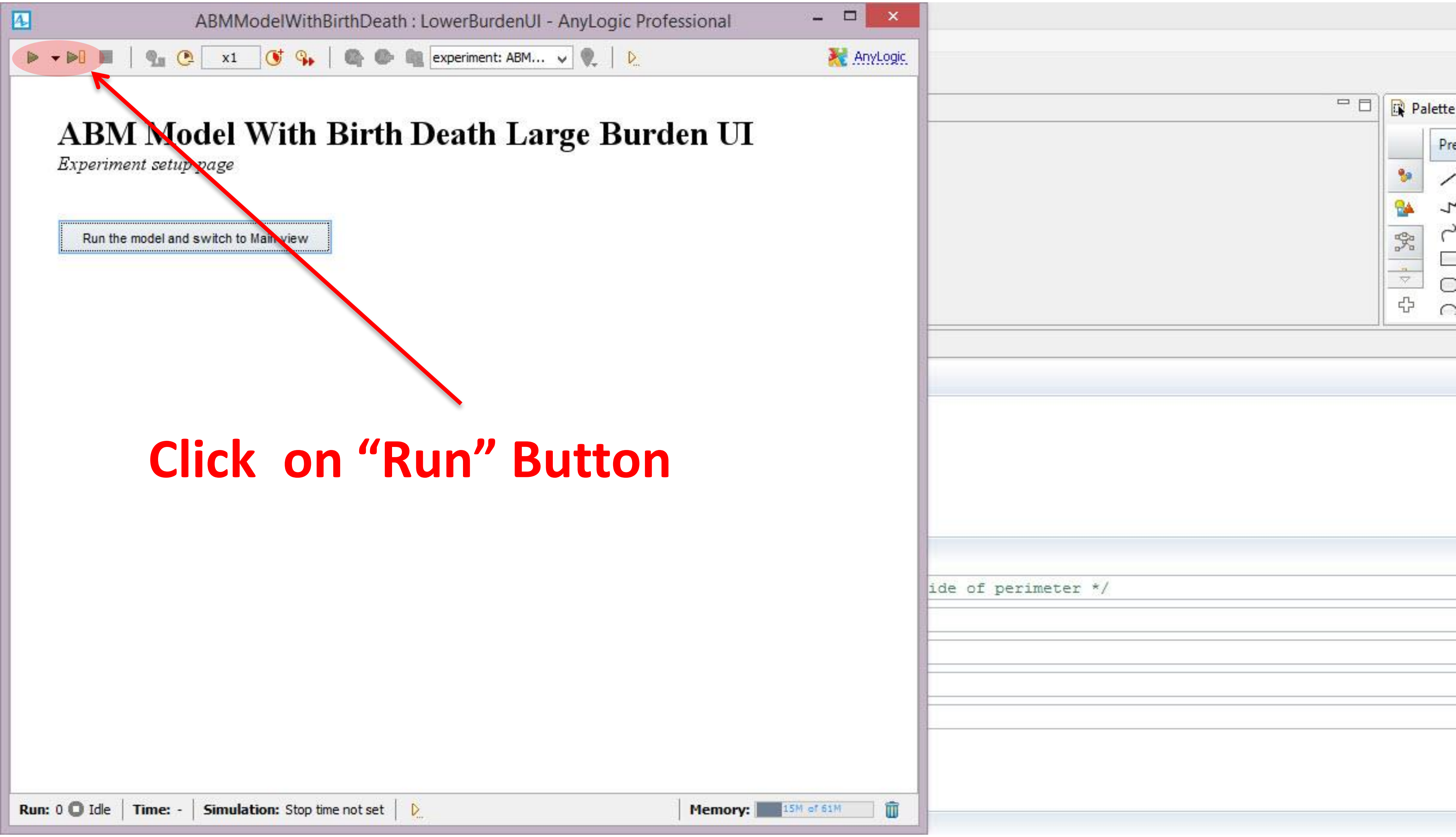
: = 0.01

= 100

ngImmigrants: = 0.10

= 80.0

Run & Time Model without Displaying UI



ABMModelWithBirthDeath : LowerBurdenUI - AnyLogic Professional
x1
experiment: ABM...
AnyLogic
Palette
Run the model and switch to Main view
Run: 0 Idle
Time: -
Simulation: Stop time not set
Memory: 15M of 61M
ide of perimeter */

ABM Model With Birth Death Large Burden UI

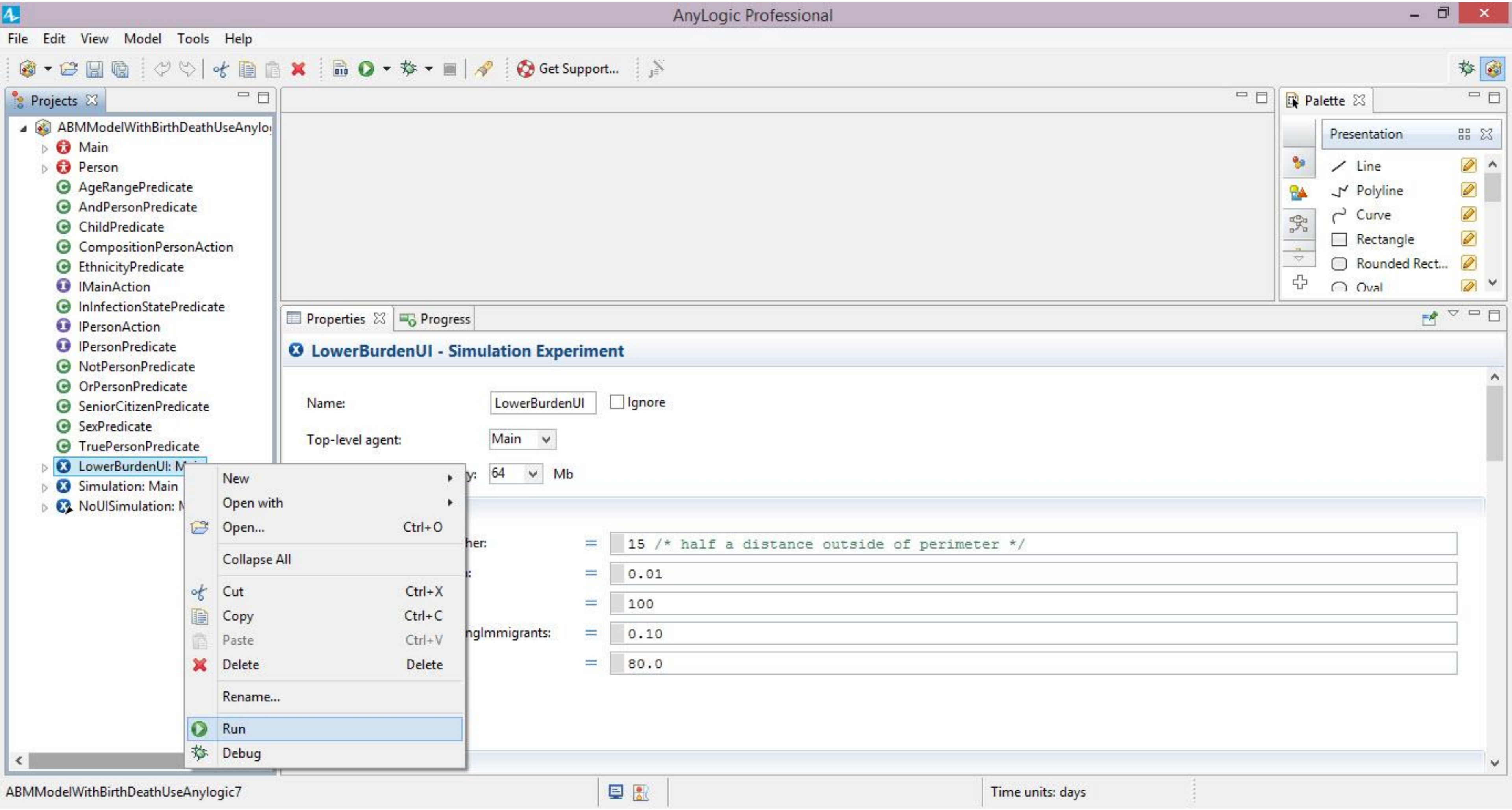
Experiment setup page

Click on "Run" Button

Without the UI, Model should be Fast!

- Note that the inserted code allows this trick to work
  - Not viewing the UI in the original simulation experiment (which is missing this inserted code) will not similarly shorten the running time!

Run the Experiment



LowerBurdenUI - Simulation Experiment

Name: LowerBurdenUI □ Ignore

Top-level agent: Main

y: 64 Mb

her: = 15 /* half a distance outside of perimeter */

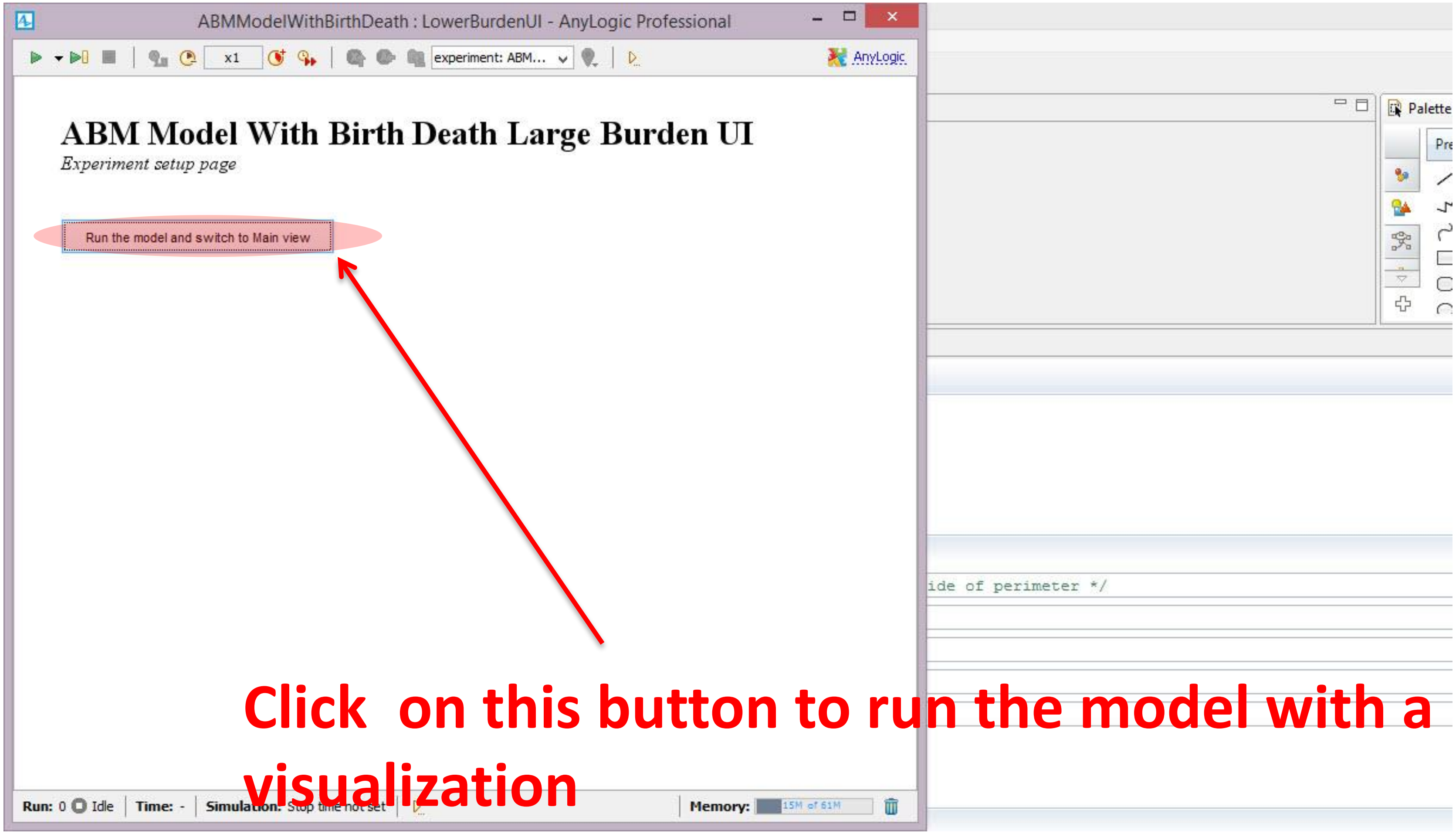
: = 0.01

= 100

ngImmigrants: = 0.10

= 80.0

Run & Time the “LowerBurdenUI” Model *with* the Visualization



ABMModelWithBirthDeath : LowerBurdenUI - AnyLogic Professional

ABM Model With Birth Death Large Burden UI

Experiment setup page

Run the model and switch to Main view

Run: 0 Idle | Time: - | Simulation: Stop time not set | Memory: 15M of 61M

Click on this button to run the model with a visualization

Visualization with New Settings

- The custom settings should significantly lower the time required to run the simulation when compared to the default settings

Significant Computational Factors

- Event-limited performance
- Statistics
- Visualization
- Network Construction
- Output of data

Network Construction

- AnyLogic’s Scale Free network requires a long time to run
- We have found gains by implementing the Barabasi-Albert algorithm ourselves

Significant Computational Factors

- Event-limited performance
- Statistics
- Visualization
- Network Construction
- Output of data

Database Output

- Batch up data to send to the database
  - Send in one big call to database, rather than multiple calls
- Use local database
- Record smaller subset of data
- Record less frequently
- Record fewer types of data

Model Space Demands

- Models with large populations can require much space
- I believe that the space demands can be particularly large when visualization is enabled
- You can enable space available for models in the "experiment" area
- Ways to reduce space demands
  - Accumulate less data (less frequently/fewer data items)
  - Write data out rather than accumulating in datasets

Exploiting Opportunities for Concurrency Using Distributed Processing

- ABM offers opportunities for parallel processing
- Two particularly manifest opportunities for concurrency require different levels of sophistication to exploit
  - "Embarassingly parallel" & easy to exploit: Concurrency between model realizations. One can readily run different realizations of a model in parallel (e.g. on different machines) & harvest results
  - Also parallelizable, but harder to exploit: Concurrency opportunities between distinct agents. While agent processing could in principle be parallelized, dependencies between agents (e.g. via message sending & joined flows) makes this more challenging to exploit.