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

Nathaniel Osgood
MIT 15.879

March 23, 2012

Agent Mobility

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

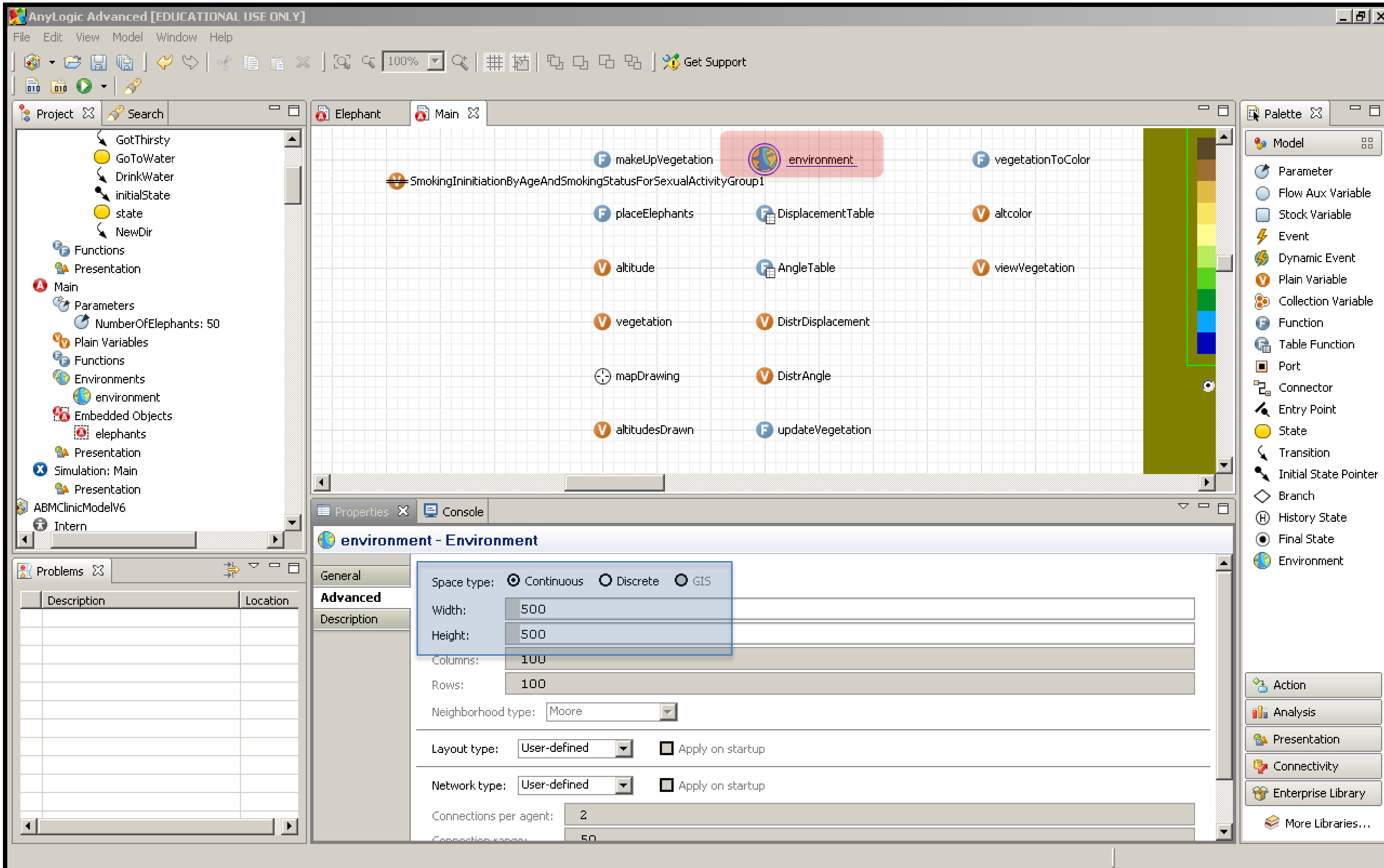
2D Spatial Embedding: Mobility Implications

- 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

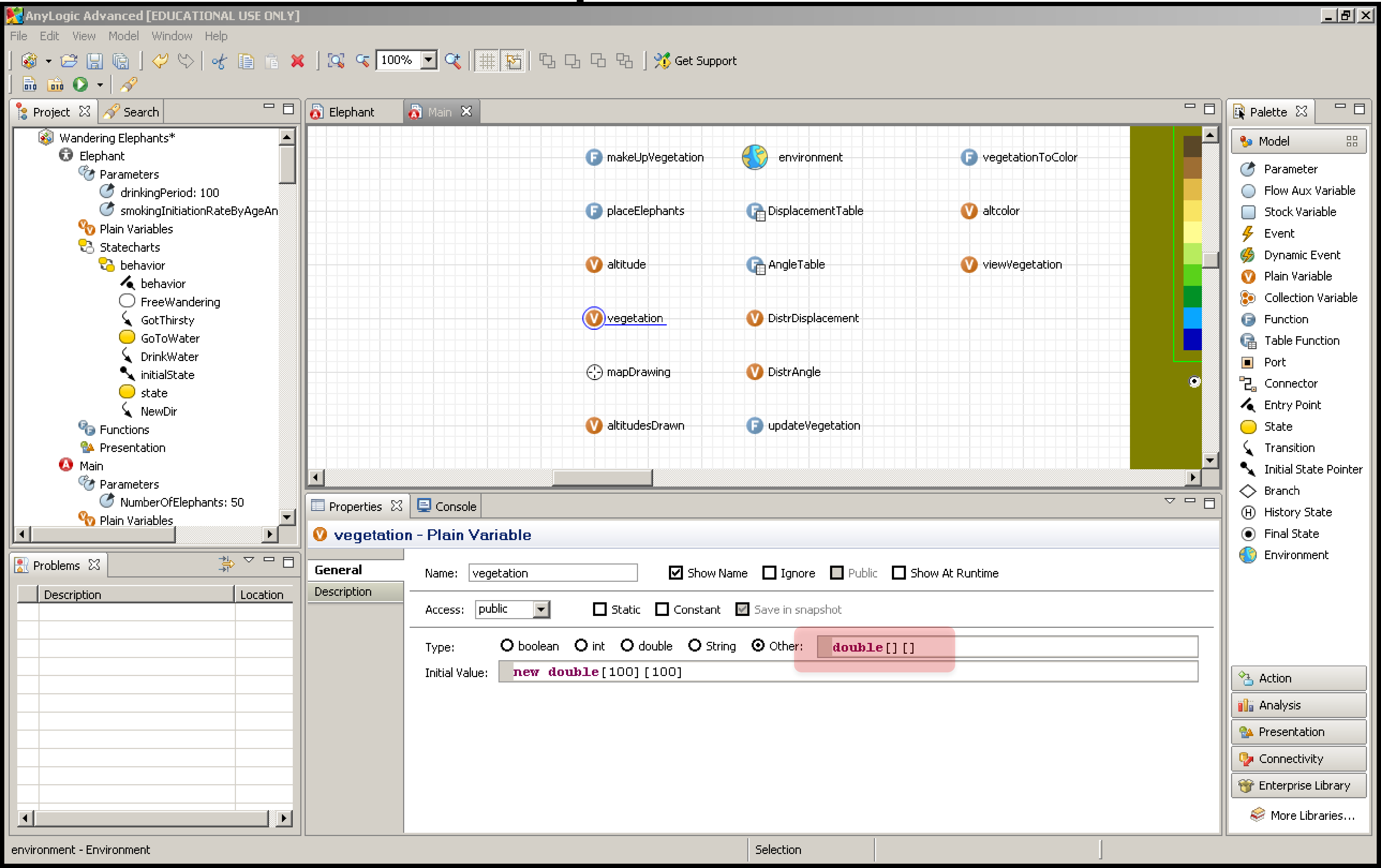
Hands on Model Use Ahead

Load model: Wandering Elephants.alp

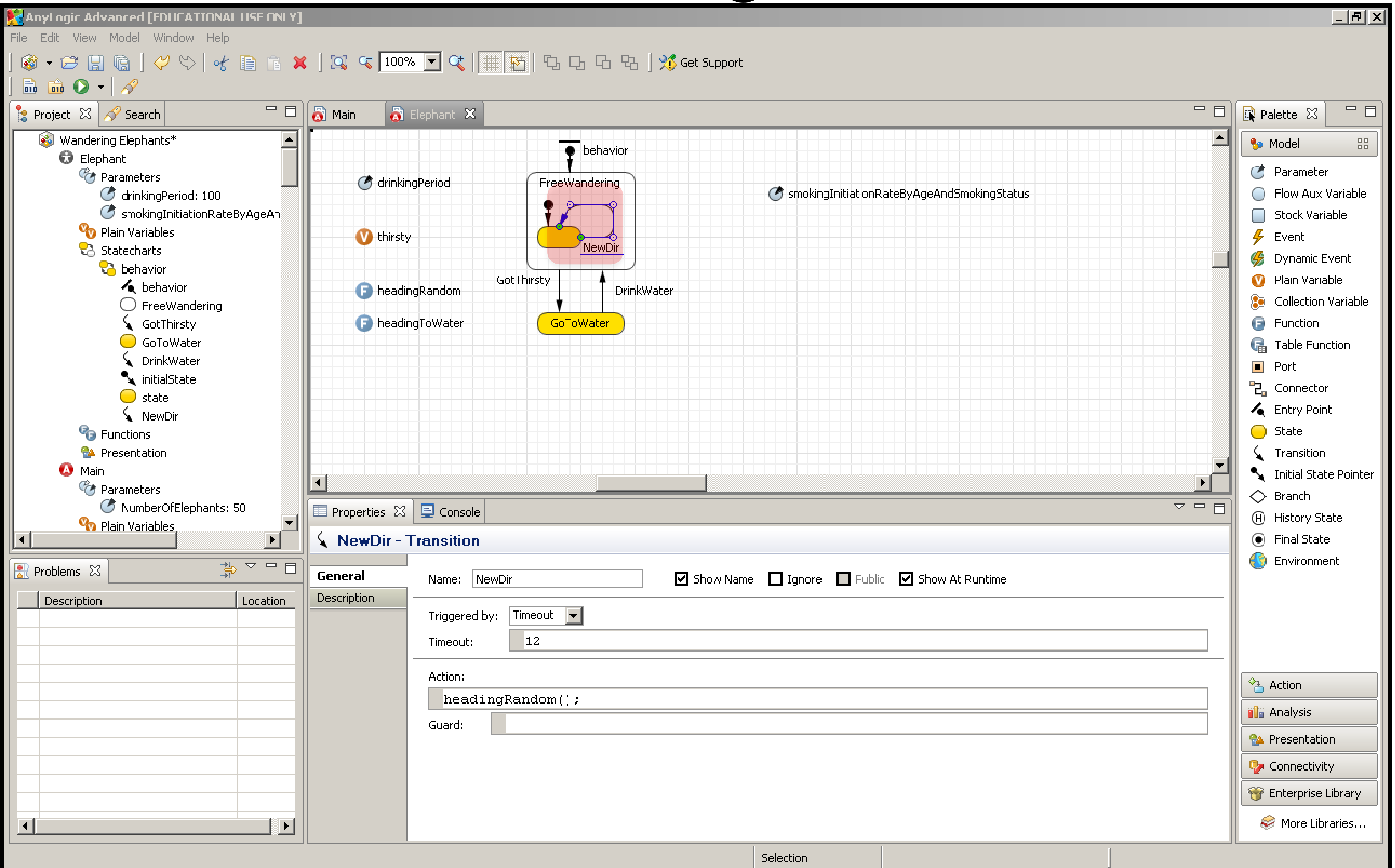
Environment



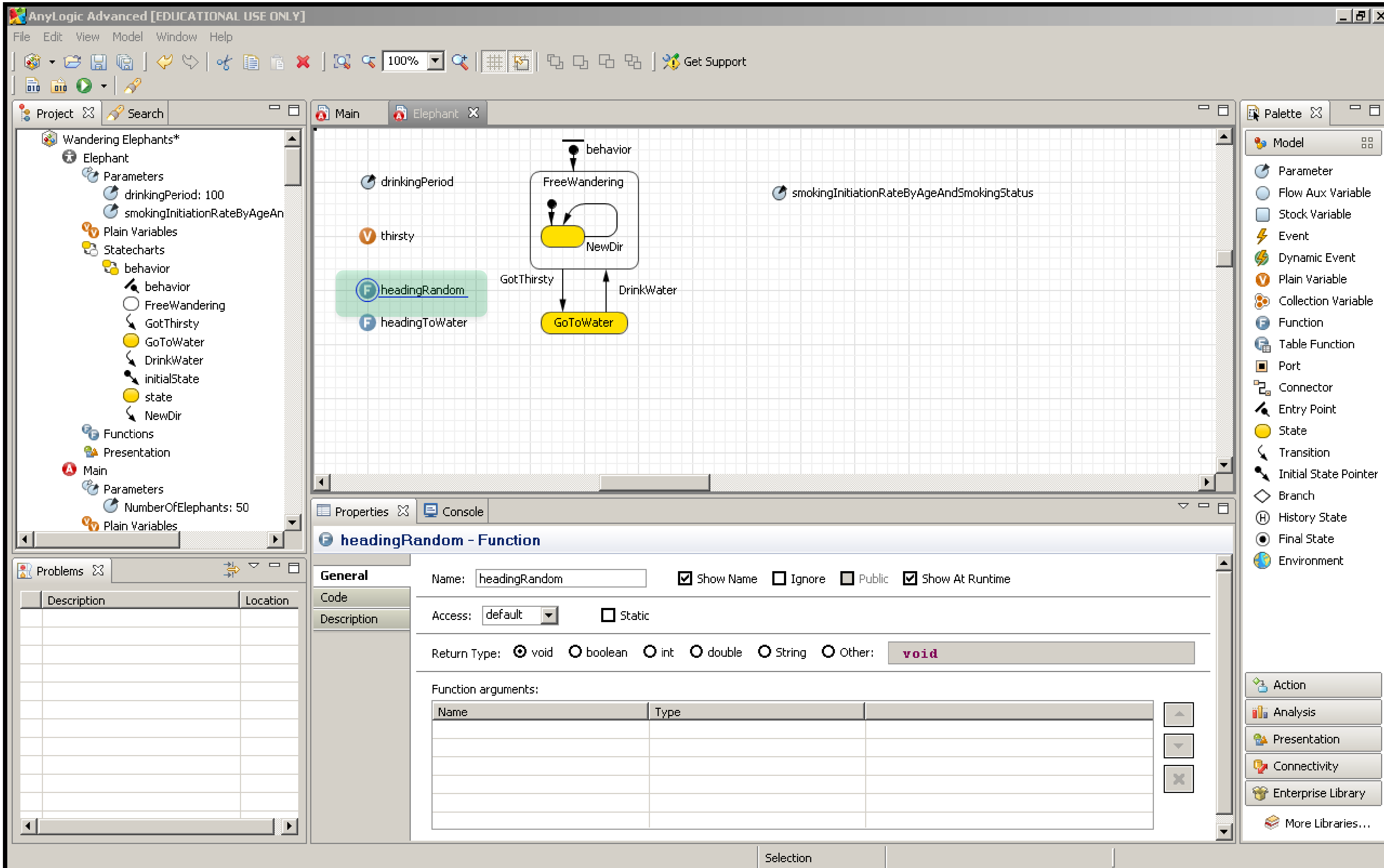
Landscape Information



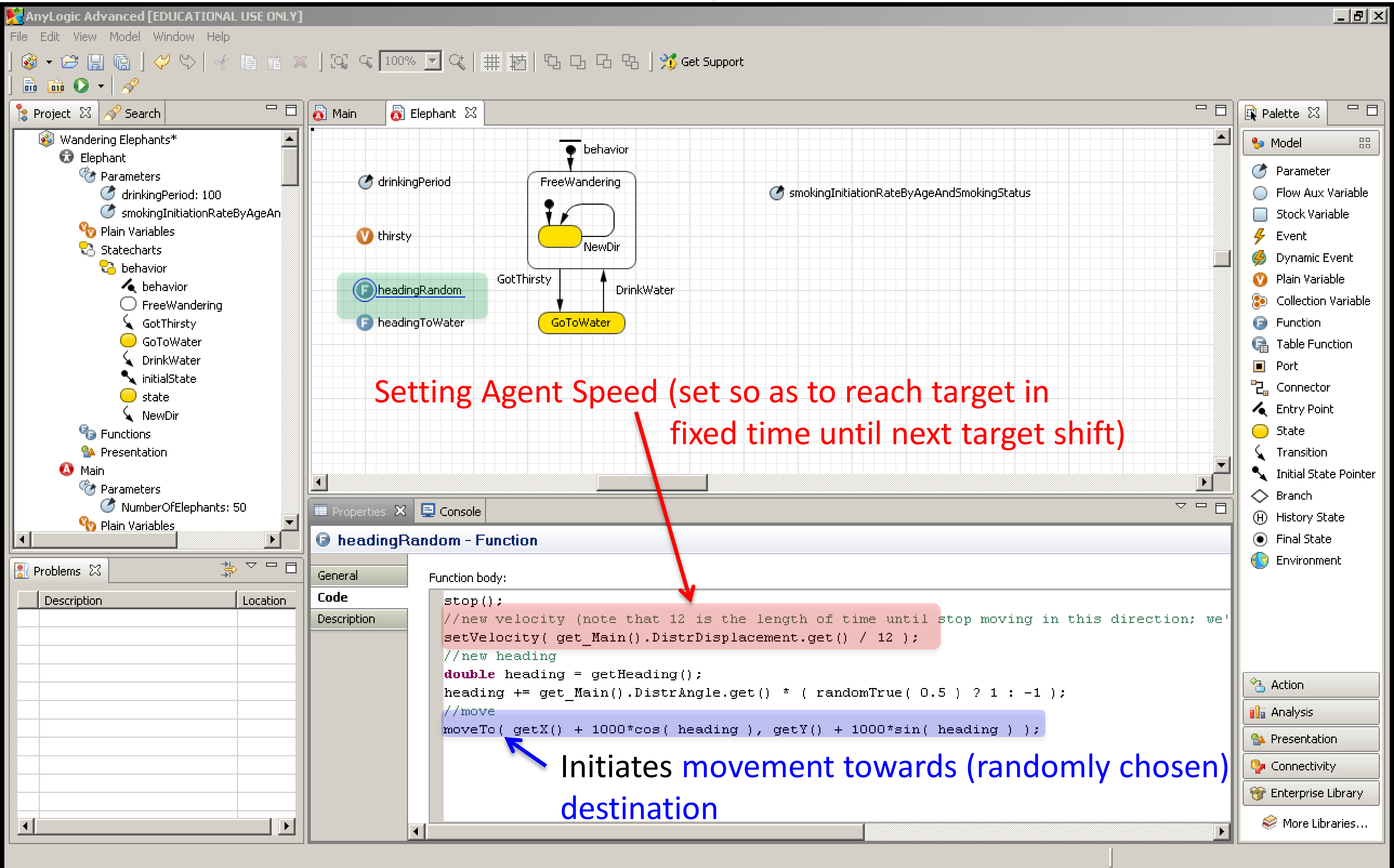
Agent Movement: Periodic Movement Changes



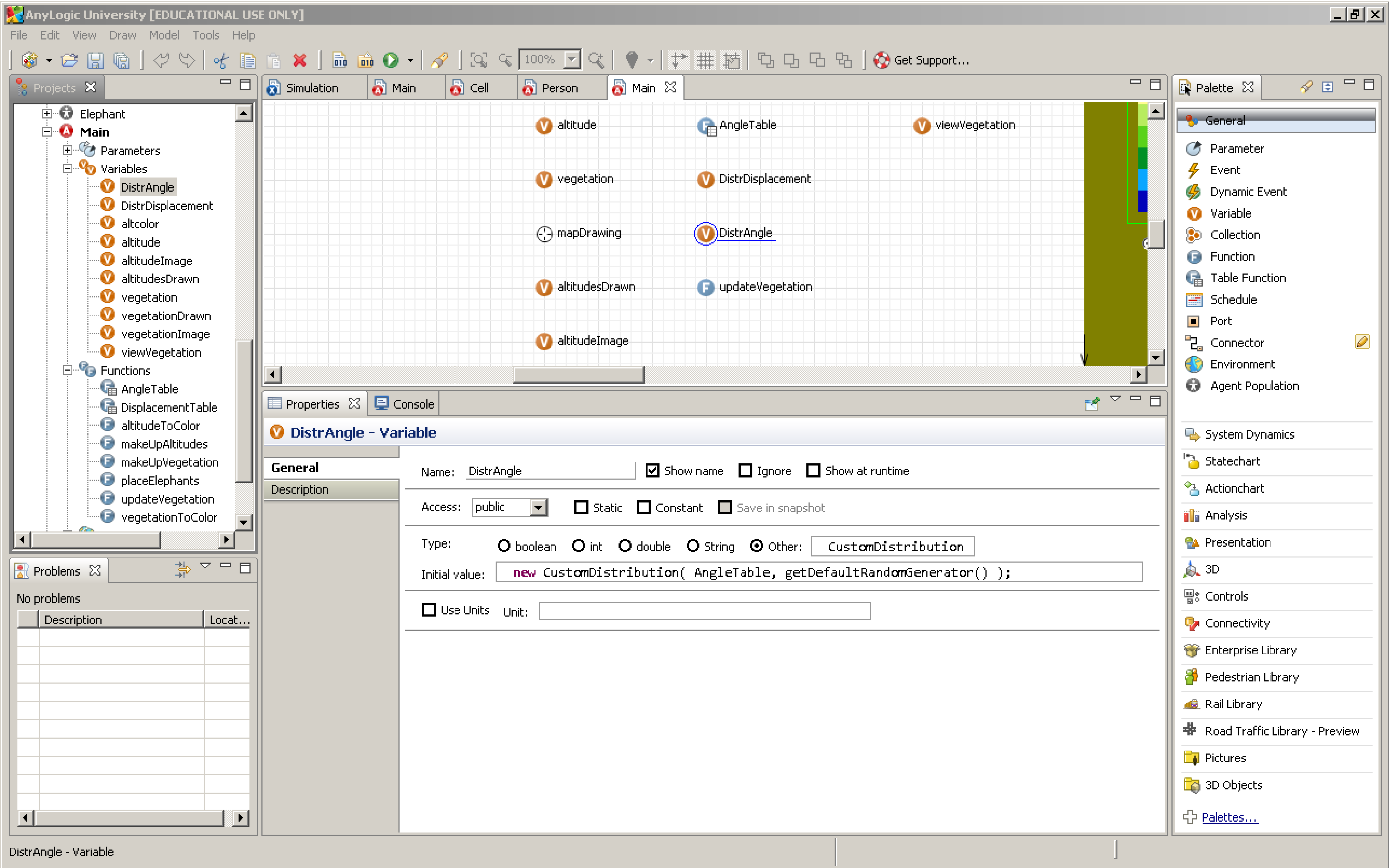
New Direction Change Function Info



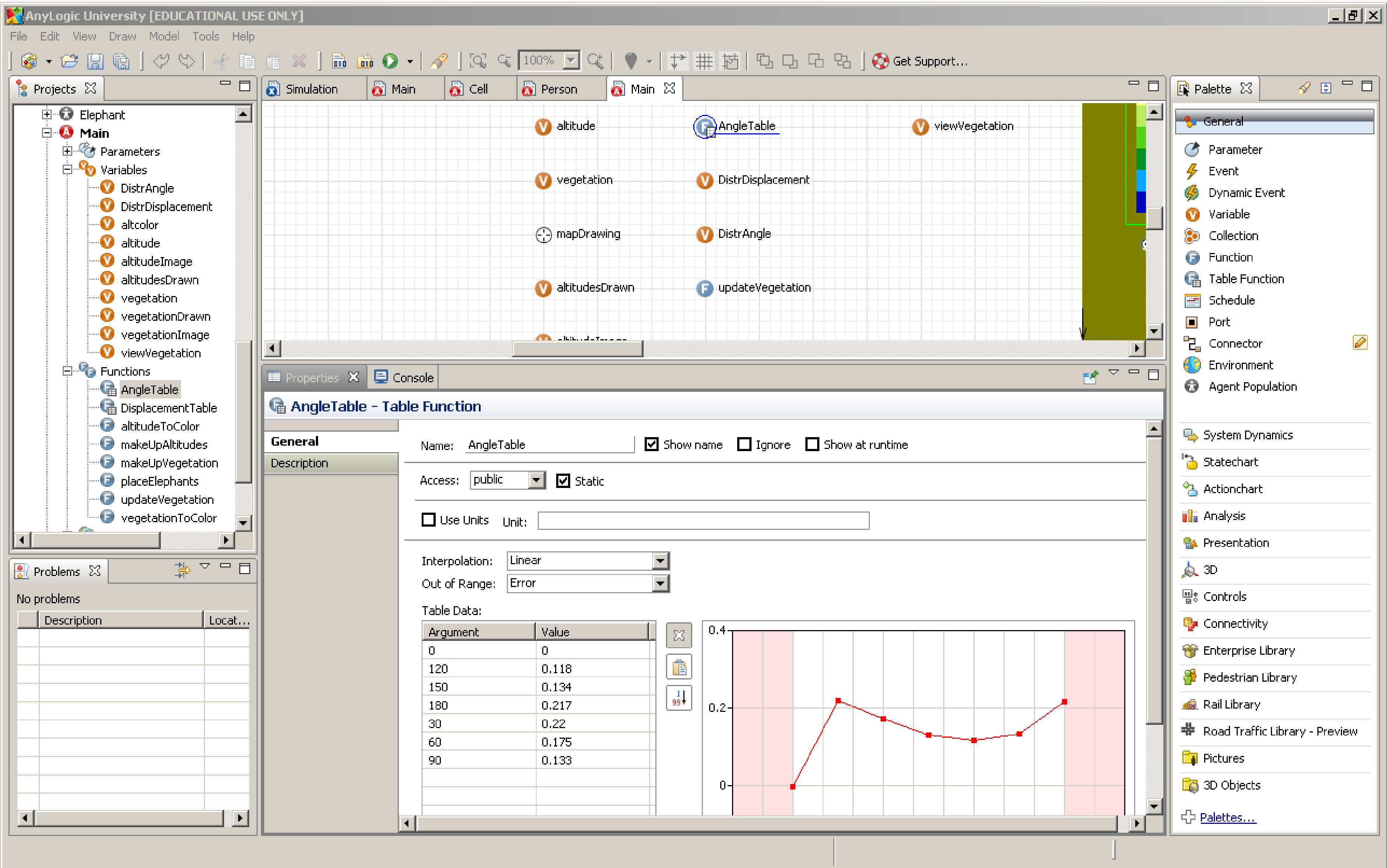
New Direction Change: Function “Body”



(Main) Defining a Custom Angle Distribution

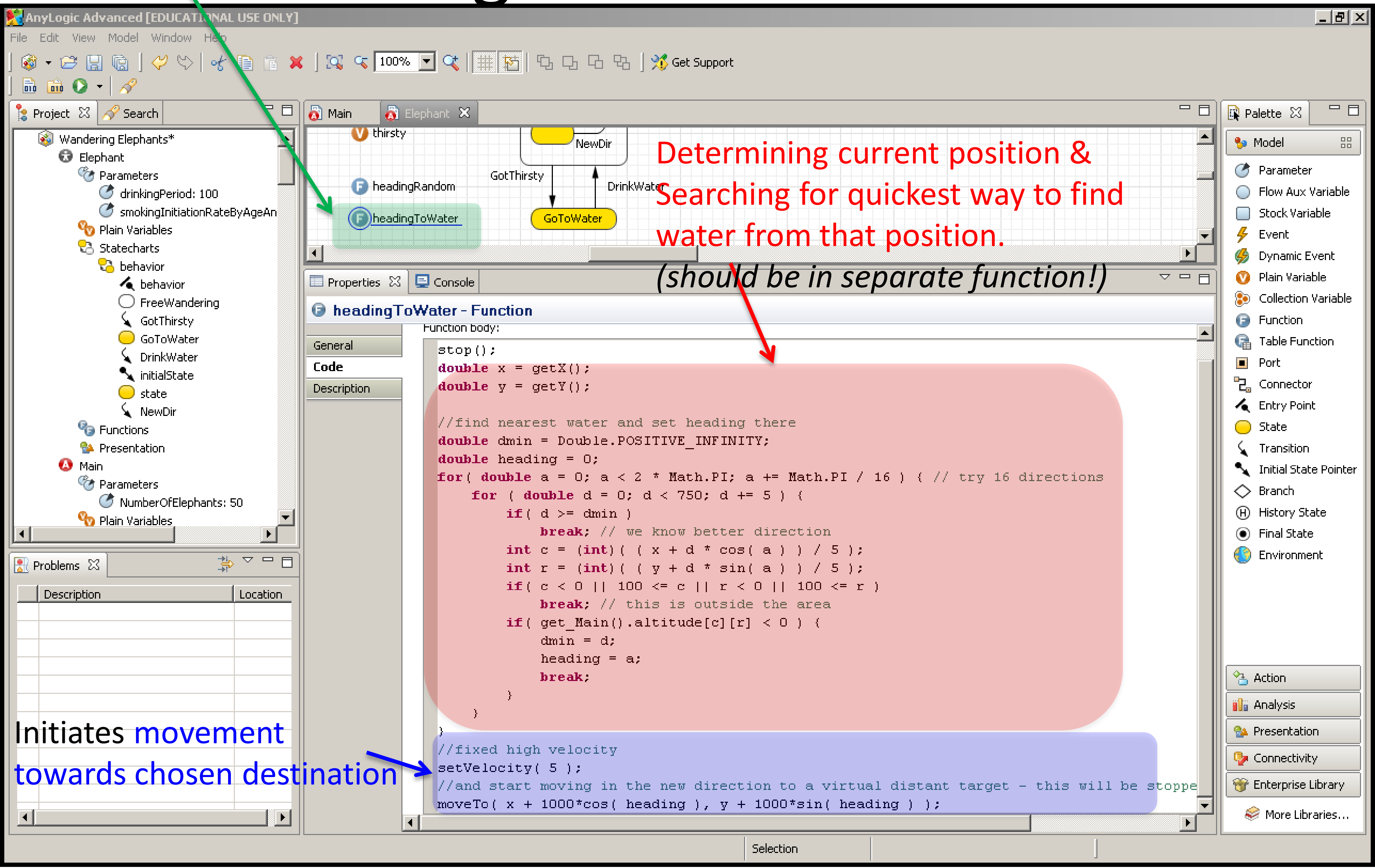


Data for Custom Distribution



Heading Towards Resource

Looking at body of this function (method)

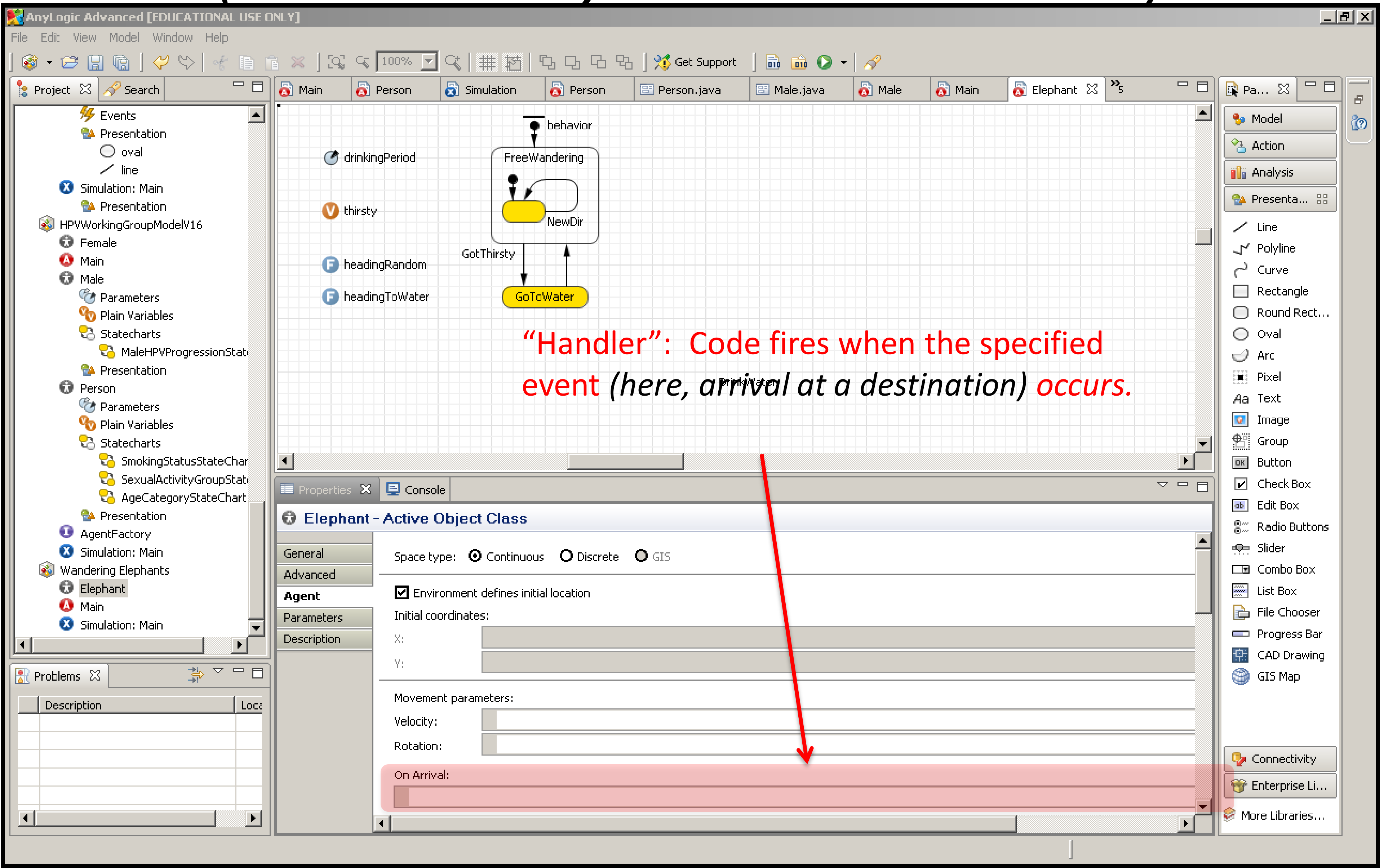


Determining current position & Searching for quickest way to find water from that position.

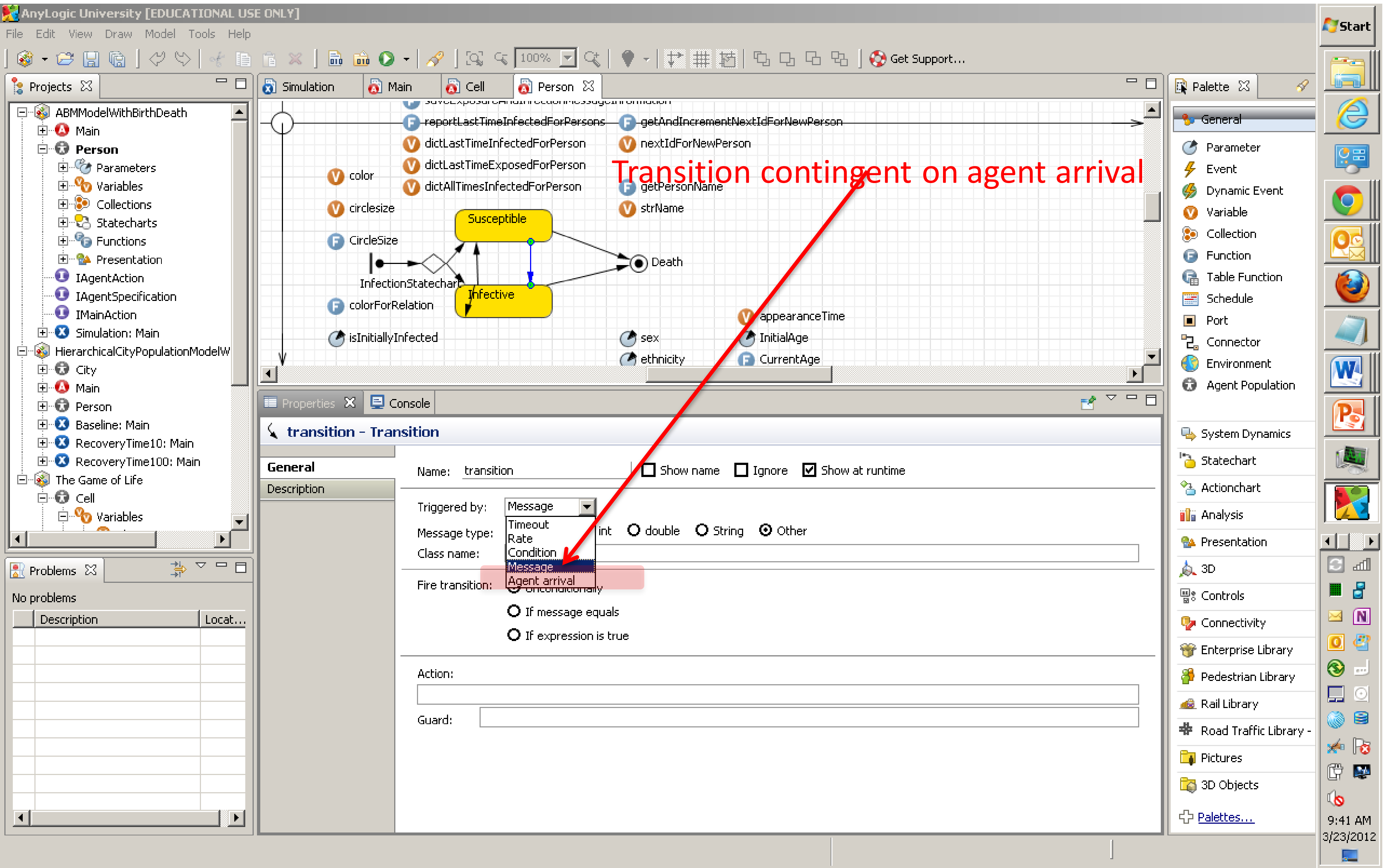
(should be in separate function!)

Initiates movement towards chosen destination

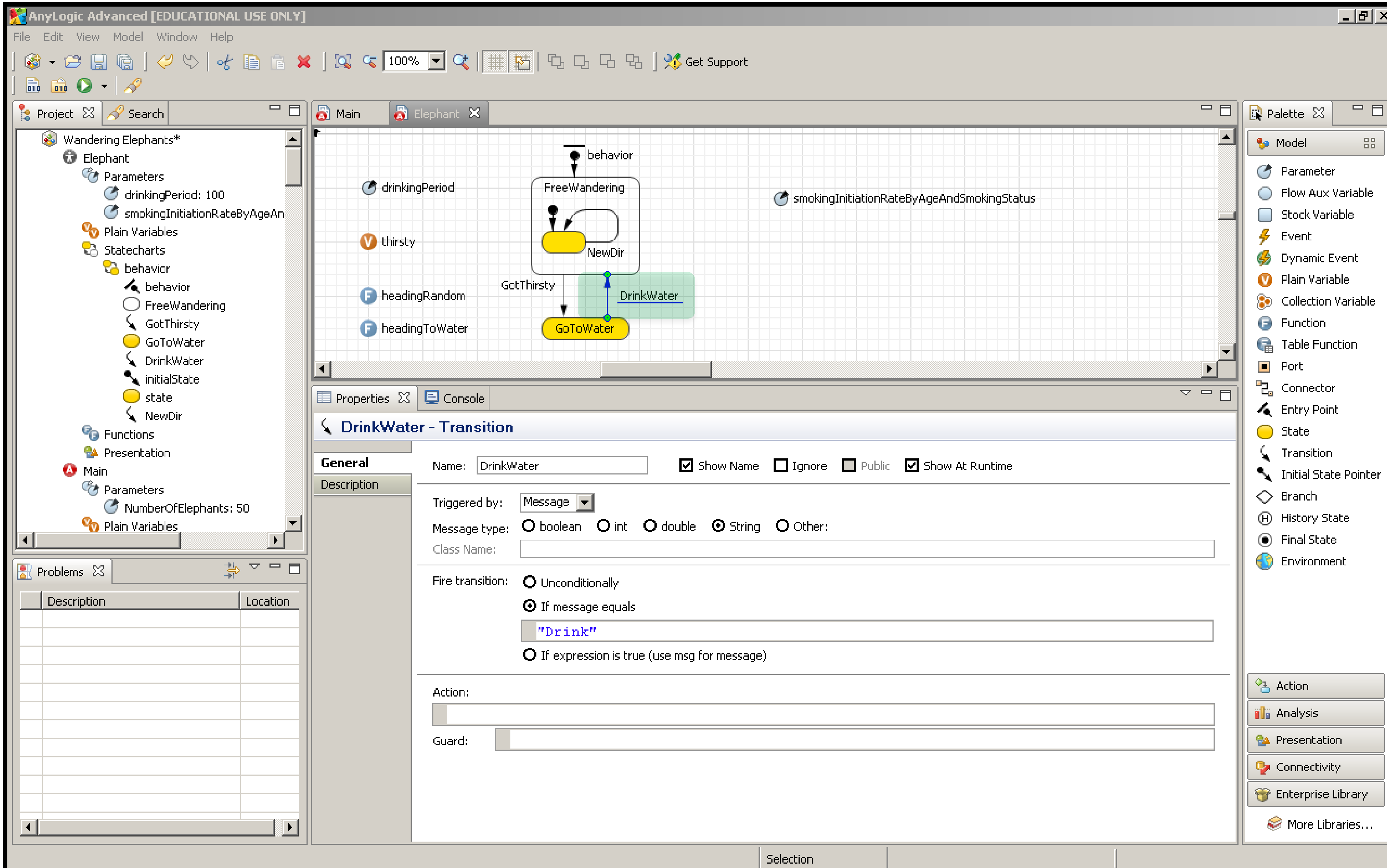
Handling Agent Arrival at Destination
(Not Currently Used in this Model)



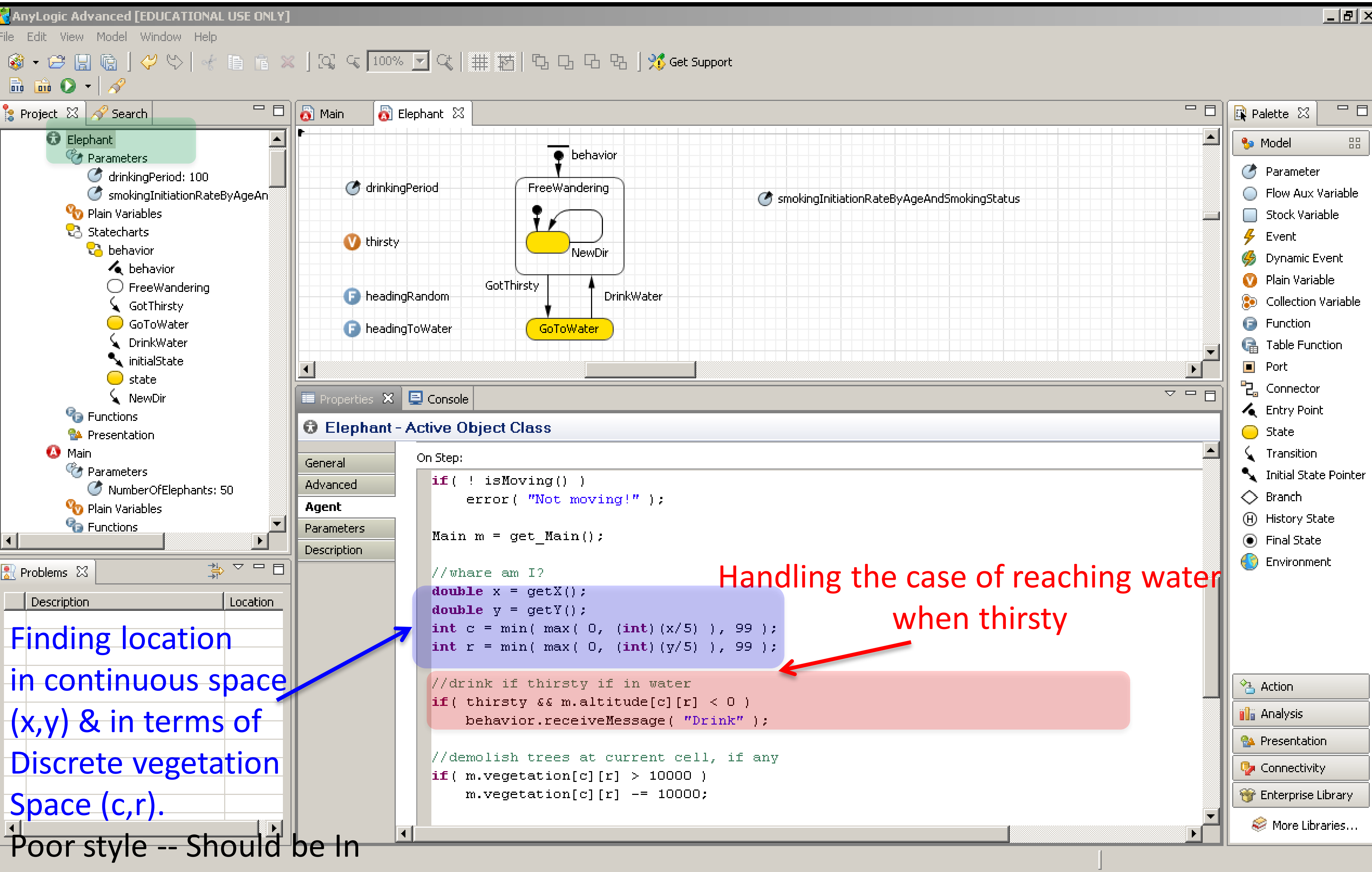
Handling Arrival Events in Statecharts



Resumption of Wandering After Slaking Thirst



Handling of Movement Logic



Handling the case of reaching water when thirsty

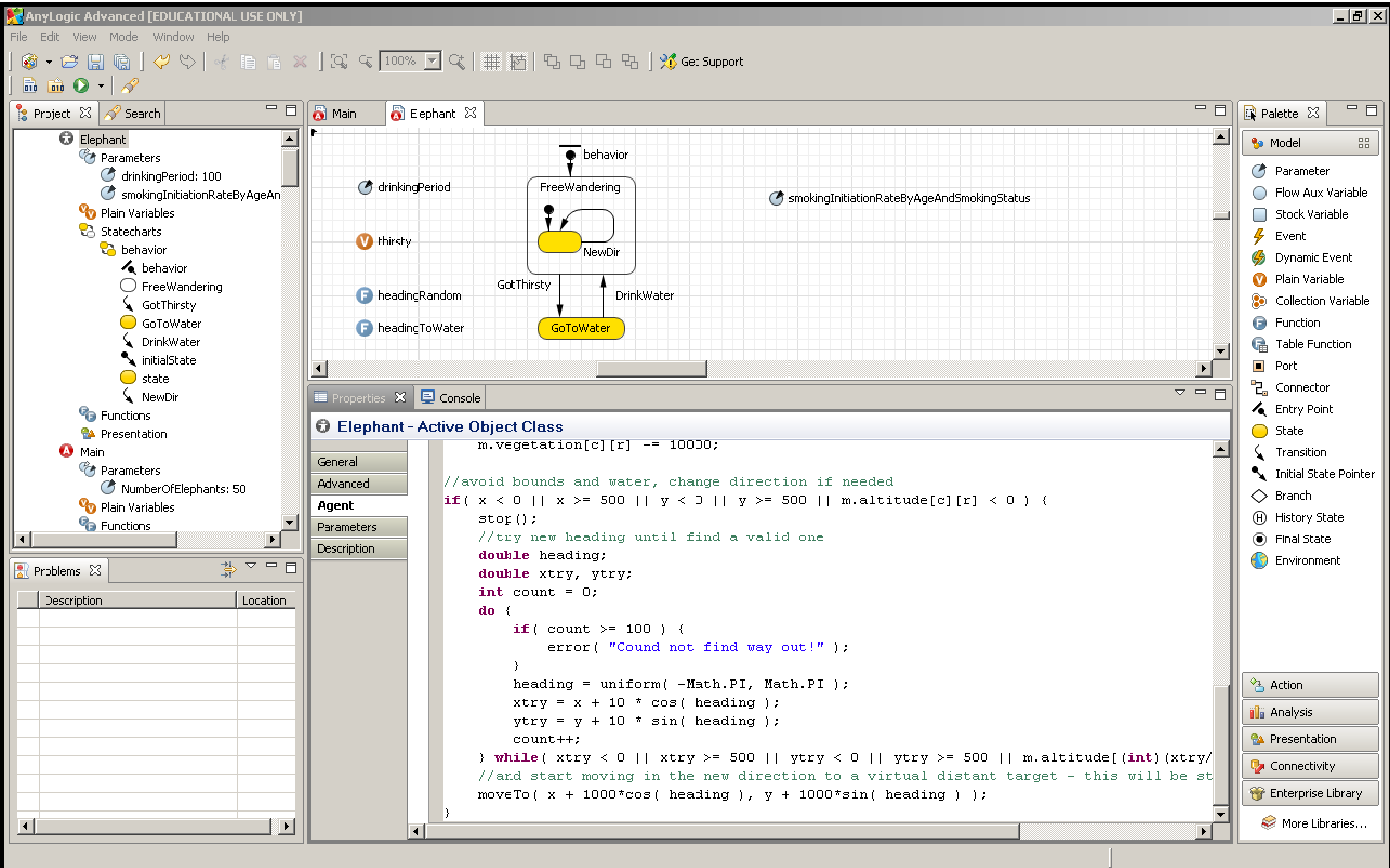
Finding location in continuous space (x,y) & in terms of Discrete vegetation Space (c,r).

Poor style -- Should be In separate function

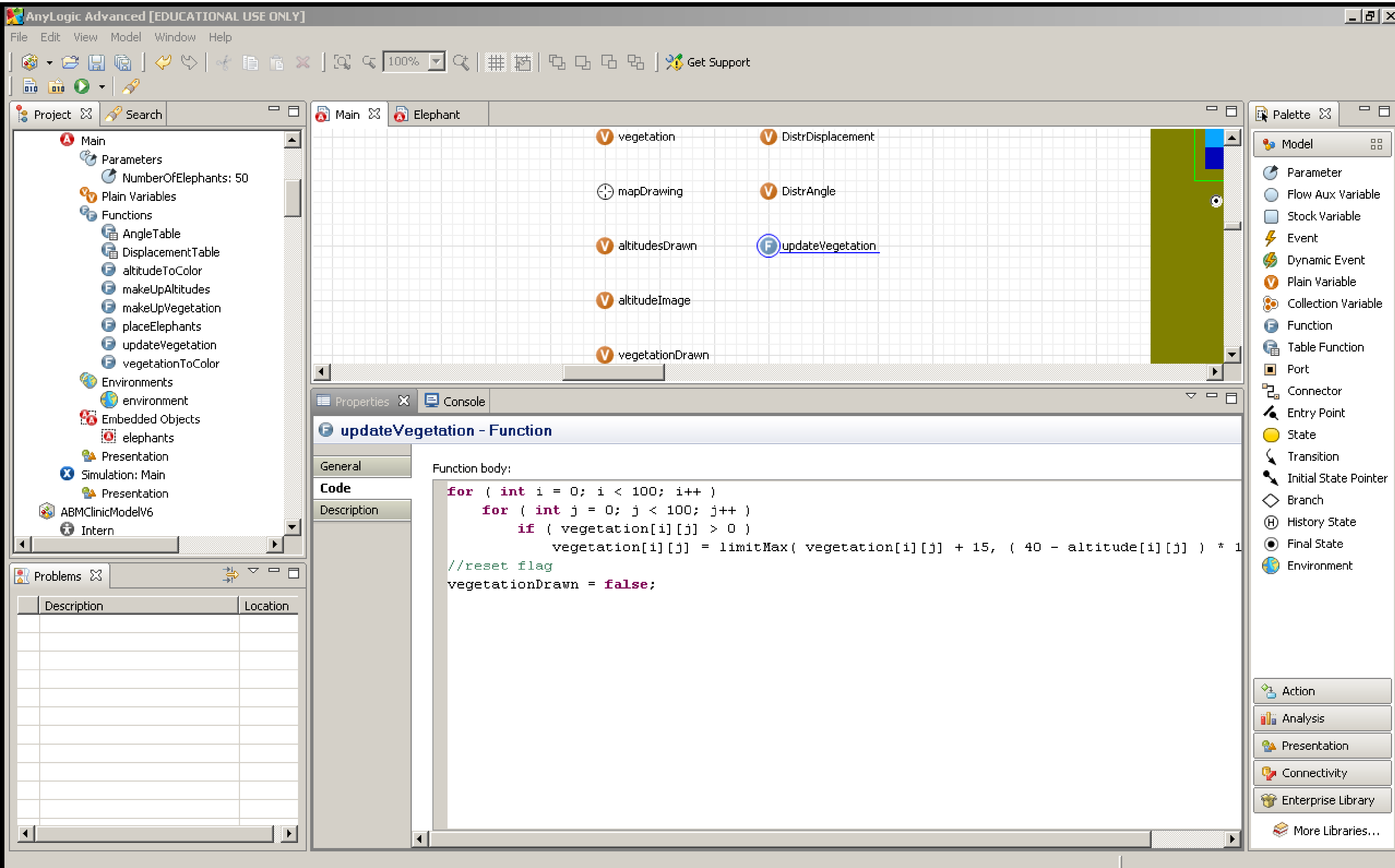
Rerouting Around Barriers (Boundaries & Water)

Poor Style – entire logic, conditions (checks on boundaries, whether water) & rerouting

Logic should all be in separate functions from this & from each other). Remove constants



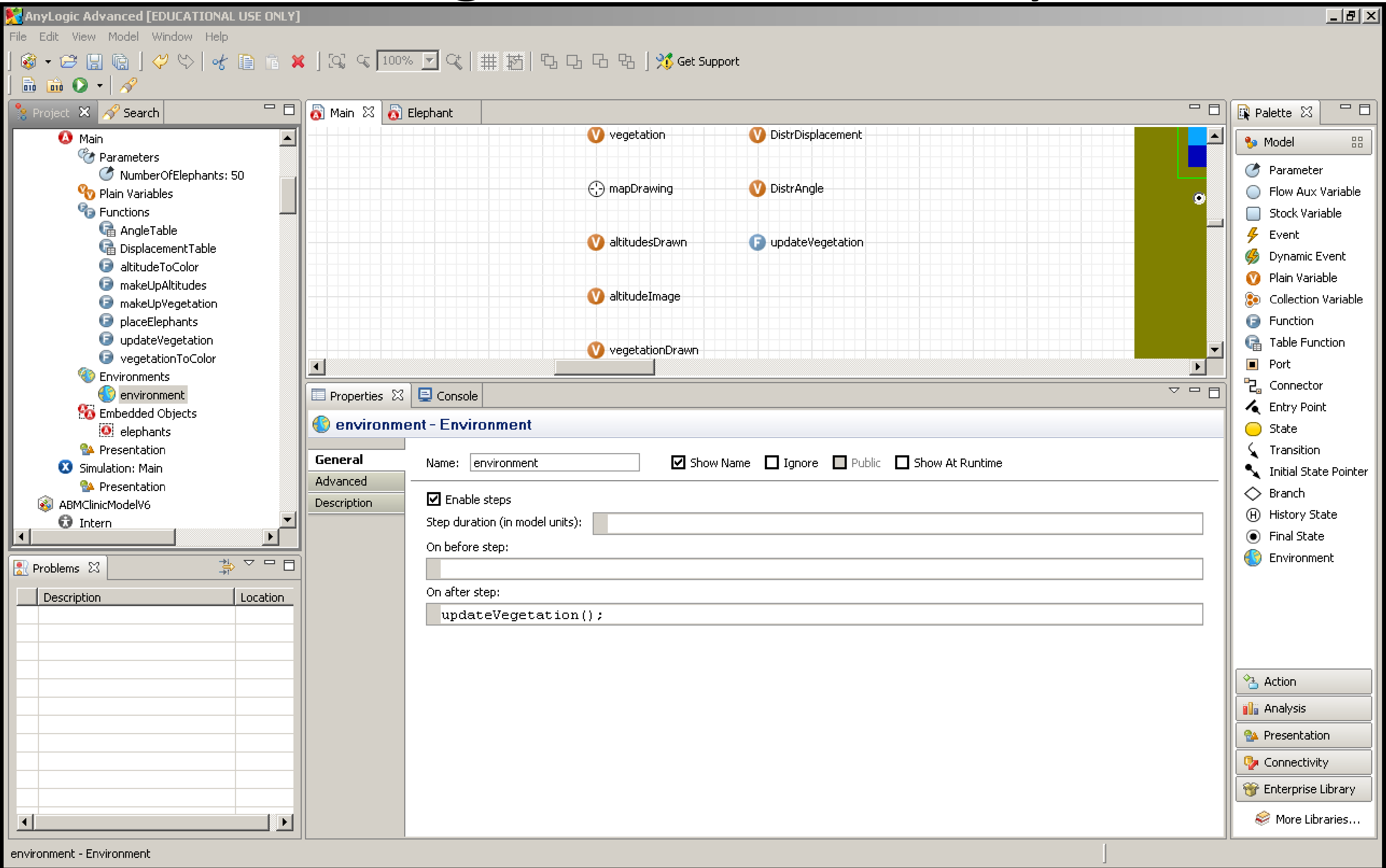
Environment: Updating Vegetation



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

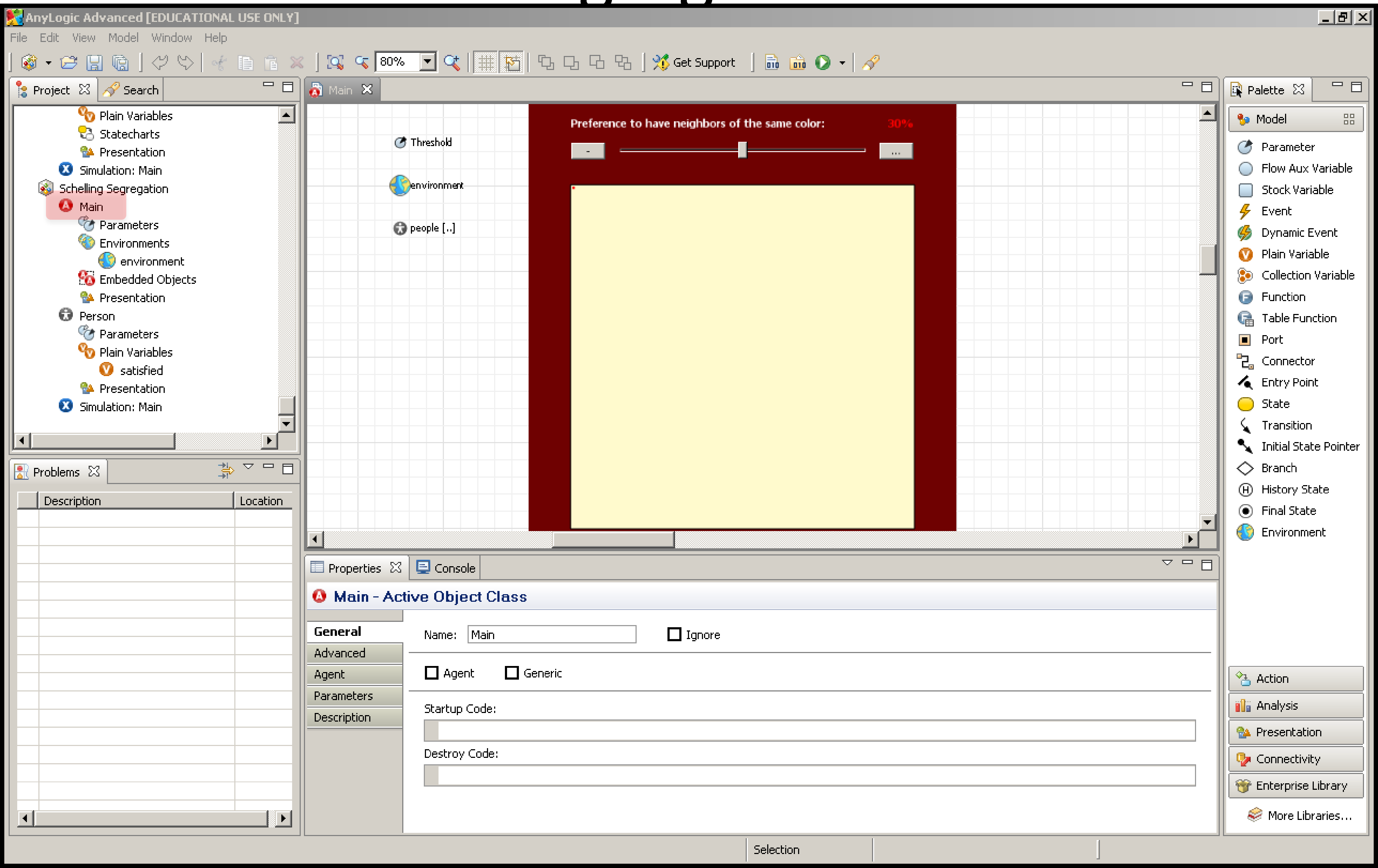
Environment Happens to Handle Process of Maintaining Environmental Dynamics



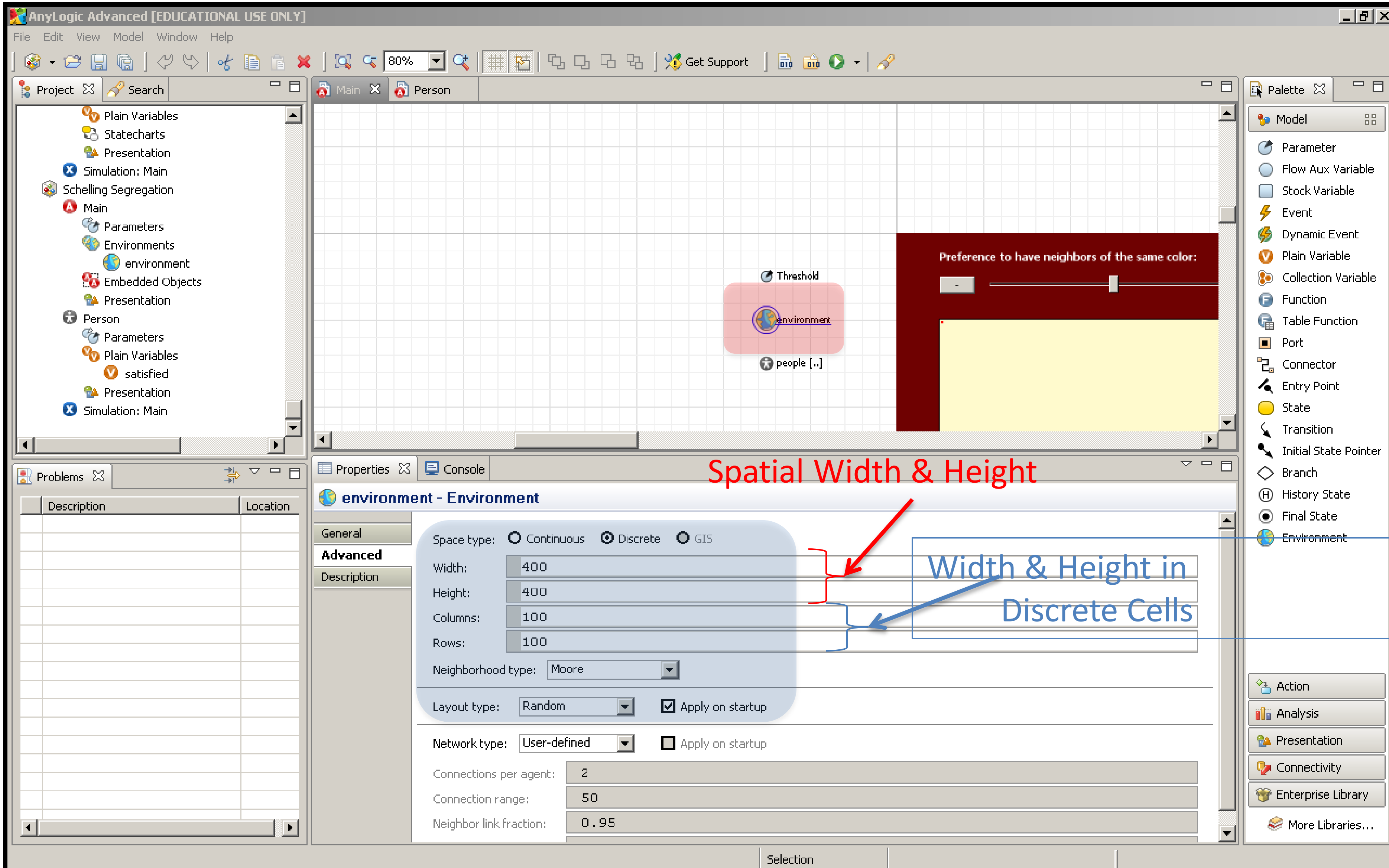
Hands on Model Use Ahead

Load model: Schelling Segregation.alp

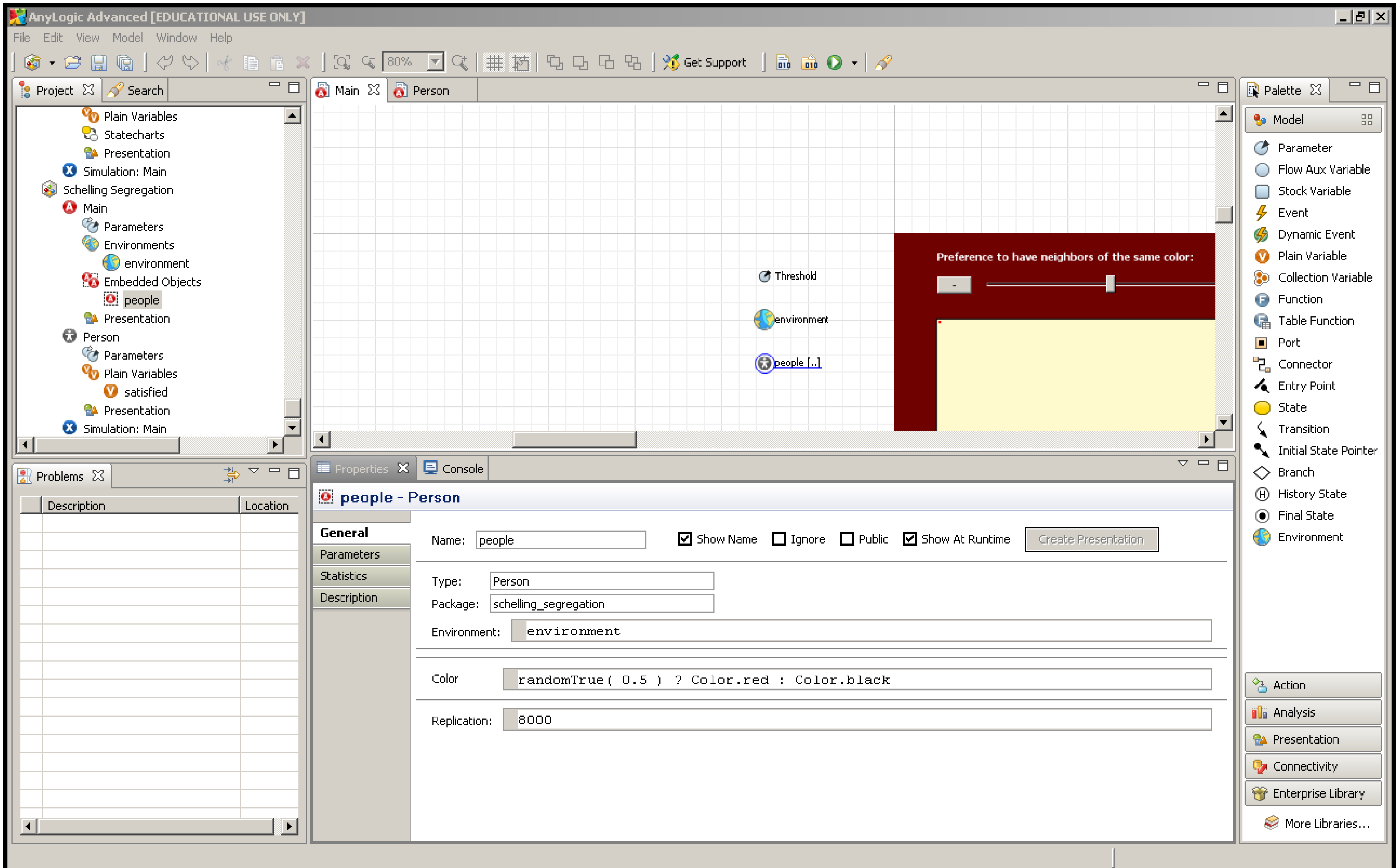
A Model to Examine the Emergence of Segregation



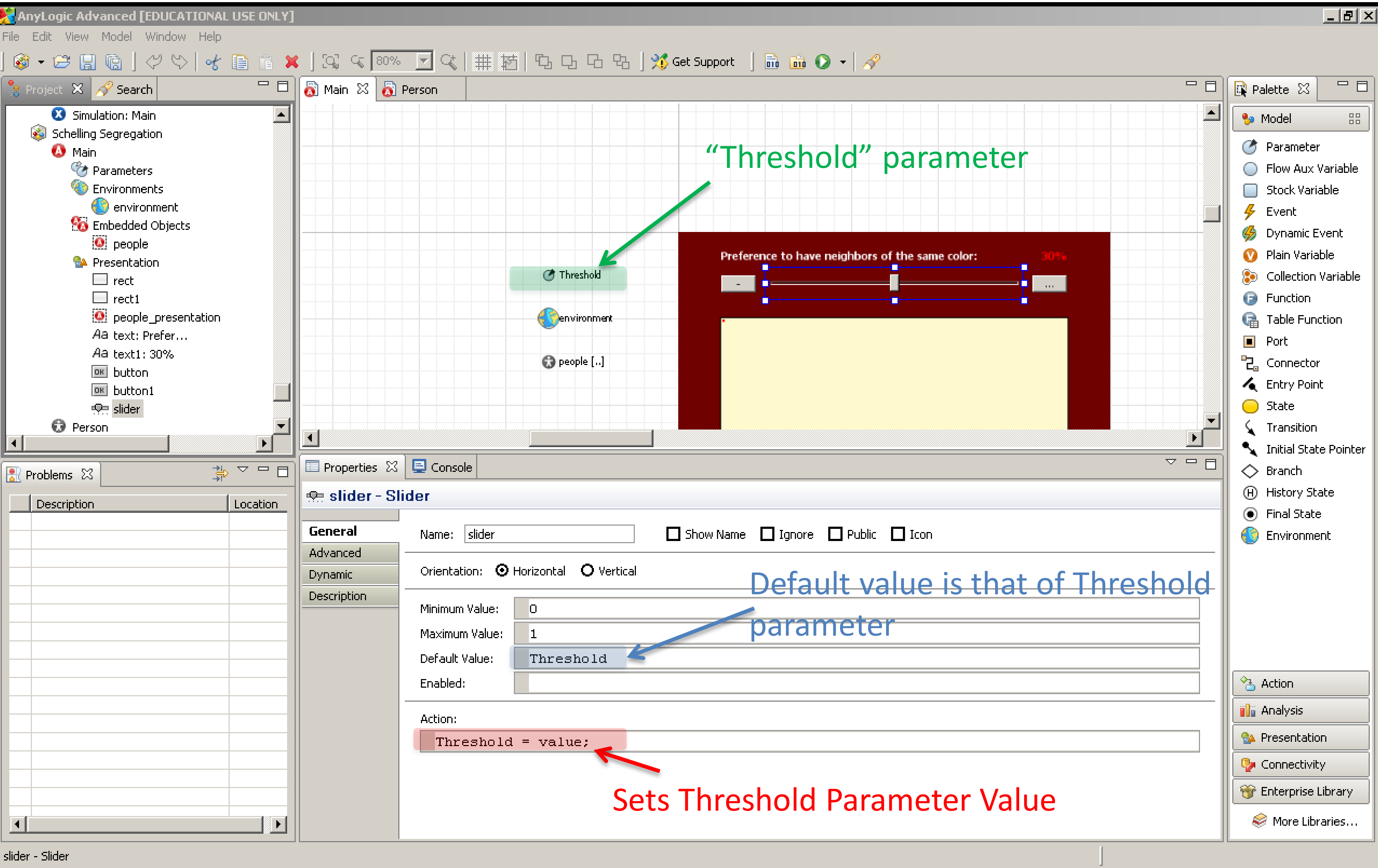
A Discrete Spatial Environment with Random Agent Positioning



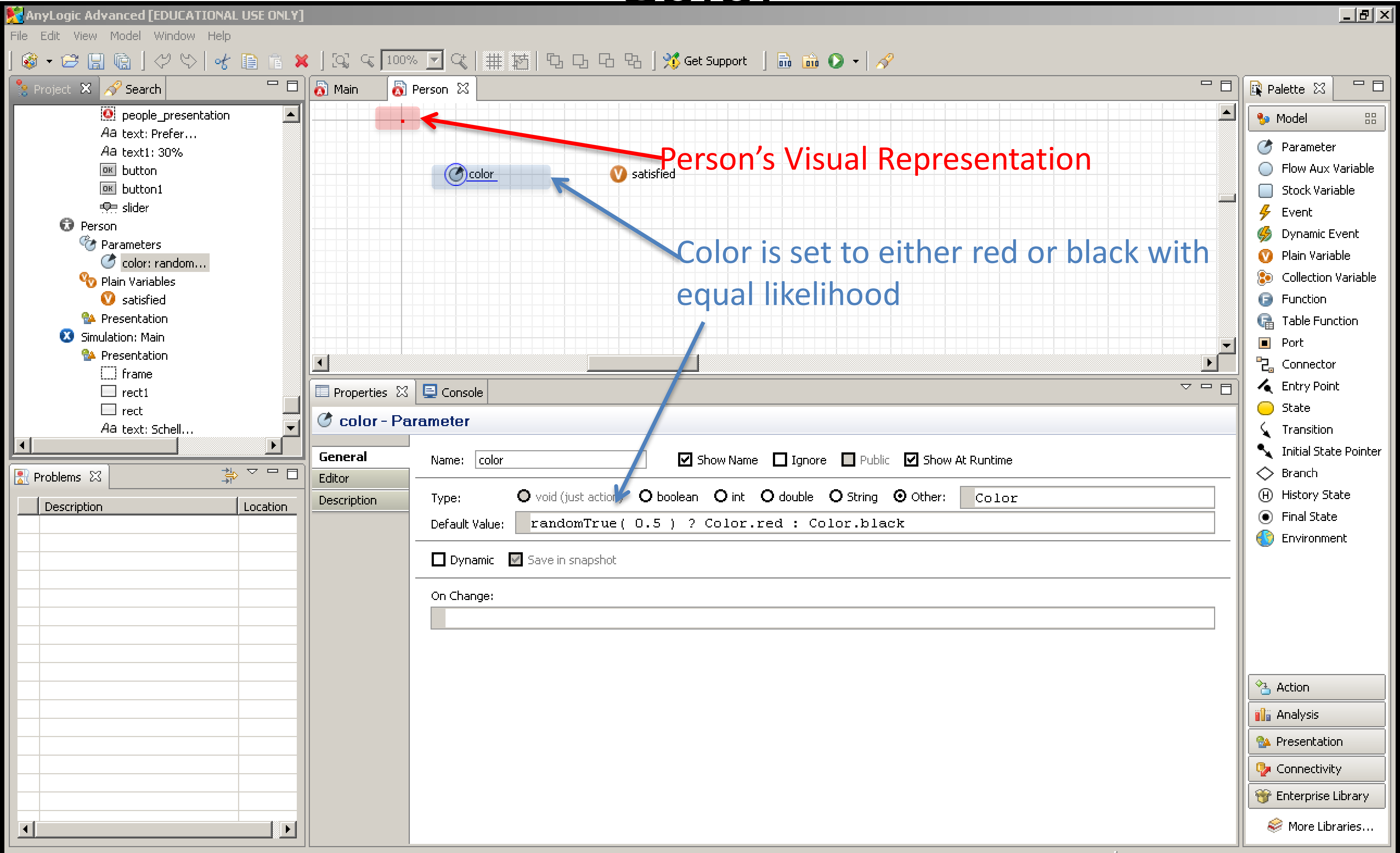
Population Dependence on the Population



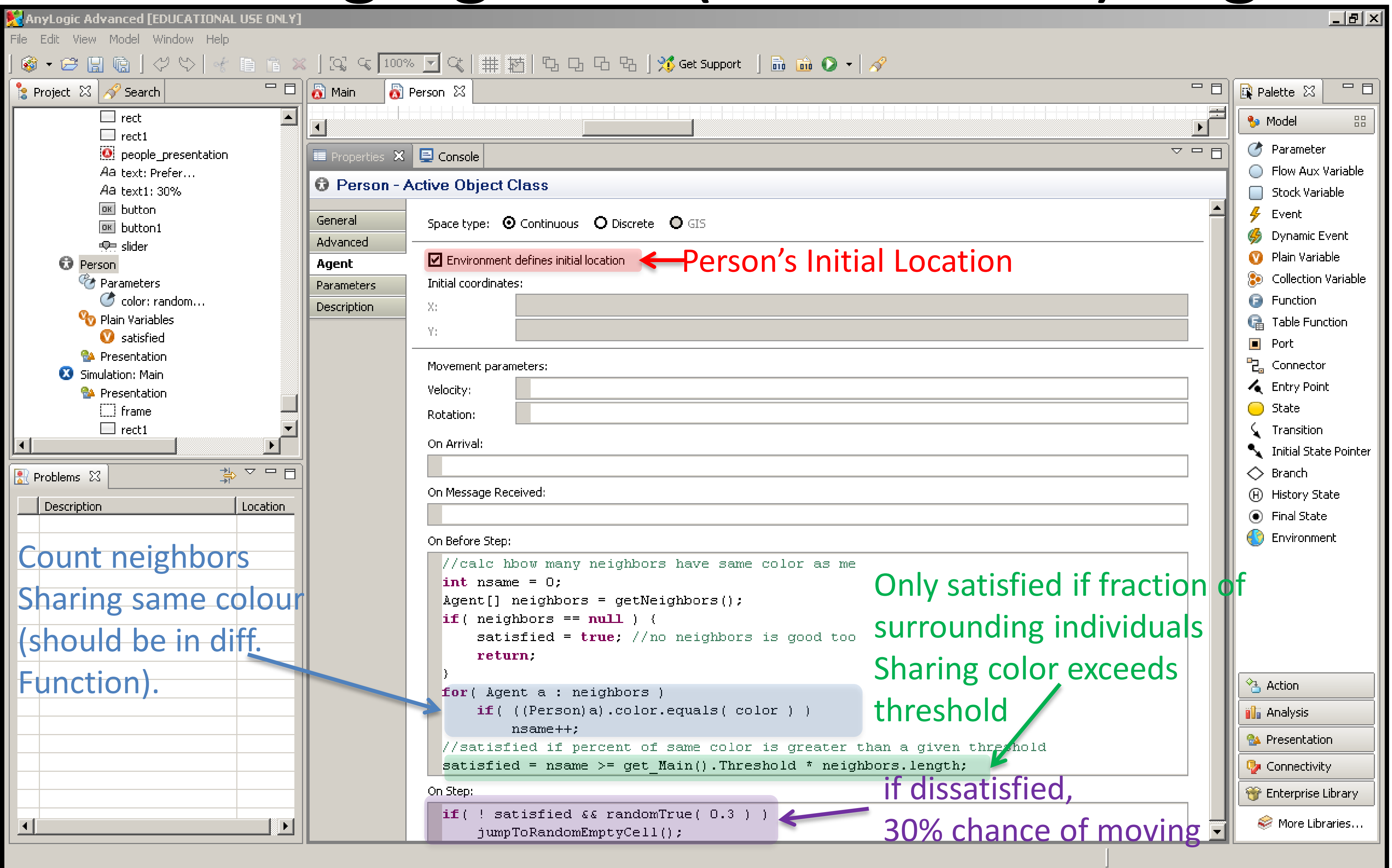
Slider Input Sets Parameter Value



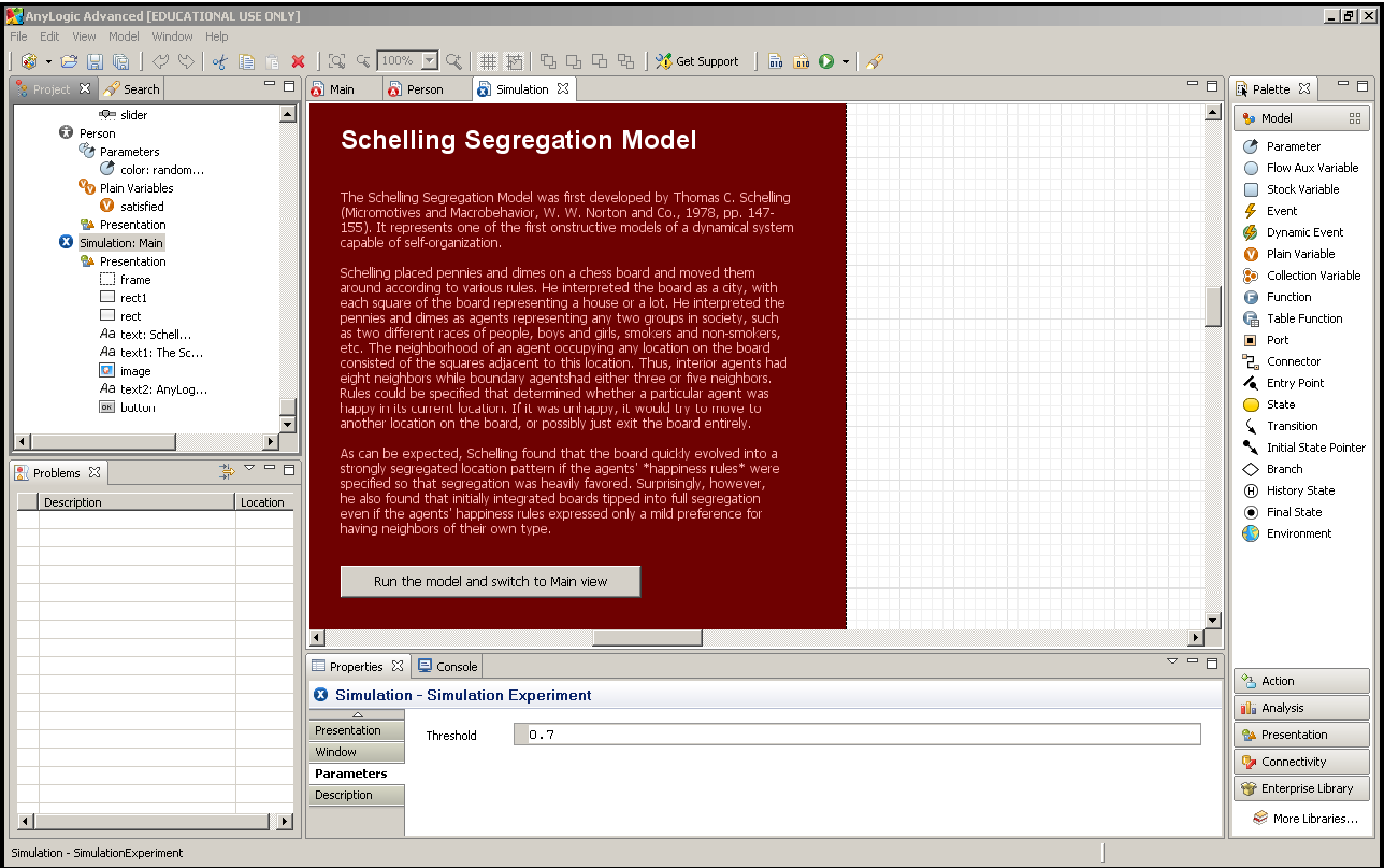
Person is Assigned a Randomly Picked Color



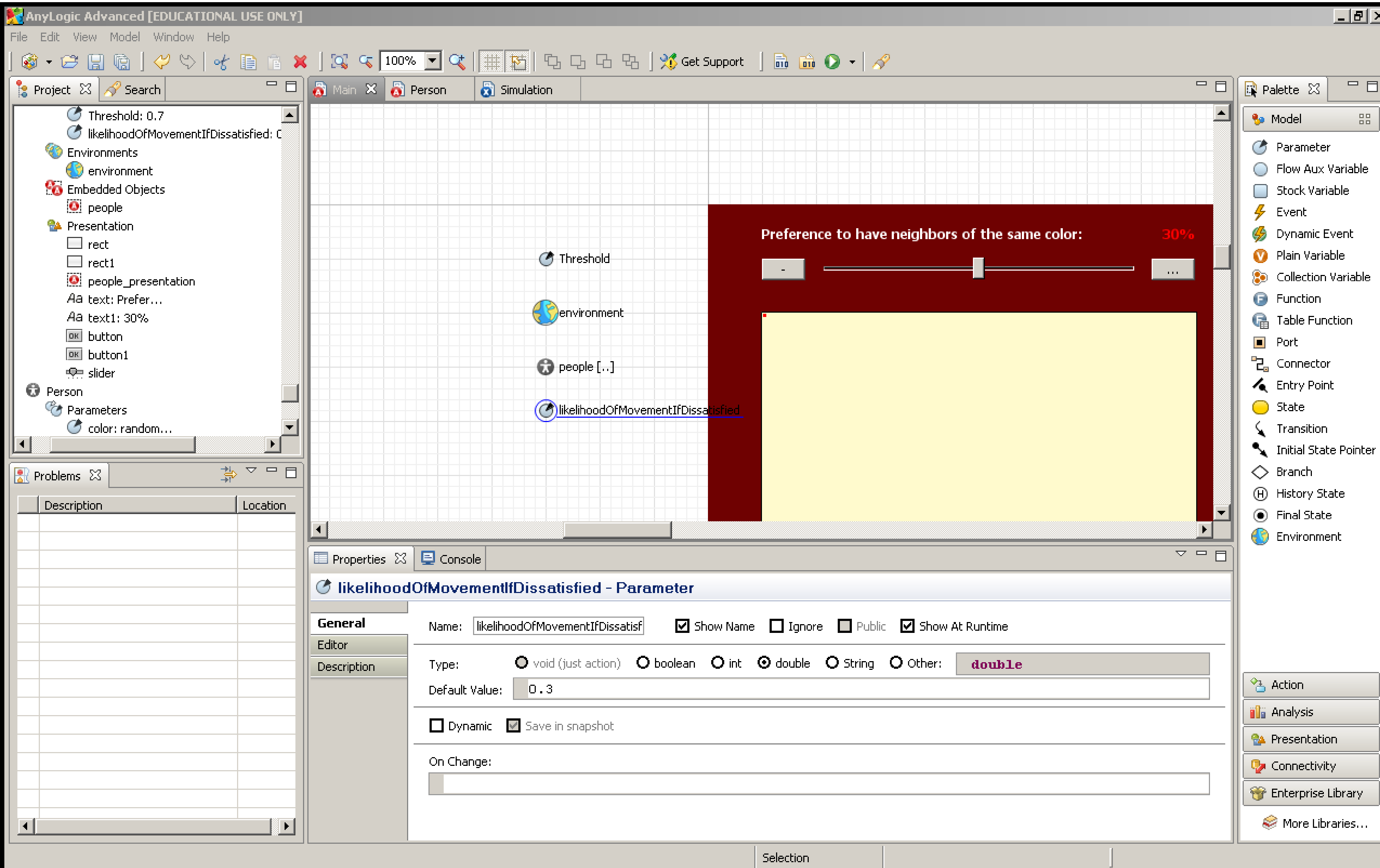
Core Segregation (Movement) Logic



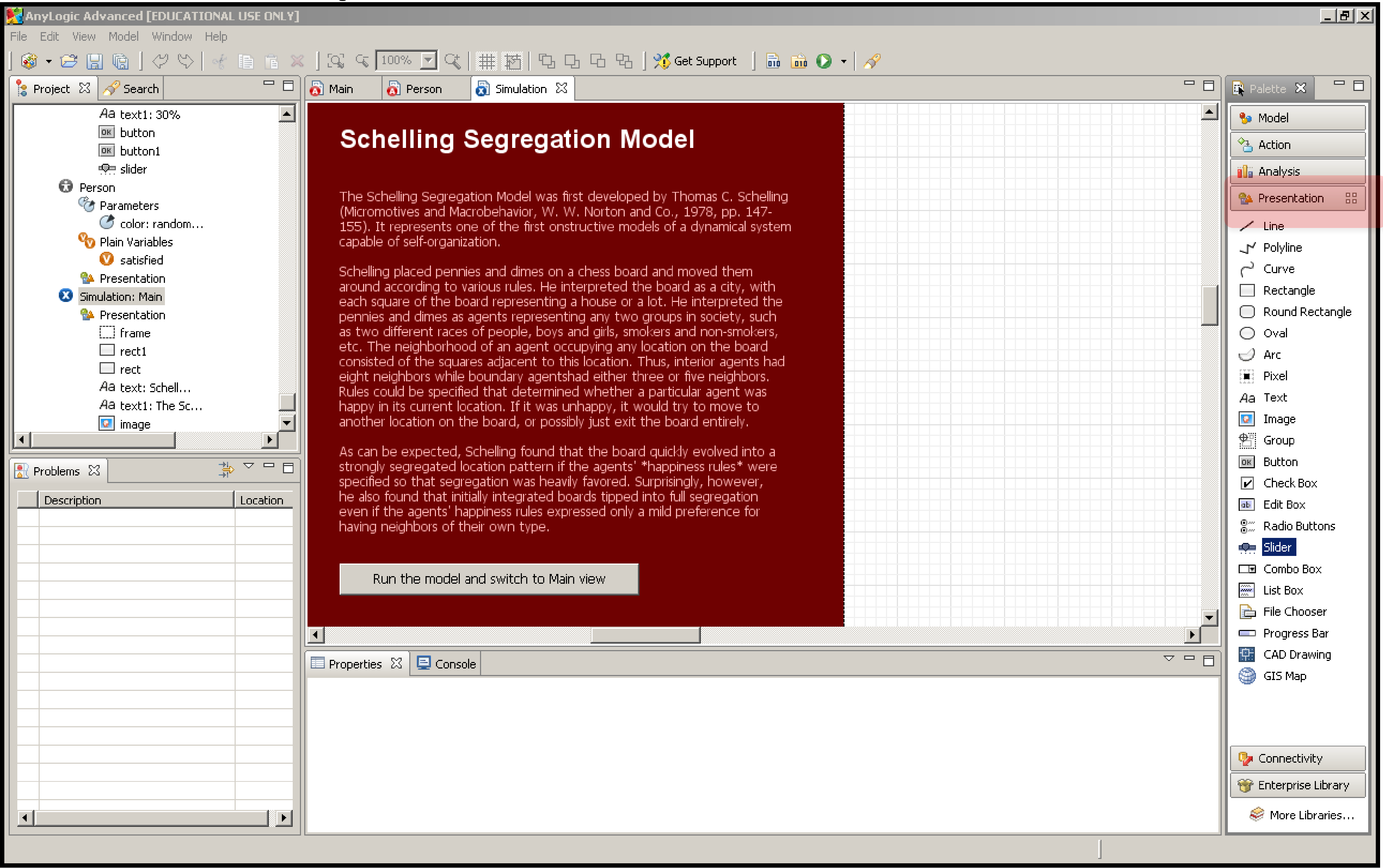
Experiment: Simulation Sets Parameter Assumptions



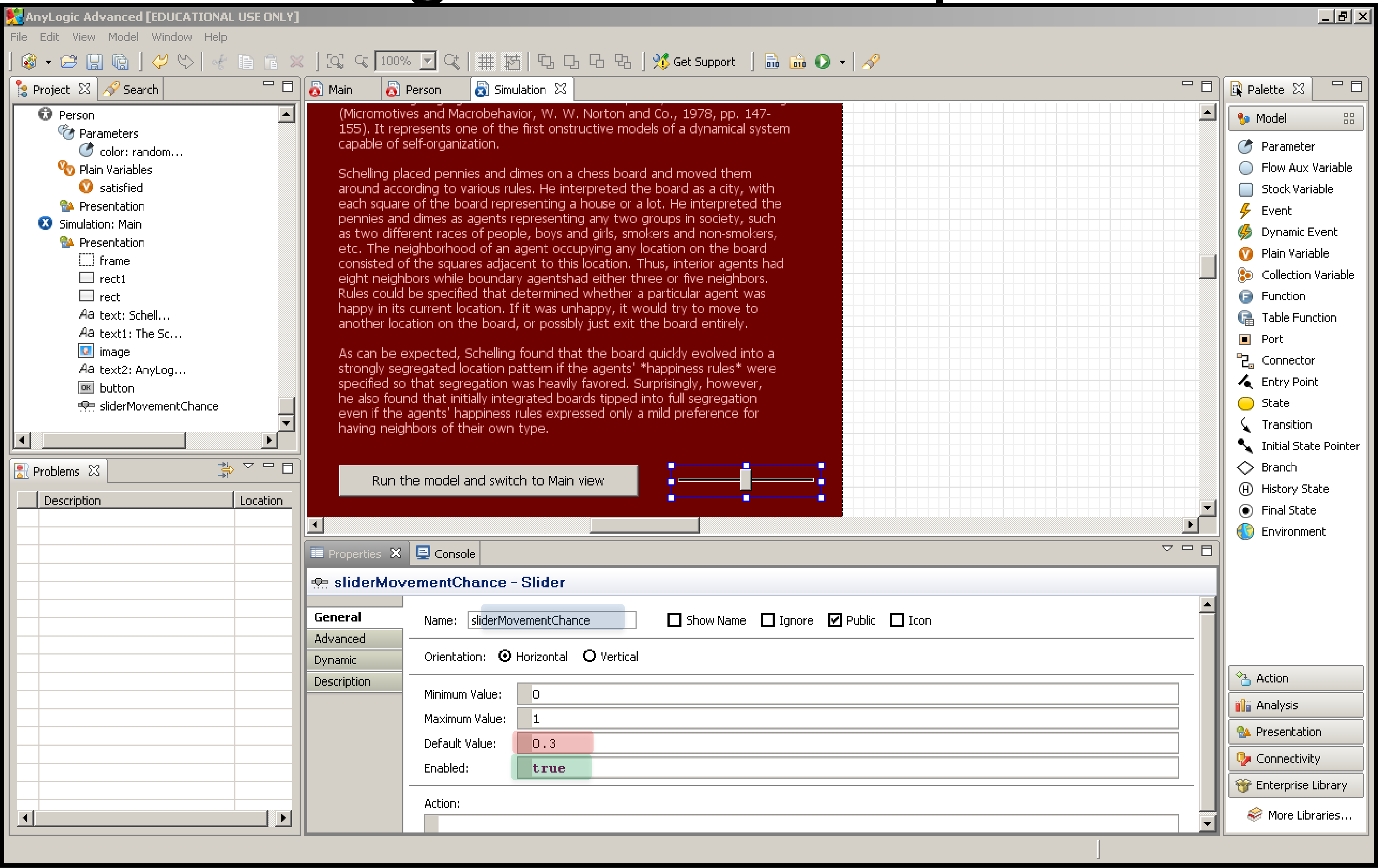
Add a Parameter to Main



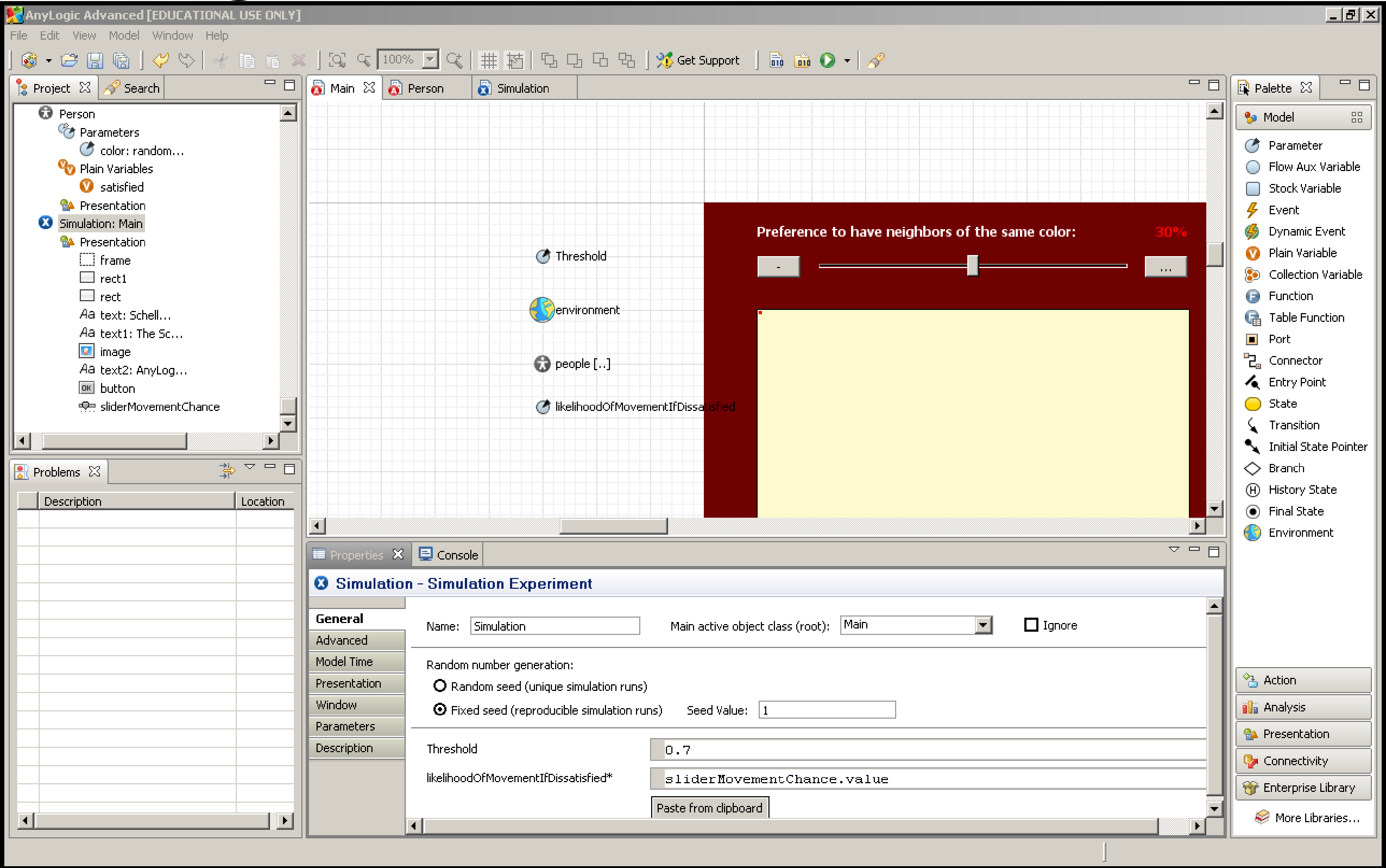
Experiment: Add a Slider!



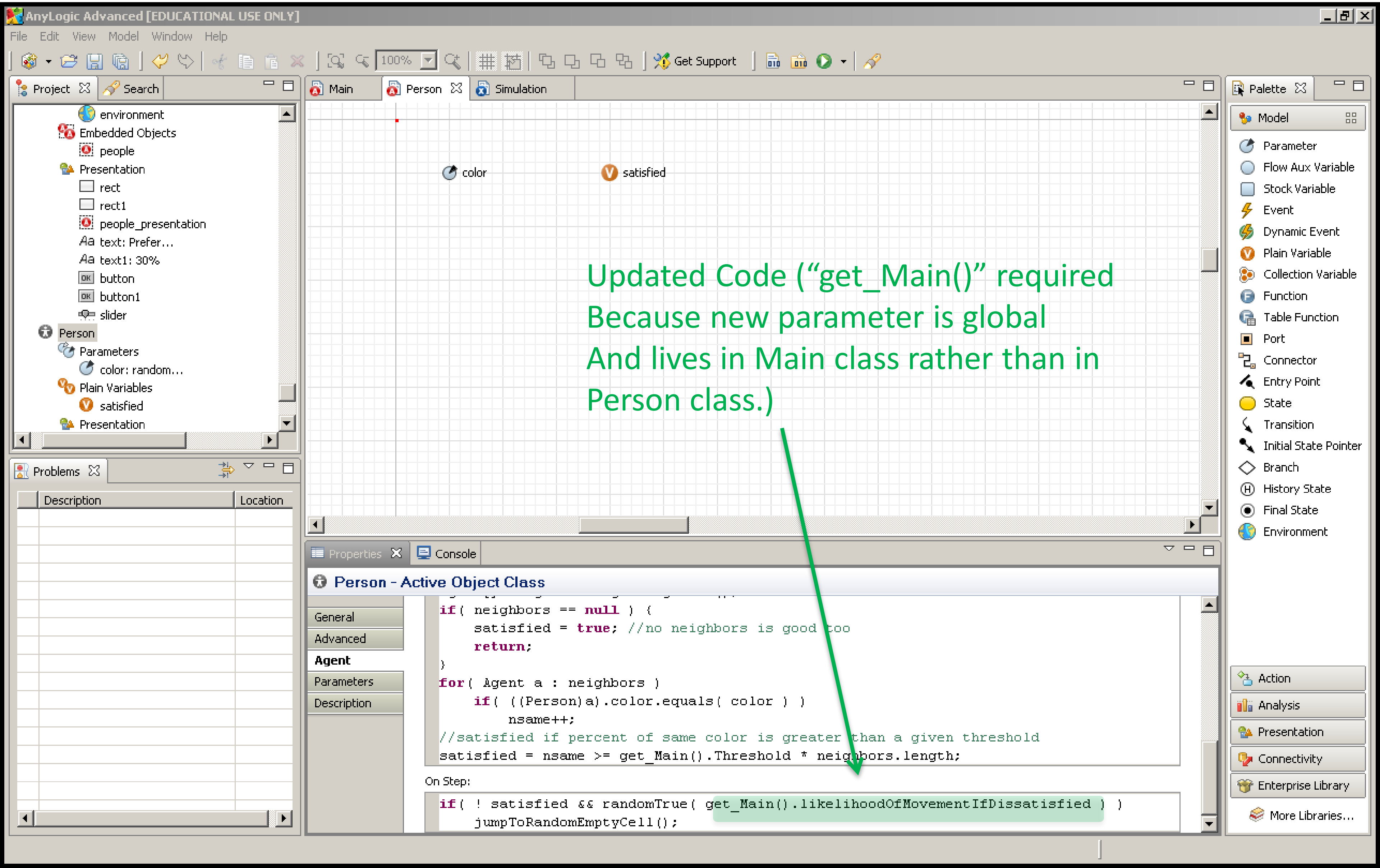
Setting the Slider Properties



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()

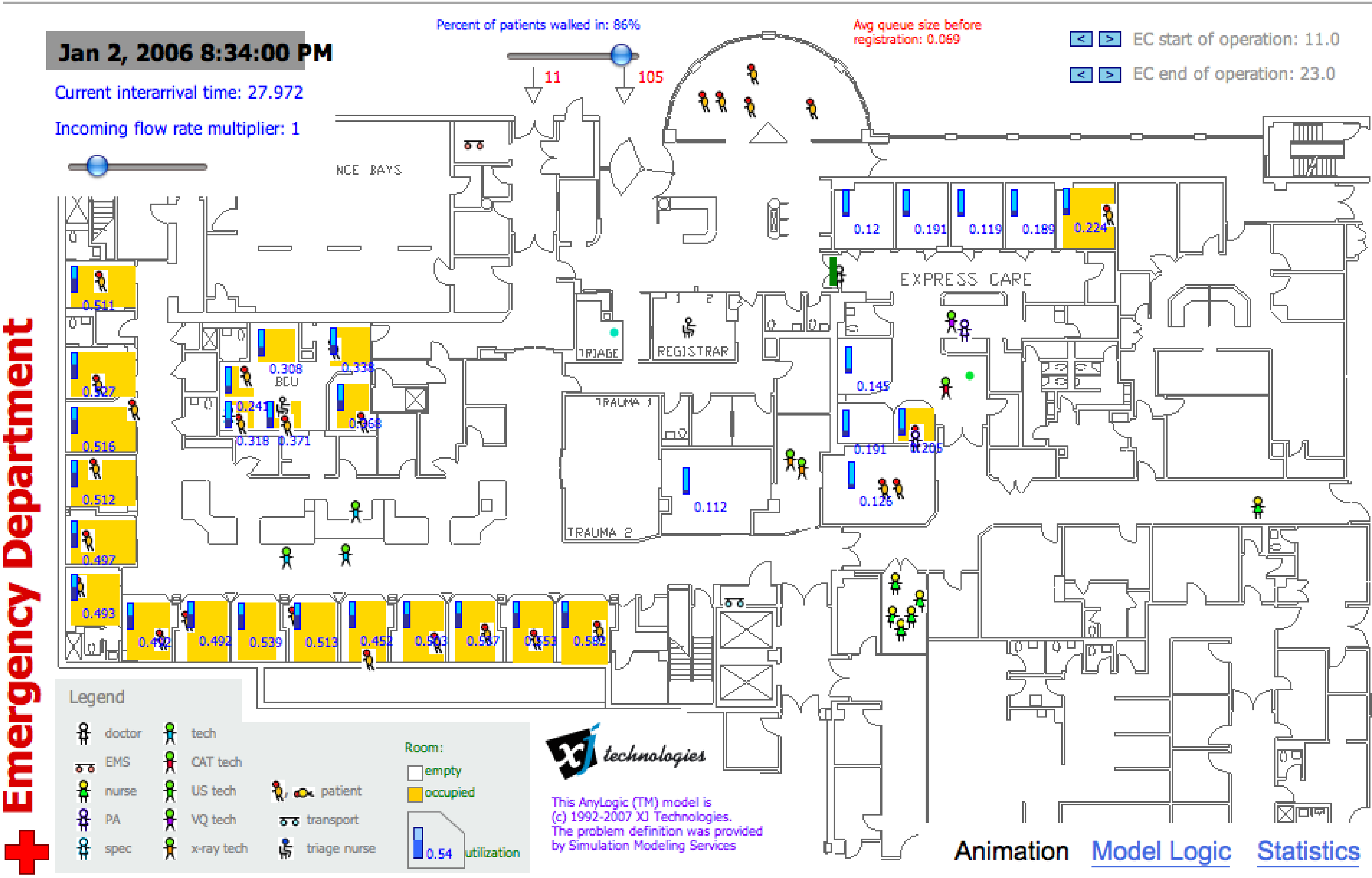
Important Distinction

- 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
  - A guest lecture with an Alzheimer's application will give a glimpse as to how this can be achieved