Agent Mobility in 2D Landscapes (Bonus: Some UI Customization)

Nathaniel Osgood

Using Modeling to Prepare for Changing Healthcare Needs

Duke-NUS

April 16, 2014

Reminder: Agent Spatial Embedding

- Spatial embedding of agents is key to
  - Expressing essential dynamics for problems Locality of influence/Transmission
  - Insight into certain phenomena (spatial concentration, percolation, spatial reference modes)
- Spatial embedding can permit GIS integration

2D Spatial Embedding: Two Options

- Continuous embedding (e.g. Wandering elephants, our built-up model)
  - No physical exclusion: Agents are assumed to be small compared to landscape scale, and exhibit arbitrary spatial density without interfering
  - We have seen this much with distributing agents initially around the space, adding agents in
- Discrete cells (e.g. The Game of Life, Agent-based predator prey, Schelling Segregation)
  - Divided into “Columns” and “Rows”
  - Physical exclusion: Only one agent in a cell at a time

The Locus of Control: Environment

- The Anylogic Environment sets the parameters for the nature of the 2D landscape
  - Width
  - Breadth
  - Continuous vs. Discrete
  - Character of discrete neighbourhoods (cardinal directions vs. Euclidian { N,NE,E,SE,S,SW,W,NW}

Reminder: Common Division

- Endogenous
  - Things whose dynamics are calculated as part of the model
- Exogenous
  - Things that are included in model consideration, but are specified externally
    - Time series
    - Constants
- Ignored/Excluded
  - Things outside the boundary of the model

Motivations for Including Endogenous Factors

- Maintaining factors as endogenous (rather than pre-specified as exogenous) lends
  - Extra flexibility for more accurately capturing effects of
    - Interventions
    - Alternative exogenous scenarios
  - Greater robustness in the context of changes
  - Support for translations to other contexts
- Keeping greater detail requires more data & implementation work, but allows our models to be translated to other contexts & times

Example

$$\begin{bmatrix} x_{11} & x_{12} & 1-x_{11}-x_{12} \\ x_{21} & x_{22} & 1-x_{21}-x_{22} \\ x_{31} & x_{32} & 1-x_{31}-x_{32} \end{bmatrix}$$

- Mixing Matrix (specifies fraction of population A's contact that occur with populations B & C
- Preference matrix
  - Scales to capture fluctuating population captures relative preference
  - can't specify where to test
- Mobility-based methods with mobility patterns hard-coded
  - this is challenged for interventions which change e.g. mixing opportunities and mobility
- Mobility-based methods with preference-based mobility model

Agent Mobility

- Thus far, we have looked at spatial dynamics where each agent remains stationary
  - Continuous space (static & dynamic populations)
  - Discrete space (cellular automata)

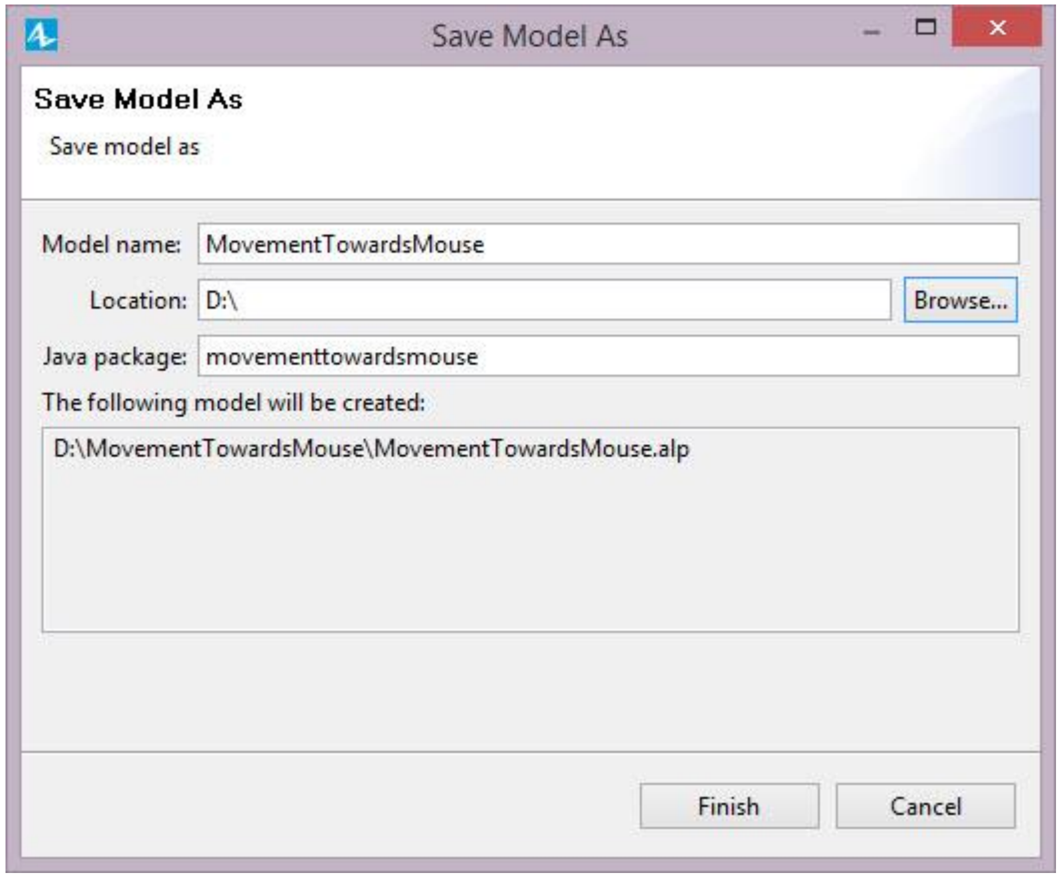
2D Spatial Embedding: Mobility Implications

- Continuous embedding (e.g. Wandering elephants)
  - No physical exclusion: Agents are assumed to be small compared to landscape scale, and exhibit arbitrary spatial density without interfering
  - Agents move
    - In a direction
    - With some speed
- Discrete cells (e.g. Agent-based predator prey, Schelling Segregation)
  - Divided into "Columns" and "Rows"
  - Physical exclusion: Only one agent in a cell at a time
  - Agents move continuously or discontinuously from cell to cell

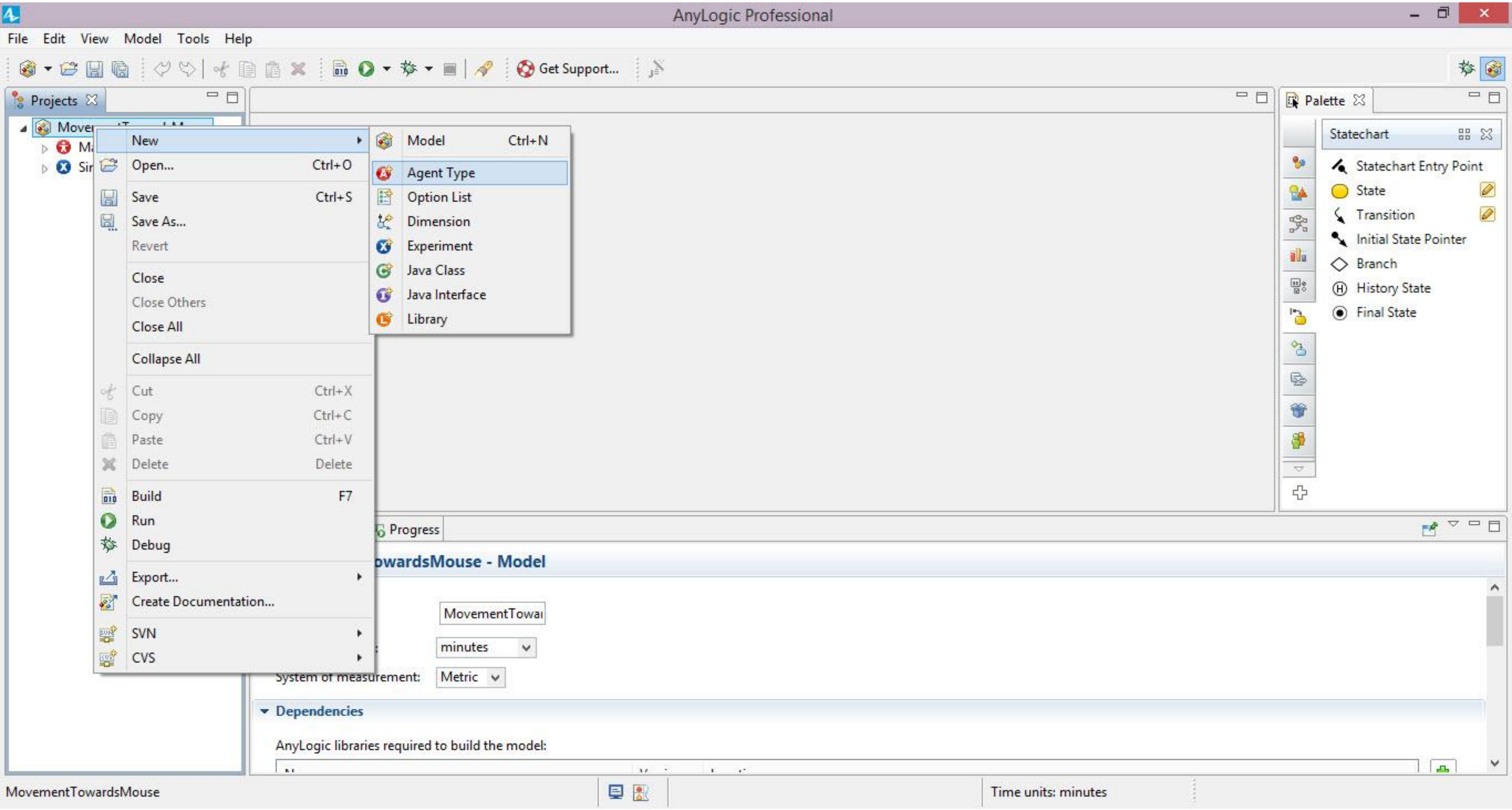
Continuous Space: Relevant Methods (To call on *Agent*)

- Controlling
  - moveTo(x,y) : initiates agent movement to location
  - setVelocity(v)
  - setXY(x,y): initial location
  - jumpTo(x,y): moves agent to location
  - setRotation()
- Basic info
  - getX()/getY()
  - isMoving()
  - getTargetX()/getTargetY()
    - Where heading to?
  - getRotation()

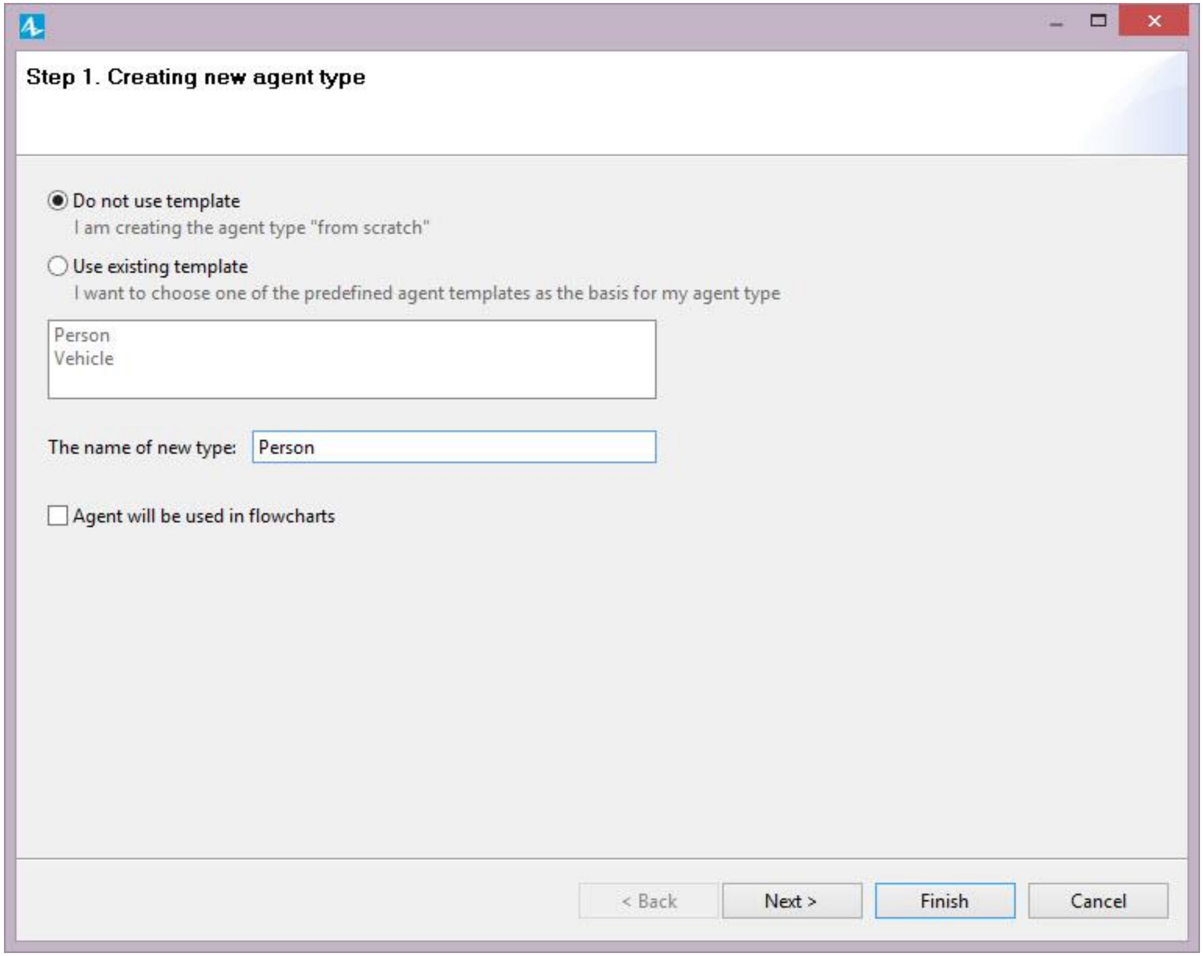
Create a New Project Called "MovementTowardsMouse"



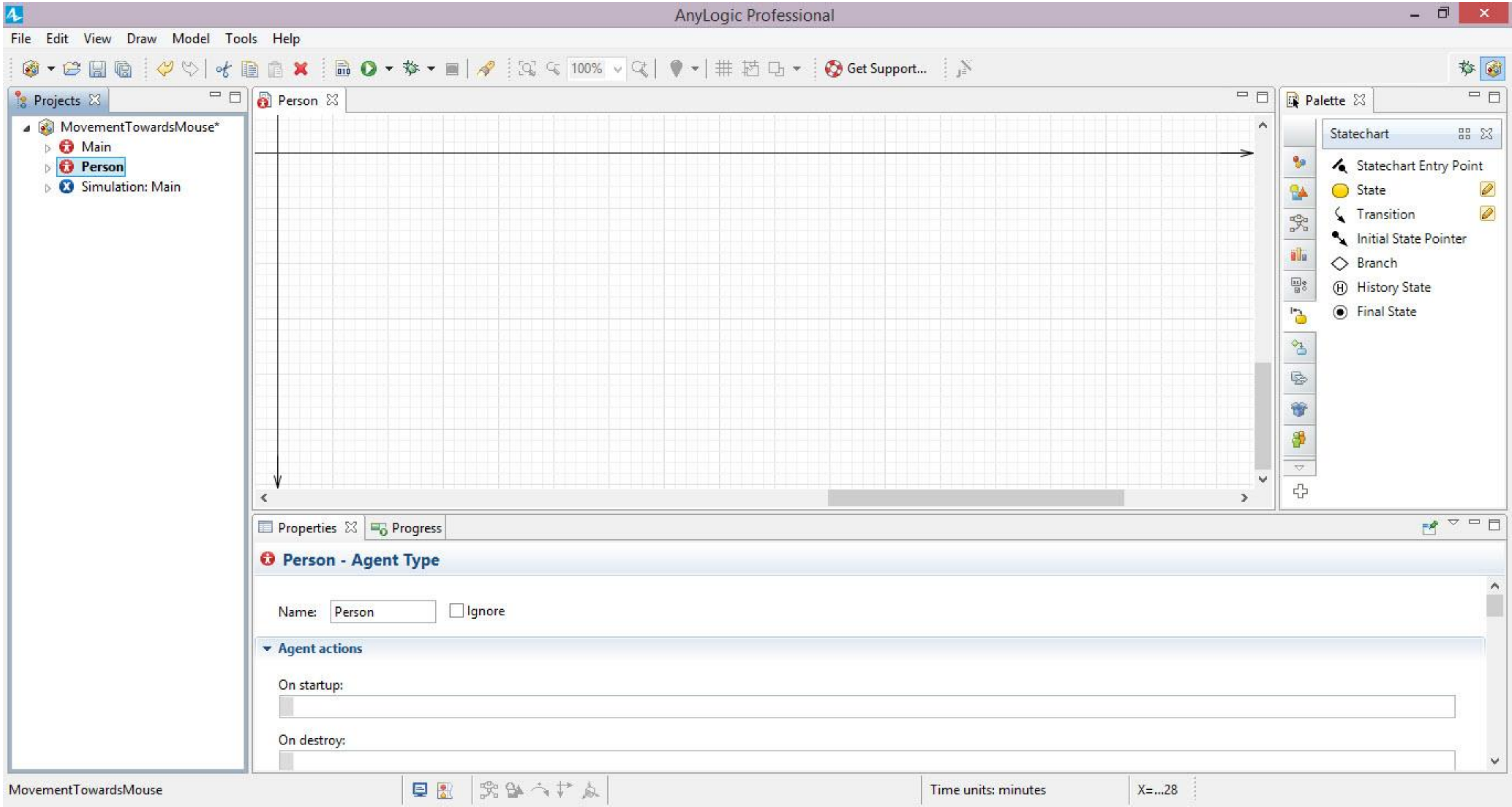
Add a New Active Object Class



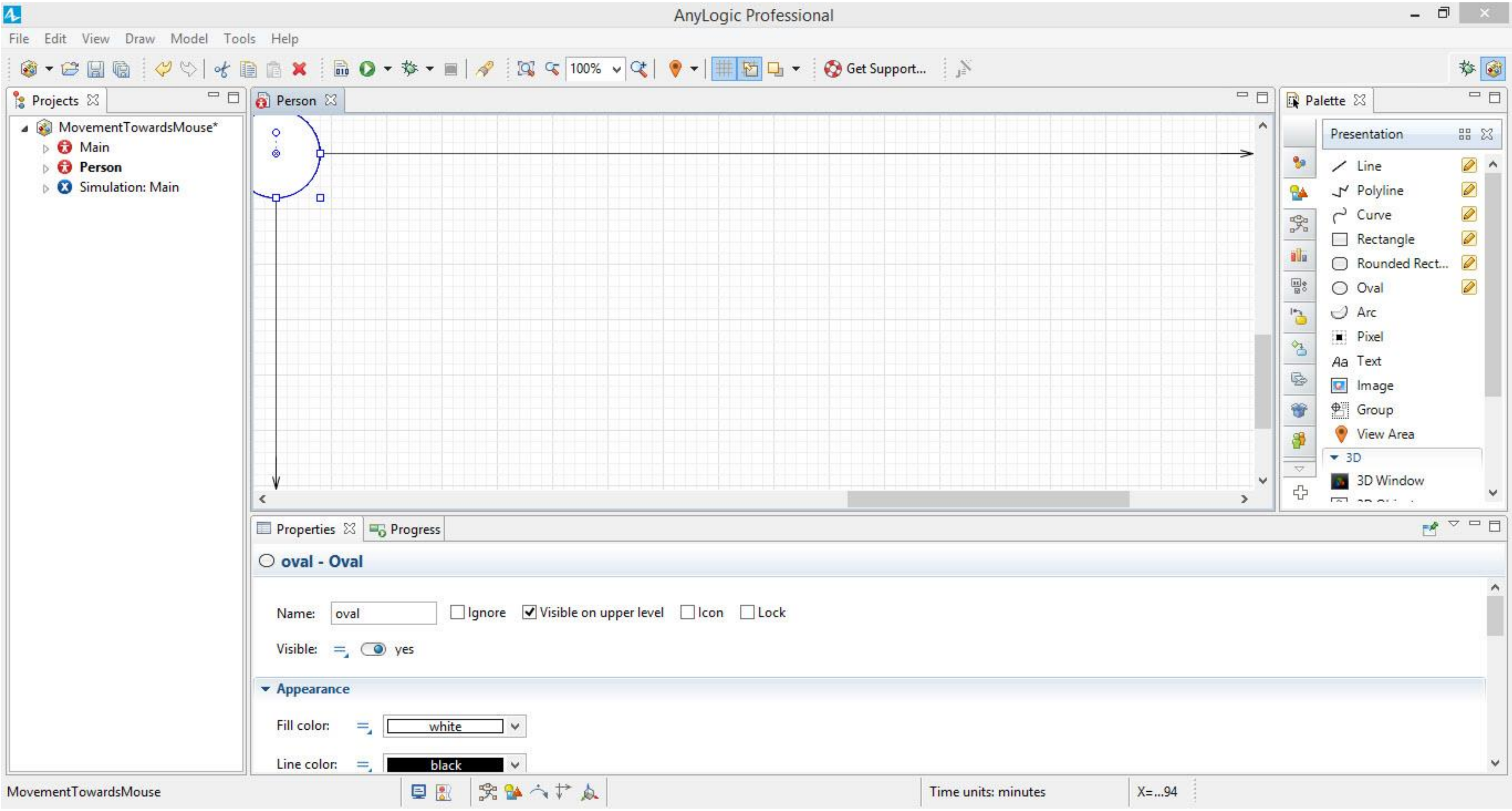
Class Properties



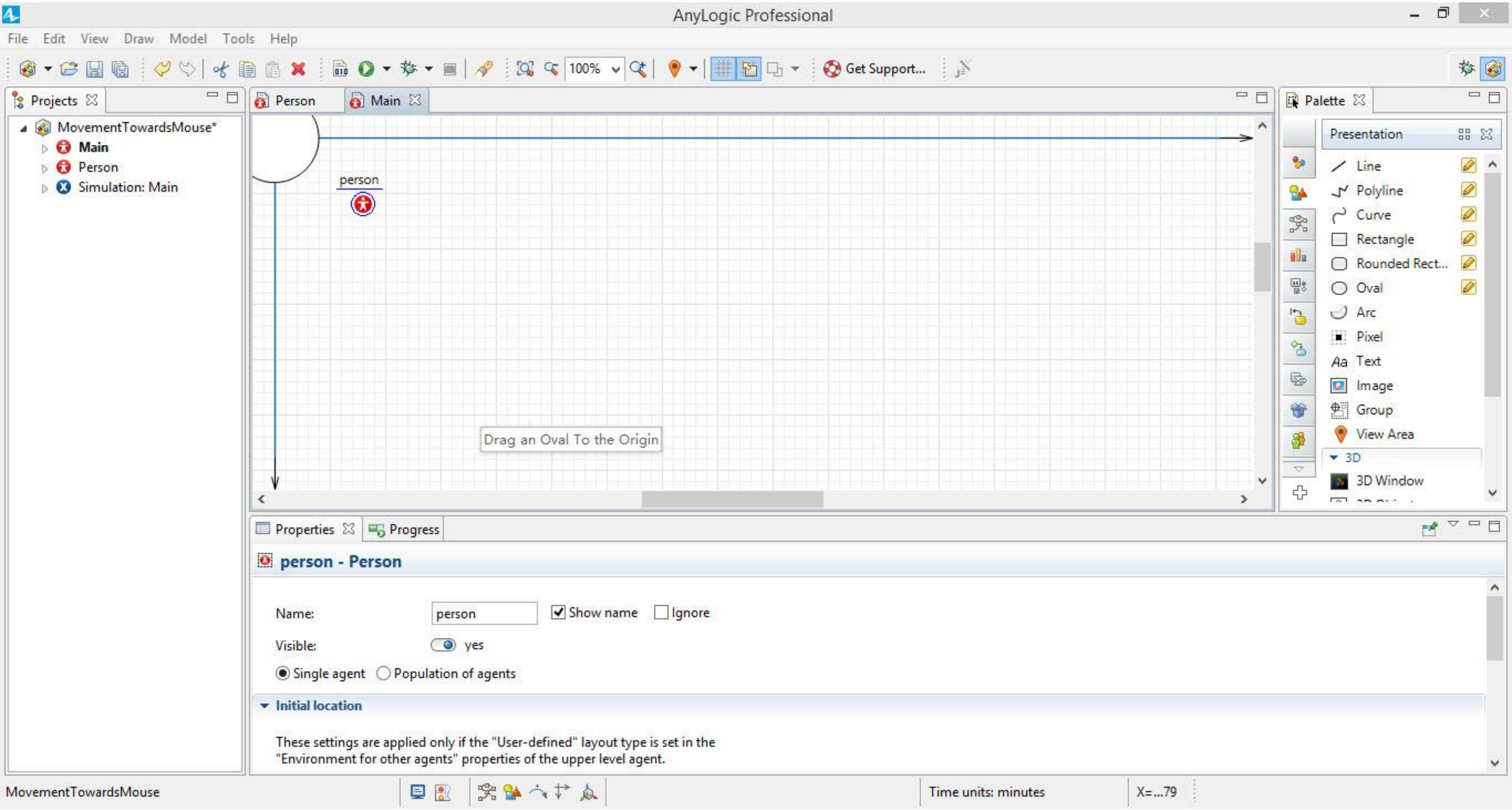
Once "Person" Class is Added



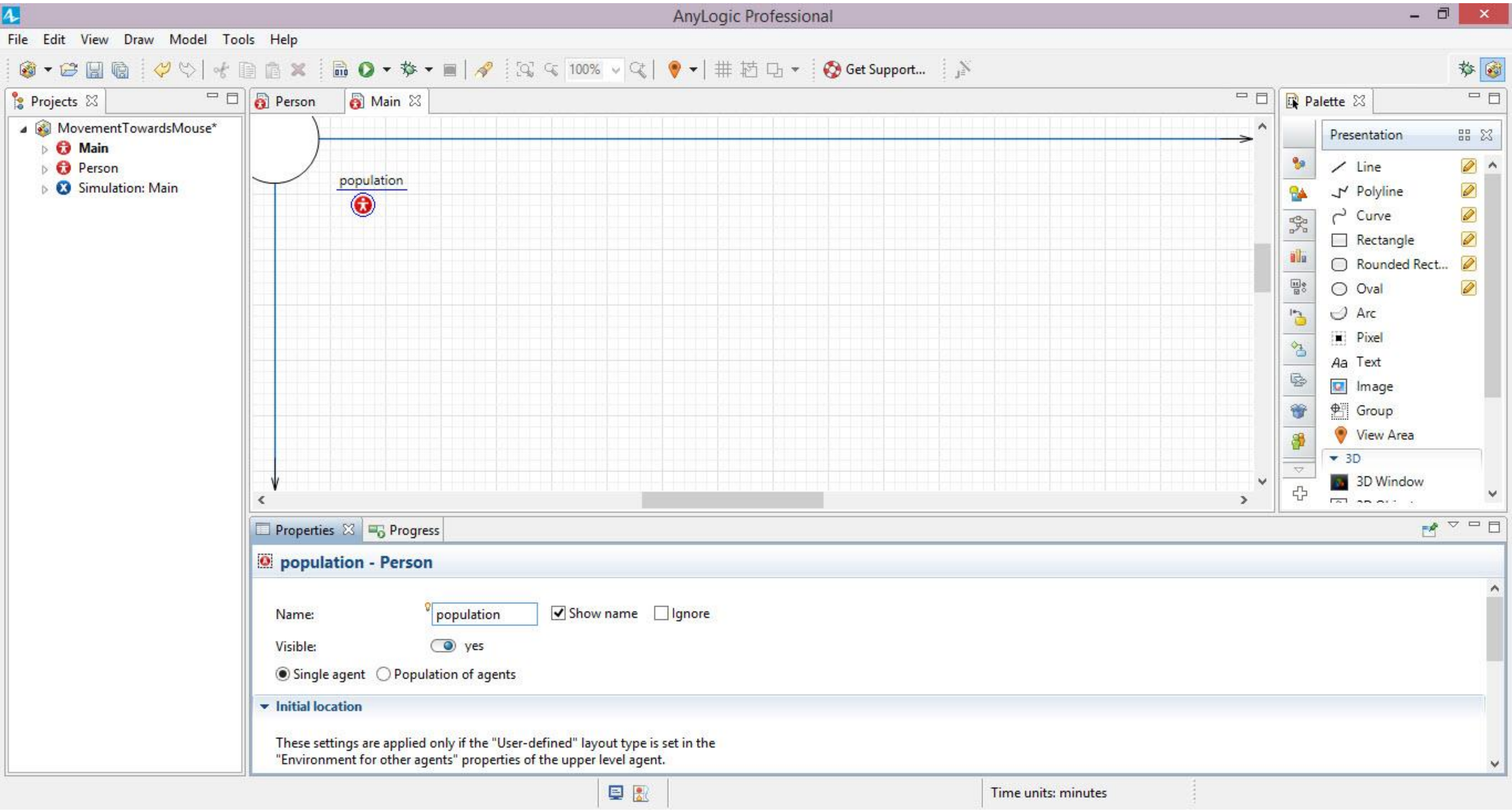
Drag an Oval To the Origin



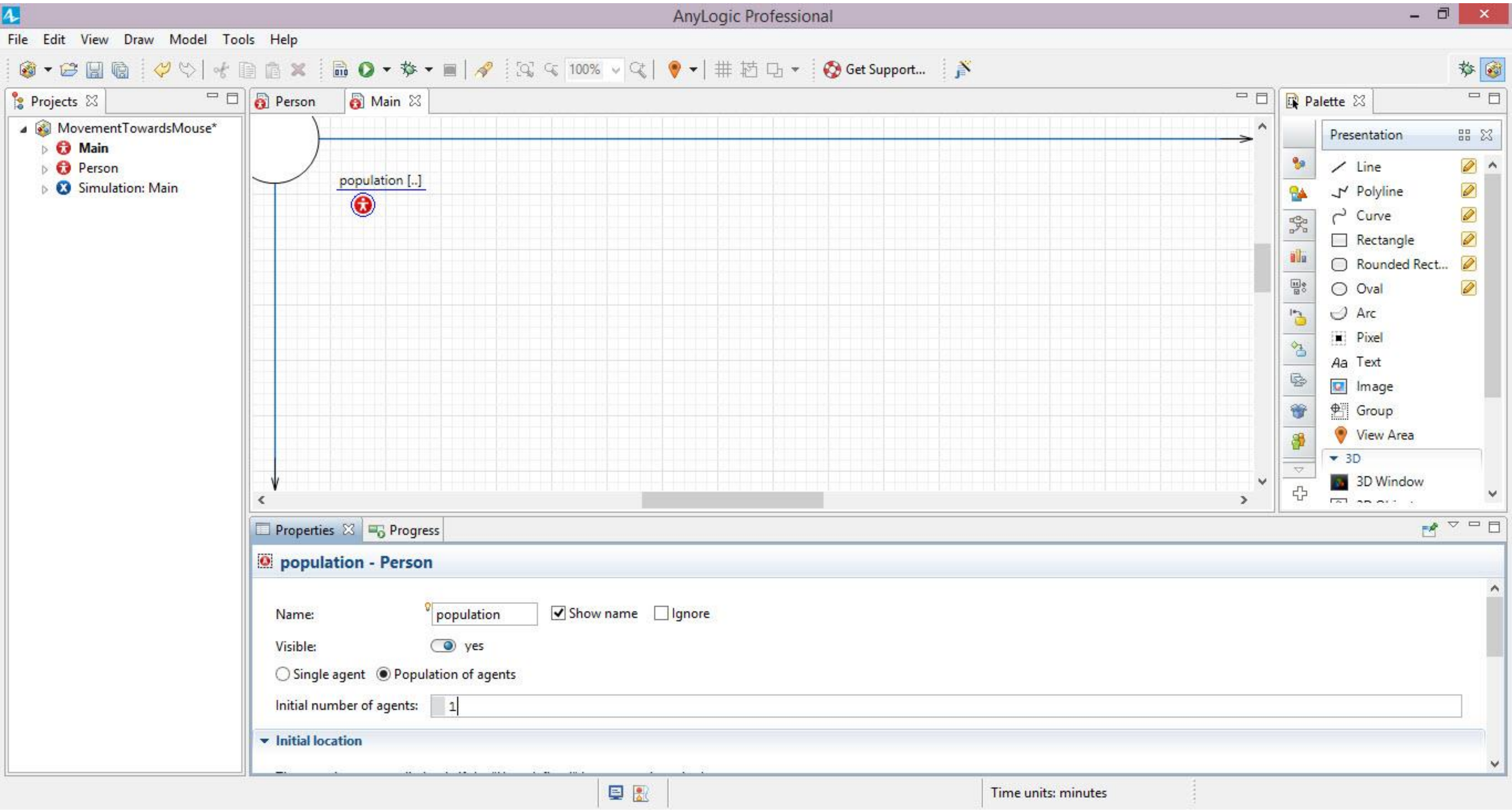
Drag "Person" Class into "Main" to Create a Population



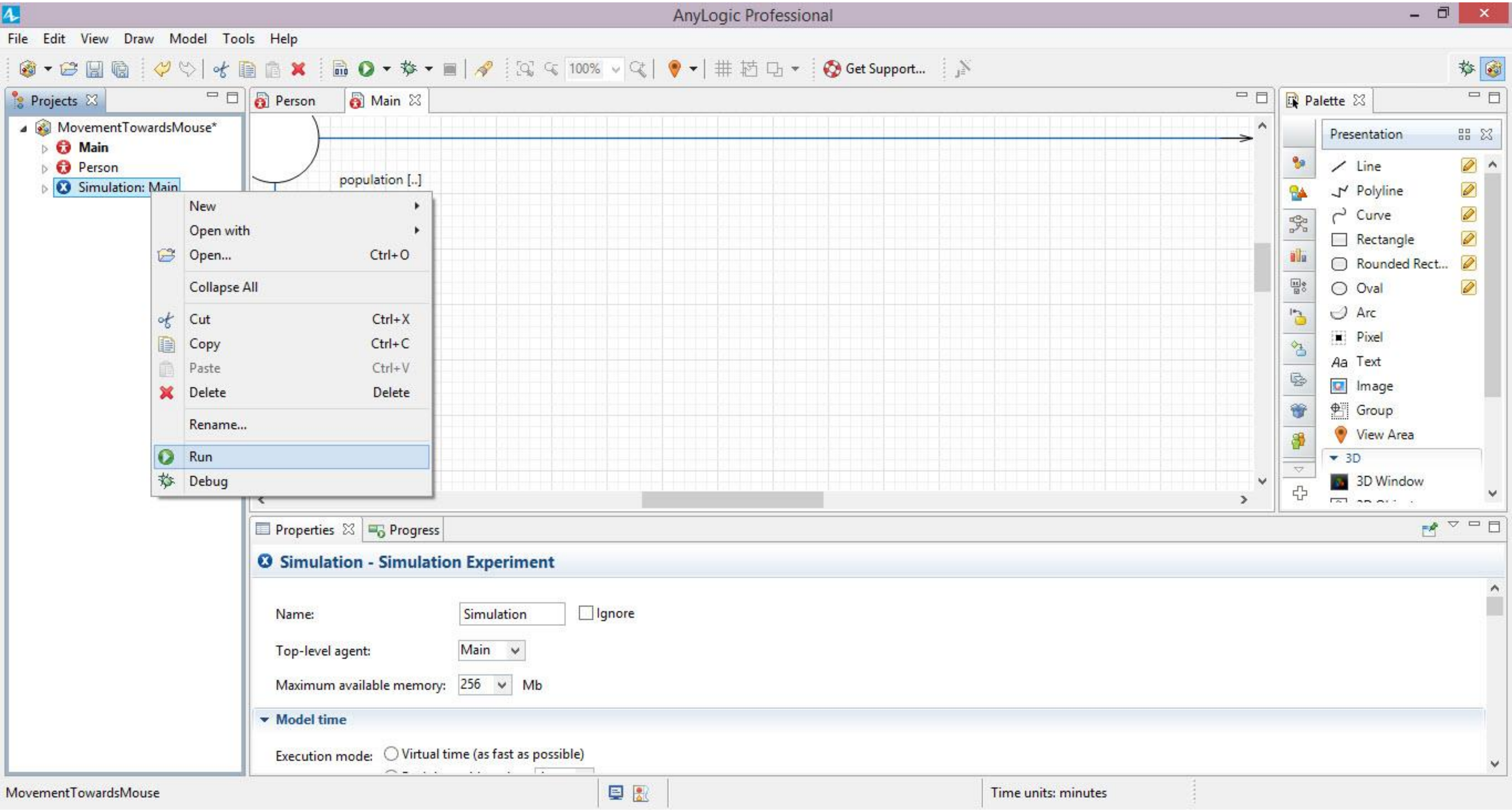
Name it “population”



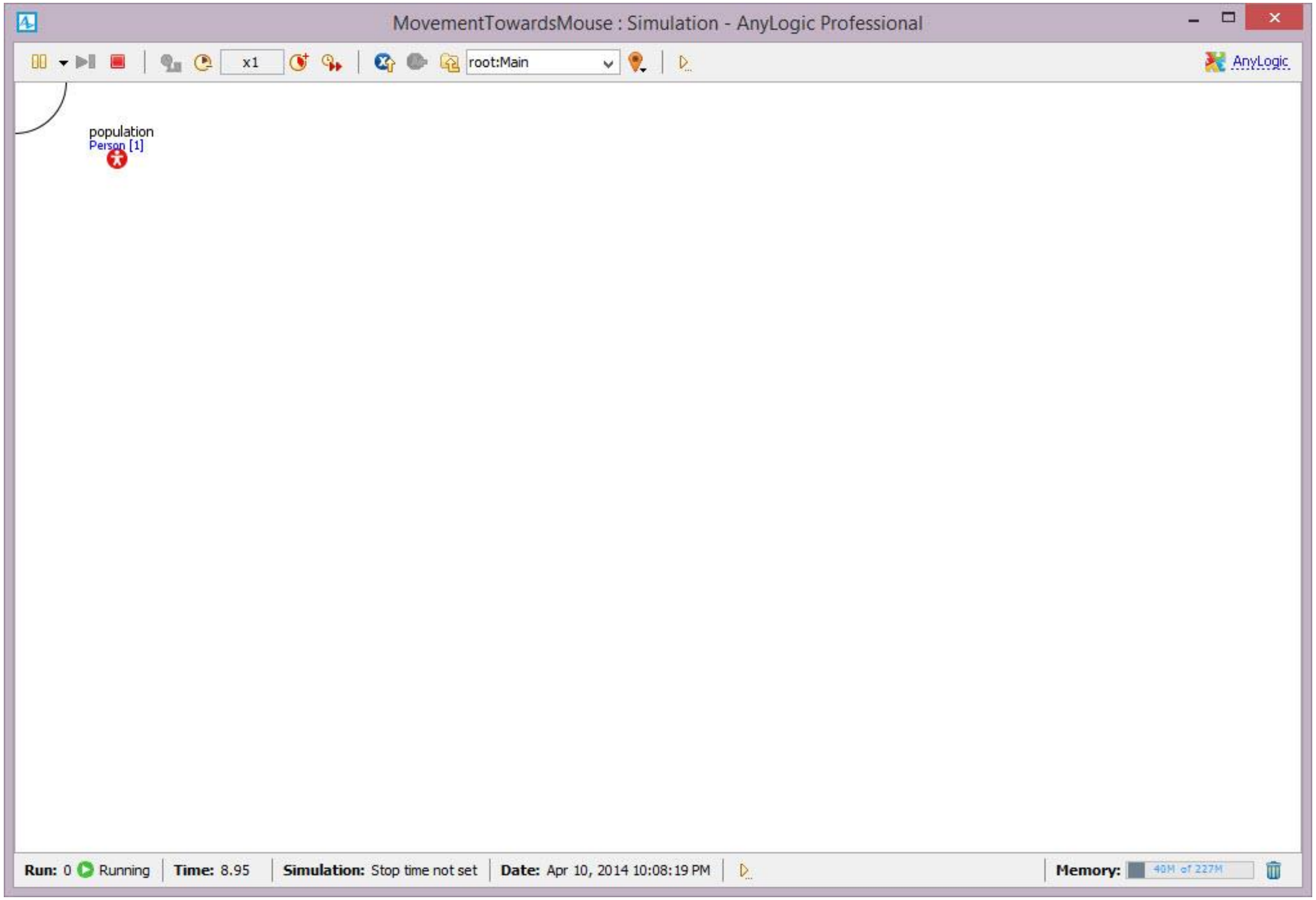
Set Population as Replicated (size 1 – for now)



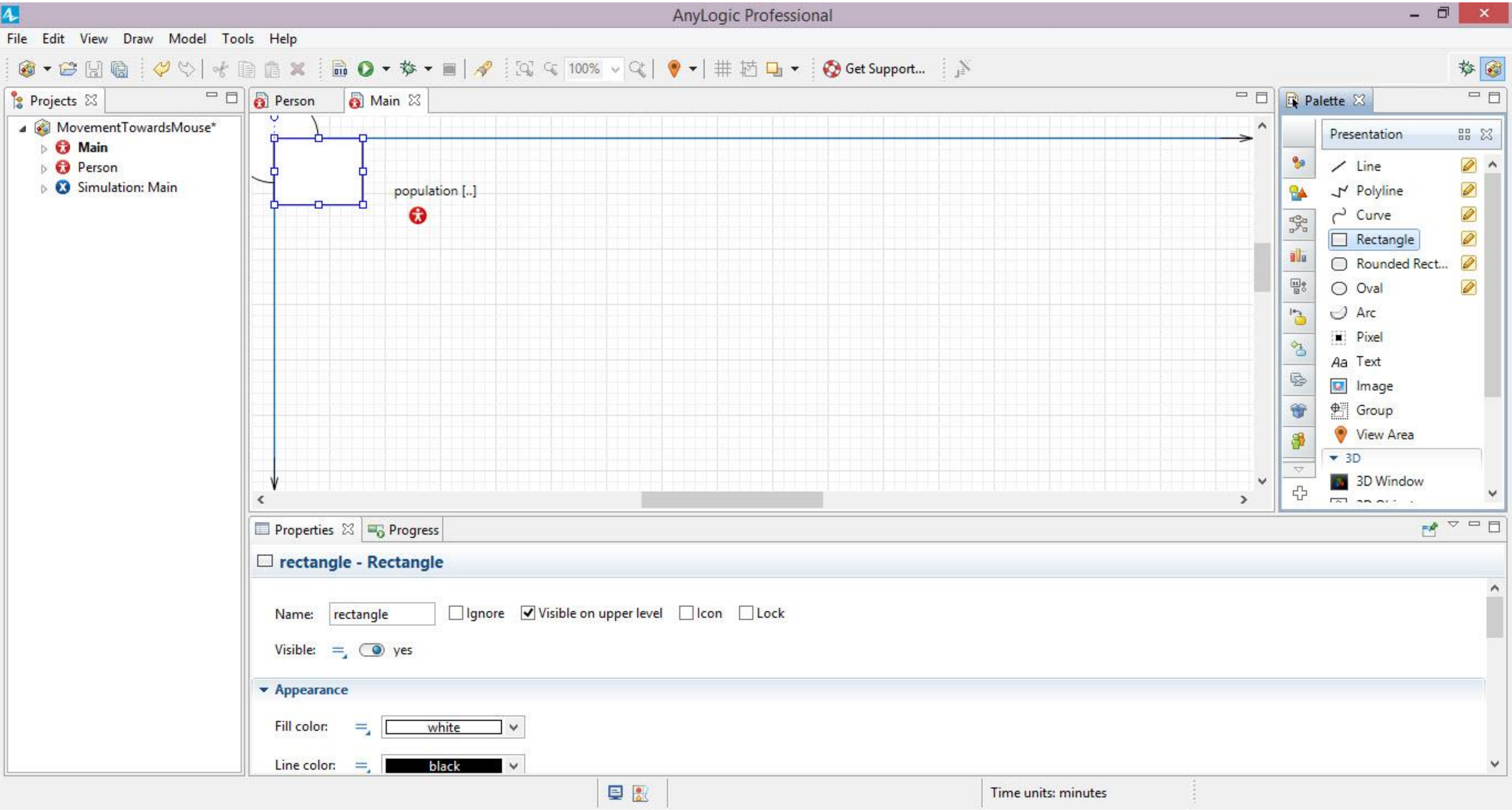
Run Experiment



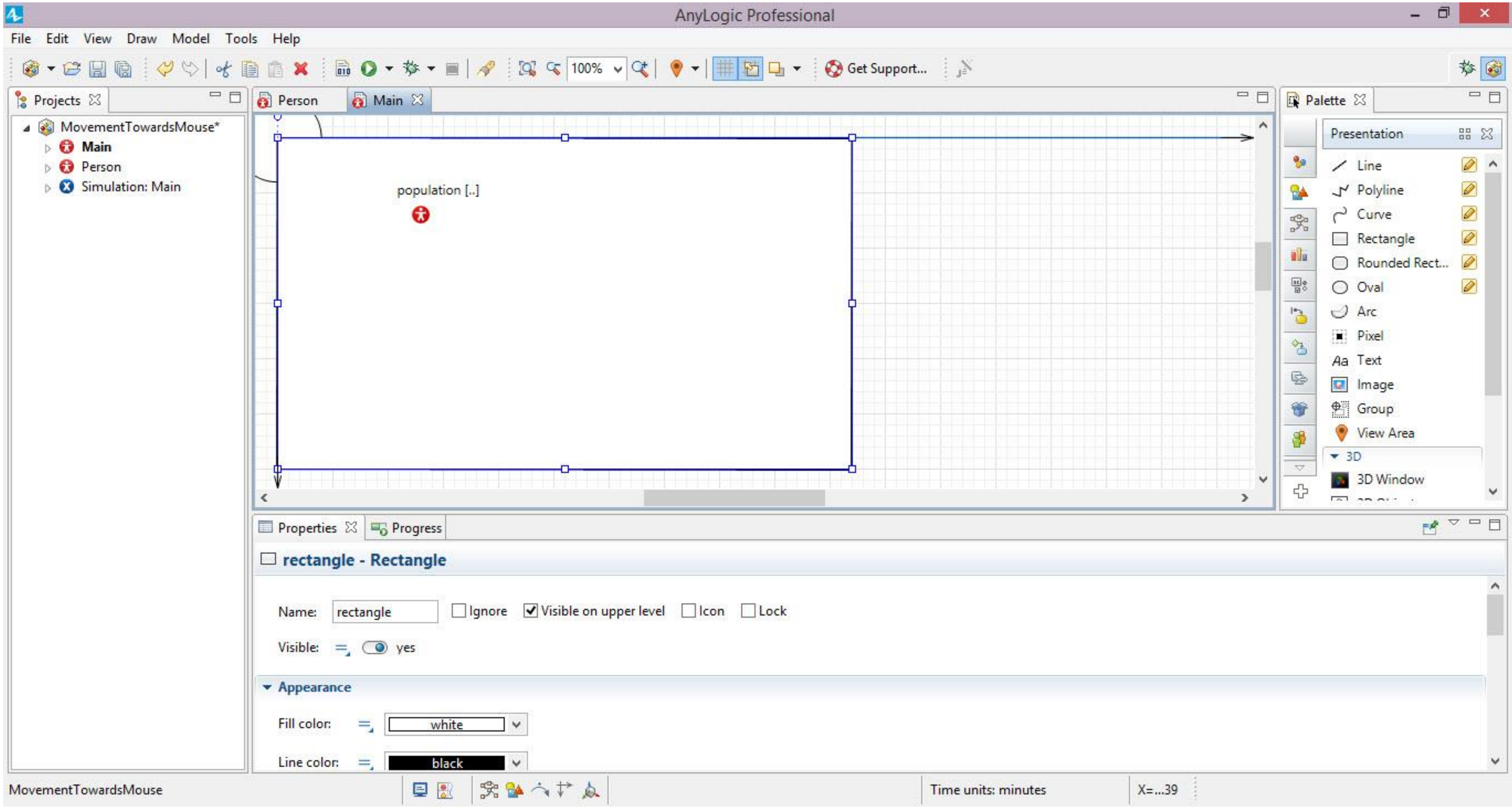
Model Appearance



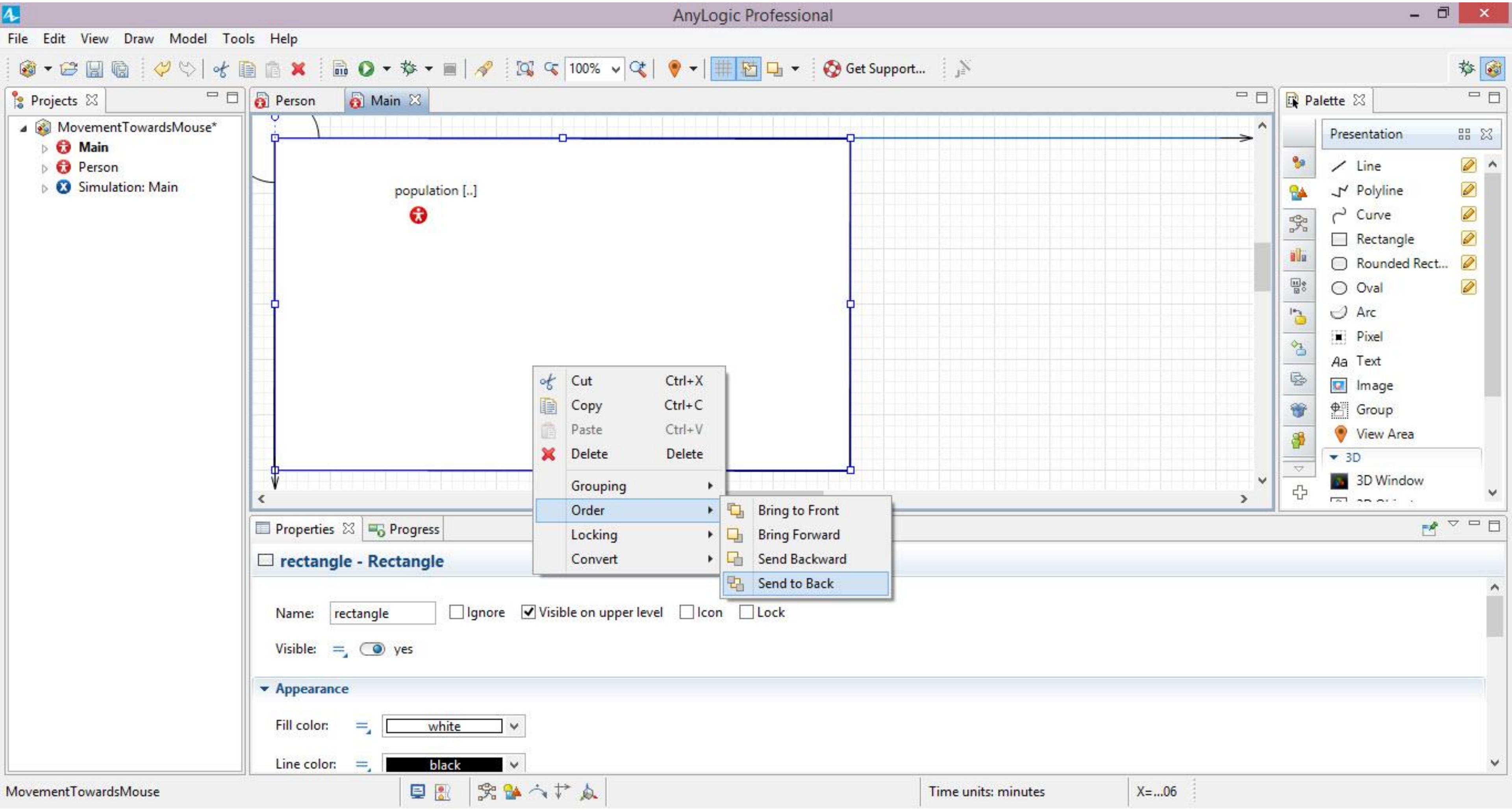
In “Main” Add a Rectangle (Origin at Same Place as Pop Origin)



Enlarge Rectangle

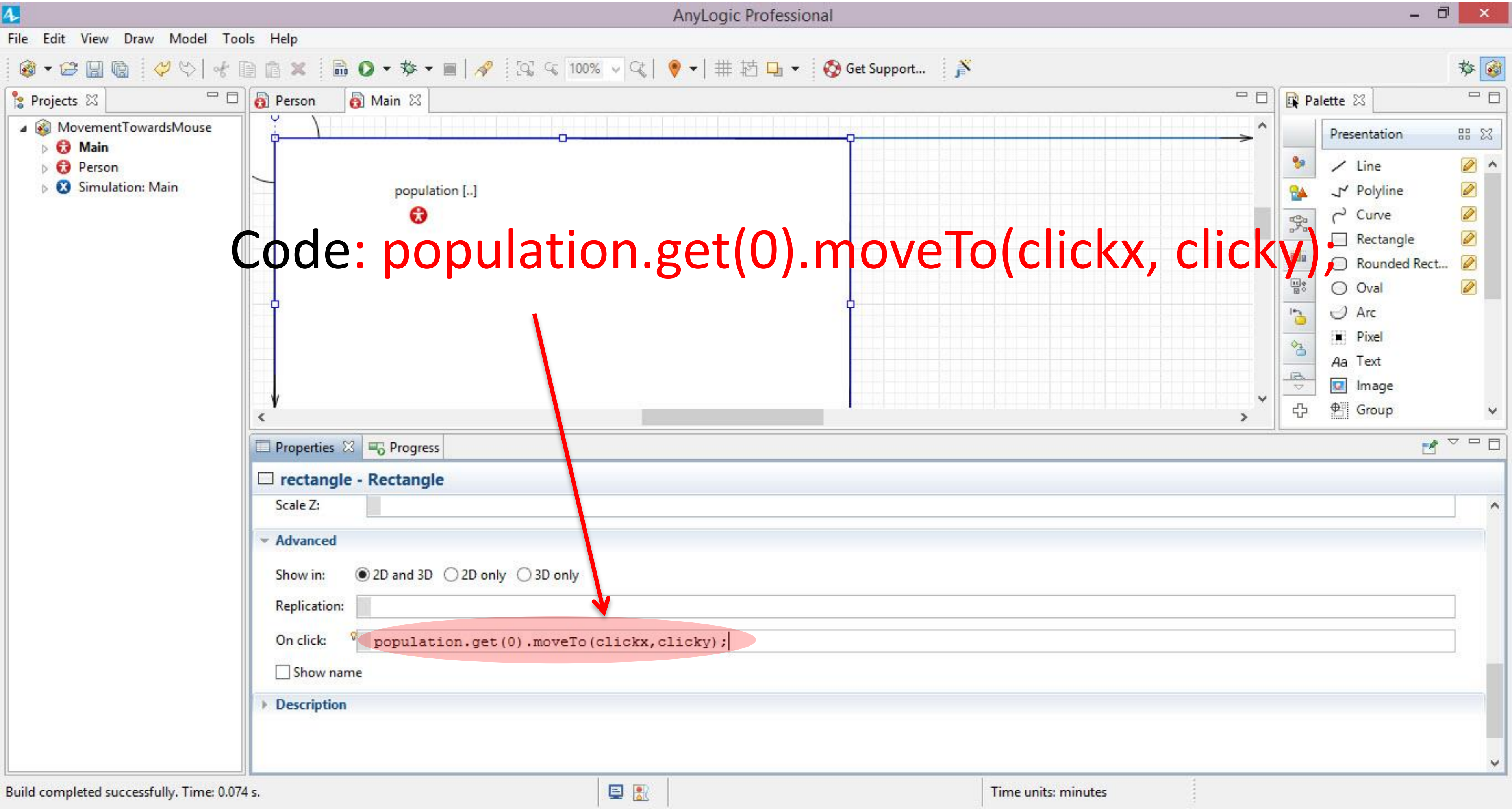


Set Rectangle's Ordering to Back

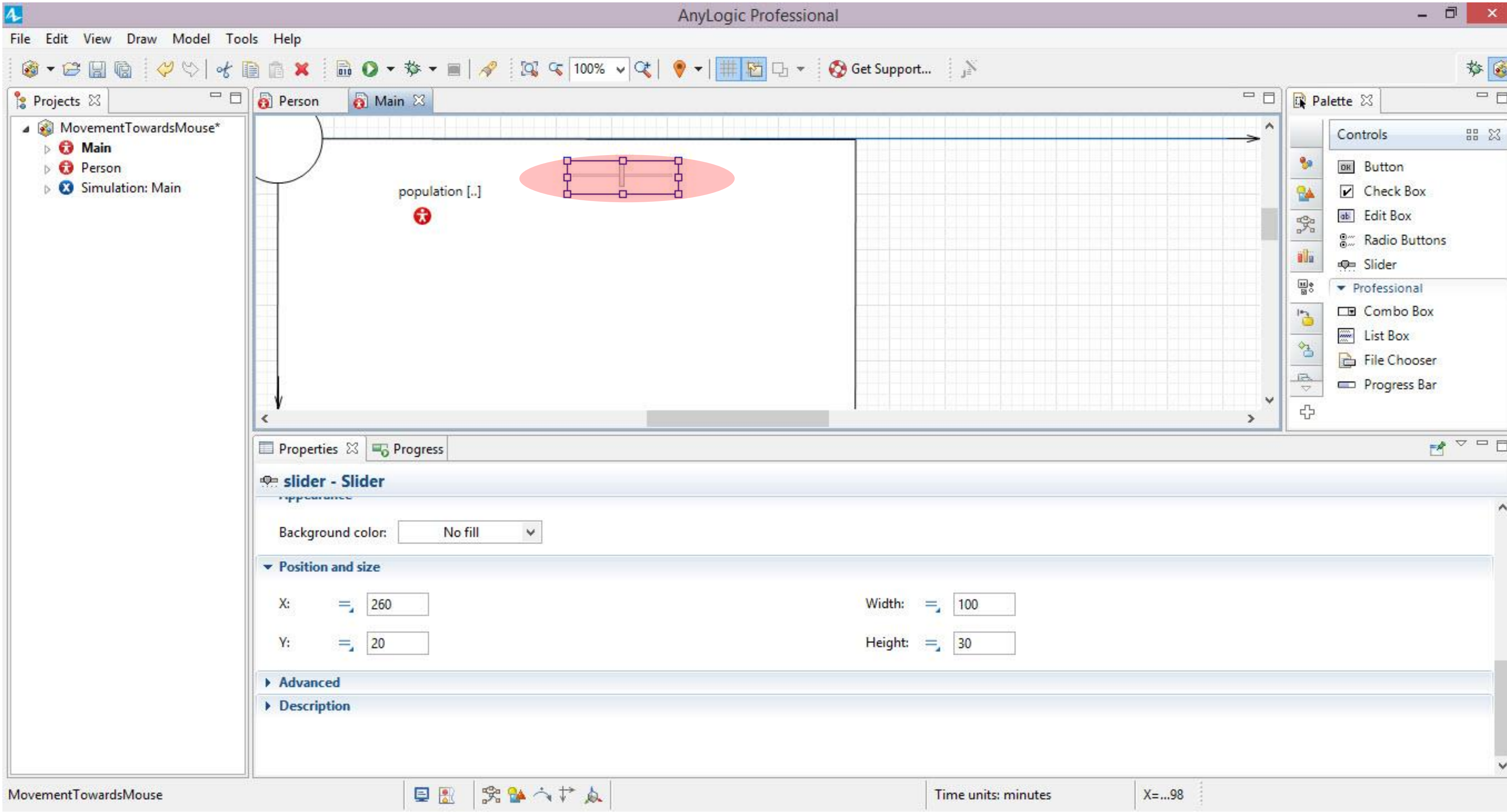


Insert Code

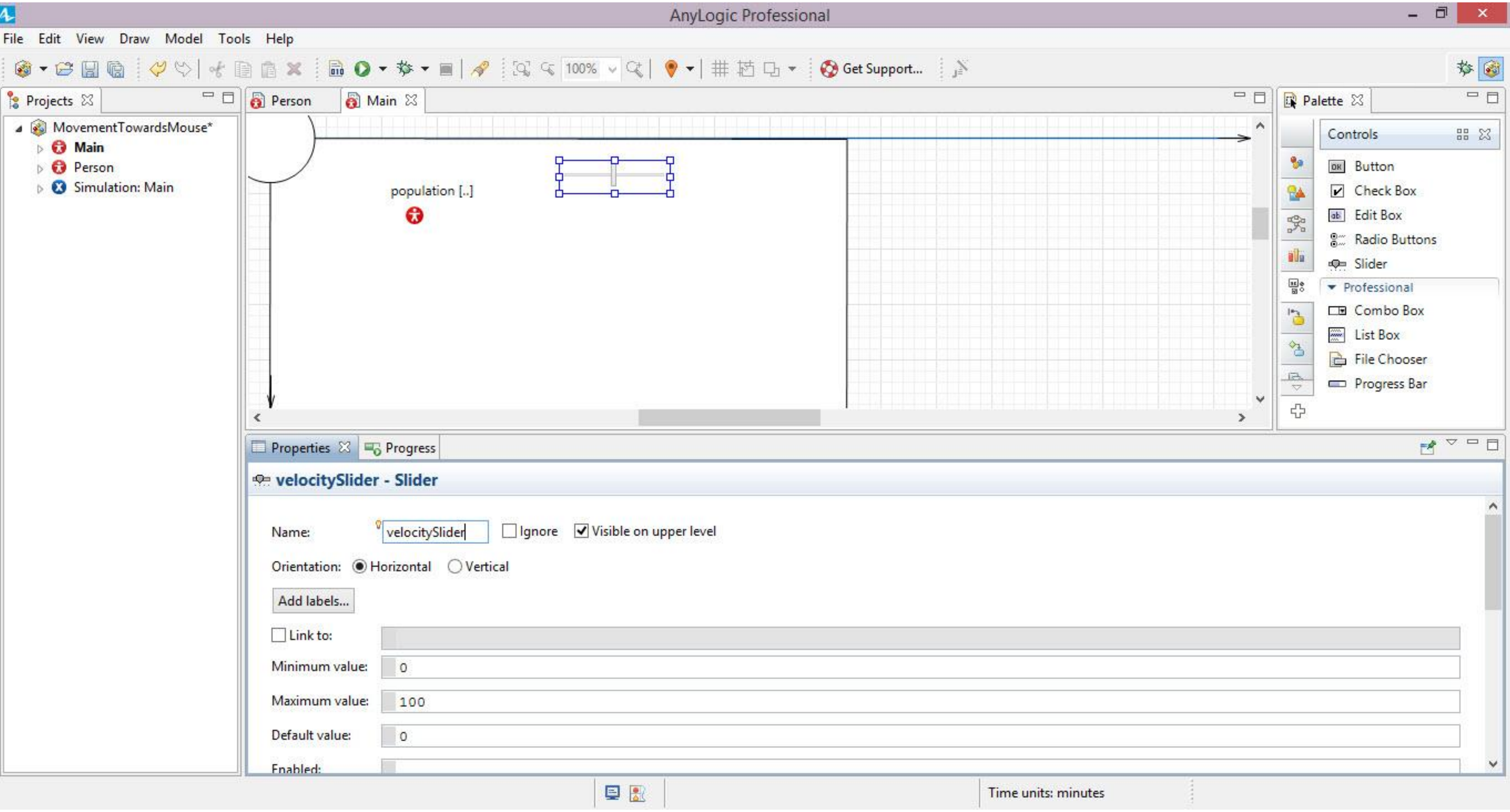
Code: population.get(0).moveTo(clickx, clicky);



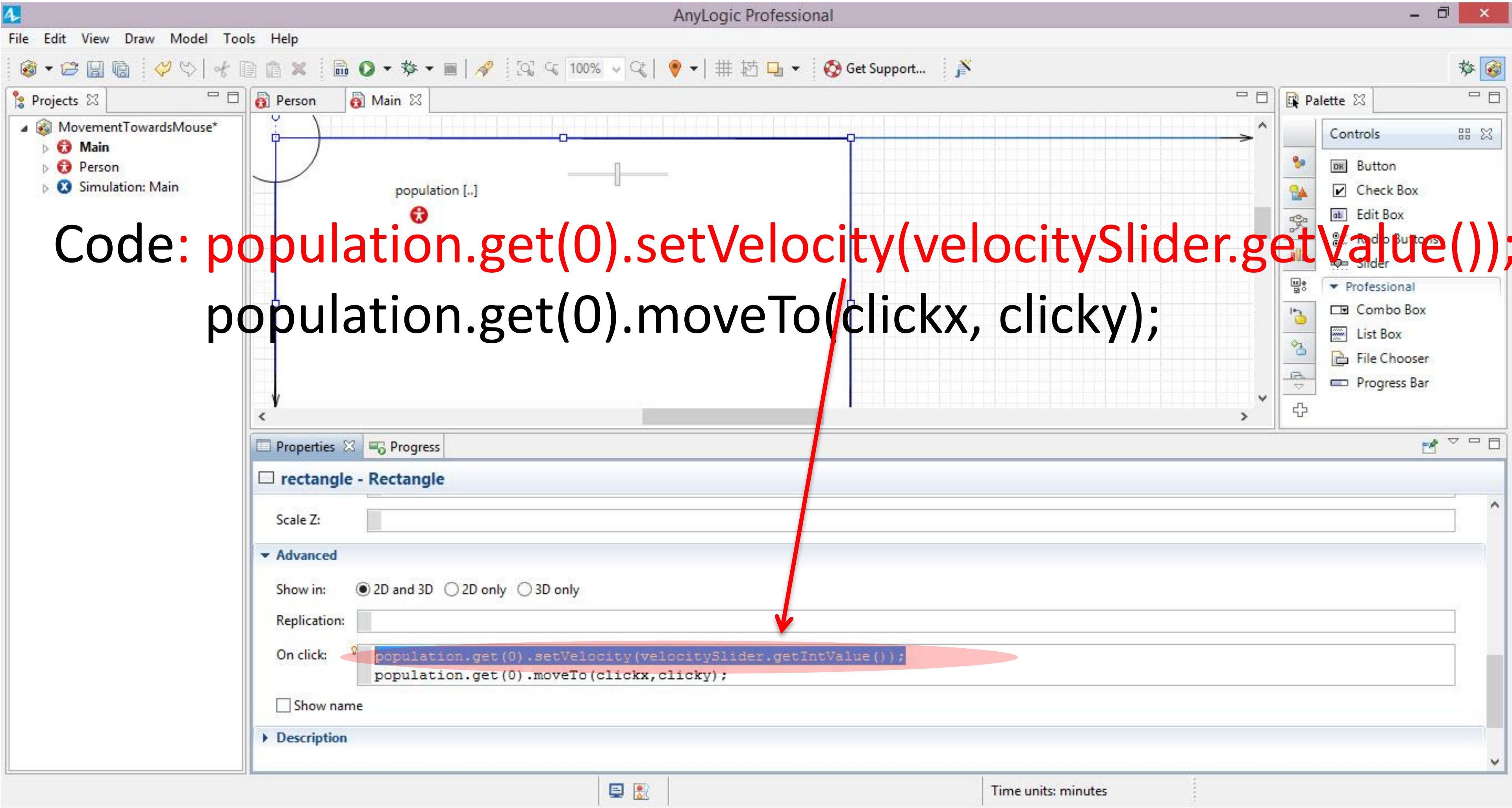
Drag a Slider from the “Controls” Area



Name it “velocitySlider”
(in “Properties”, under “General” tab)

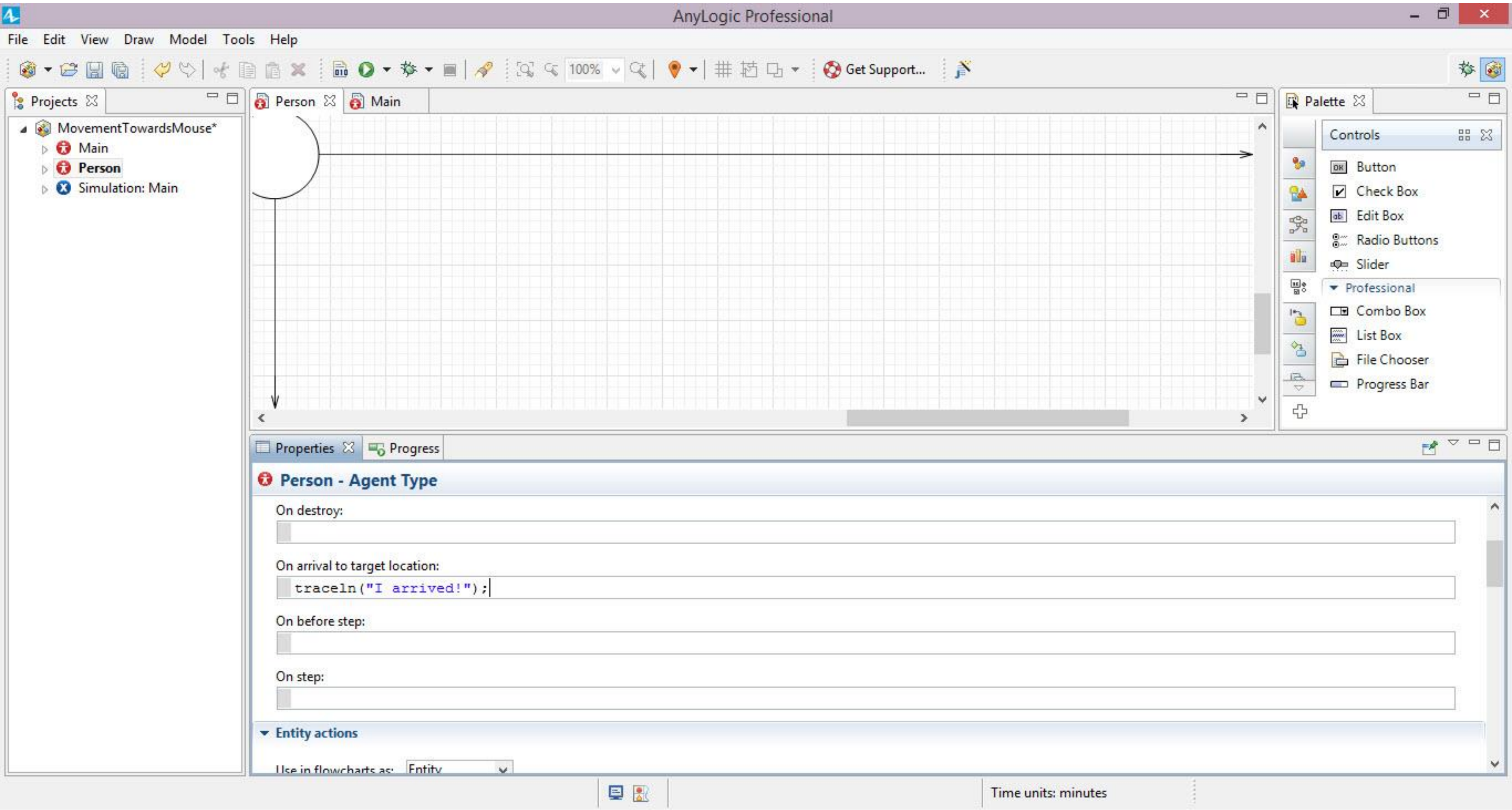


Add a New Line of Code to “On Click” Handler of Rectangle

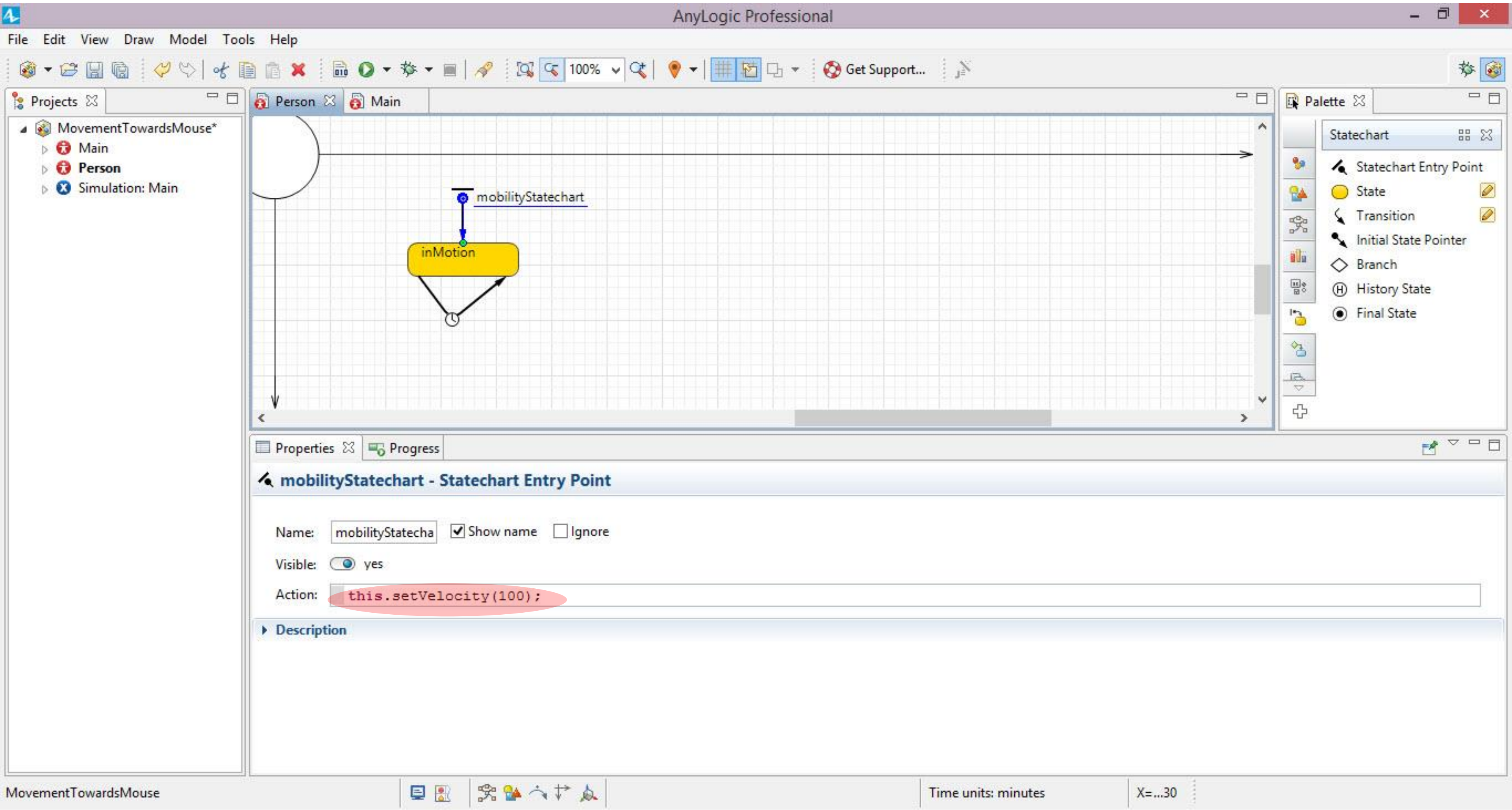


Code: population.get(0).setVelocity(velocitySlider.getValue());
population.get(0).moveTo(clickx, clicky);

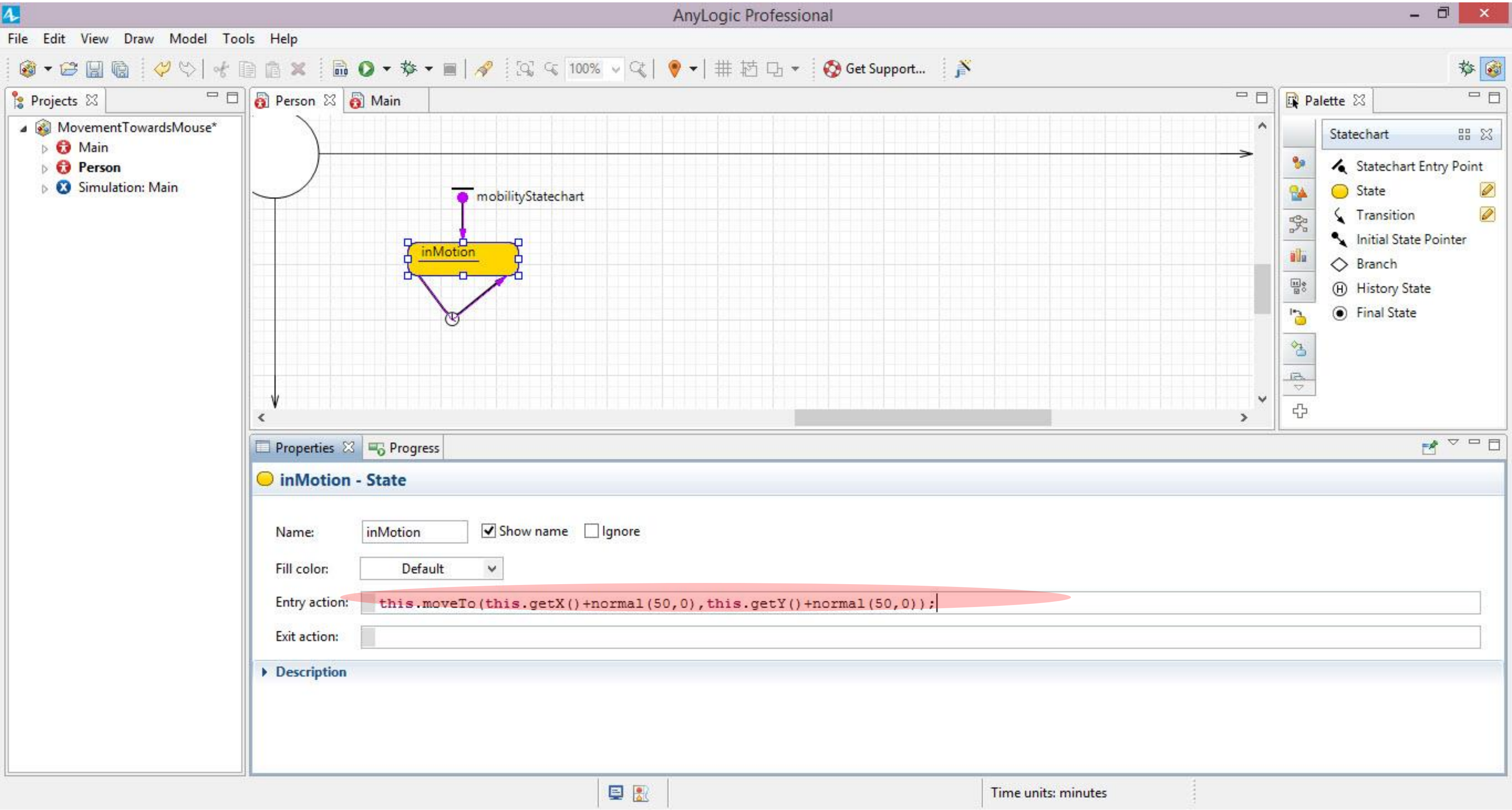
On "Person" "Agent" Properties Add code to indicate Arrival



Add Random Walks



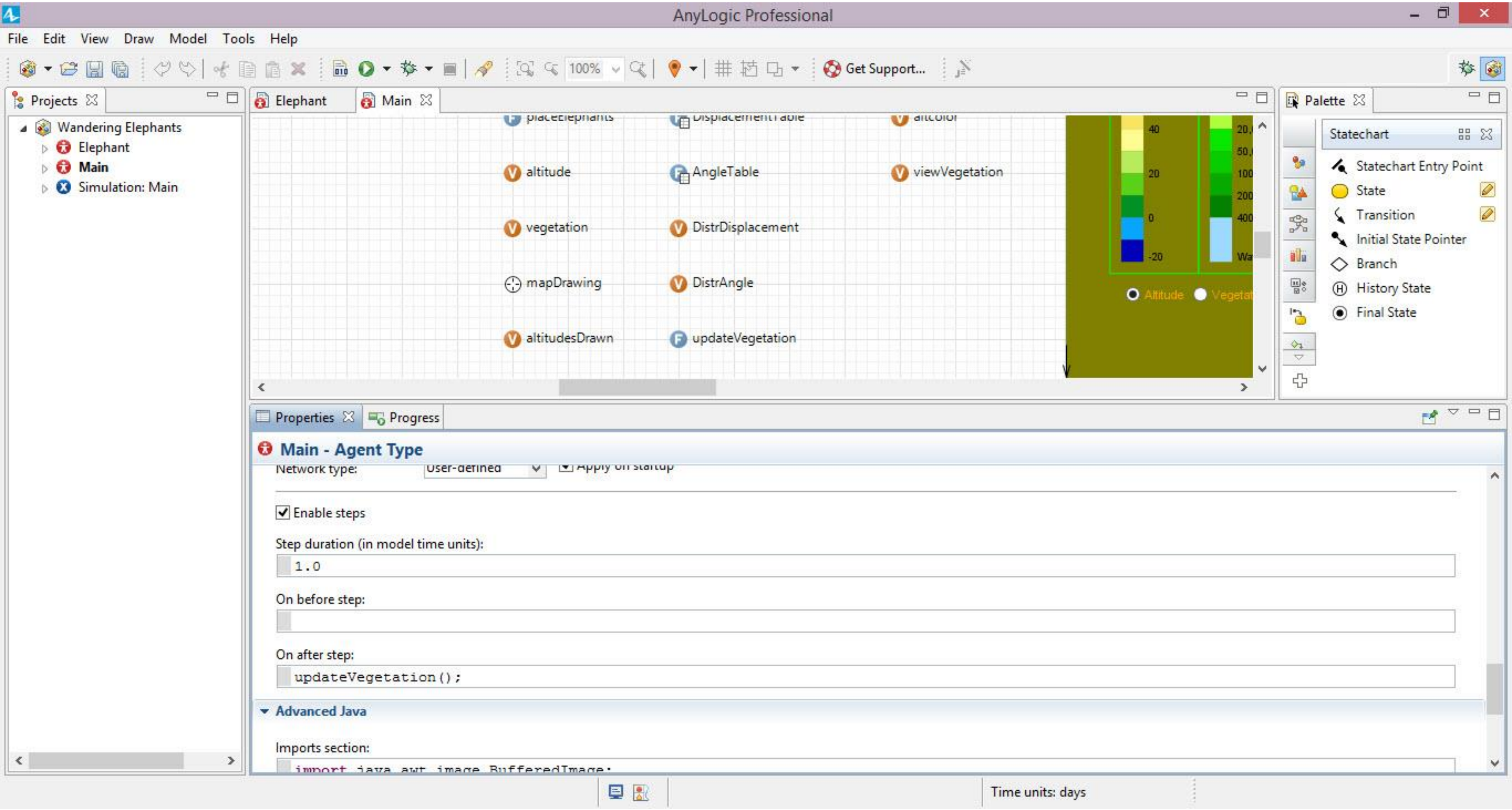
Random Walk Models



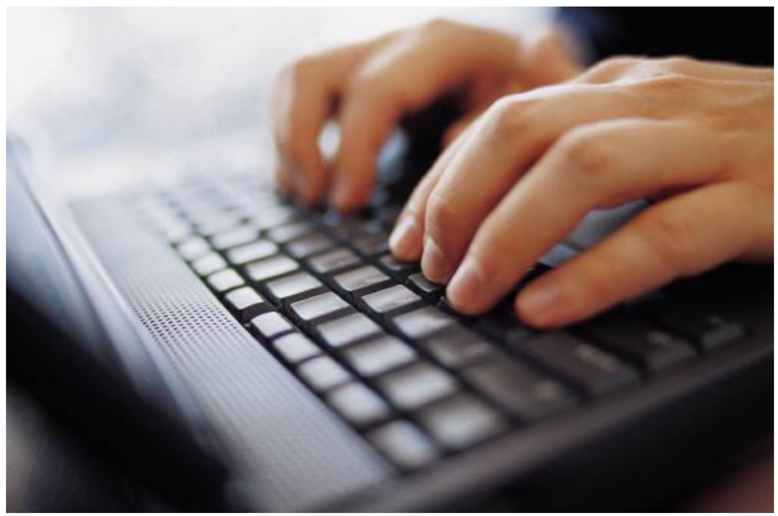
Continuous Space: Relevant Methods (To call on Instances of *Agent*)

- Already covered
  - moveTo(x,y) : initiates agent movement to location
  - setVelocity(v)
- Basic info
  - getX()/getY()
  - setXY(x,y): initial location
  - jumpTo(x,y): moves agent to location
  - isMoving()
  - getTargetX()/getTargetY()
    - Where heading to?
  - setRotation()/ getRotation()

Environment Happens to Handle Process of Maintaining Environmental Dynamics

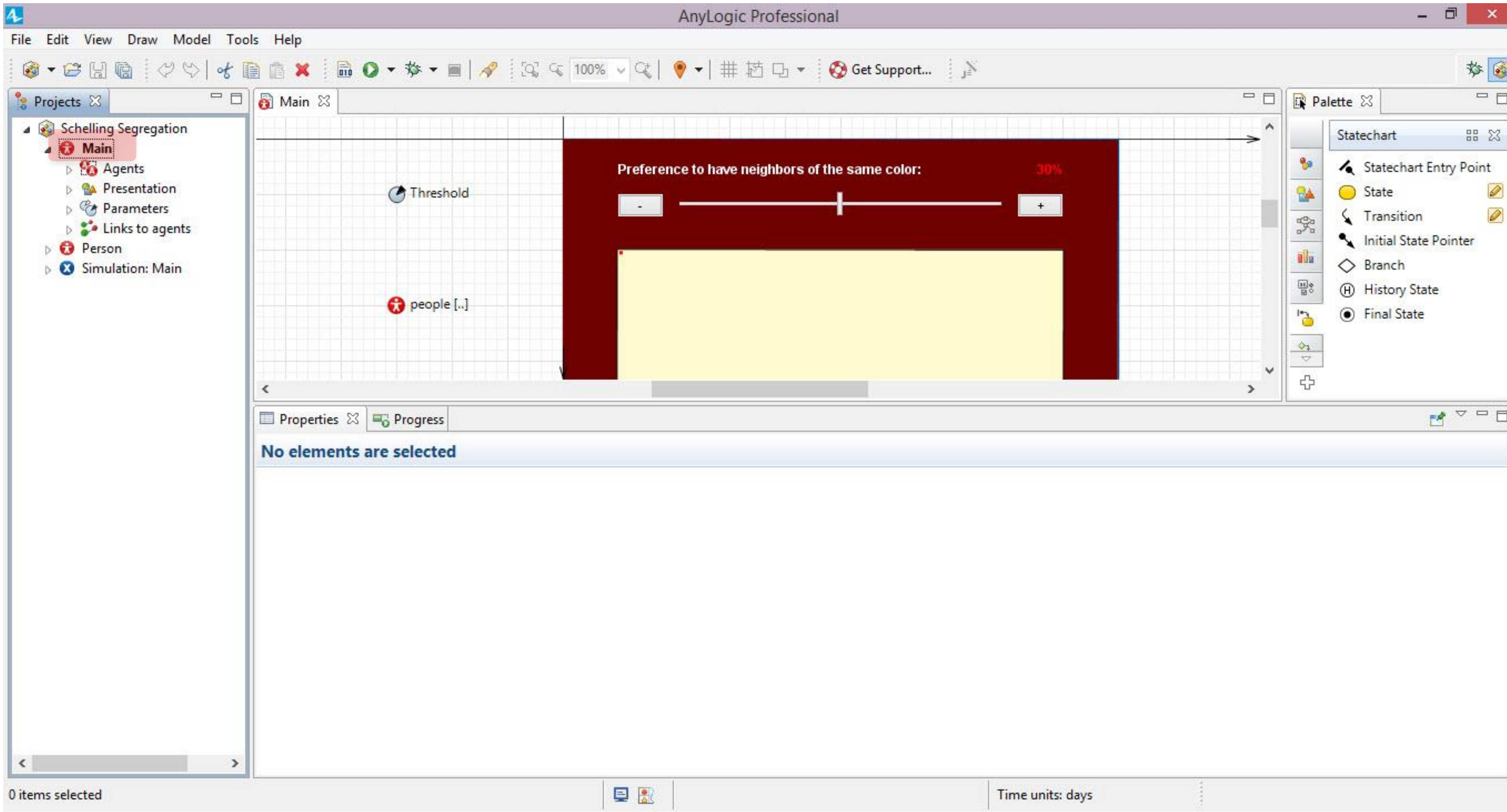


Hands on Model Use Ahead

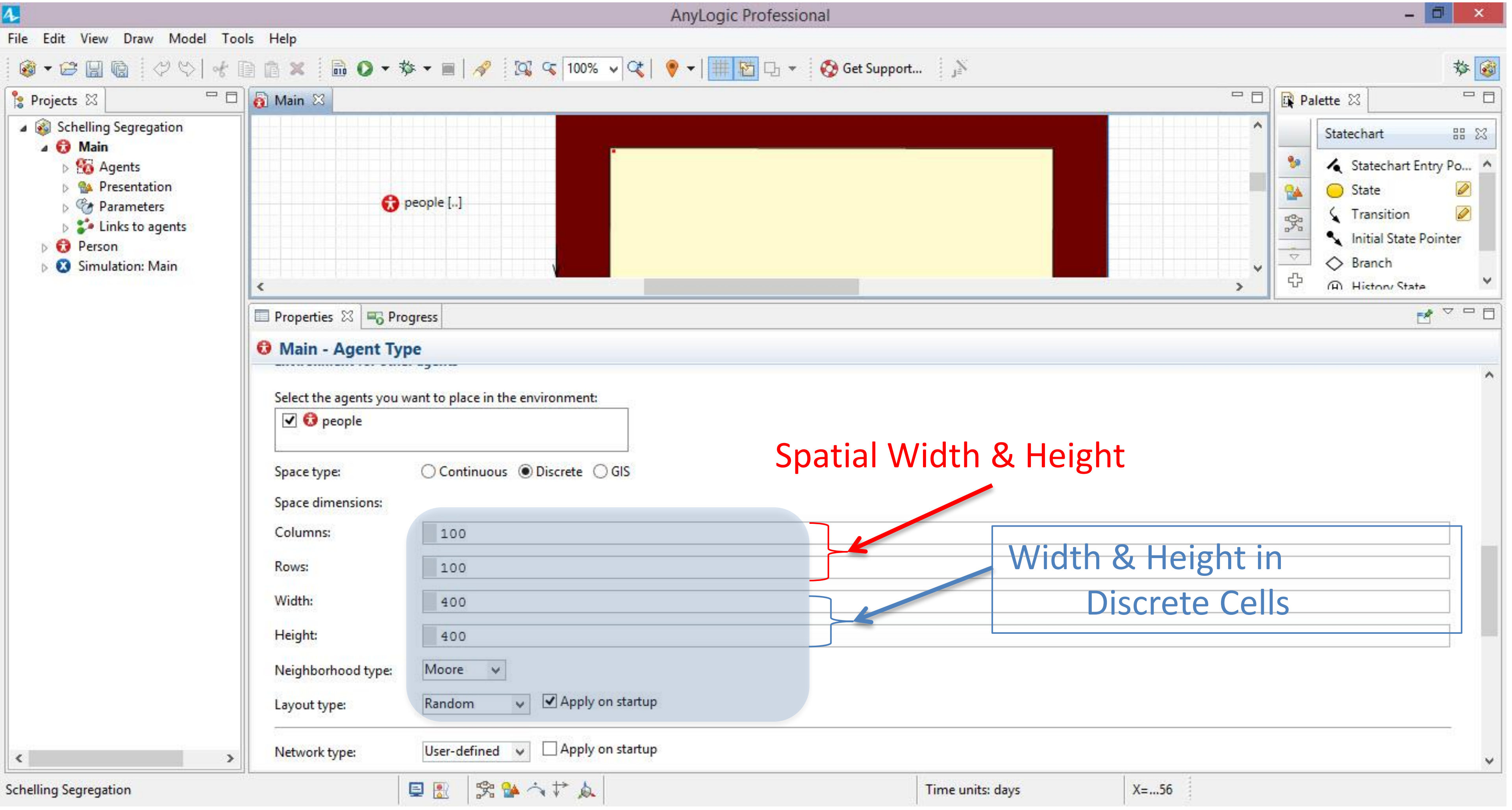


Load model: Schelling Segregation.alp

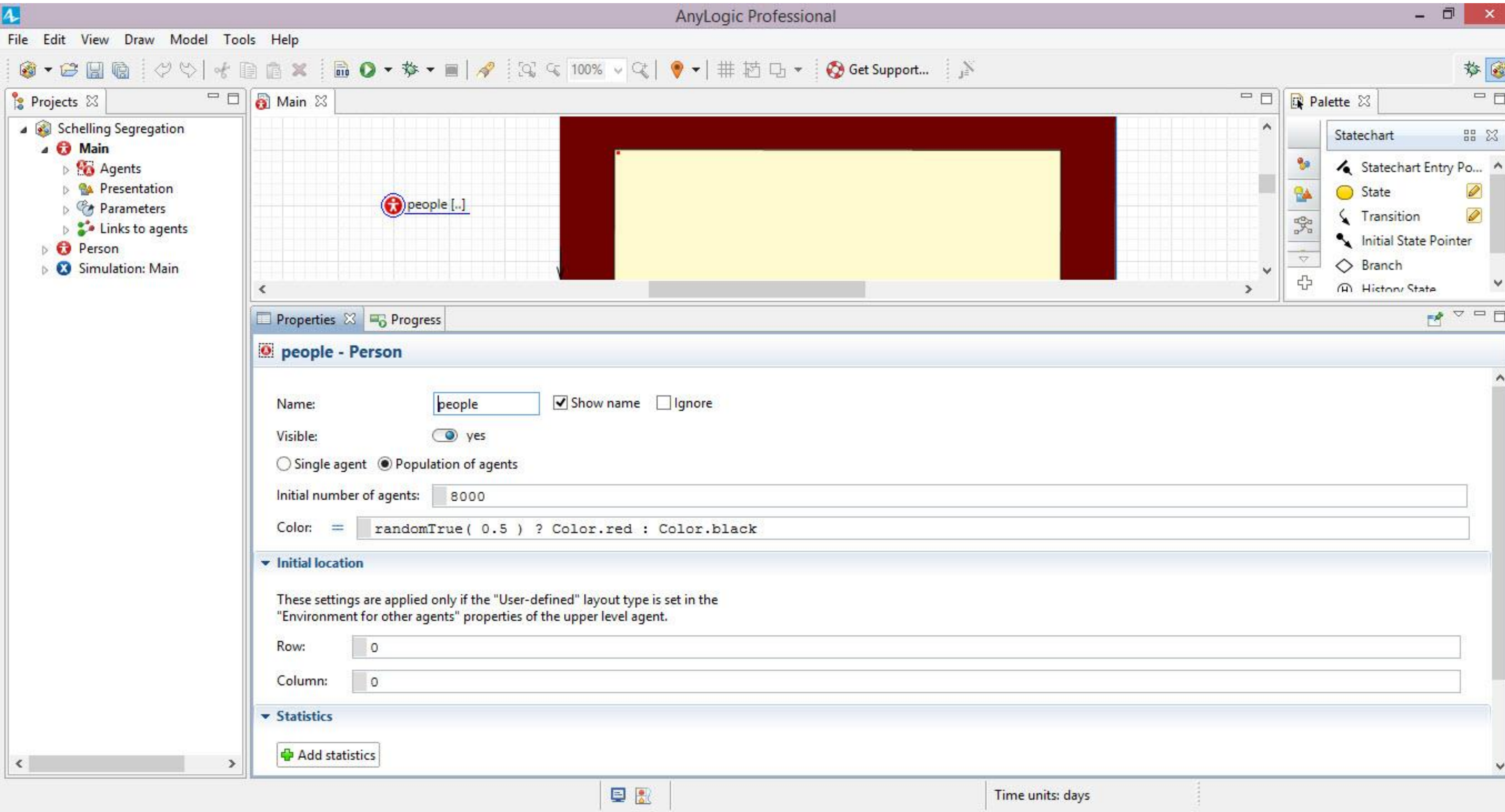
A Model to Examine the Emergence of Segregation



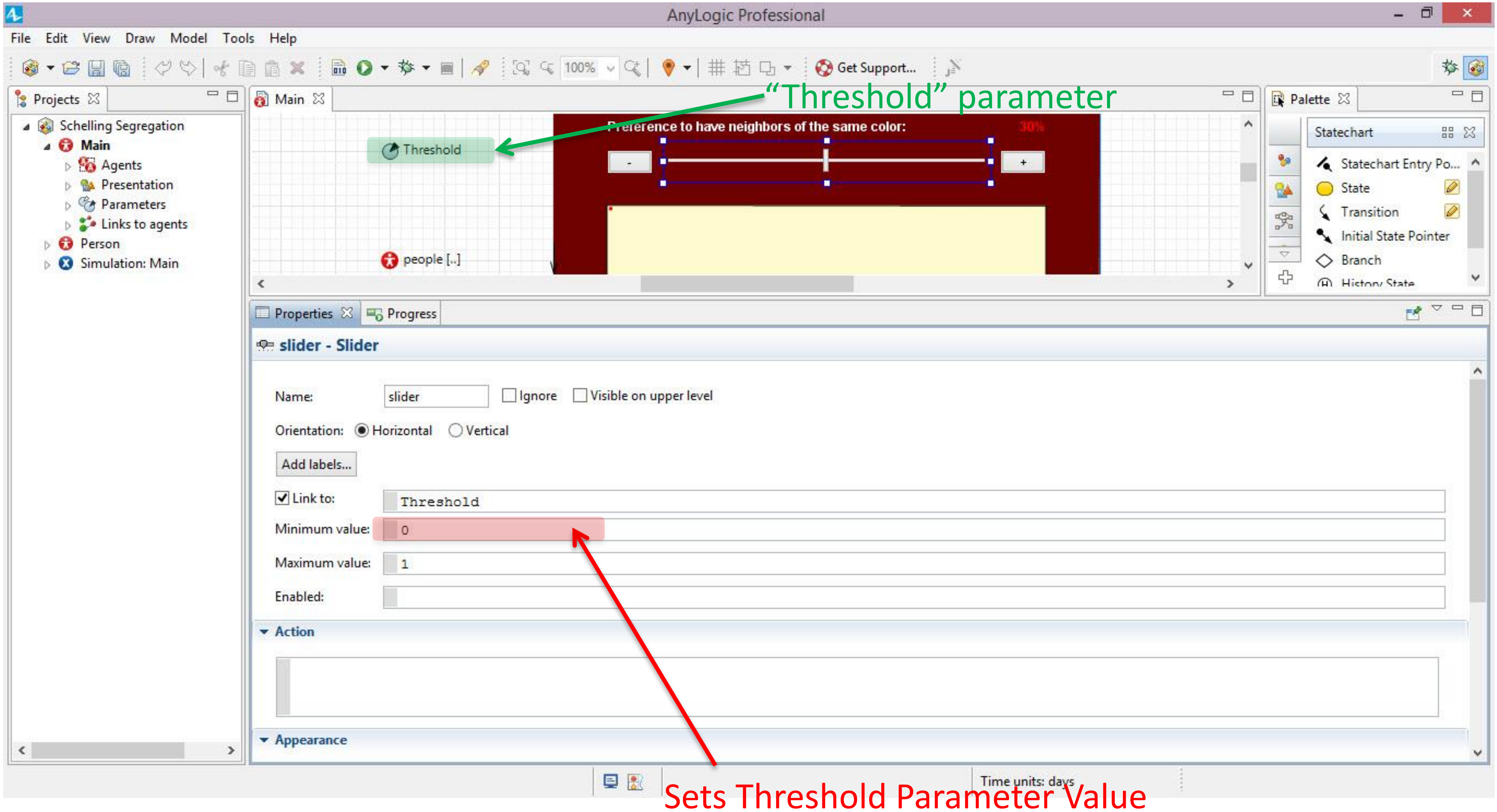
A Discrete Spatial Environment with Random Agent Positioning



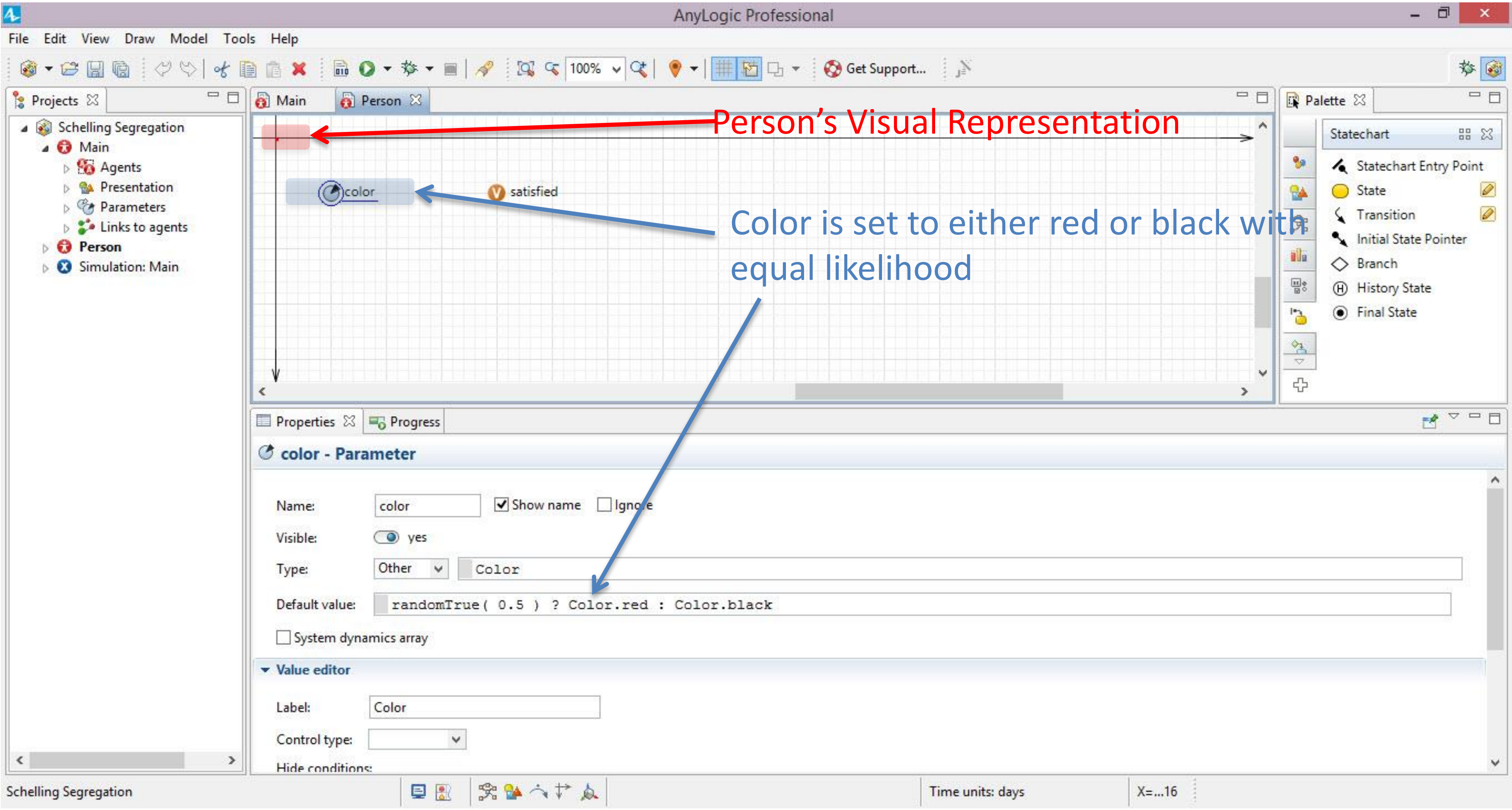
Population Dependence on the Population



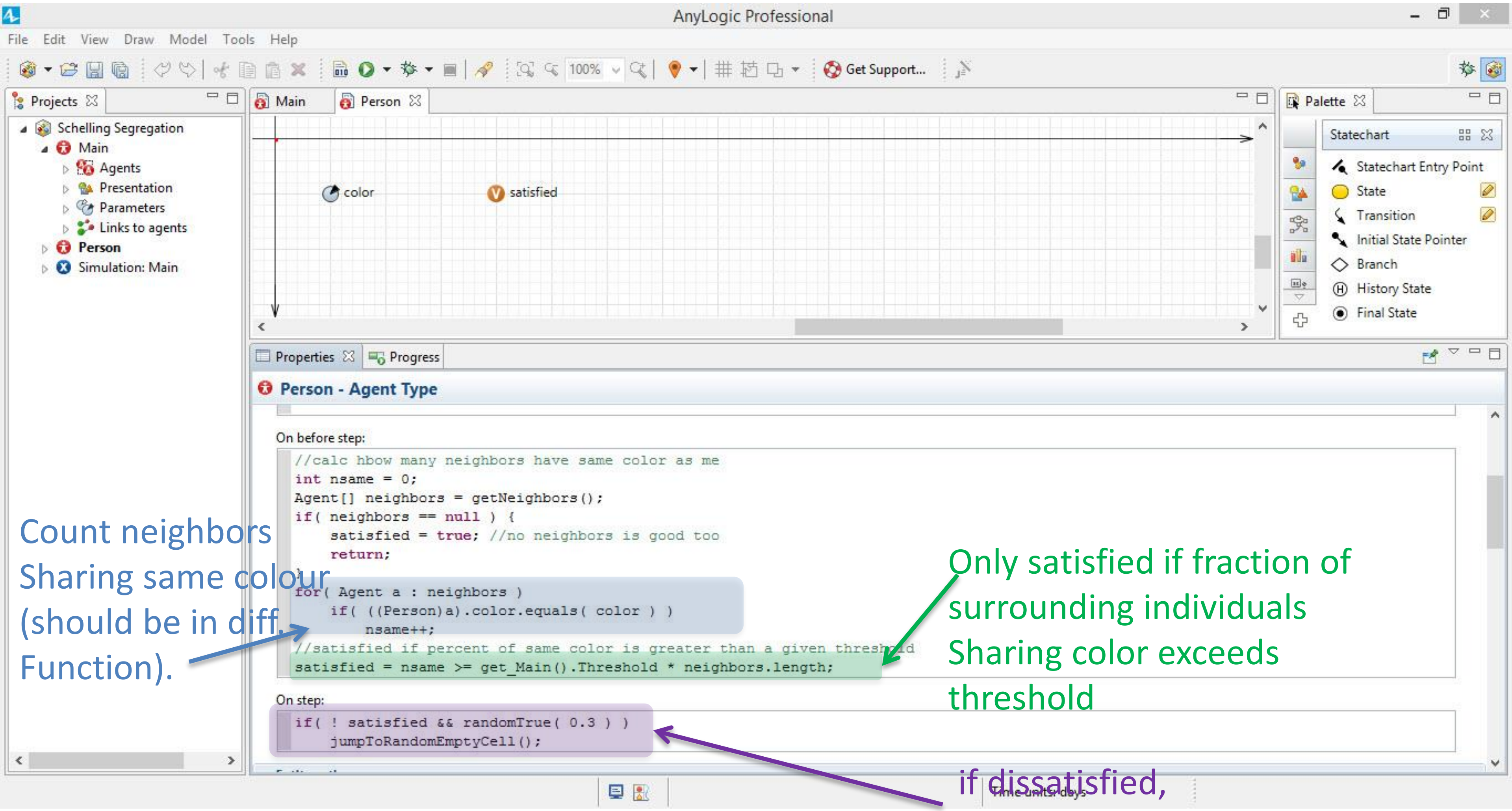
Slider Input Sets Parameter Value



Person is Assigned a Randomly Picked Color



Core Segregation (Movement) Logic

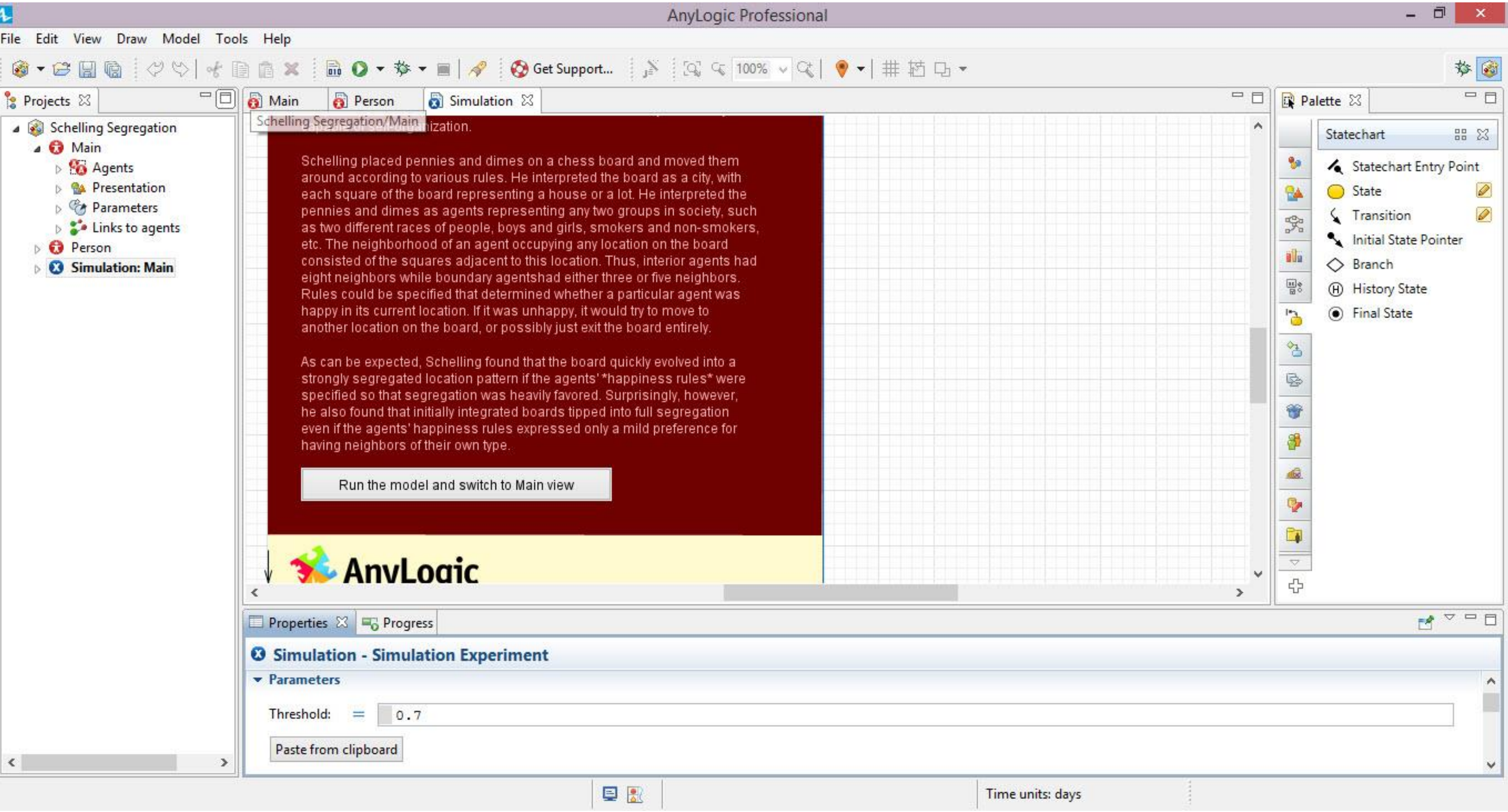


Count neighbors
Sharing same colour
(should be in diff.
Function).

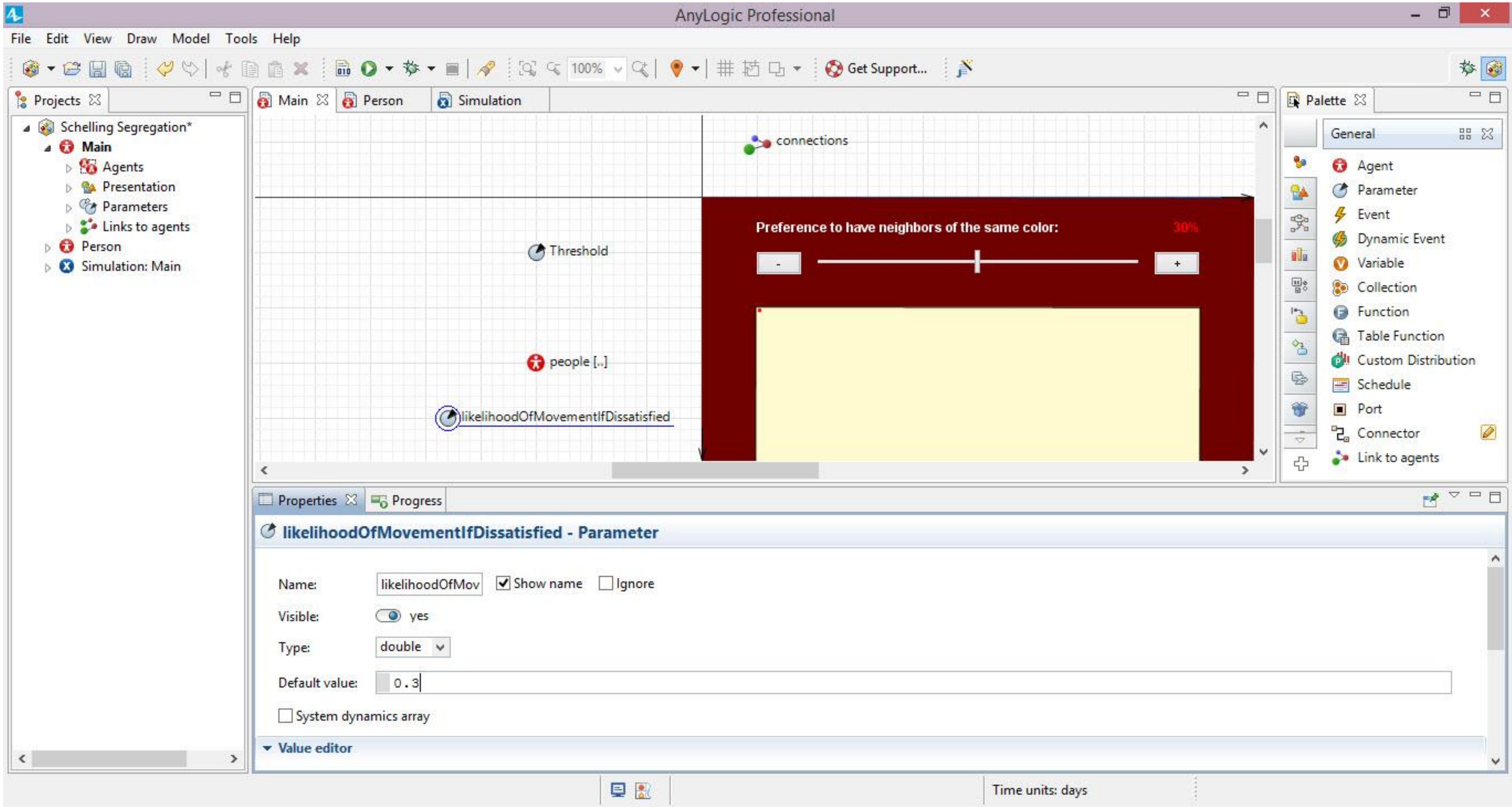
Only satisfied if fraction of
surrounding individuals
Sharing color exceeds
threshold

if dissatisfied,
30% chance of moving

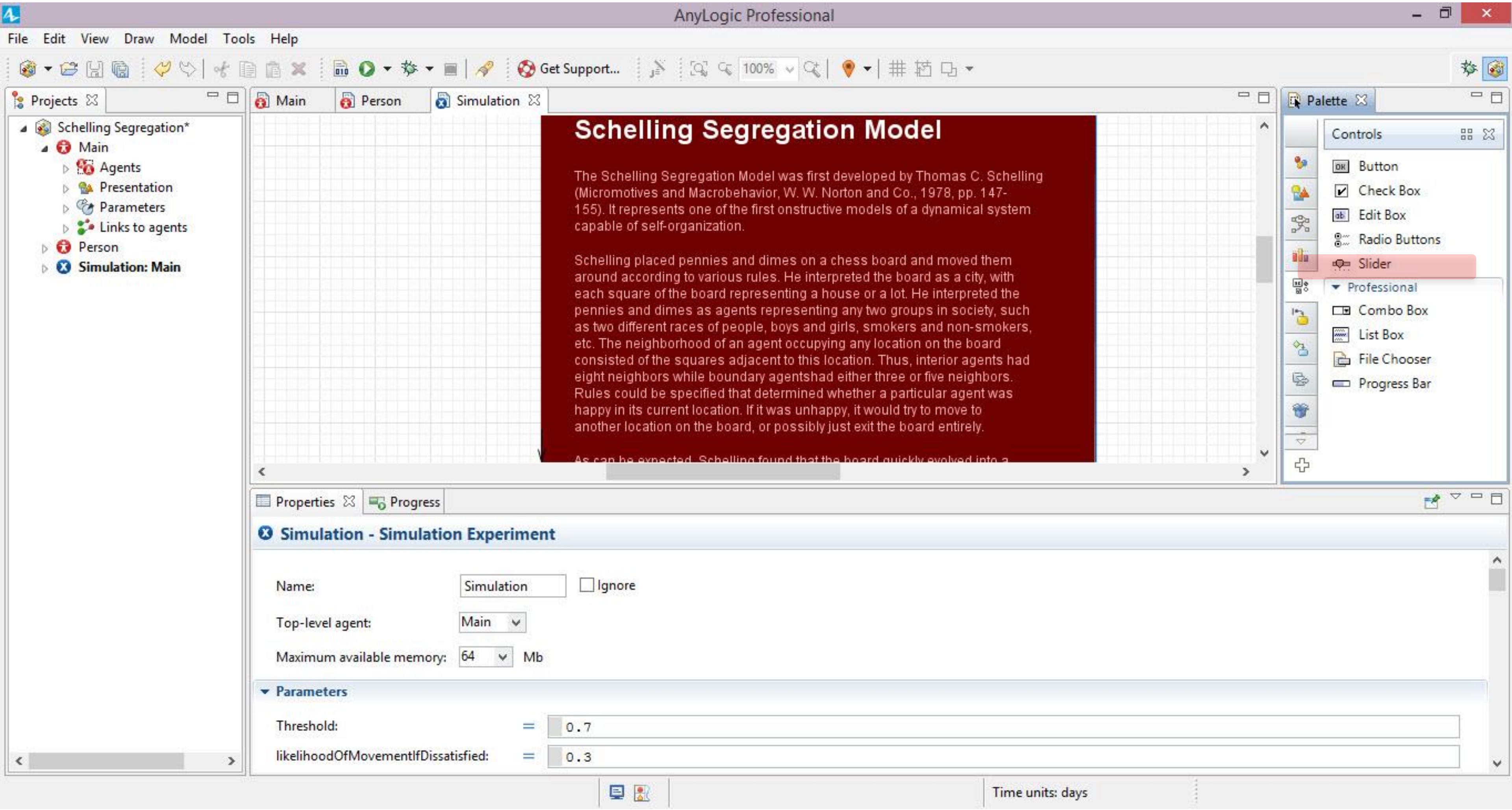
Experiment: Simulation Sets Parameter Assumptions



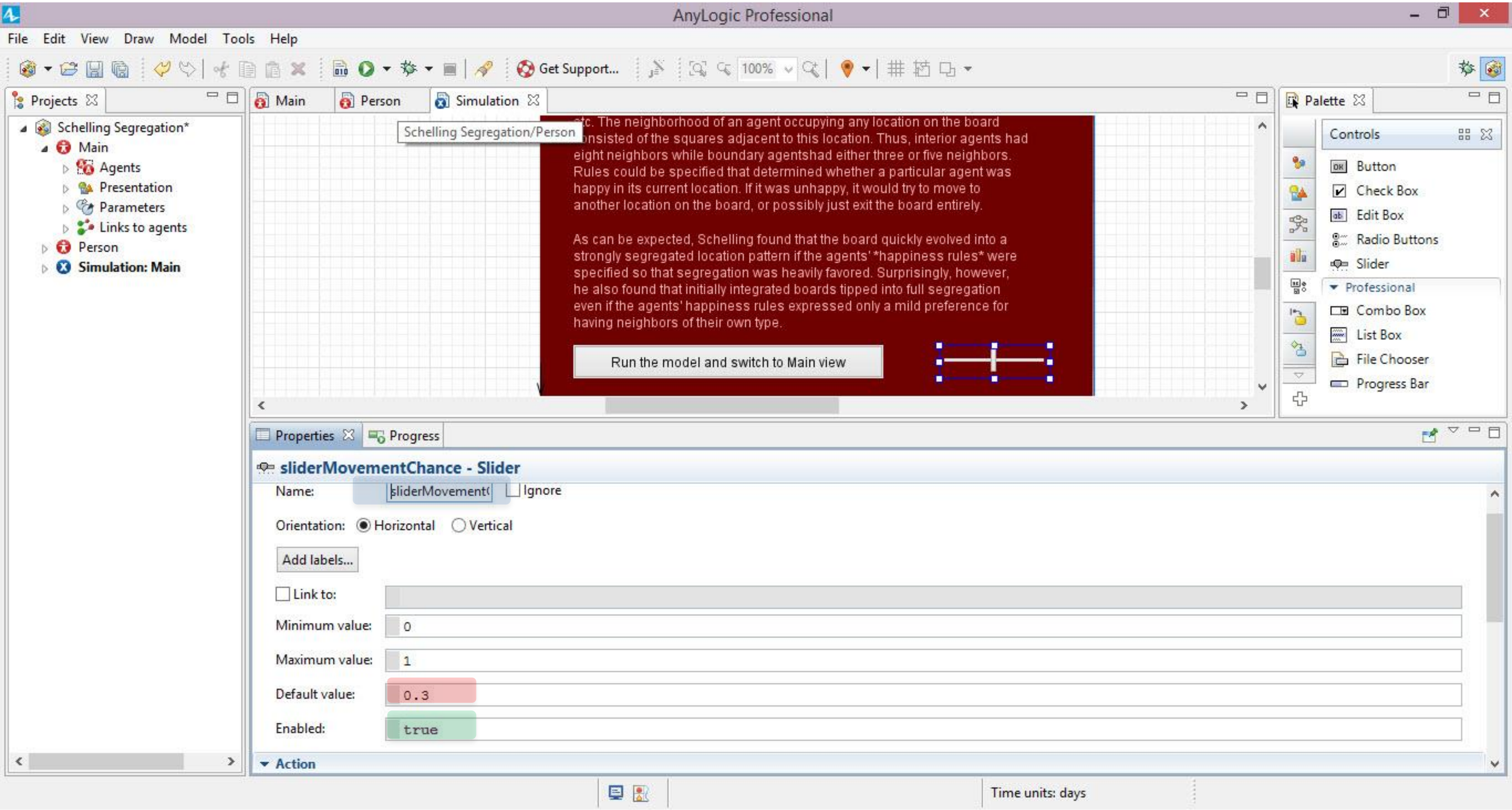
Add a Parameter to Main



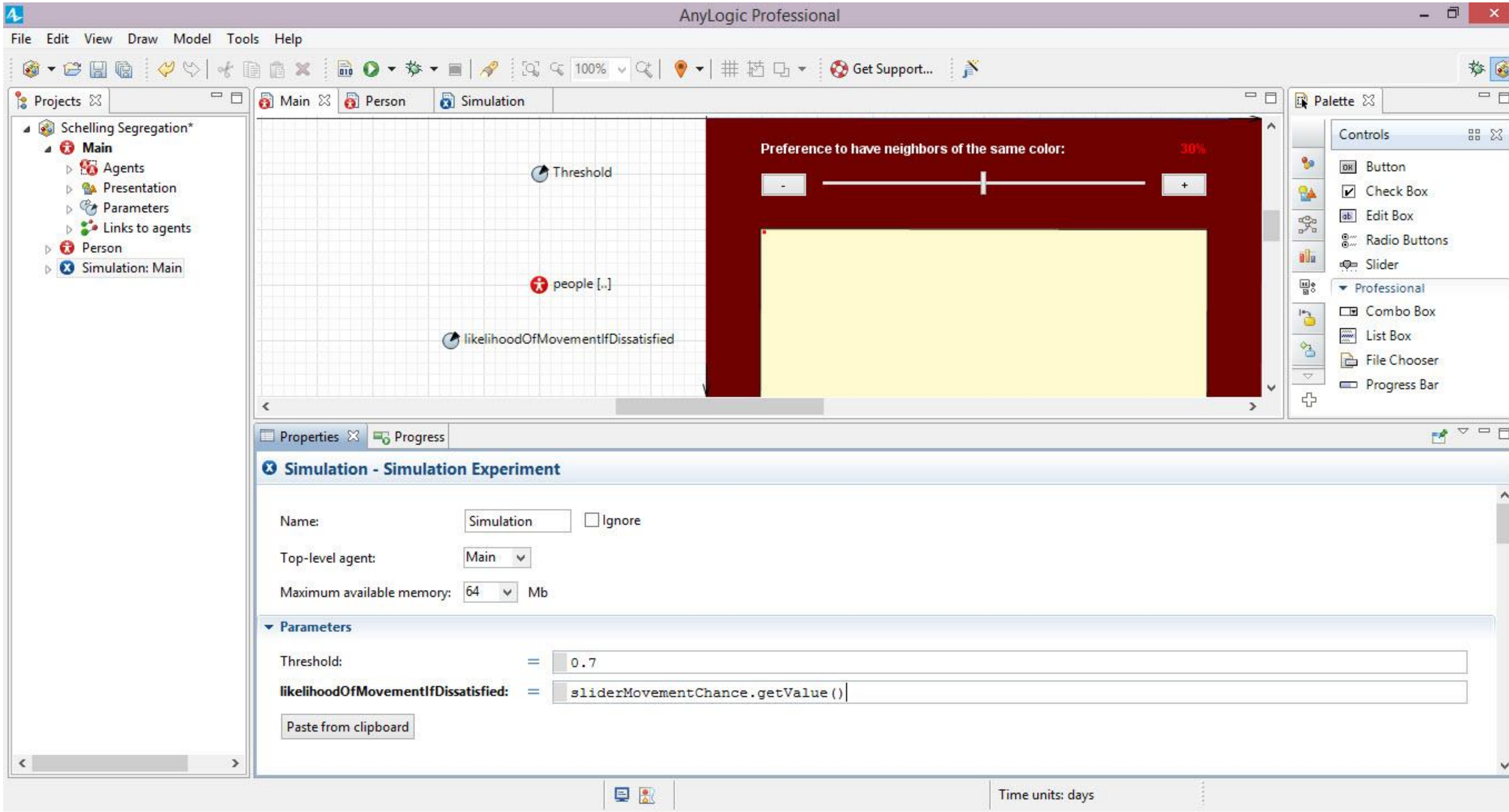
Experiment: Add a Slider!



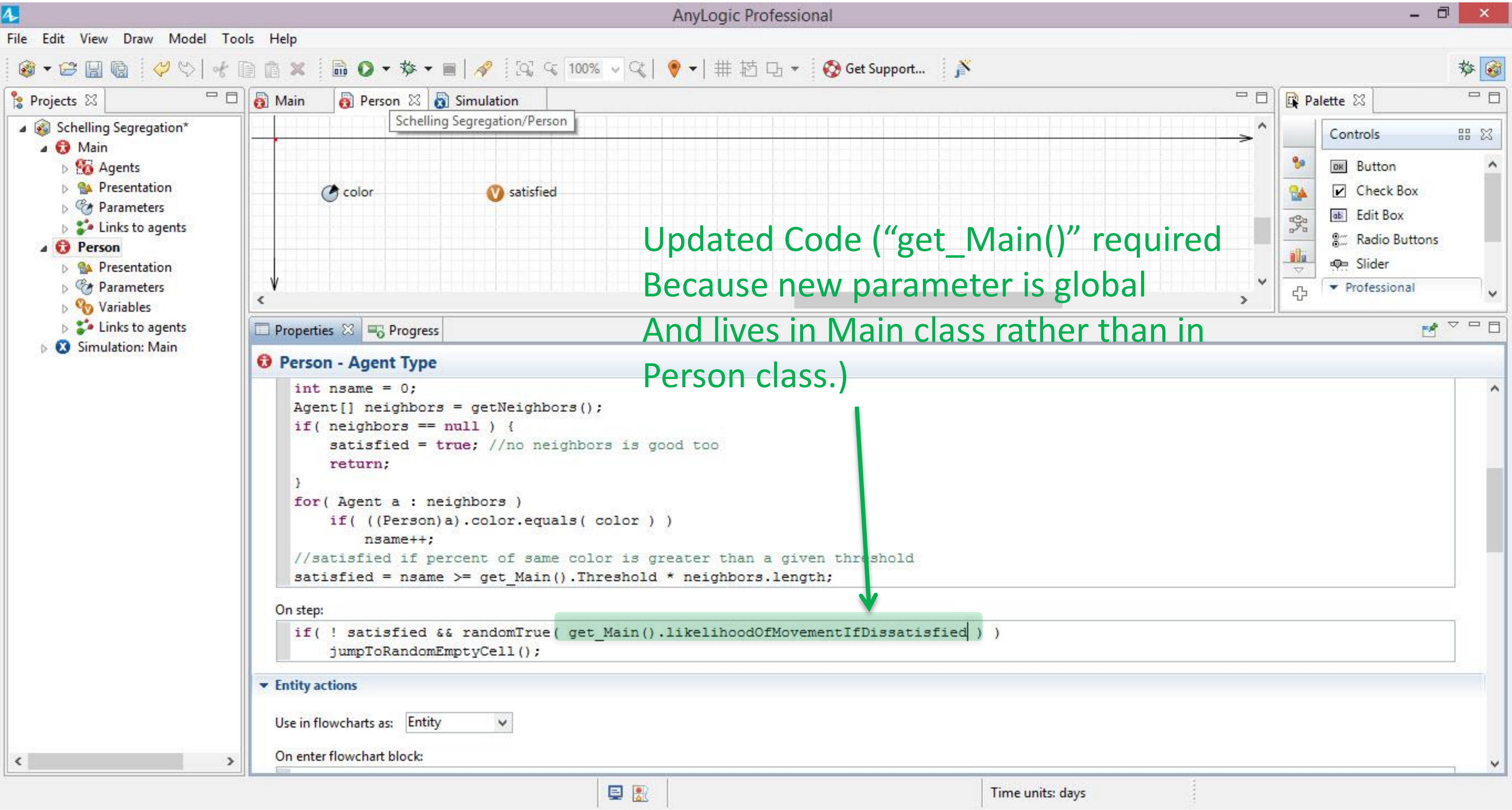
Setting the Slider Properties



Setting Value for Parameter from Slider



Modify Person's Behavior to Depend on New Parameter



Movement in Discrete Space

- jumpToCell(int row, int column)
  - Jumps to a particular unoccupied cell
  - **Precondition: destination cell is unoccupied**
- moveToNextCell(int direction)
  - Moves agent into a neighbouring cell in a given direction
  - Directions: NORTH, SOUTH, EAST, WEST, NORTHEAST, NORTHWEST, SOUTHEST,SOUTHWEST
  - **Precondition: destination cell is unoccupied**
- jumpToRandomEmptyCell()
  - Jumps to randomly selected empty cell (returning true), returns false if no empty cell can be located

Discovery in Discrete Space

- int []findRandomEmptyCell
  - Returns row & column of an unoccupied cell
- Getting agents in cell or direction
  - getAgentAtCell(int row, int column)
  - getAgentNextToMe(int direction)
  - getNeighbors()

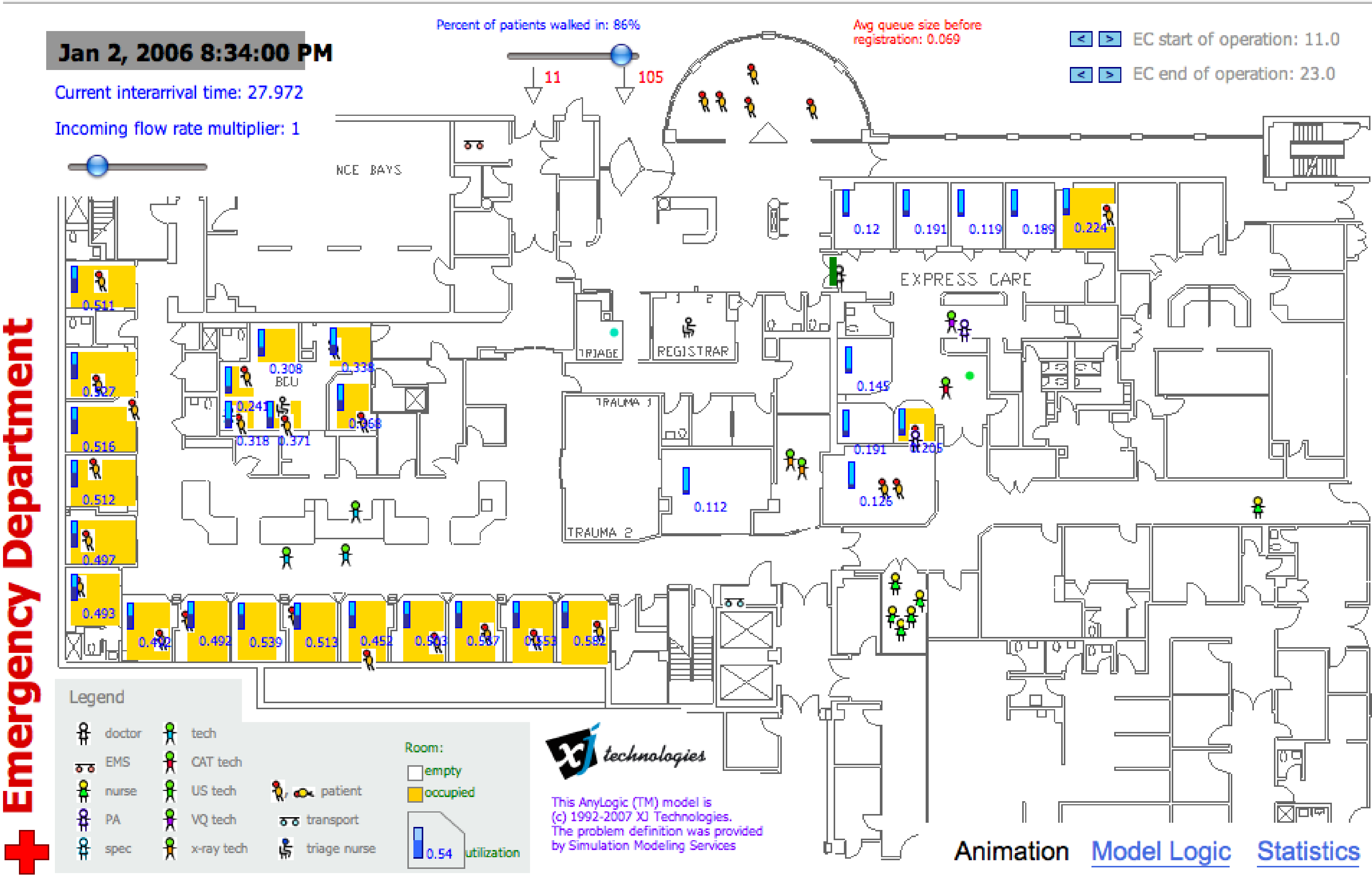
Important Distinction

- Suppose an agent is moving in discrete 2D space and need to be concerned about moving into the same cell as another agent
- We can readily prevent this agent from moving into another cell currently occupied
- But can we prevent this agent from colliding with another agent that wishes to move into the same cell?
  - To answer this, we need to be clear about the model of time used by agents

Synchronization & Discrete Agent Movement

- In Synchronous mode, it is difficult to know if two agents will collide using data on the current timestep
  - Even if we know where the other object was during the current timestep, it's possible it will move into the cell we wish to occupy in the next timestep
- It is simpler to handle this asynchronously
  - Here, we can have each agent update at slightly different times, and observe the location of the other agents at the current time – without any significant chance that they will move to the same place at the same time.
- Issue only arises for discrete agent movement, as this is the only case where cells are limited to contain 1 agent

Irregular Spatial Embedding



Realizing Irregular Spatial Embedding in AnyLogic

- Basic idea: people moving around follow networks of *paths*
- Irregular spatial embedding is supported directly by "Network Based Modeling" (Discrete Event Simulation)
  - This approach is individual-based, but treats agents either as flowing through and being operated on by a process or as (often interchangeable) process resources
  - We will have a brief introduction to this approach later in the week, showing how it can be combined with ABM
- With a modest amount of custom coding, irregular spatial embedding can be achieved within ABM by routing the agent along a network of "polylines"