Agent Spatial Embedding & Movement in 2D Landscapes (Bonus: Discrete Time & some UI customization)

Nathaniel Osgood
MIT 15.879

March 23, 2012

Lecture Outline

- AnyLogic's Spatial embedding types
  - Overview
  - Reminder of continuous space
  - A glimpse of a discrete space & discrete time model
- Agent Mobility

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

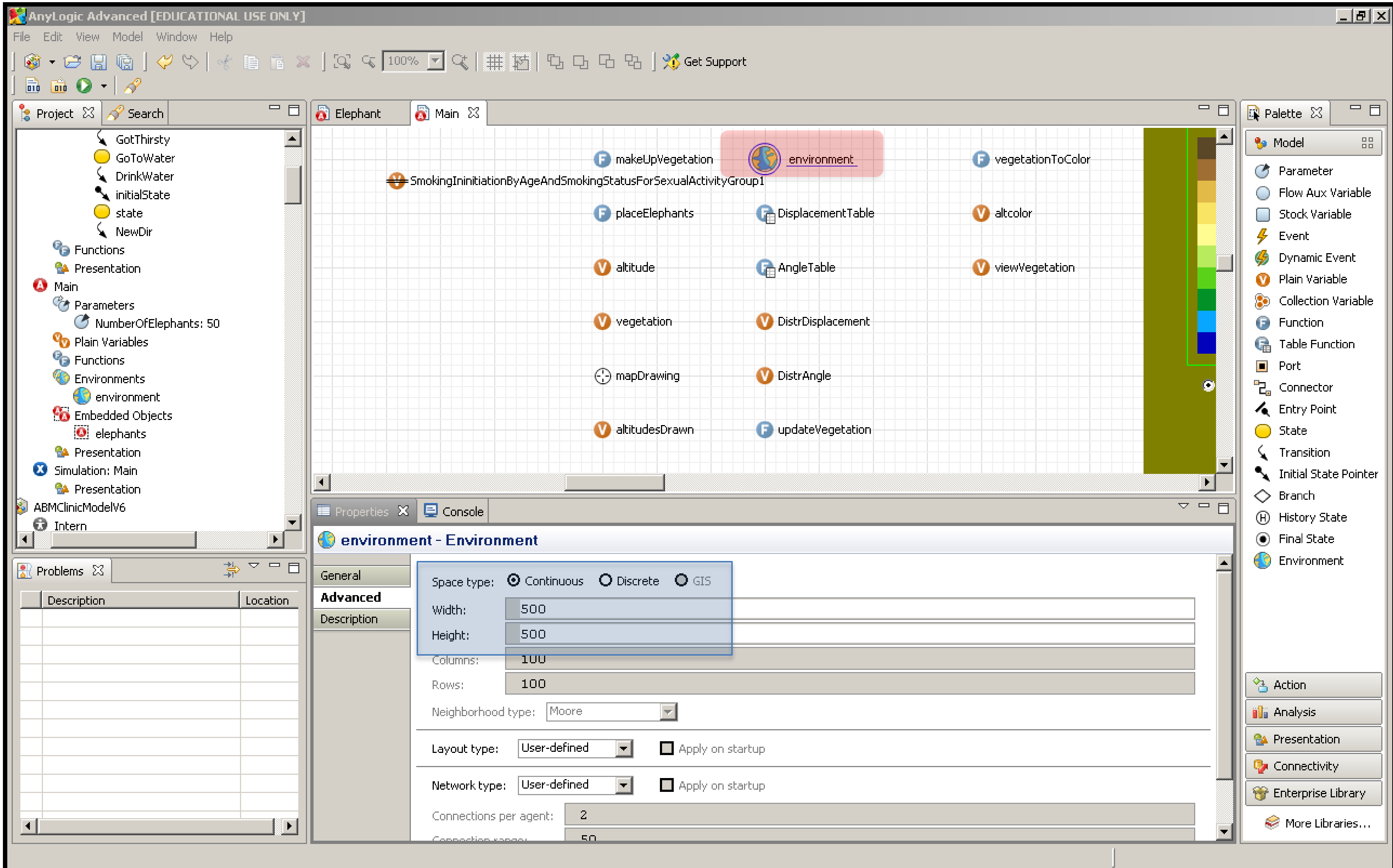
The Locus of Control: Environment

- 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}

Lecture Outline

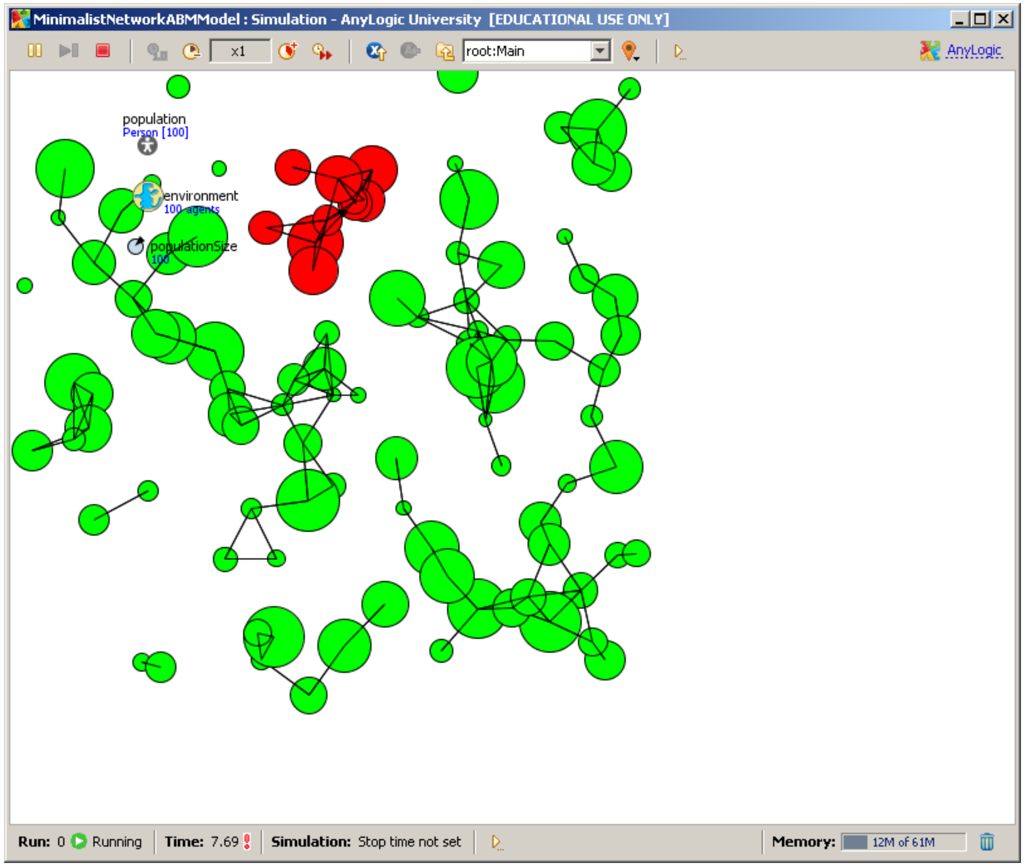
- AnyLogic's Spatial embedding types
  - √ Overview
  - – Reminder of continuous space
  - – A glimpse of a discrete space & discrete time model
- Agent Mobility

Continuous Environment



Continuous Environment: Your Model

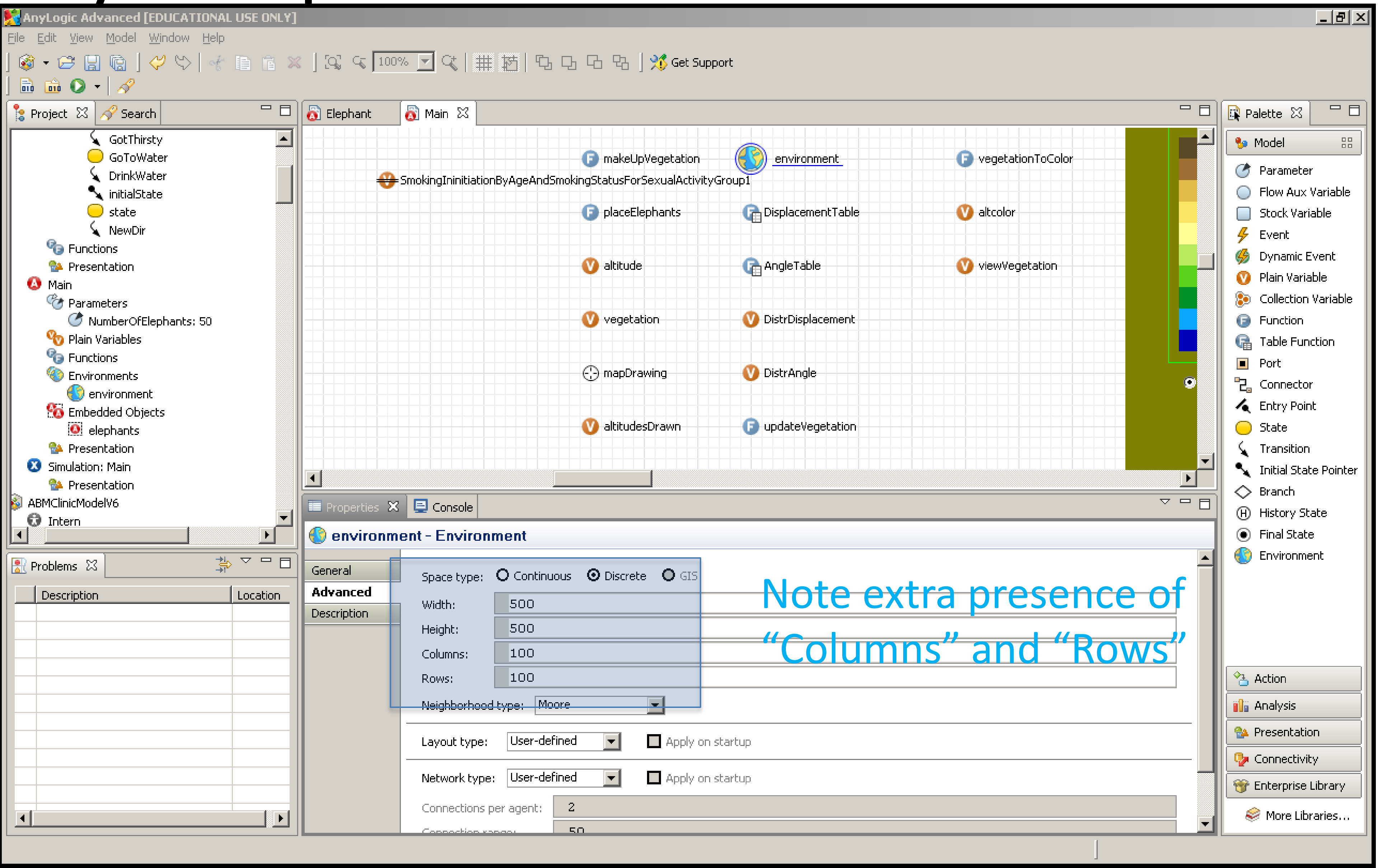
- We've already seen the continuous embedding in our built up model.



Lecture Outline

- AnyLogic's Spatial embedding types
  - √ Overview
  - √ Reminder of continuous space
  - – A glimpse of a discrete space & discrete time model
- Agent Mobility

By Comparison: Discrete Environment



Hands on Model Use Ahead

Load AnyLogic Sample Model: The Game of Life

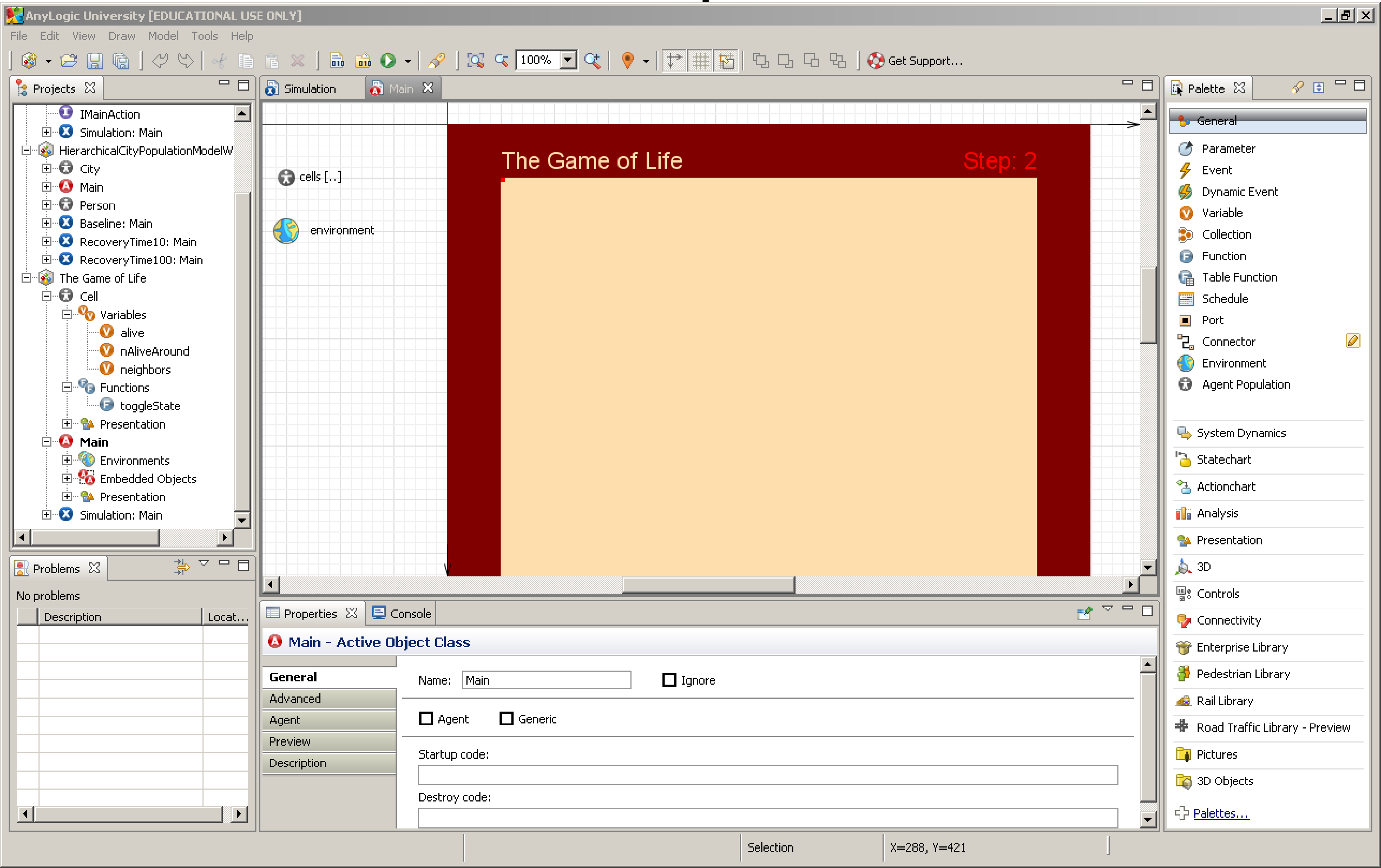
The "Game" of Life: Background

- Invented in 1970 by Mathematician Conway (modifying ideas from Von Neumann)
- Inspiration: Lifecourse of cells
  - Key dichotomy: A space contains a living element or not
  - Stylized rules for birth, death
- Cellular automaton: Uses Discrete Time (Steps) & Discrete Space (Cells) with evolving cell state
- Deterministic rules
- Illustrates the emergence of tremendous complexity from very simple rules
  - Computationally universal

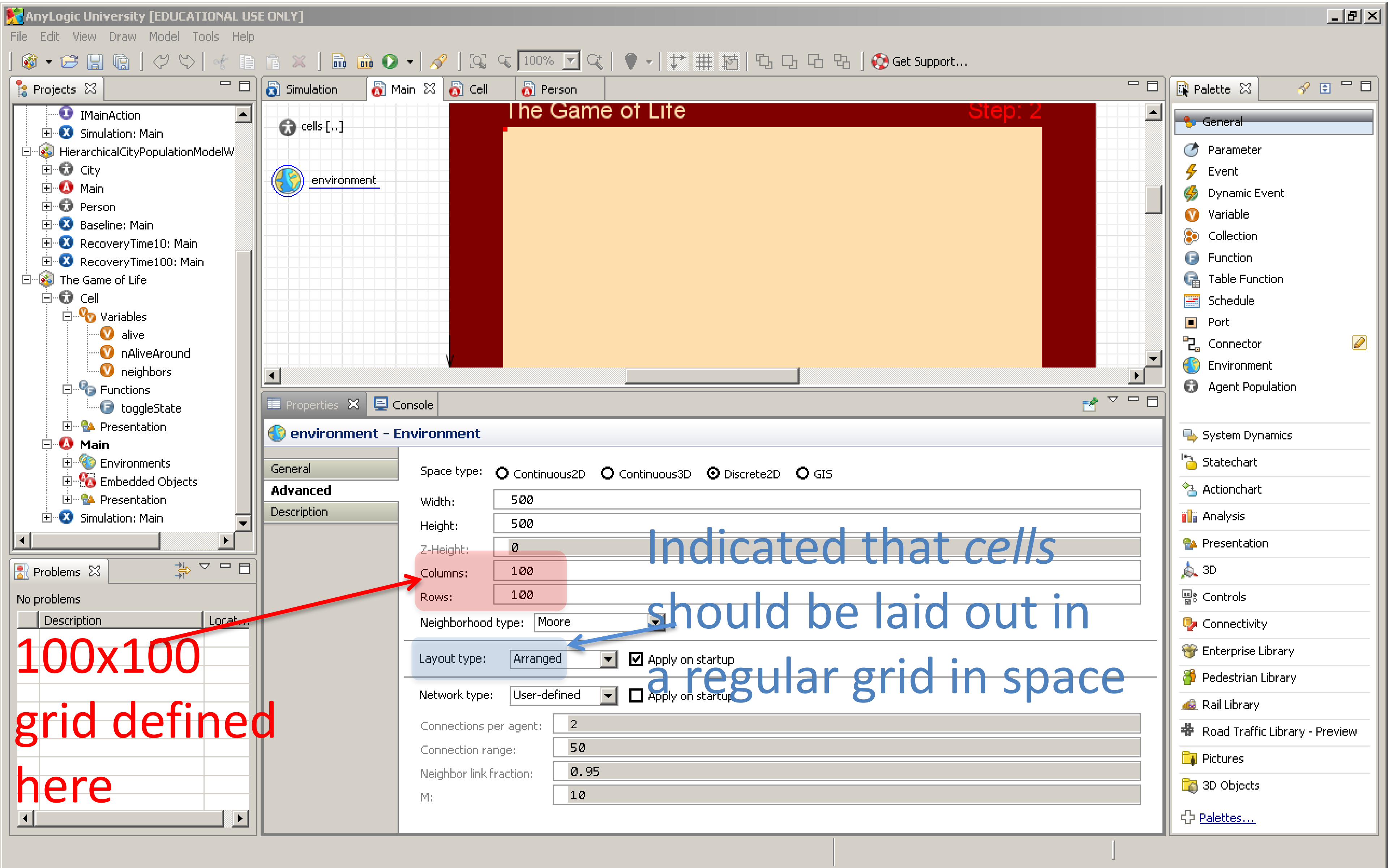
The Behavioral Rules of the Game of Life

- Cells are viewed as surrounded by 4 neighbors (in cardinal directions)
- Living cells require some neighboring empty space, but also some proximity to nearby living cells
- Birth: An empty cell becomes occupied if it has an "ideal" nurturing environment around it (3 surrounding cells)
- An existing cell dies if
  - Too isolated: It has too few neighbors (1 or 0)
  - Too crowded: It is surrounded by other cells (4 surrounding cells)
- No mobility: Cells are born, live and die in same location

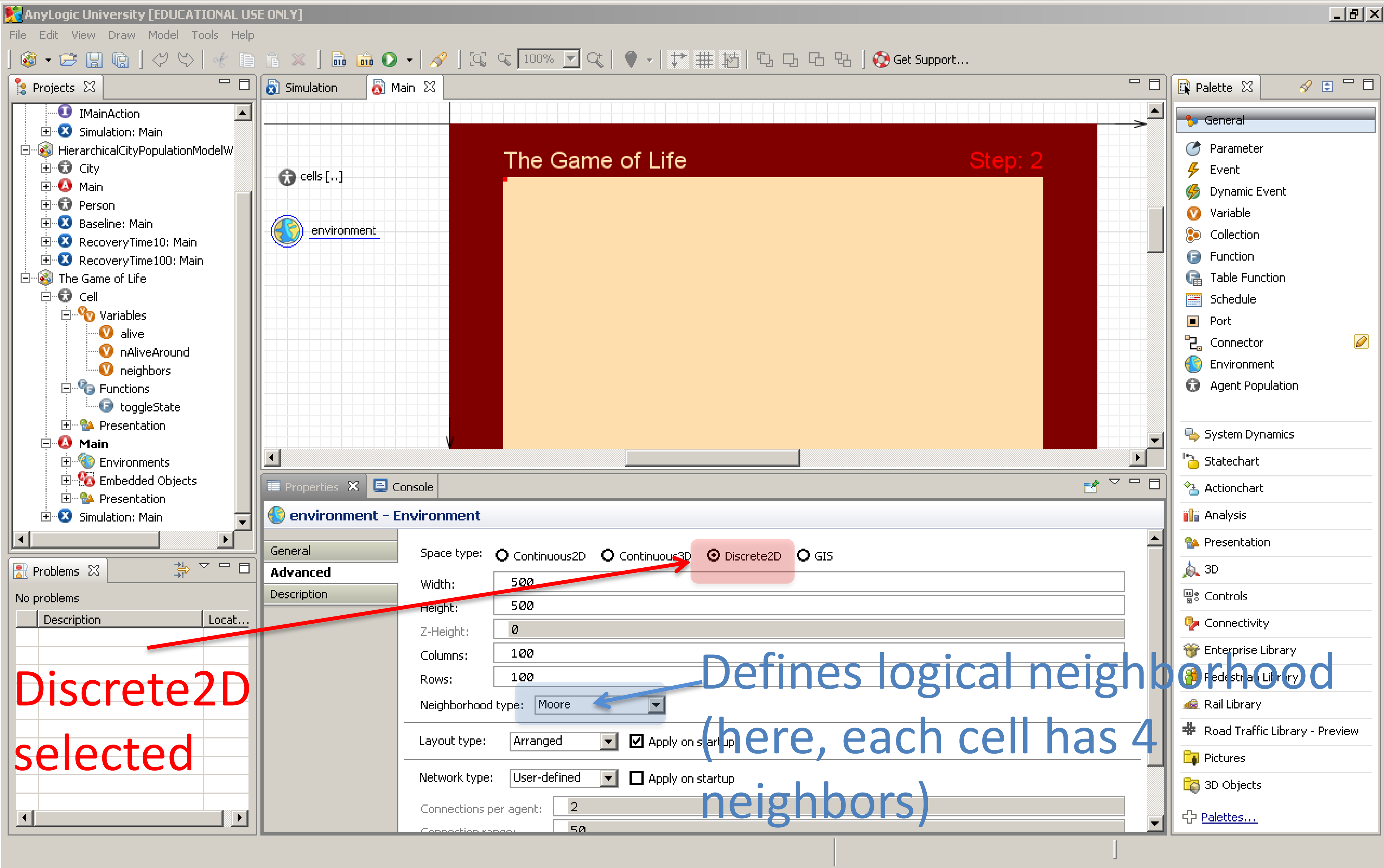
Open "Main" Class
Scroll Left to See Population & Environ.



Imposing the Regular 2D Structure



Environment: Enabling Discrete **Space** (Cells)

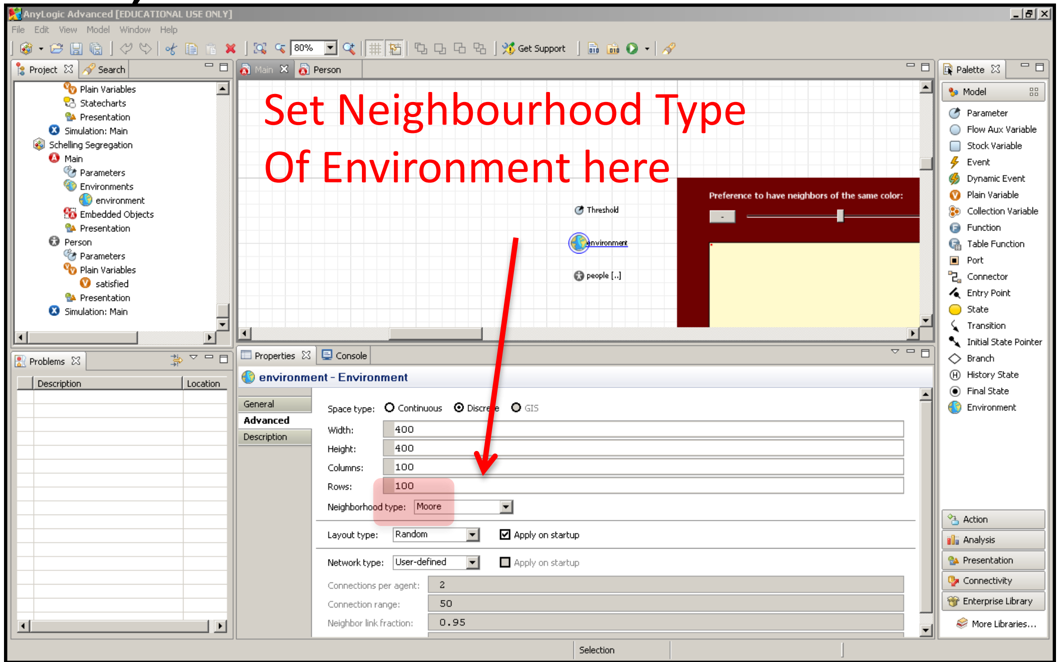


Discrete2D selected

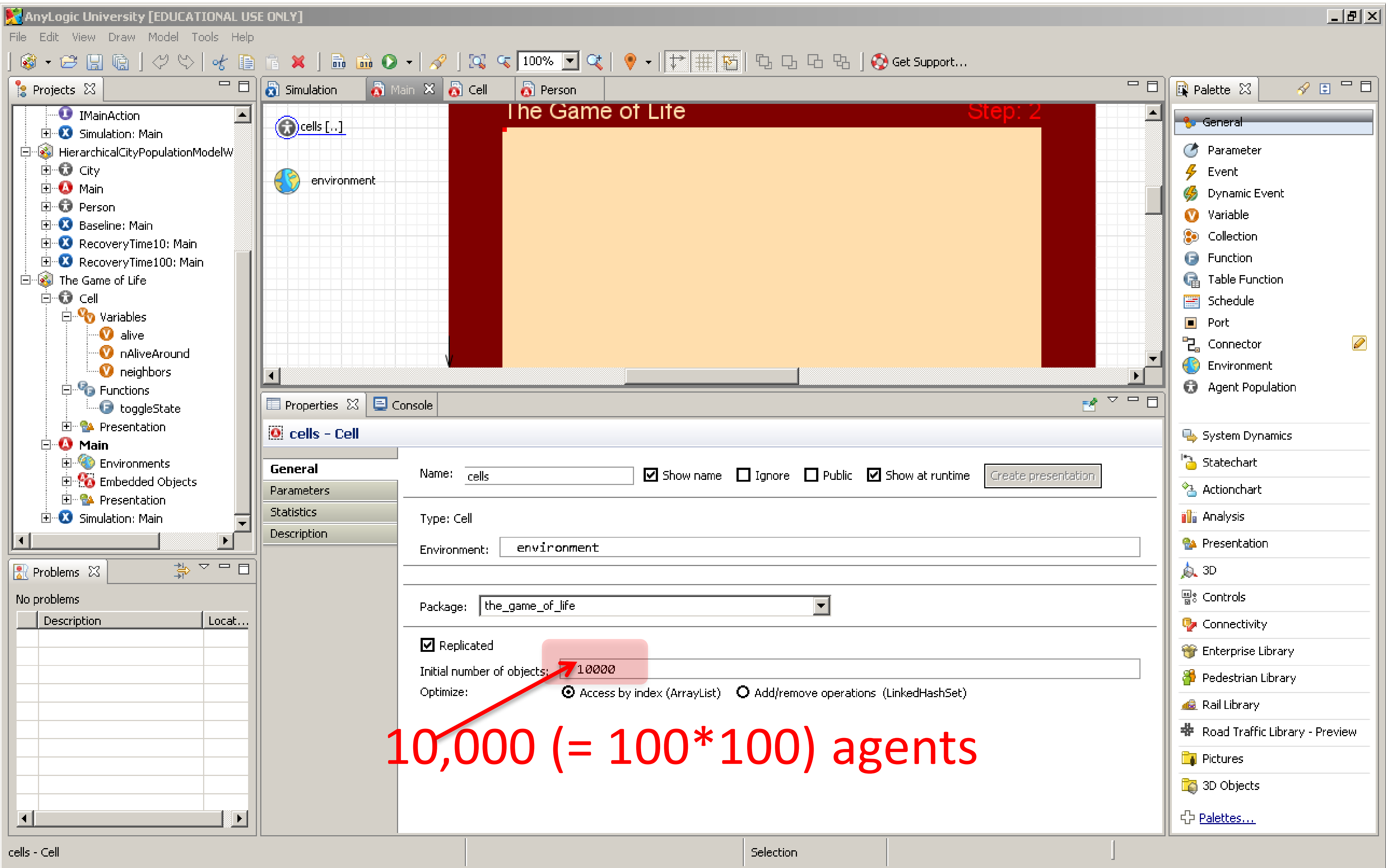
Defines logical neighborhood (here, each cell has 4 neighbors)

Neigbourhood Models

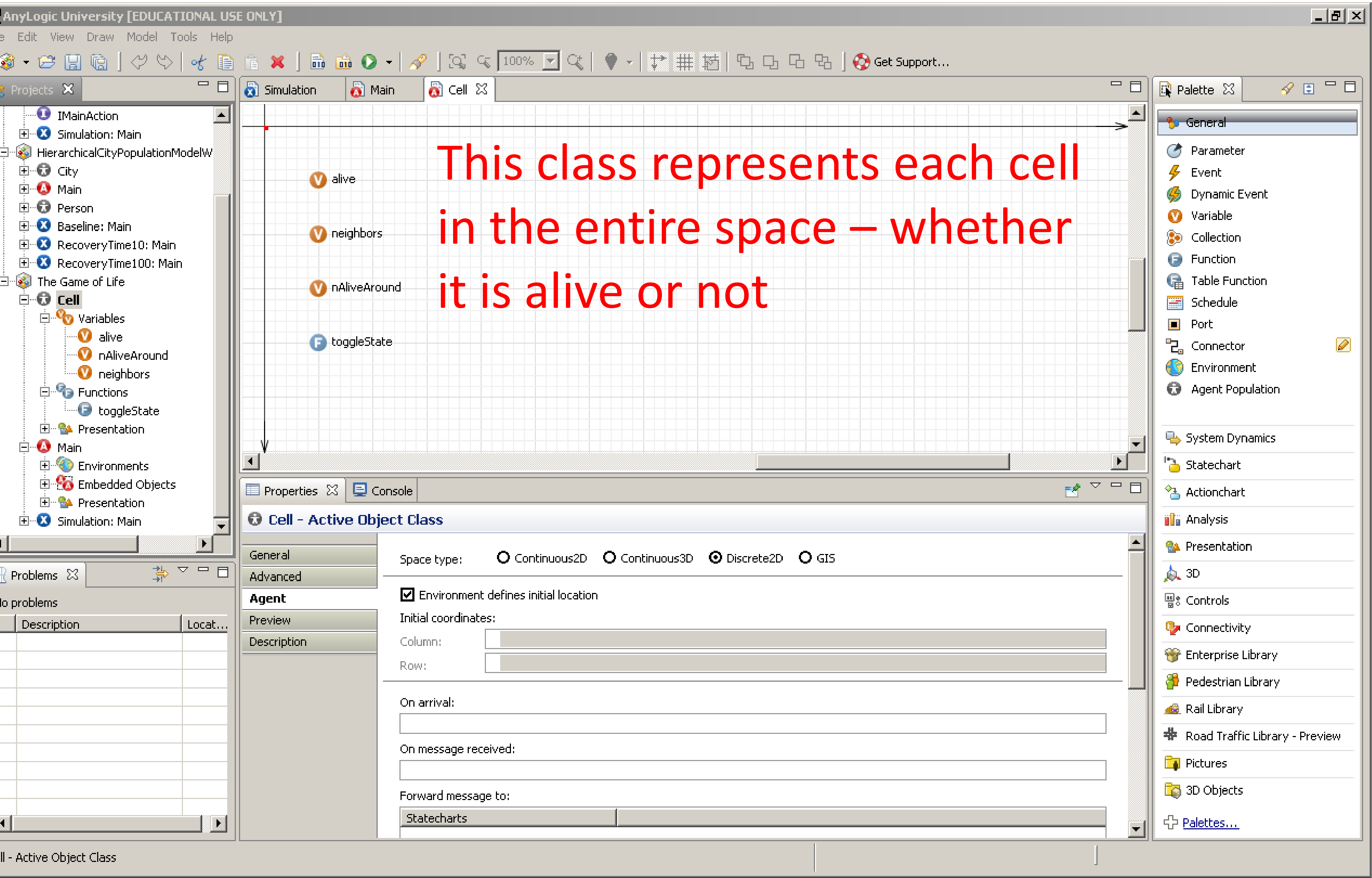
- Moore: Cardinal directions
  - NORTH,SOUTH,EAST, WEST
- Euclidean
  - NORTH, SOUTH, EAST, WEST, NORTHEAST, NORTHWEST, SOUTHEST,SOUTHWEST



Population: One Cell Agent per Grid Point

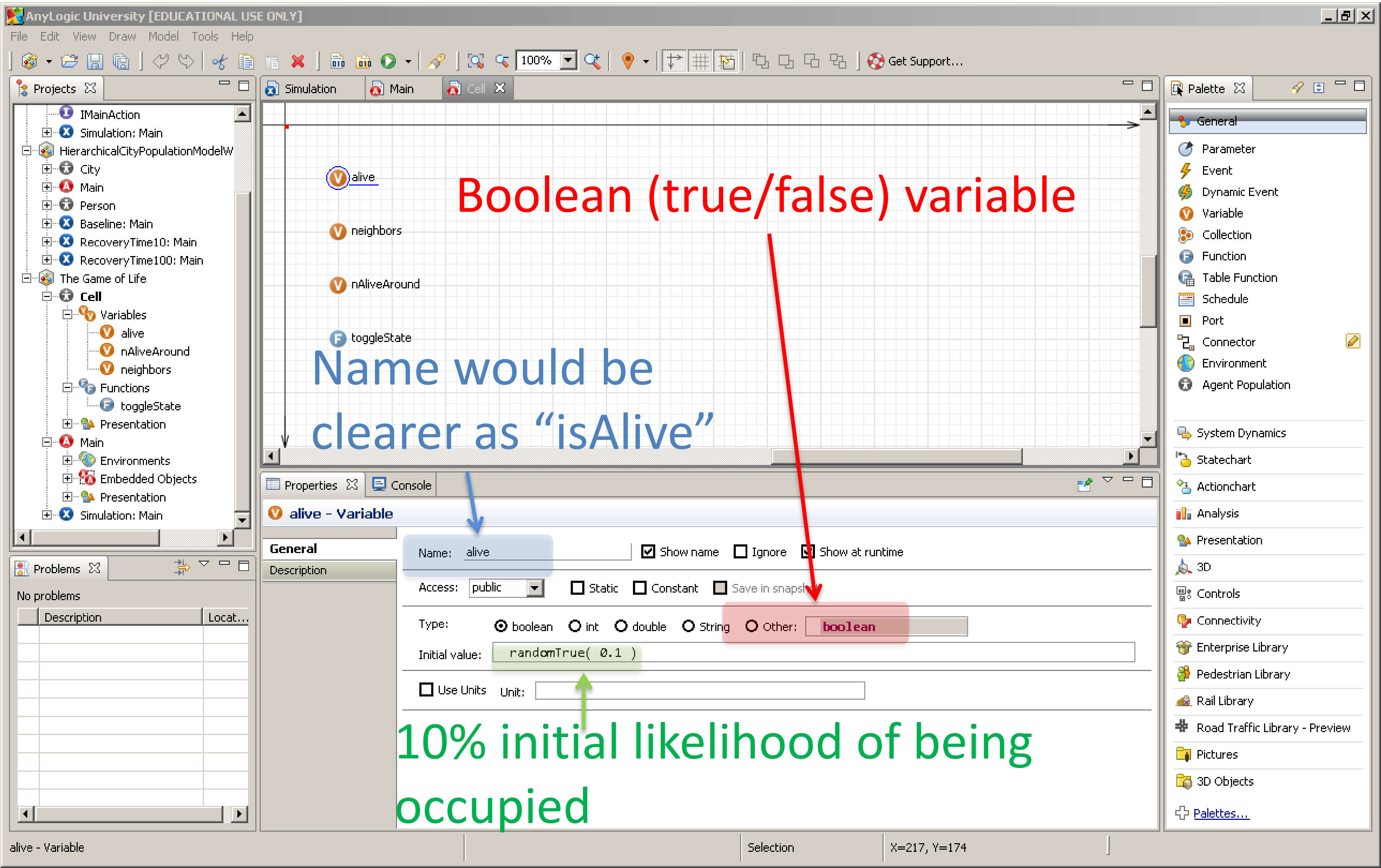


View the “Cell” Class

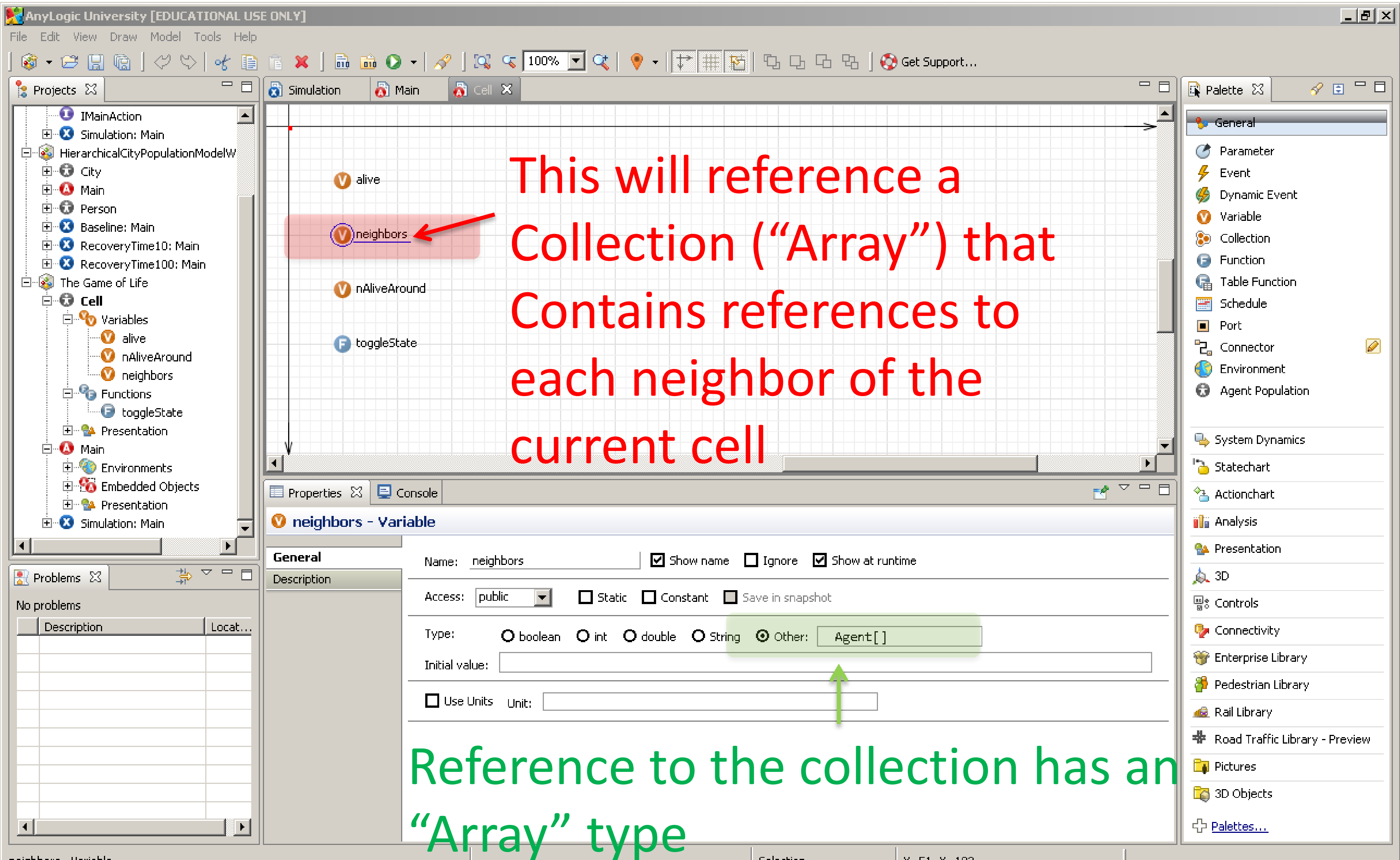


This class represents each cell in the entire space – whether it is alive or not

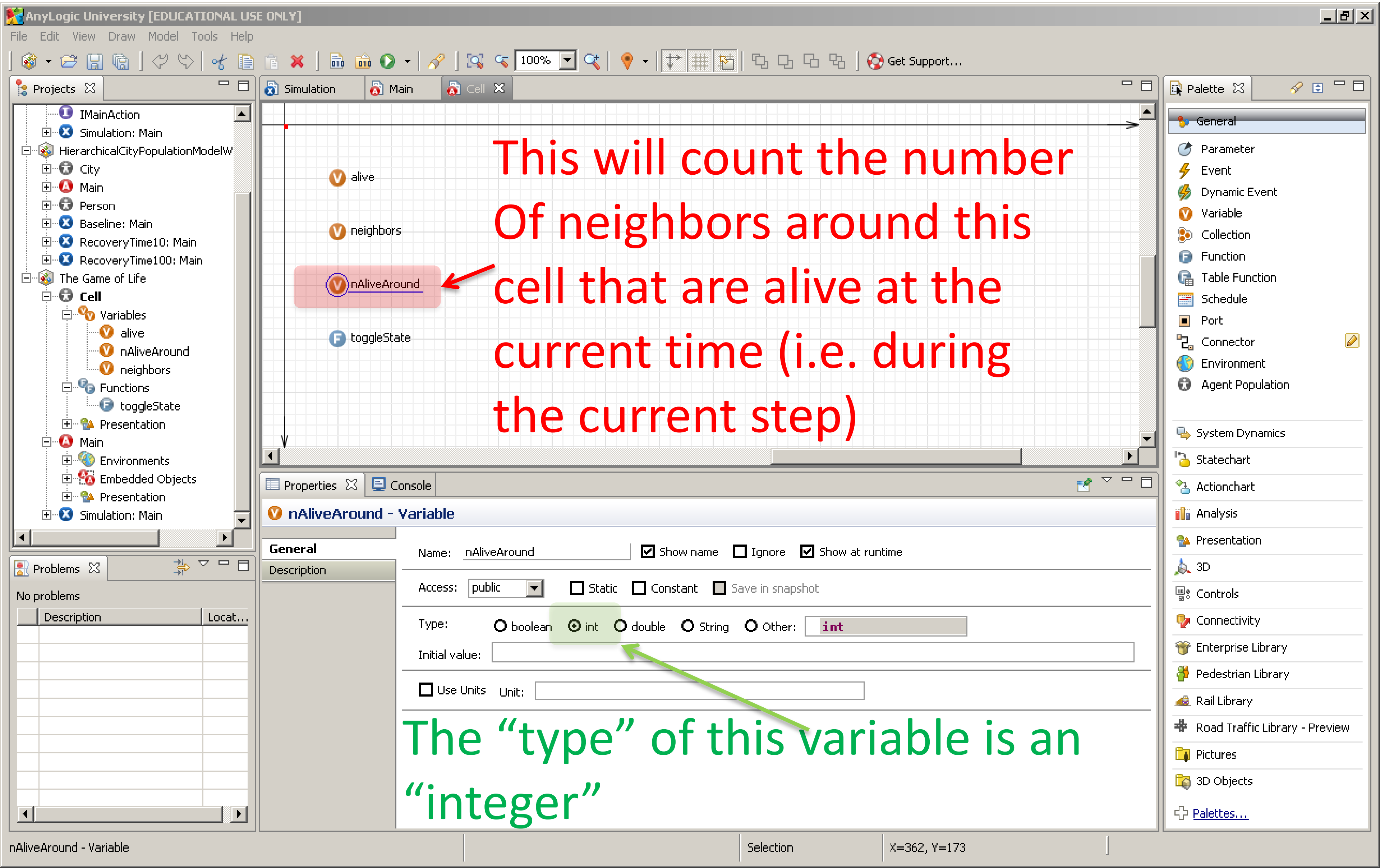
Cell Variables: “alive”



Cell Variables: “neighbors”



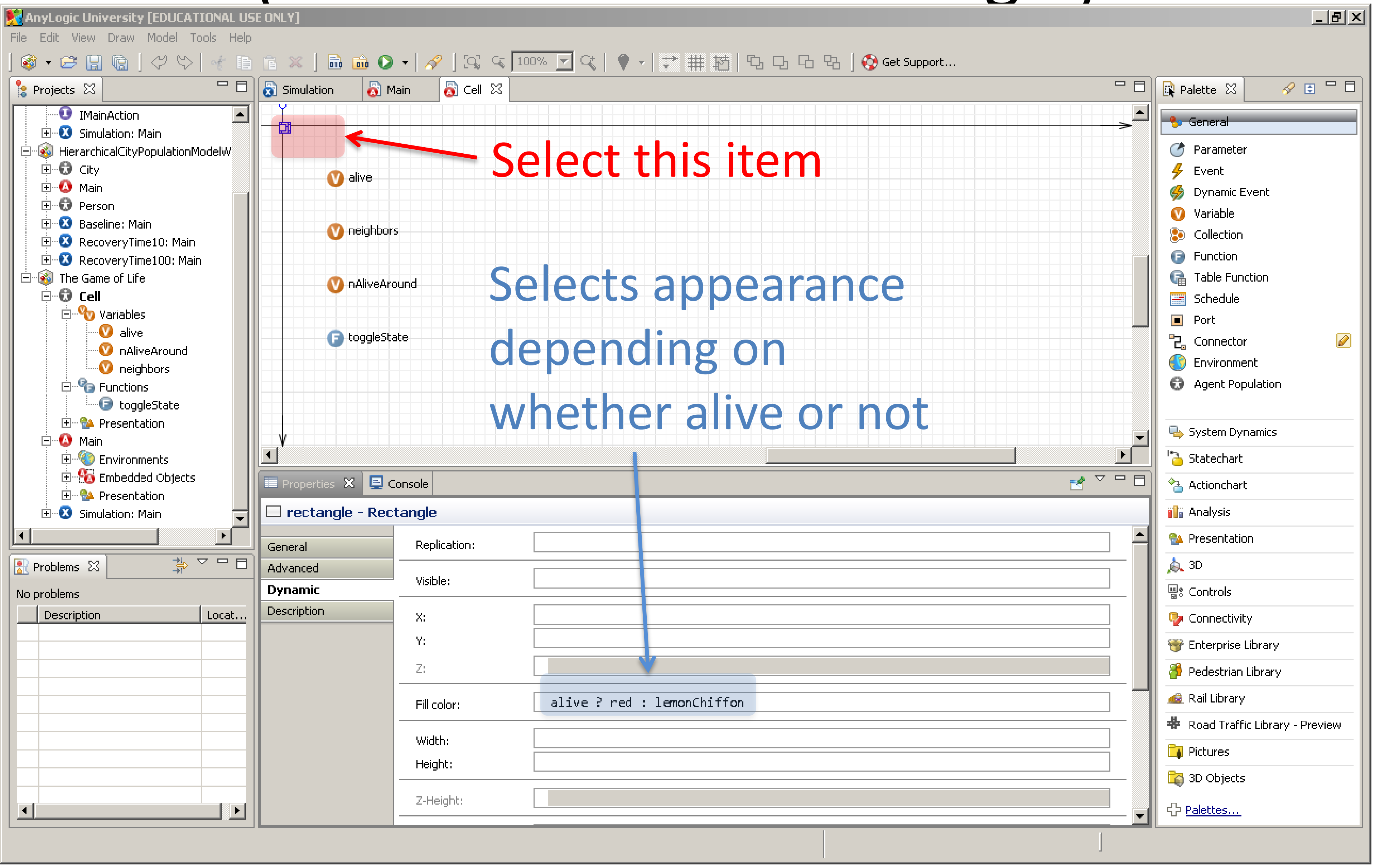
Cell Variables: “nAliveAround”



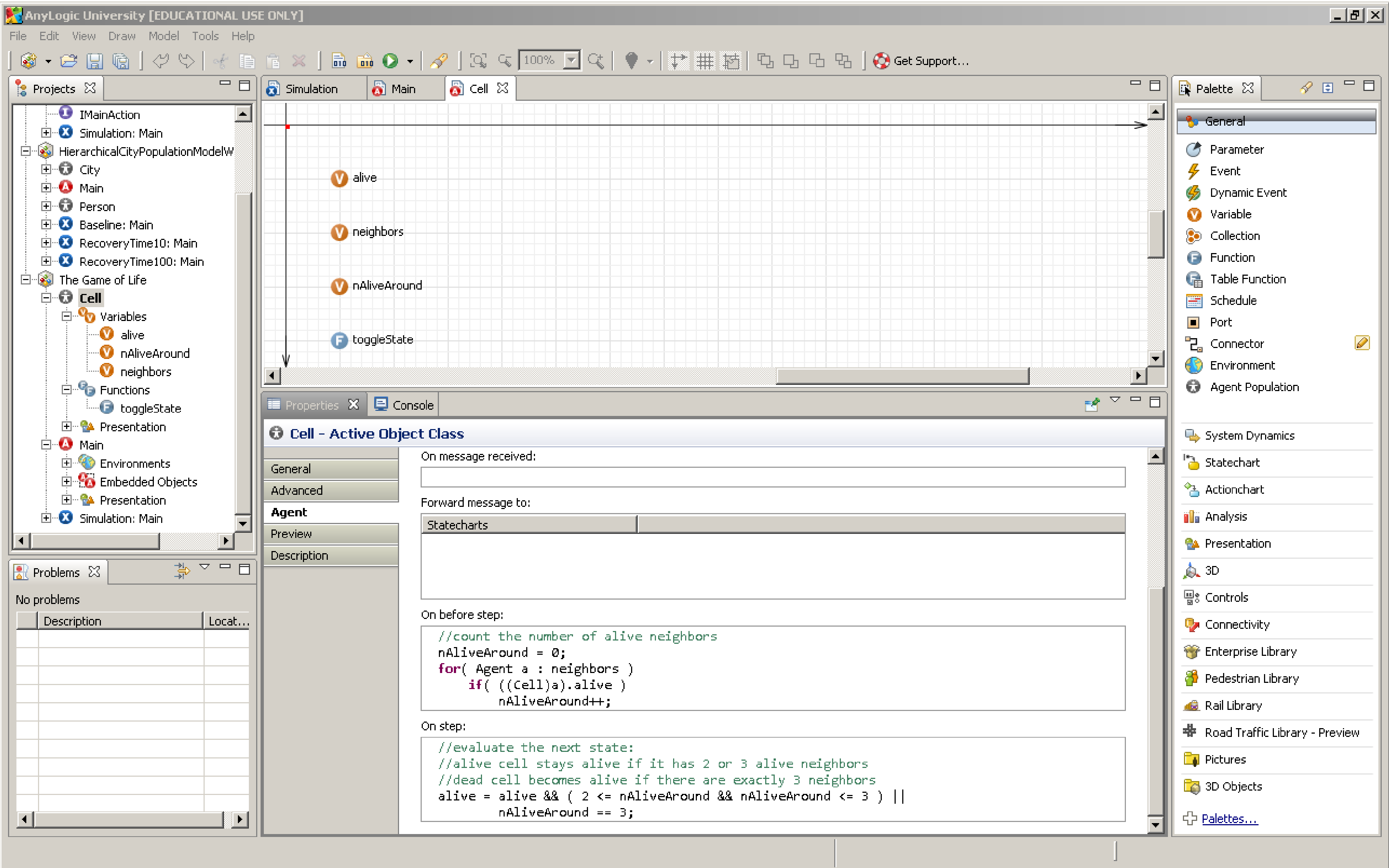
This will count the number Of neighbors around this cell that are alive at the current time (i.e. during the current step)

The “type” of this variable is an “integer”

Visual Representation of Cell (Click on Cell Icon at Origin)



Cell Update Logic
("Agent" Properties of "Cell")



Two Key Models of Time in Anylogic: Continuous (Asynchronous) Time

- This is what we have dealt with to this point
- Here, every agent is updated at a different time, according to events
- No two agents are typically likely to be updated at exactly the same time during most of model execution, so when considering the state of other agents they “see” the last situation where the other agent has been updated

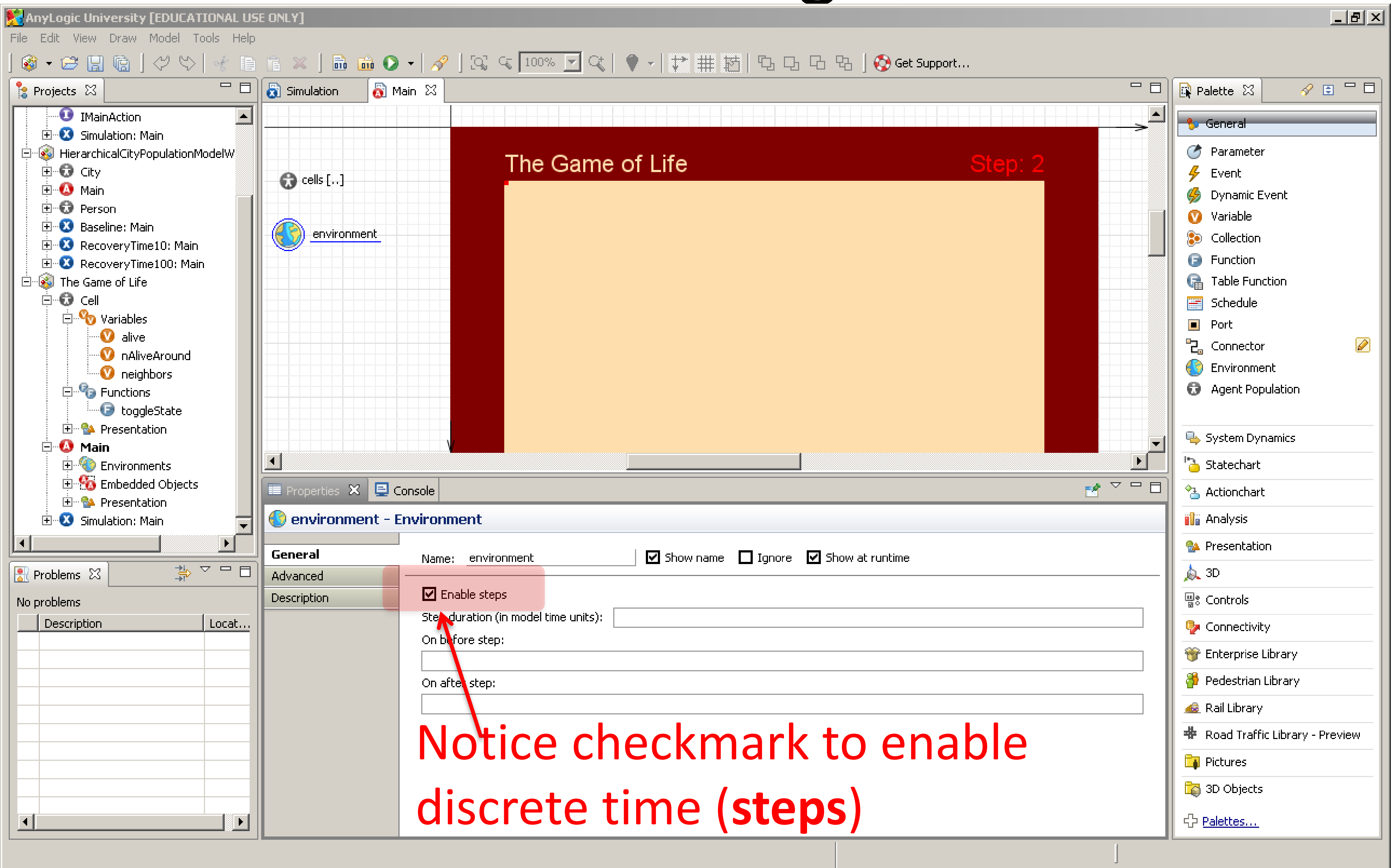
Two Key Models of Time in Anylogic: Discrete (Synchronous) Time

- Here, agents all change in lockstep, separated by fixed "time steps"
- When computing agent behavior (to determine agent state in the next timestep), our enquiries about agent state (e.g. using *getAgentAtCell* or *getAgentNextToMe*) need to ask about the situation ***in the current timestep***
  - We gather needed information regarding current state in "On Before Step", and changes are performed in "On Step".
- This is similar to what we saw in System Dynamics – the changes over the next small interval of time (Δt) depend on the current value of the stocks
  - These changes are then applied at once, and all stocks are updated

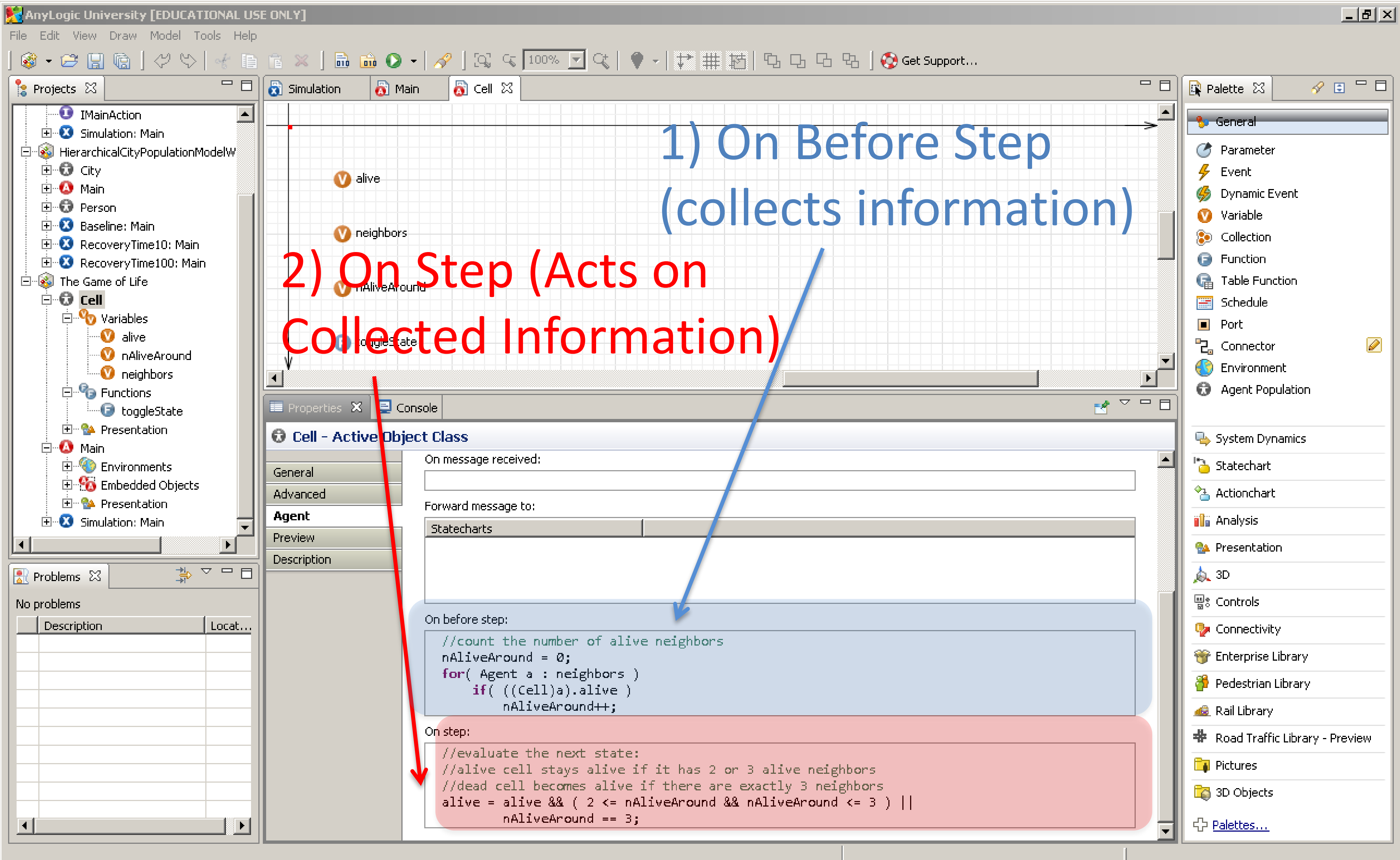
Enabling Discrete (Synchronous) Time

- When enable the steps, the various handlers for synchronized time (e.g. “On before step”, “On step”, “On after step”) etc.) are executed
  - Both environment and agents have “On before step” and “On after step” handlers
  - “On before step” for environments is executed before the corresponding method for agents
  - “On after step” for environments is executed after the corresponding method for agents
- Synchronous time can be enabled via the **environment** “General” page
  - Click checkbox “Enable steps”

Environment: Enabling Discrete **Time**



Cell Update Logic
("Agent" Properties of "Cell")



On Before Step: Collecting the Information

This records a running count of # seen so far (initially 0)

On before step:

```
//count the number of alive nei
nAliveAround = 0;
for( Agent a : neighbors )
    if( ((Cell)a).alive )
        nAliveAround++;
```

2) Loops through each of the neighbors. Every time we see a live neighbor, increment the count of alive neighbors

On Step: Performing the Update based on Observed Information

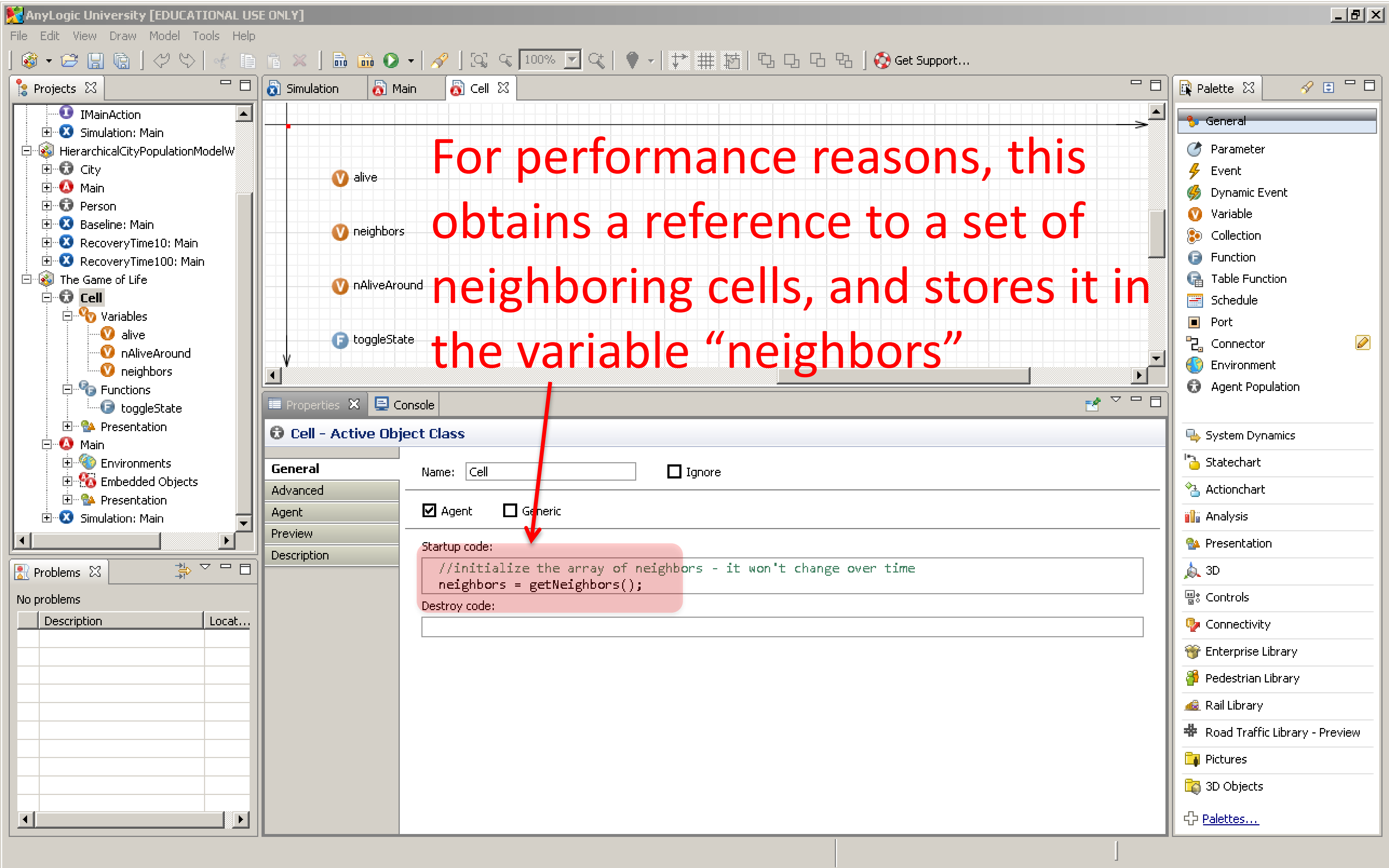
Reminder: This is the information collected in "On Before Step"

On step:

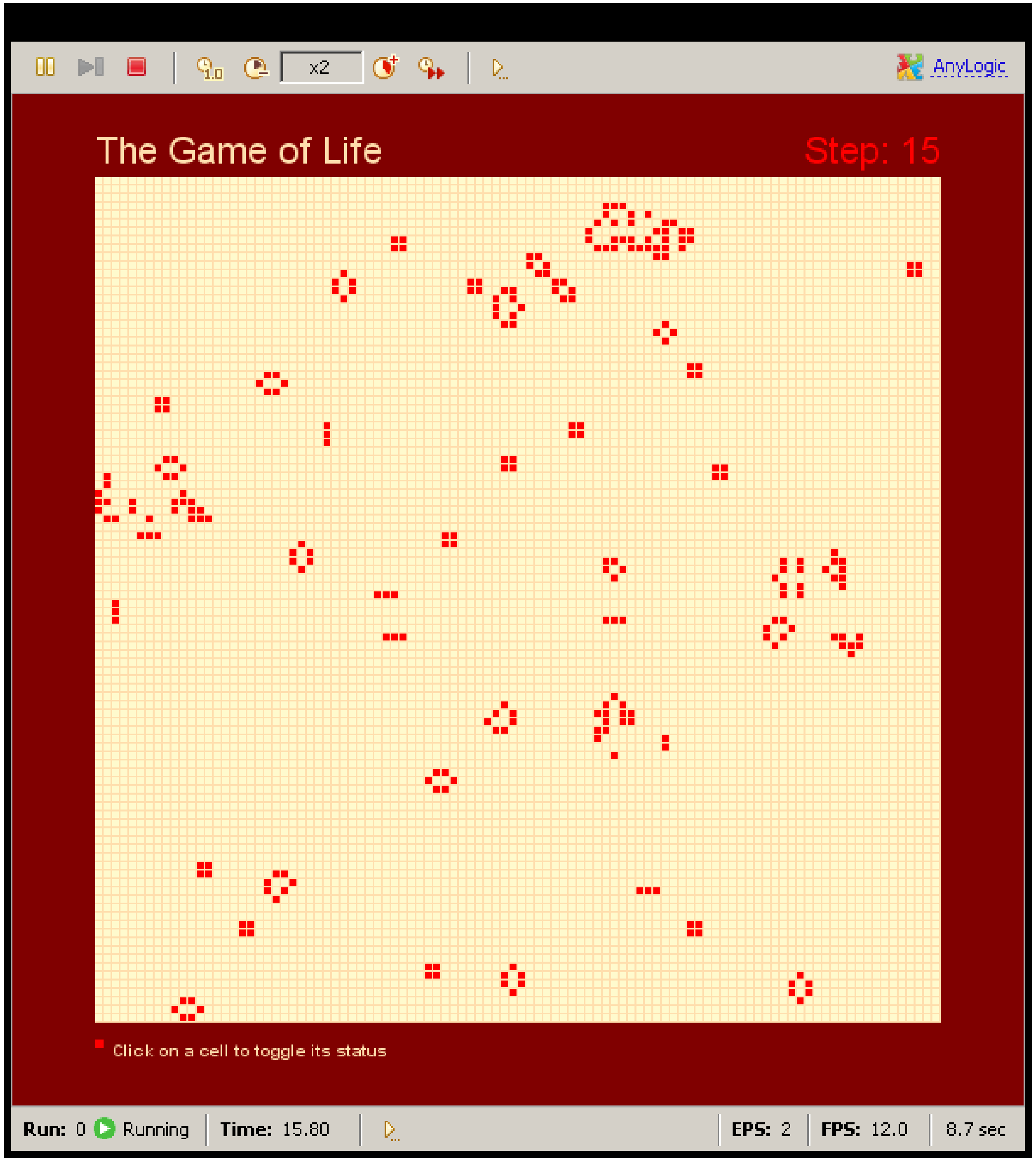
```
//evaluate the next state:
//alive cell stays alive if it has 2 or 3 alive neighbors
//dead cell becomes alive if there are exactly 3 neighbors
alive = alive && ( 2 <= nAliveAround && nAliveAround <= 3 ) ||
     nAliveAround == 3;
```

Here, we are updating our aliveness status (represented by the "alive" variable) based on our current status & characteristics of the local environment.

Obtaining the List of Neighboring Cells at Startup



Running the Model



Lecture Outline

- AnyLogic's Spatial embedding types
  - √ Overview
  - √ Reminder of continuous space
  - √ A glimpse of a discrete space & discrete time model
- Agent Mobility

Agent Mobility

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

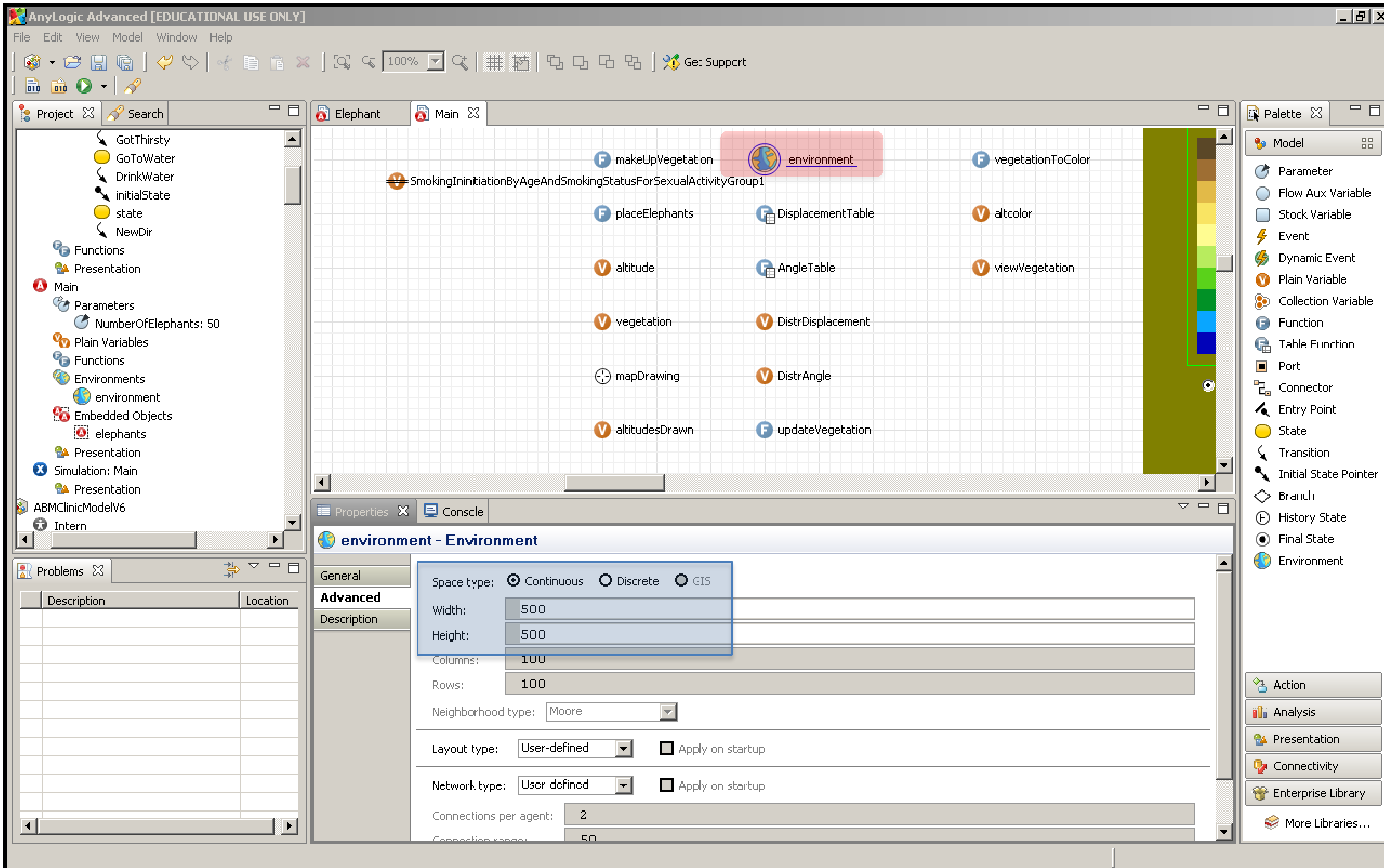
2D Spatial Embedding: Mobility Implications

- 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

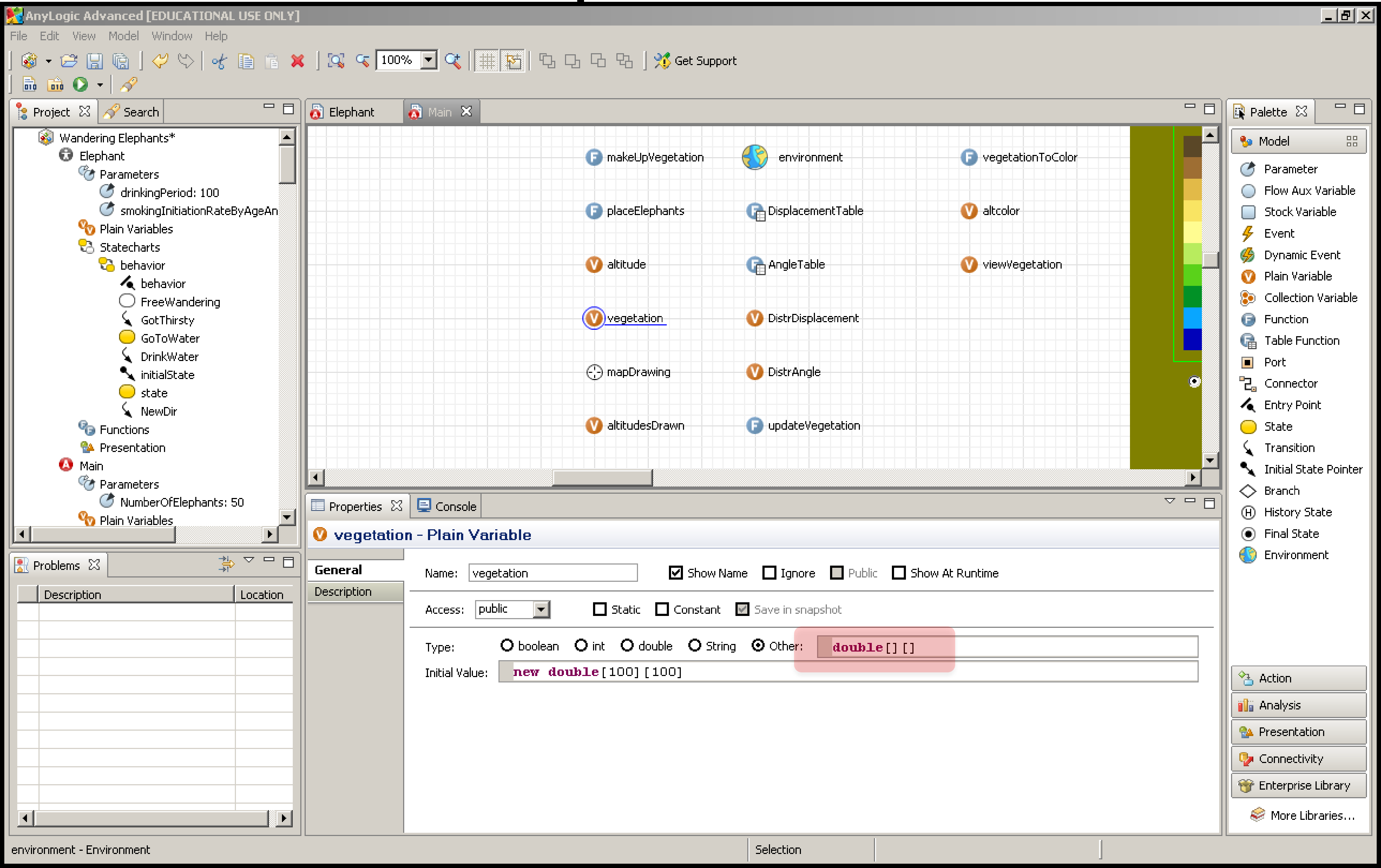
Hands on Model Use Ahead

Load model: Wandering Elephants.alp

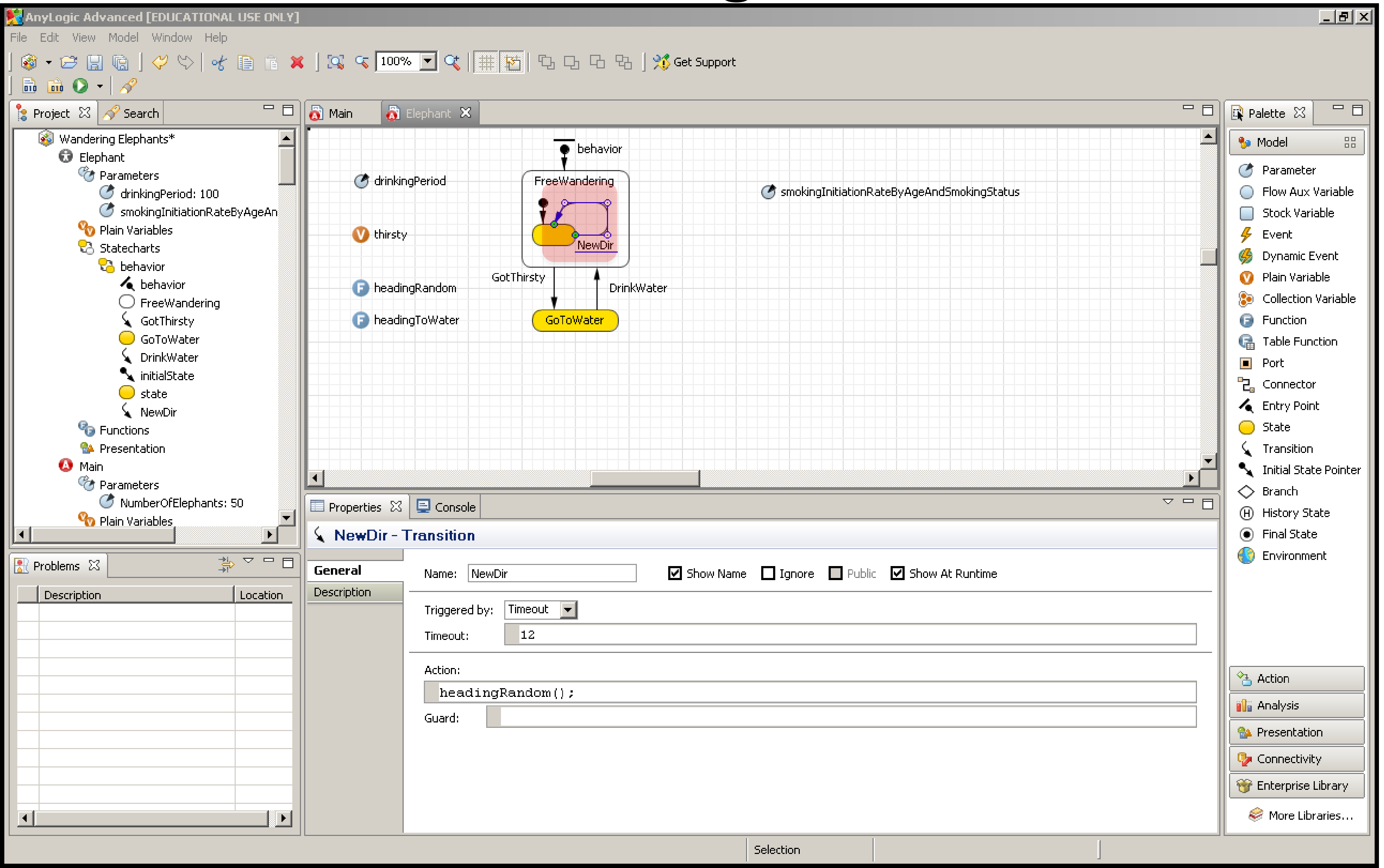
Environment



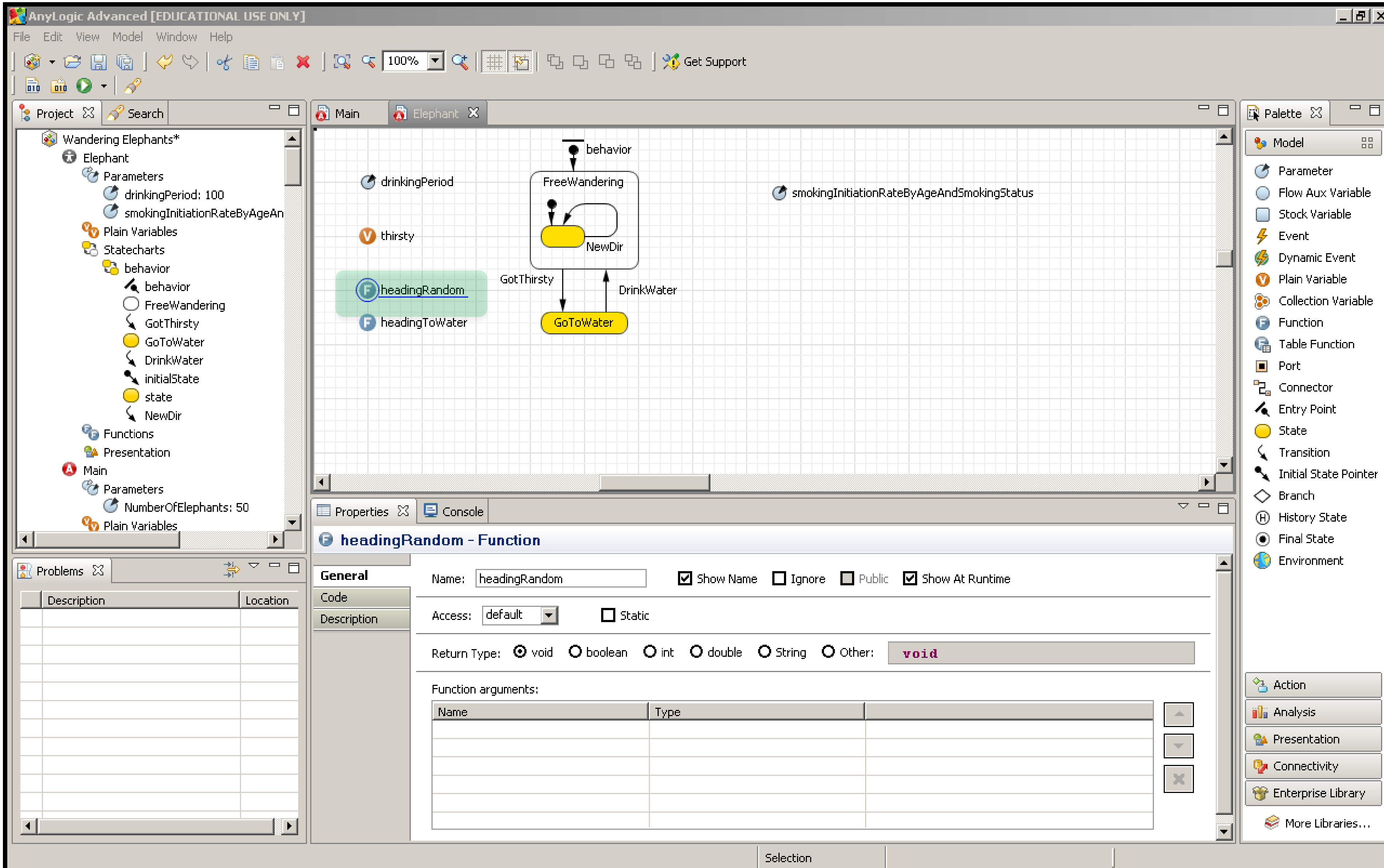
Landscape Information



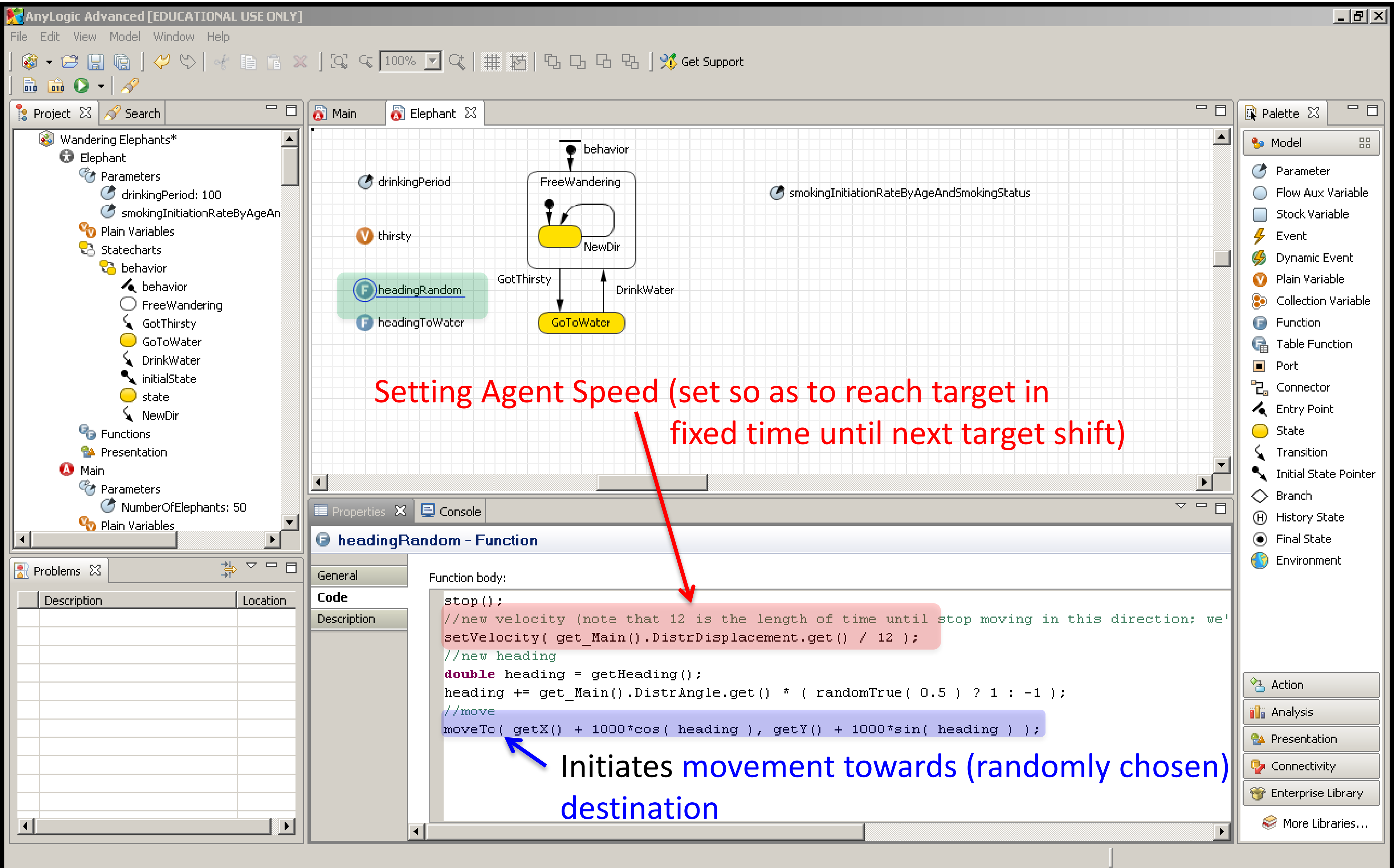
Agent Movement: Periodic Movement Changes



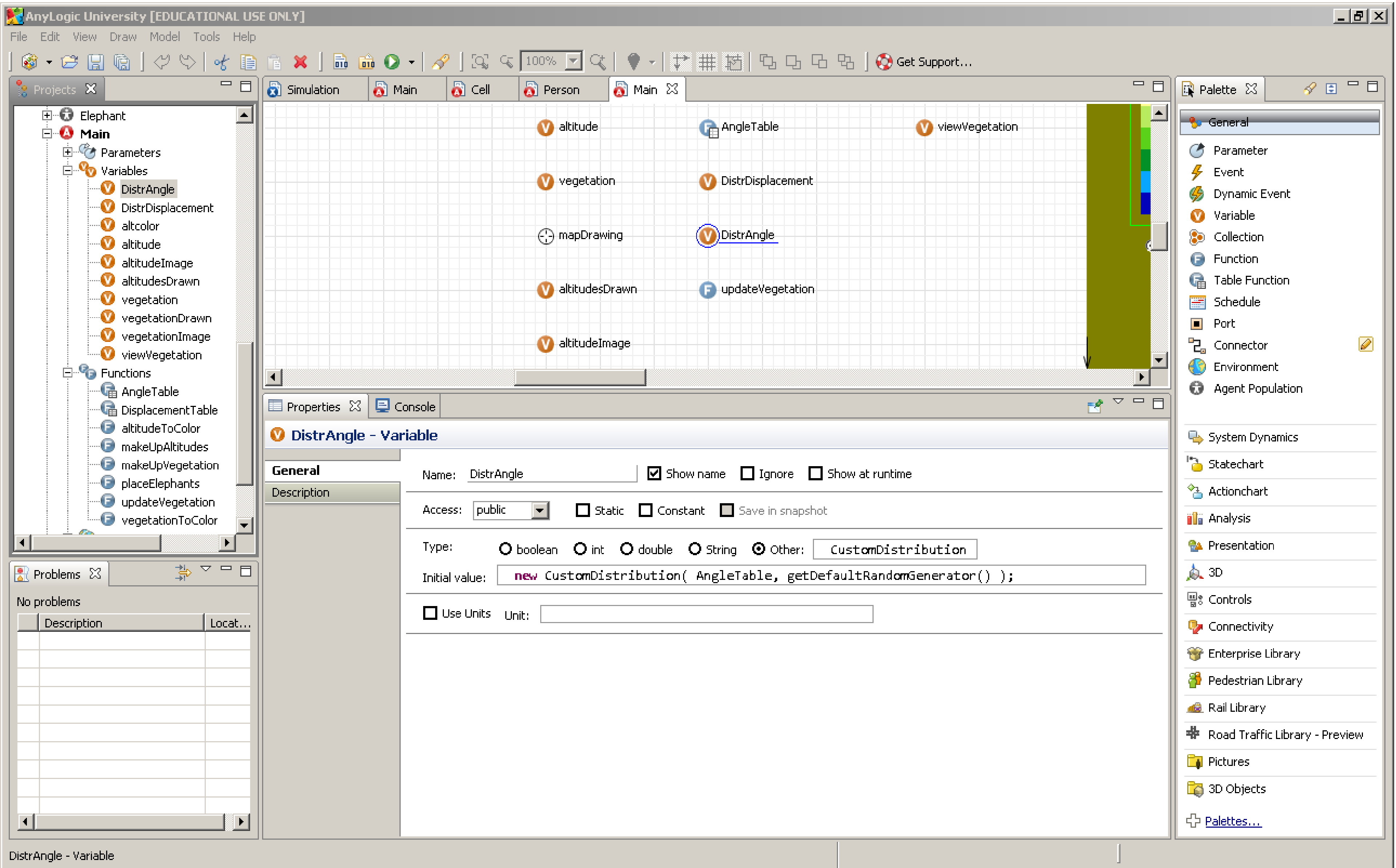
New Direction Change Function Info



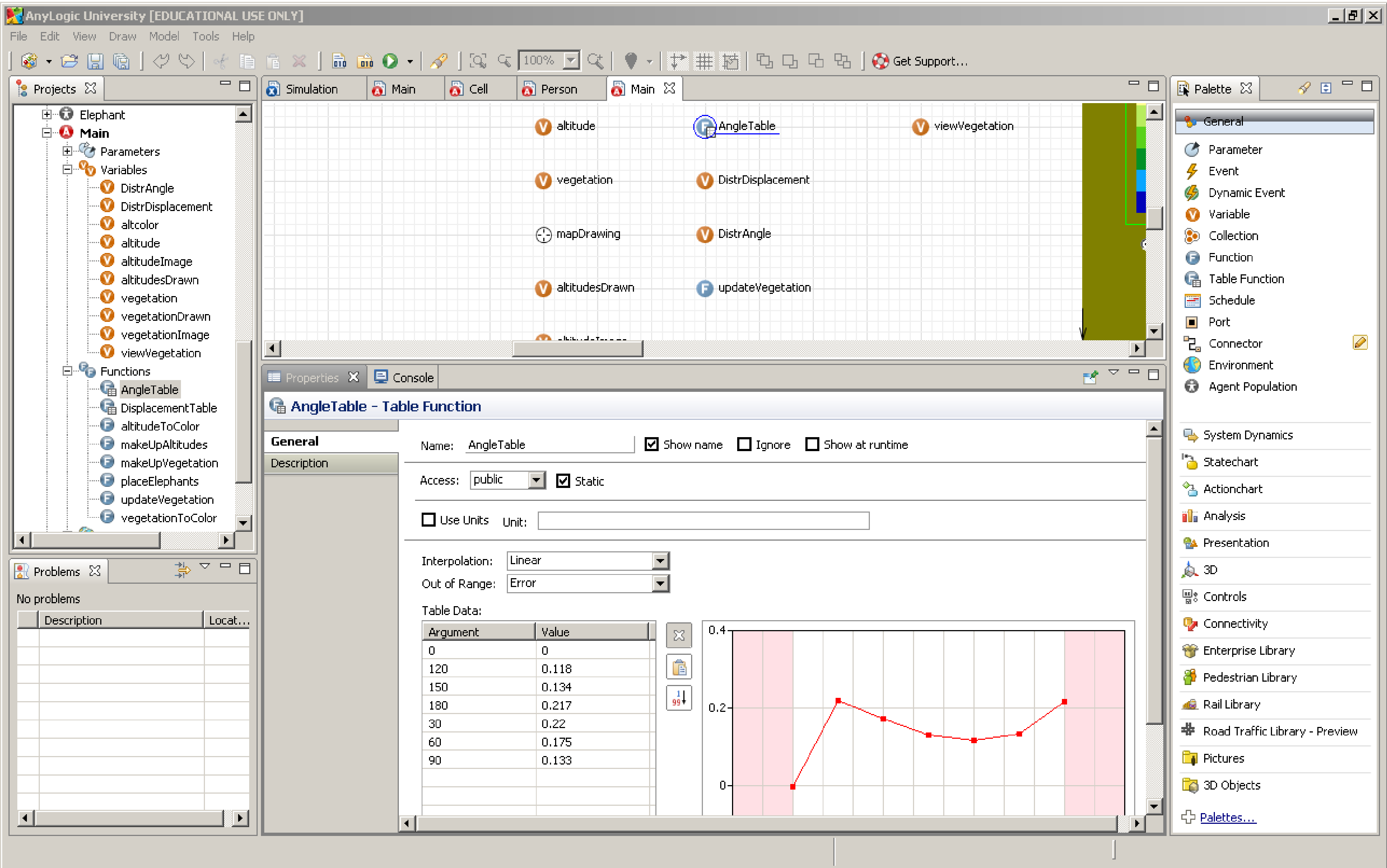
New Direction Change: Function “Body”



(Main) Defining a Custom Angle Distribution

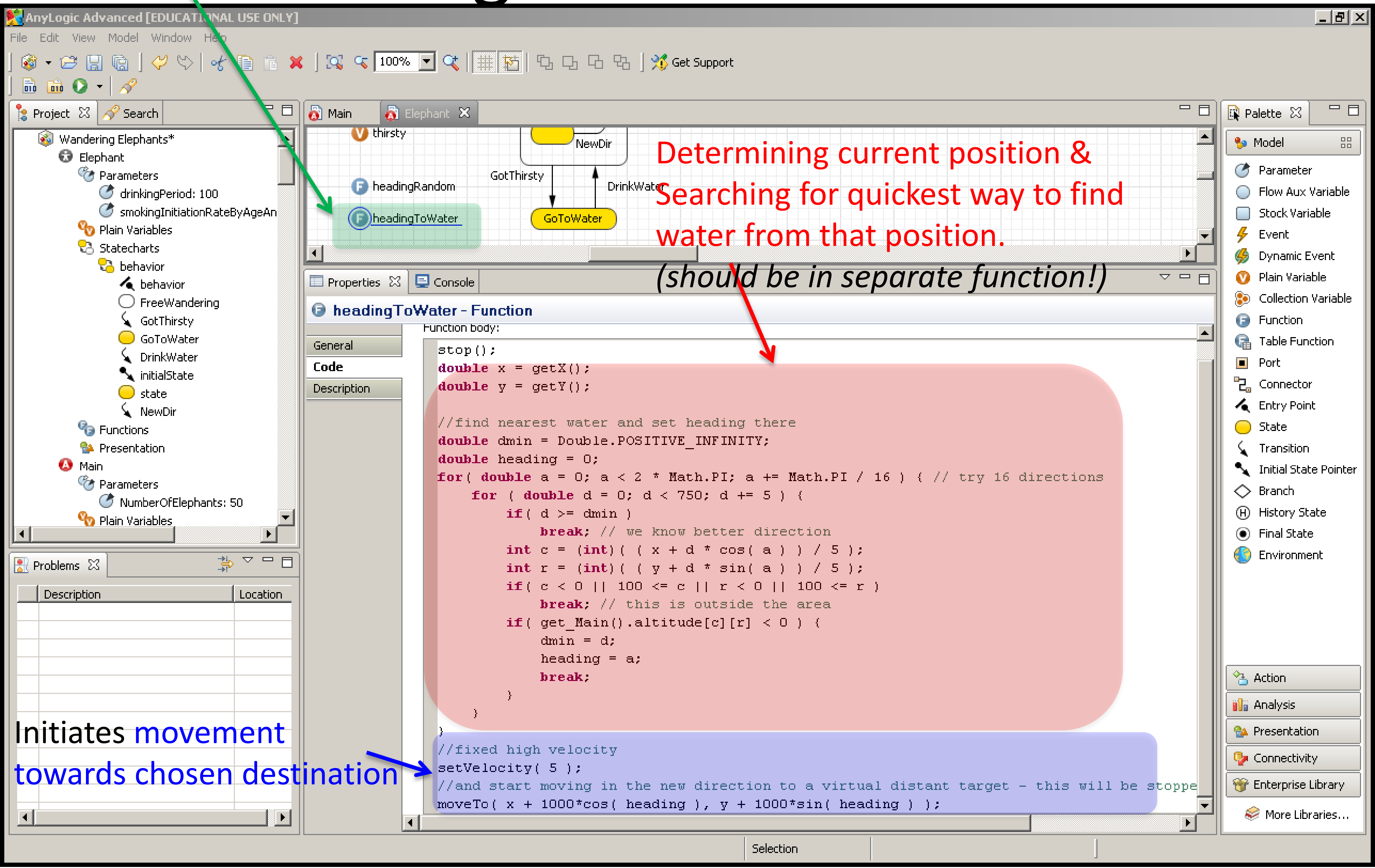


Data for Custom Distribution



Heading Towards Resource

Looking at body of this function (method)

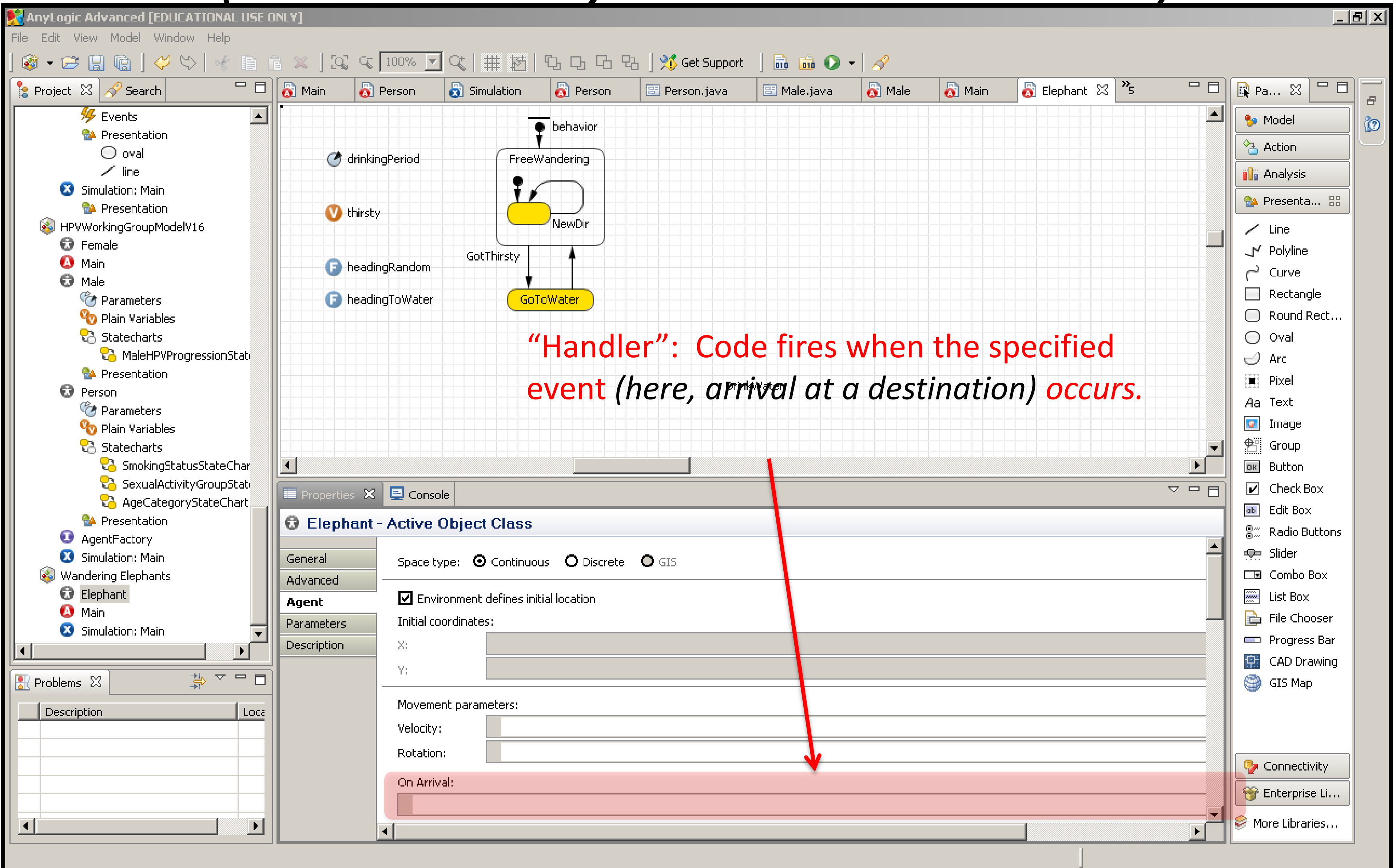


Determining current position & Searching for quickest way to find water from that position.

(should be in separate function!)

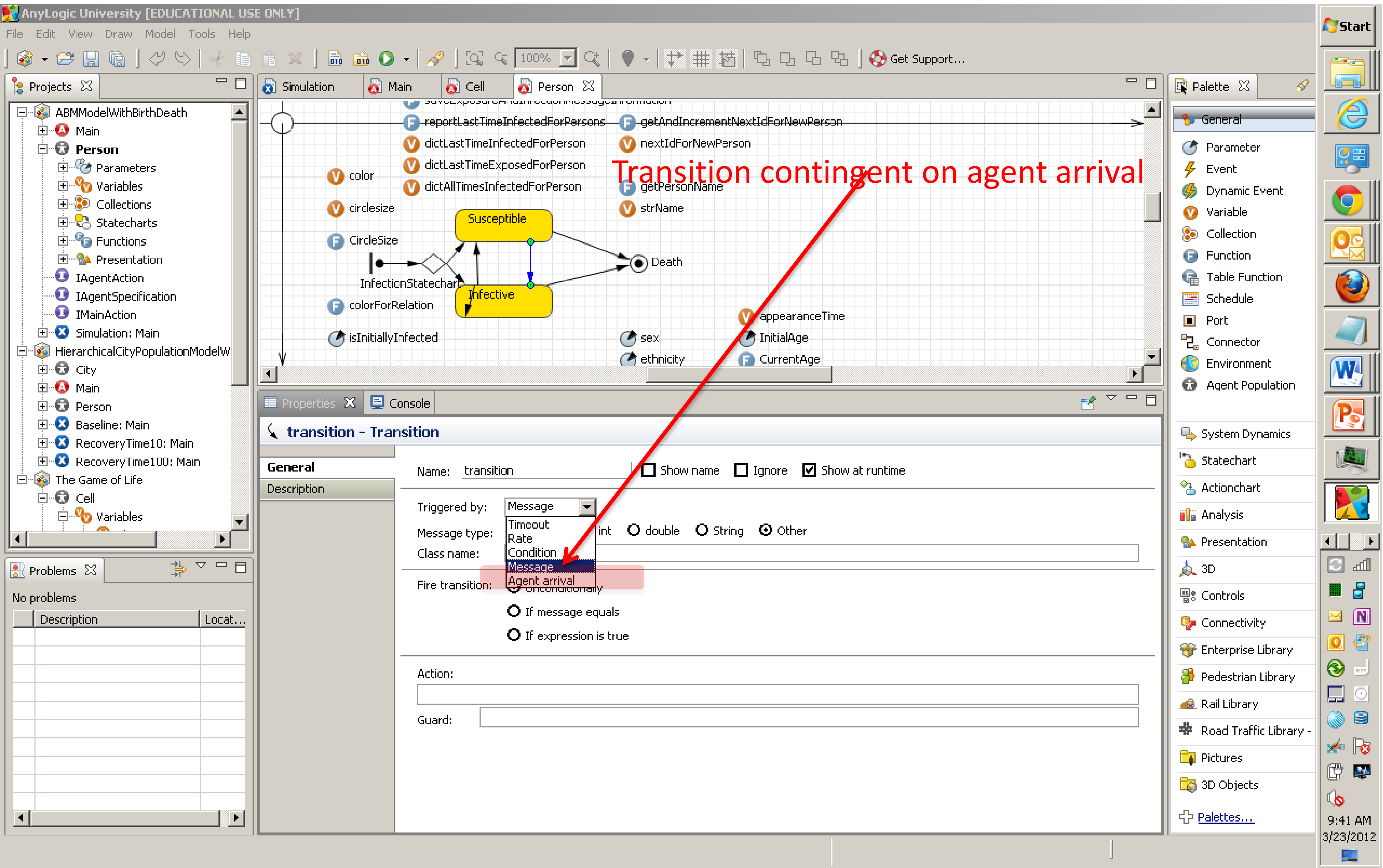
Initiates movement towards chosen destination

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

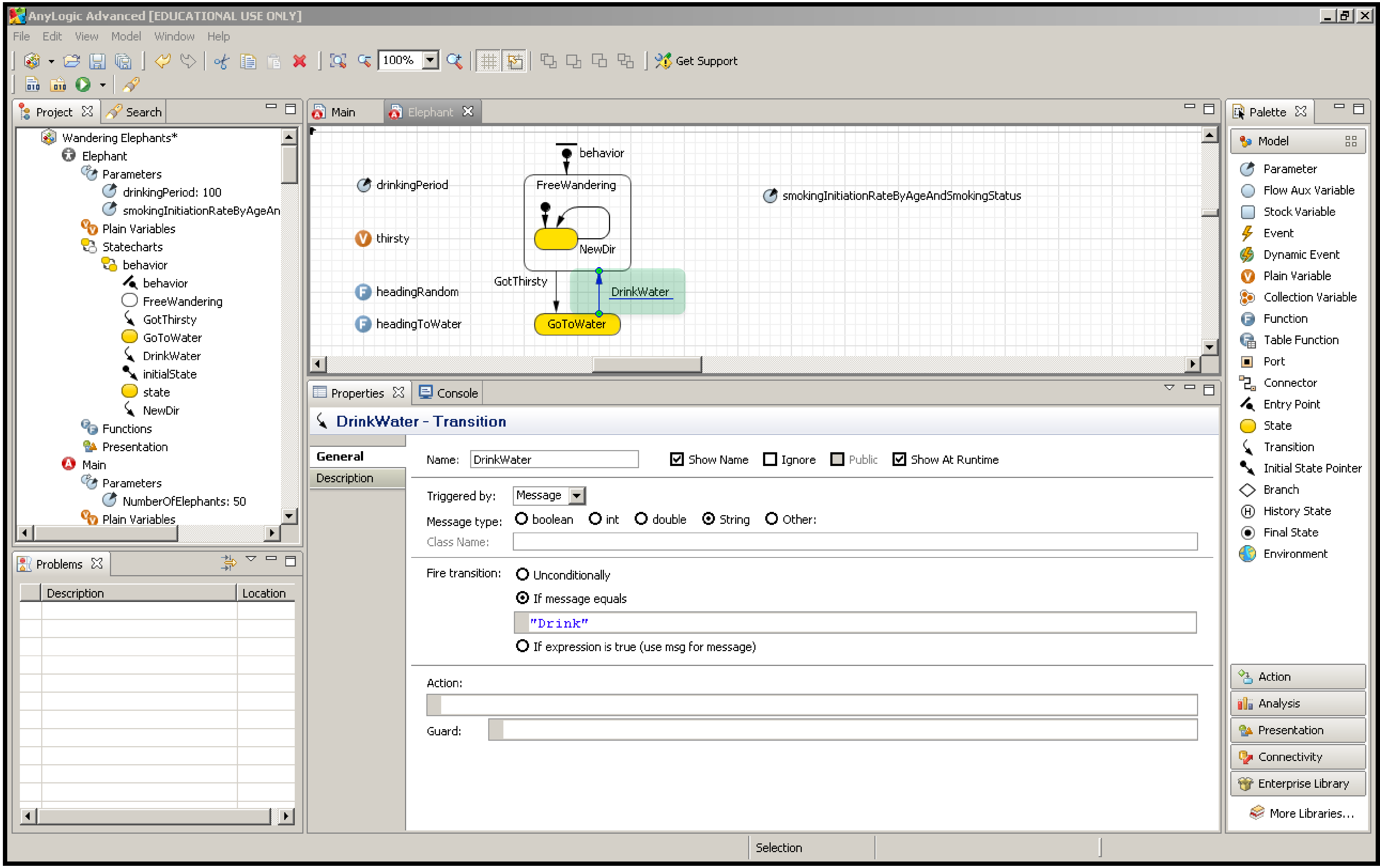


"Handler": Code fires when the specified event *(here, arrival at a destination)* *occurs.*

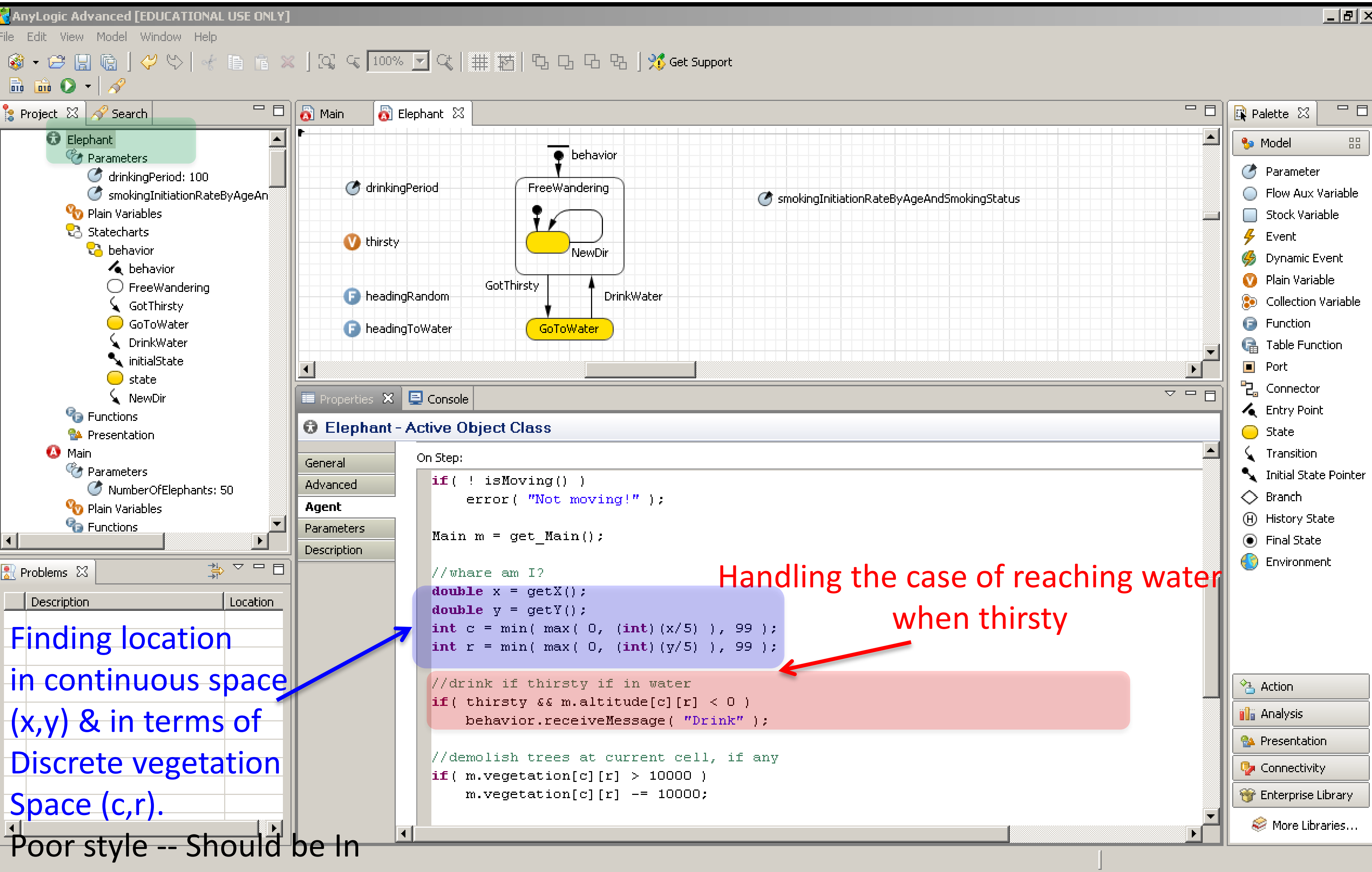
Handling Arrival Events in Statecharts



Resumption of Wandering After Slaking Thirst



Handling of Movement Logic



Handling the case of reaching water when thirsty

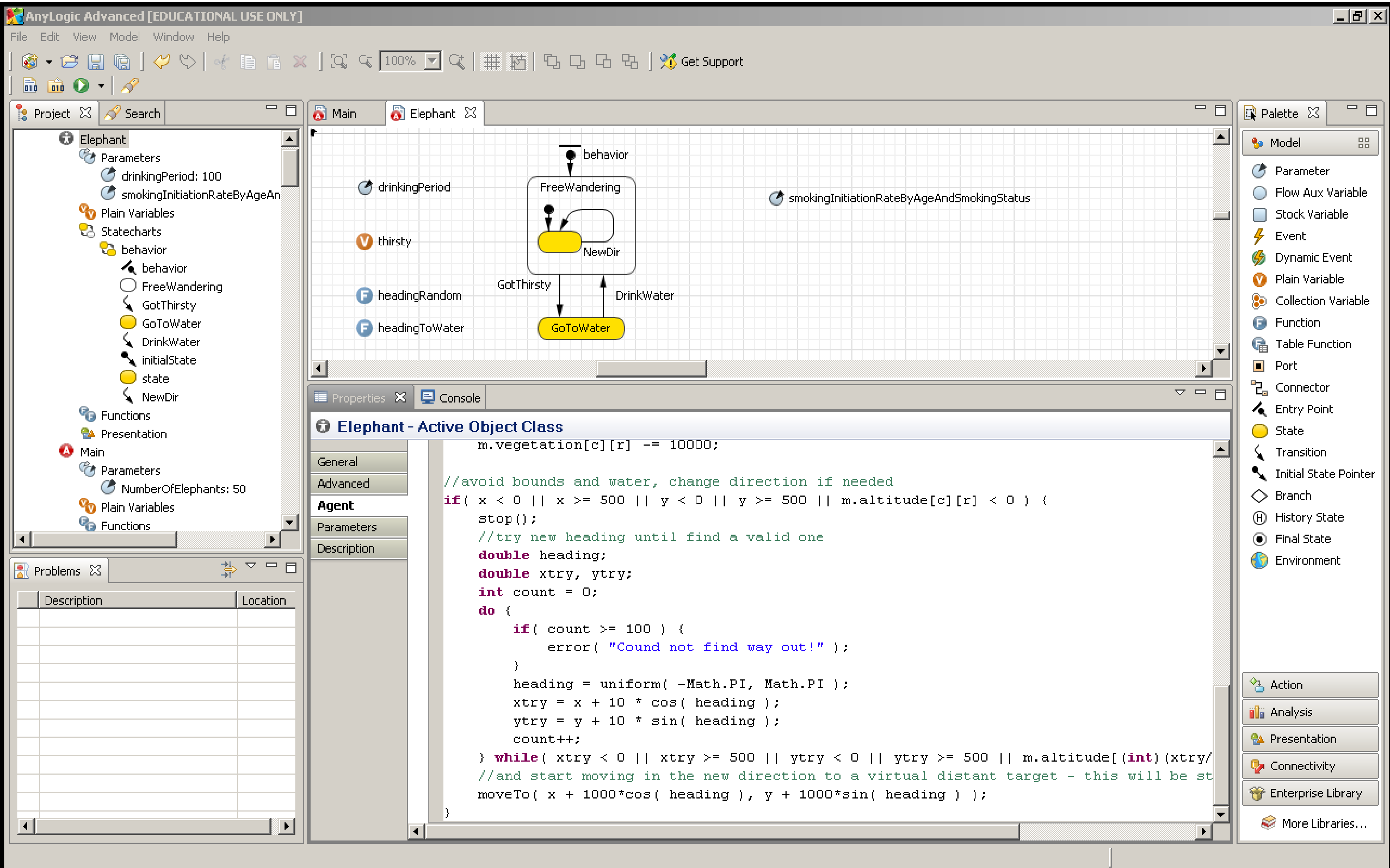
Finding location in continuous space (x,y) & in terms of Discrete vegetation Space (c,r).

Poor style -- Should be In separate function

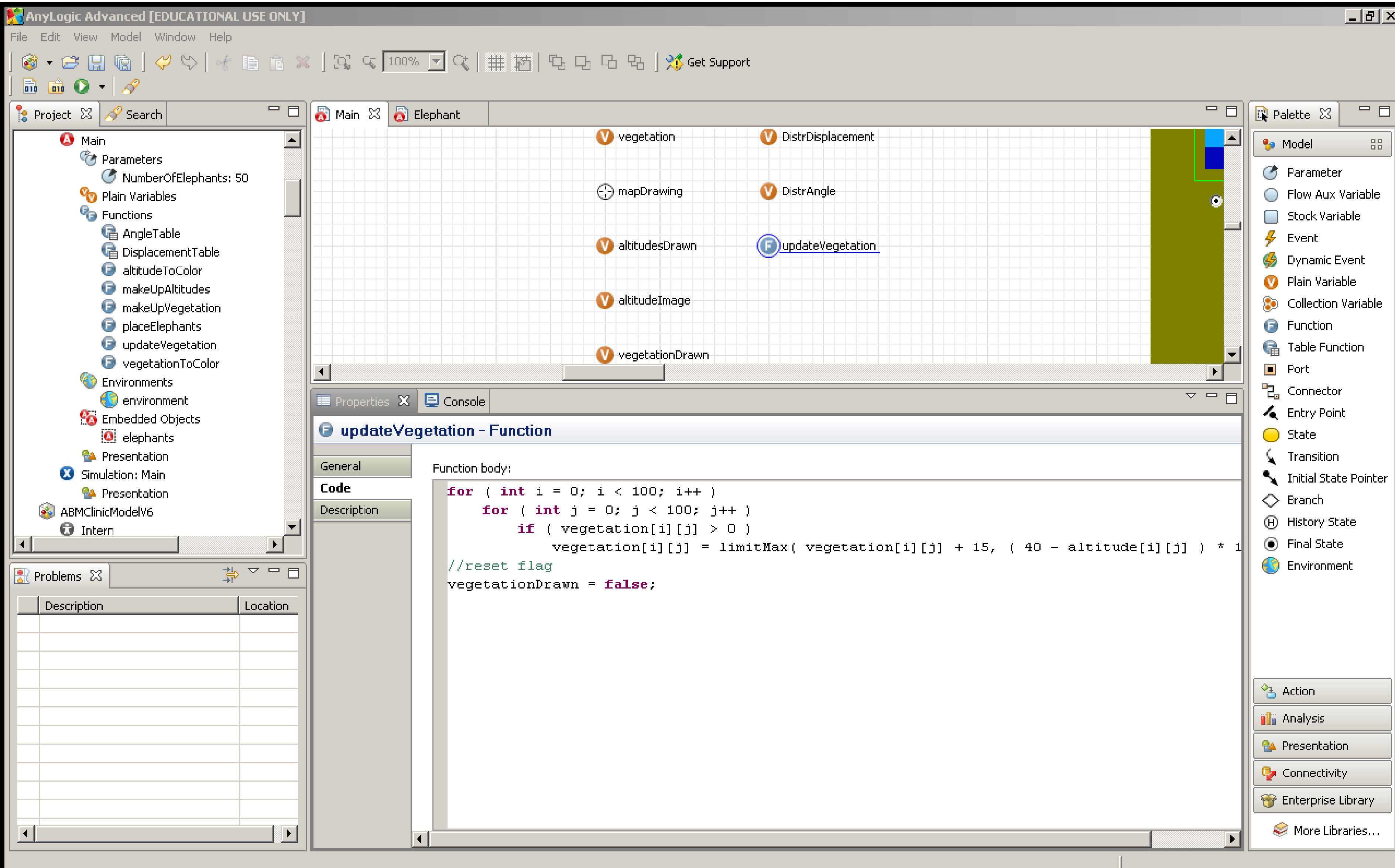
Rerouting Around Barriers (Boundaries & Water)

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



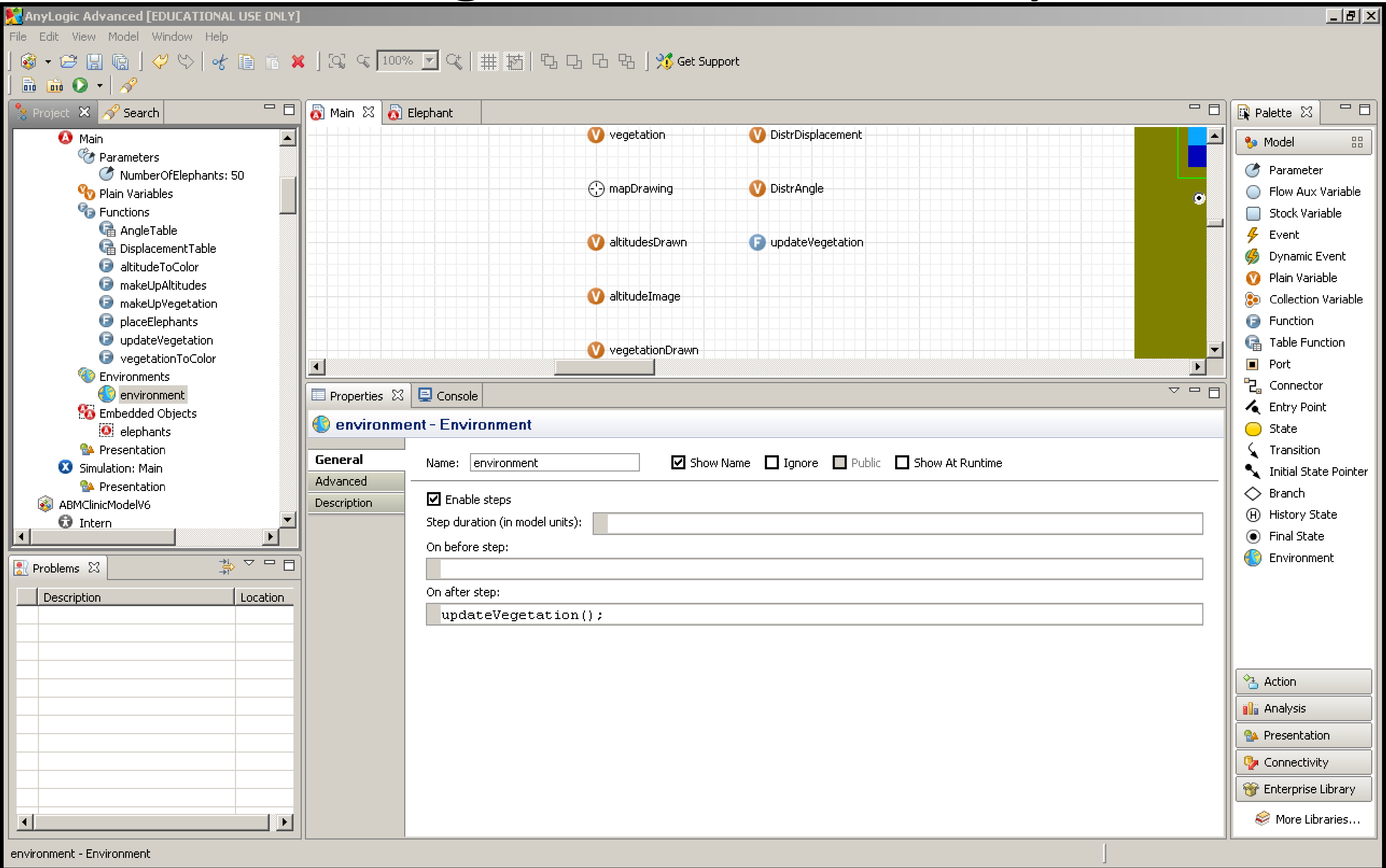
Environment: Updating Vegetation



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()

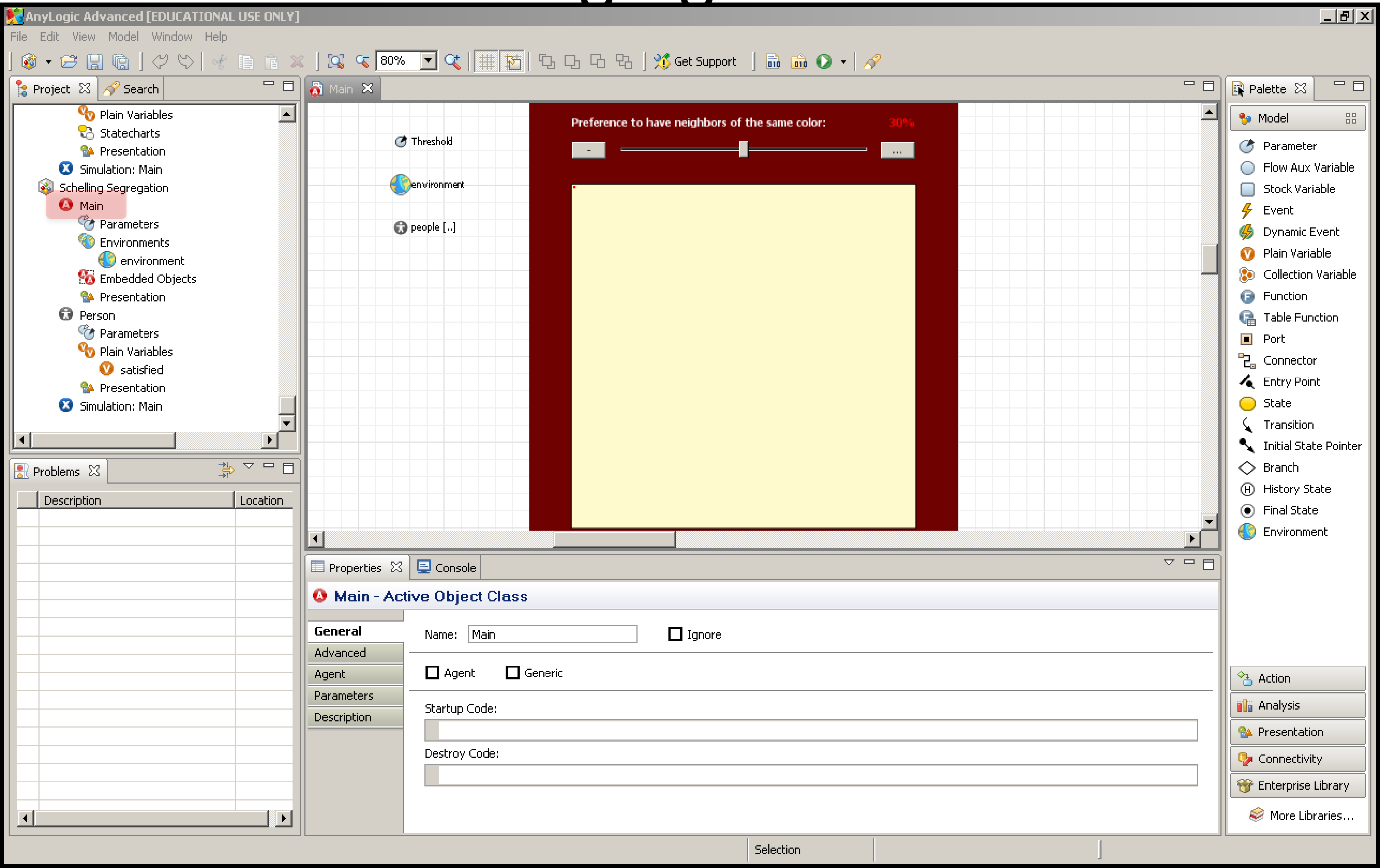
Environment Happens to Handle Process of Maintaining Environmental Dynamics



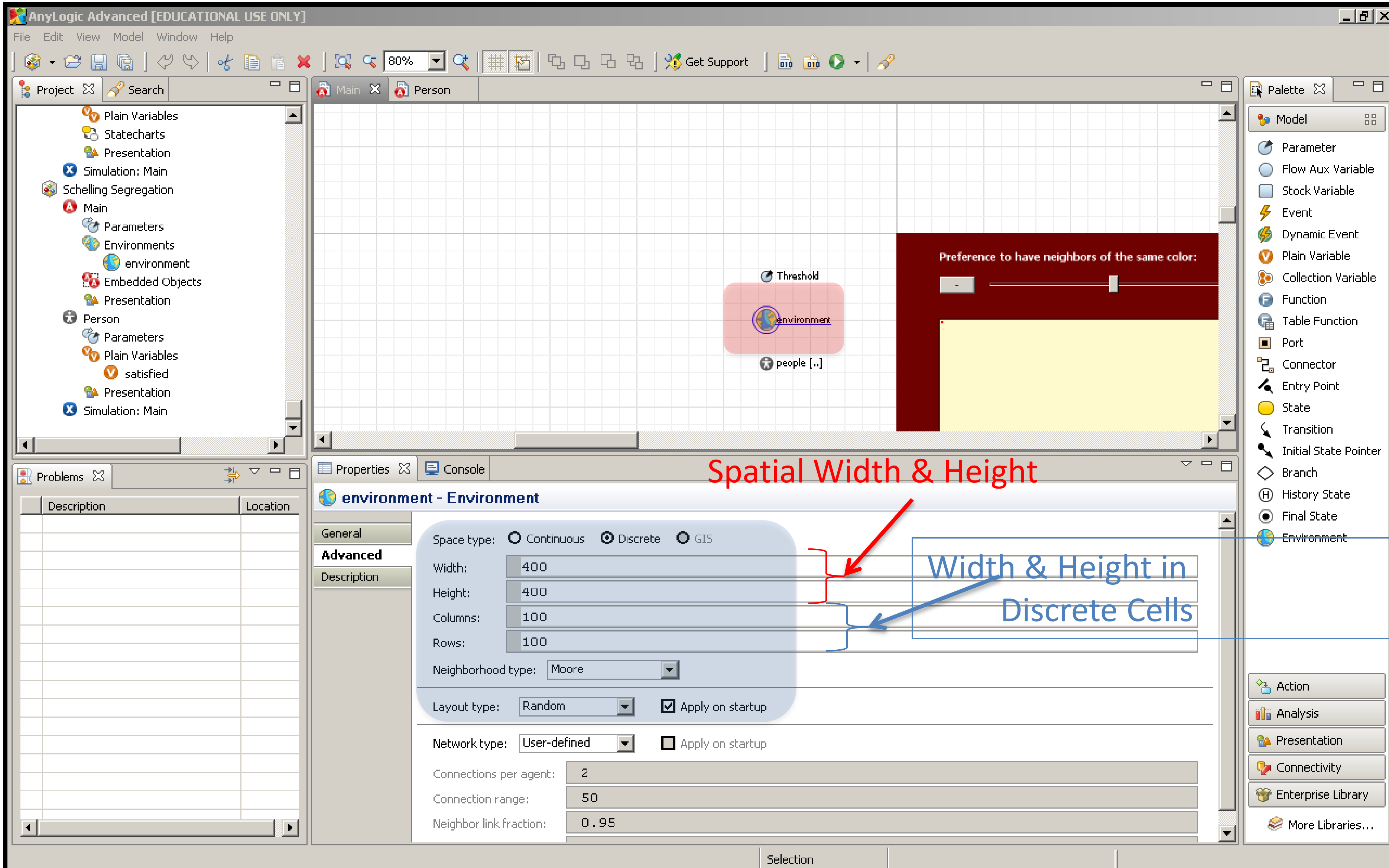
Hands on Model Use Ahead

Load model: Schelling Segregation.alp

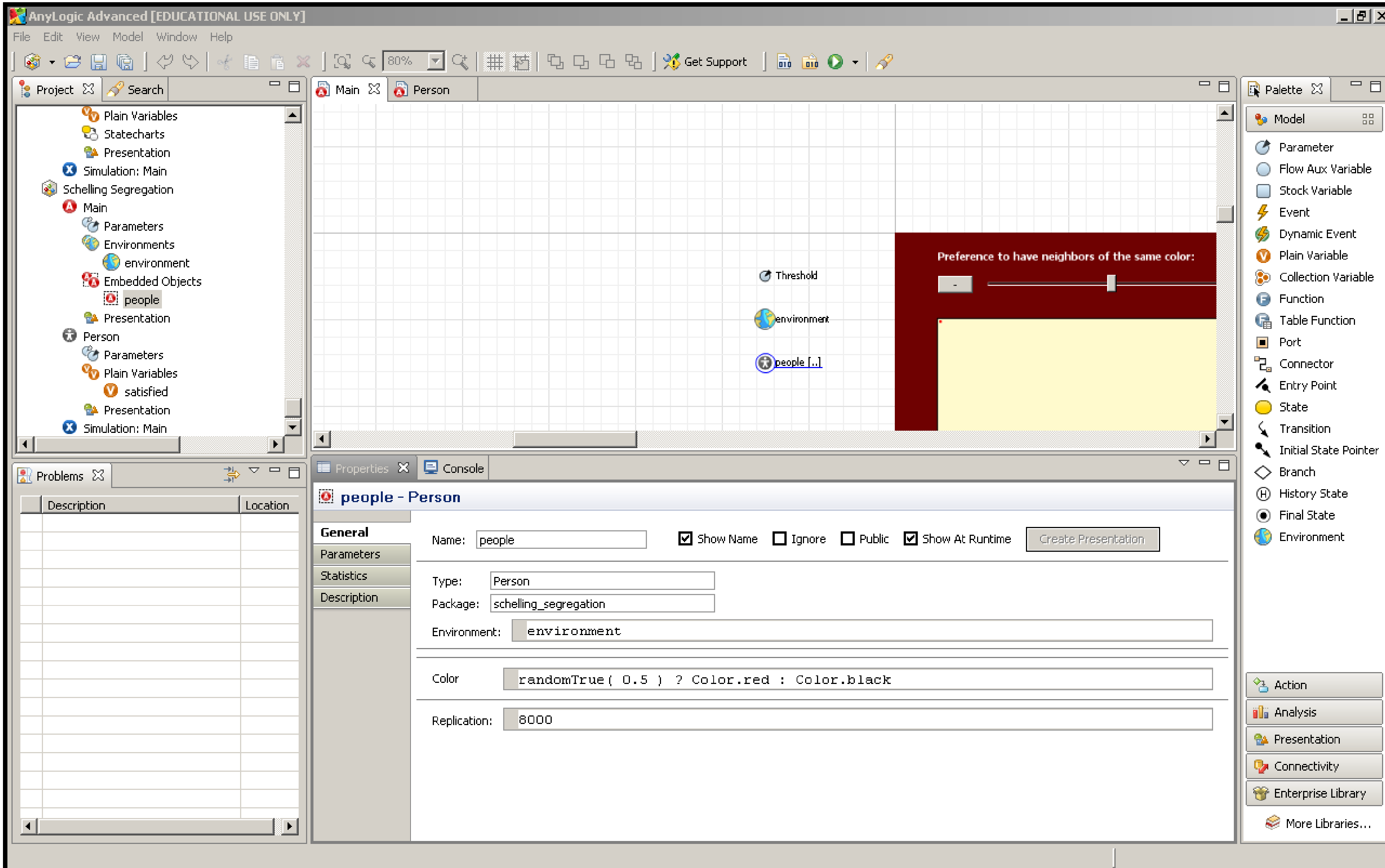
A Model to Examine the Emergence of Segregation



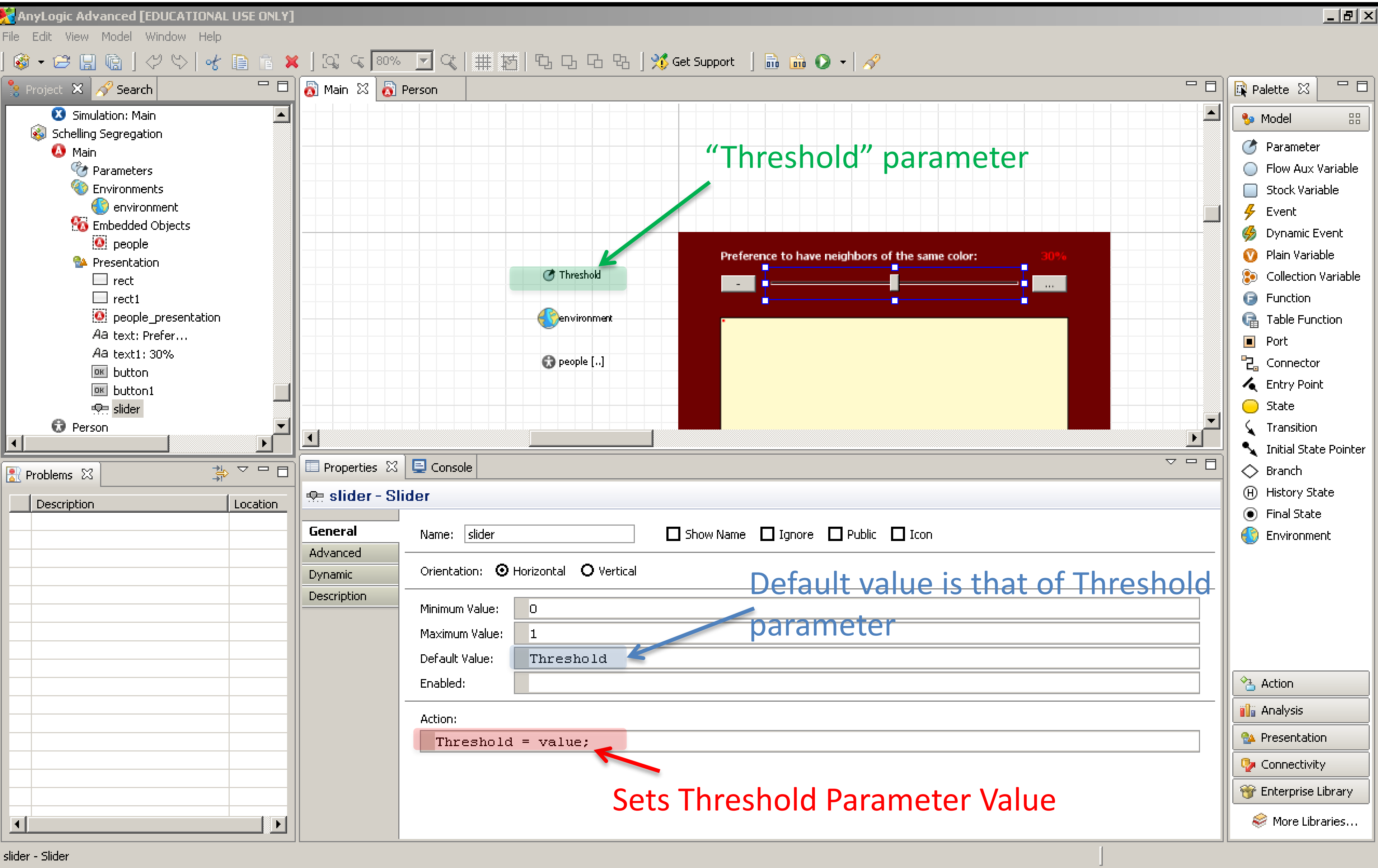
A Discrete Spatial Environment with Random Agent Positioning



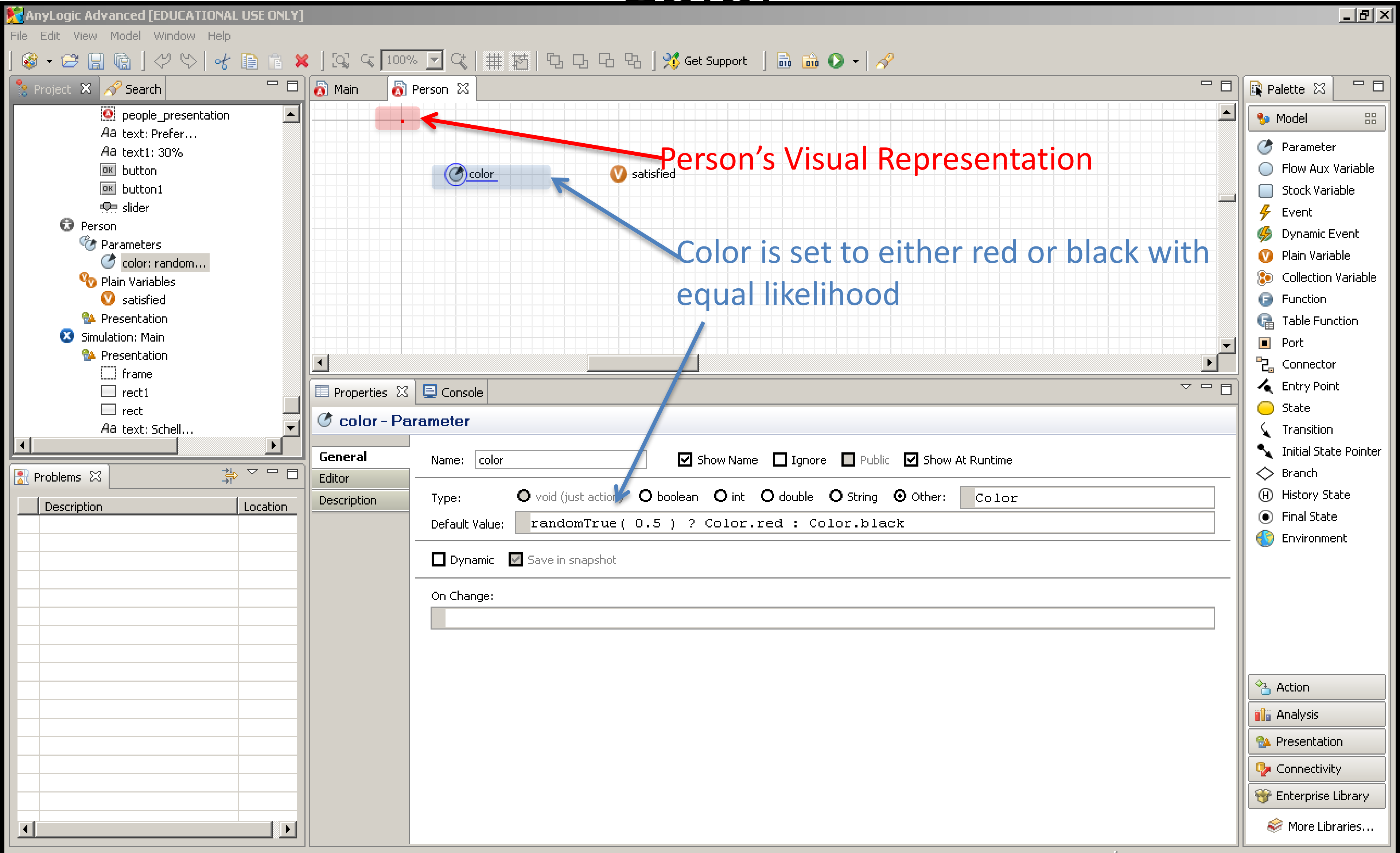
Population Dependence on the Population



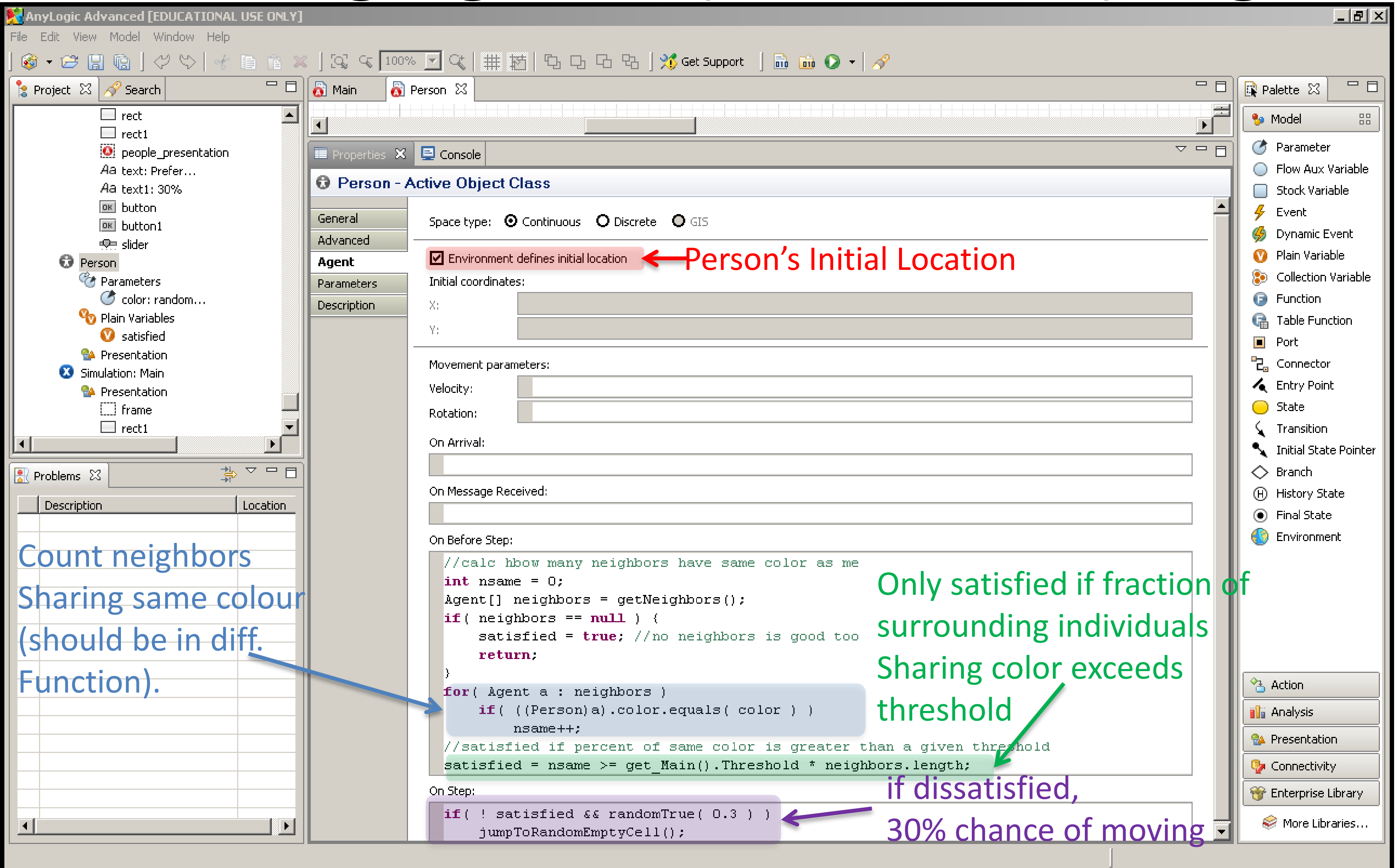
Slider Input Sets Parameter Value



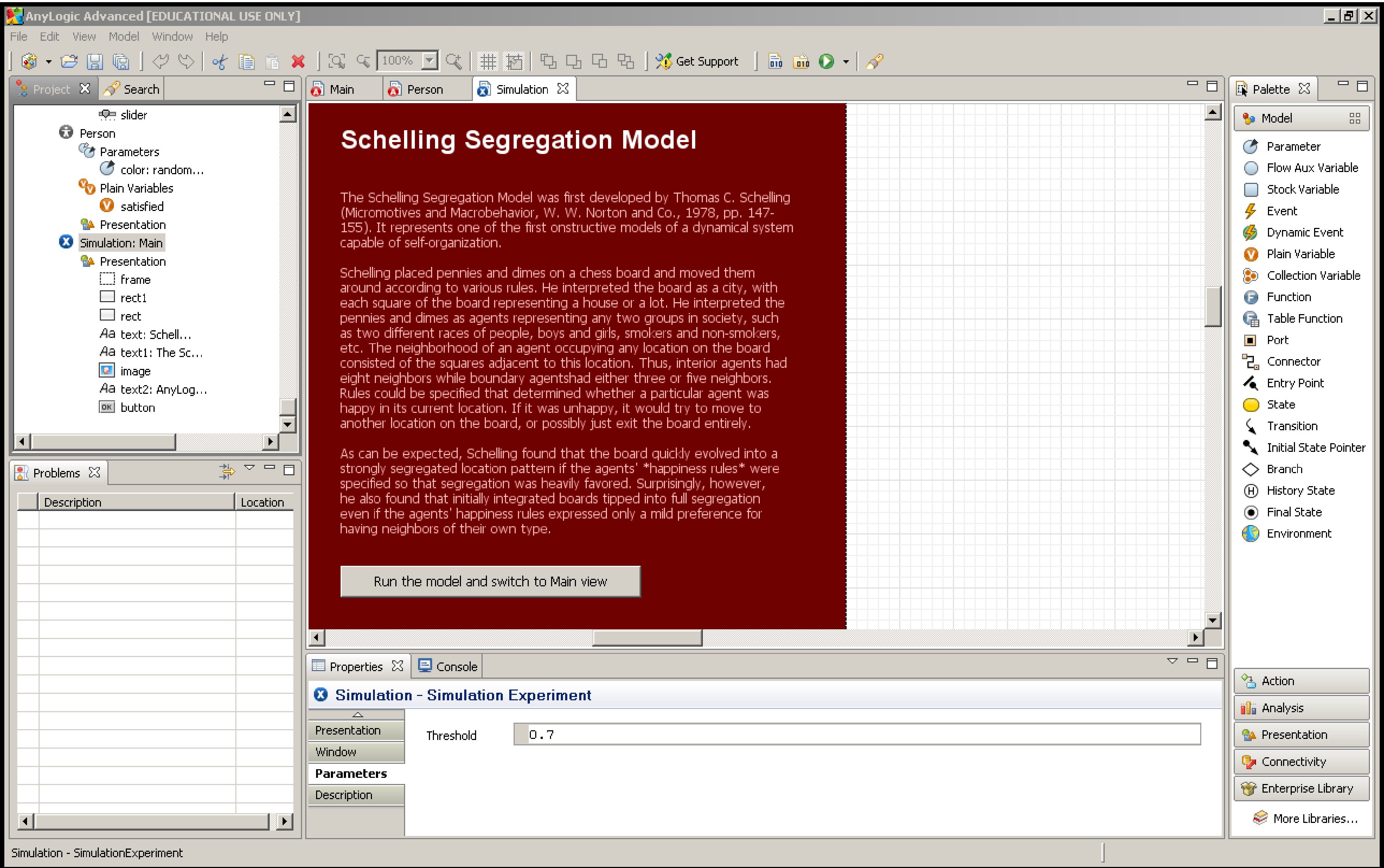
Person is Assigned a Randomly Picked Color



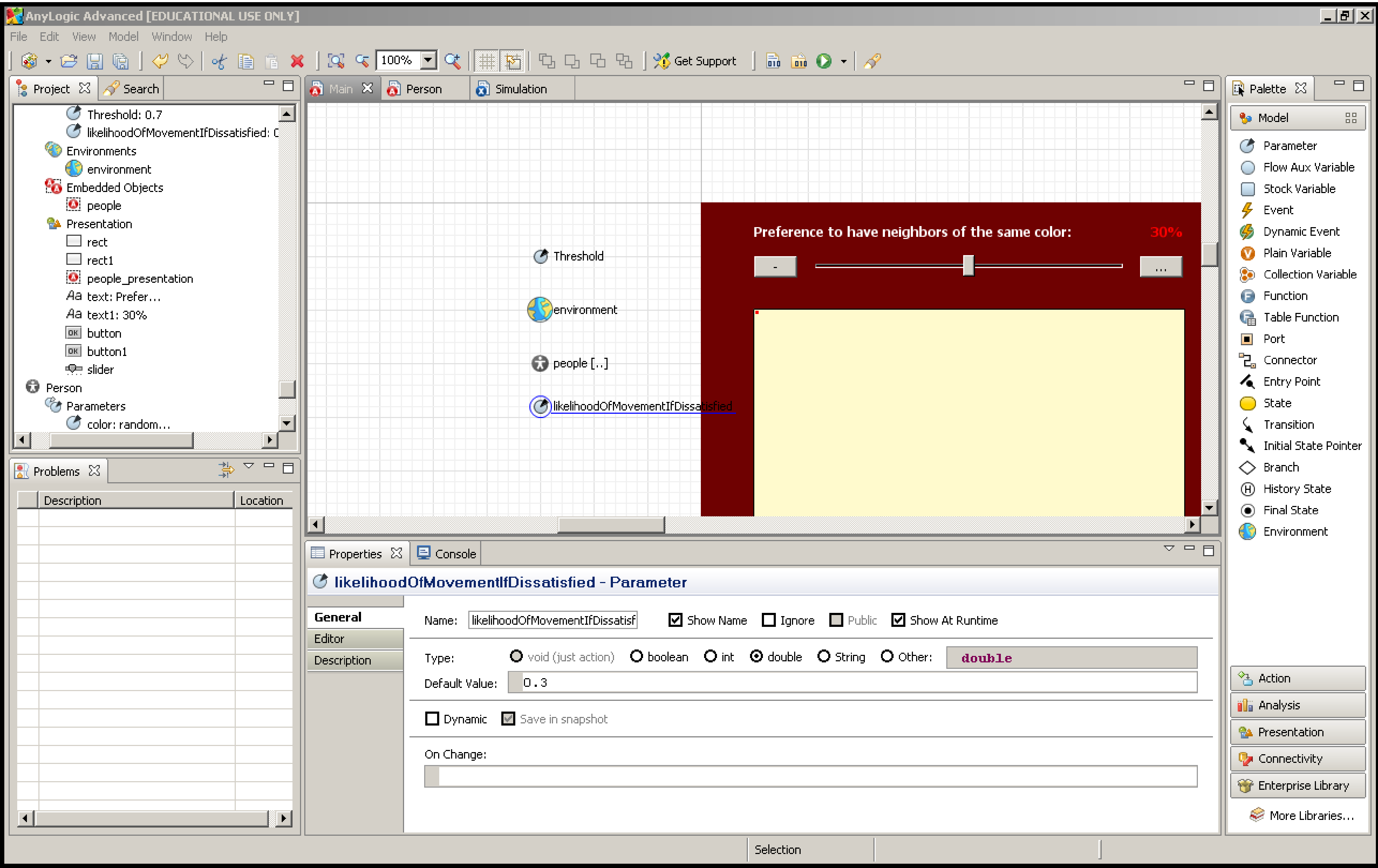
Core Segregation (Movement) Logic



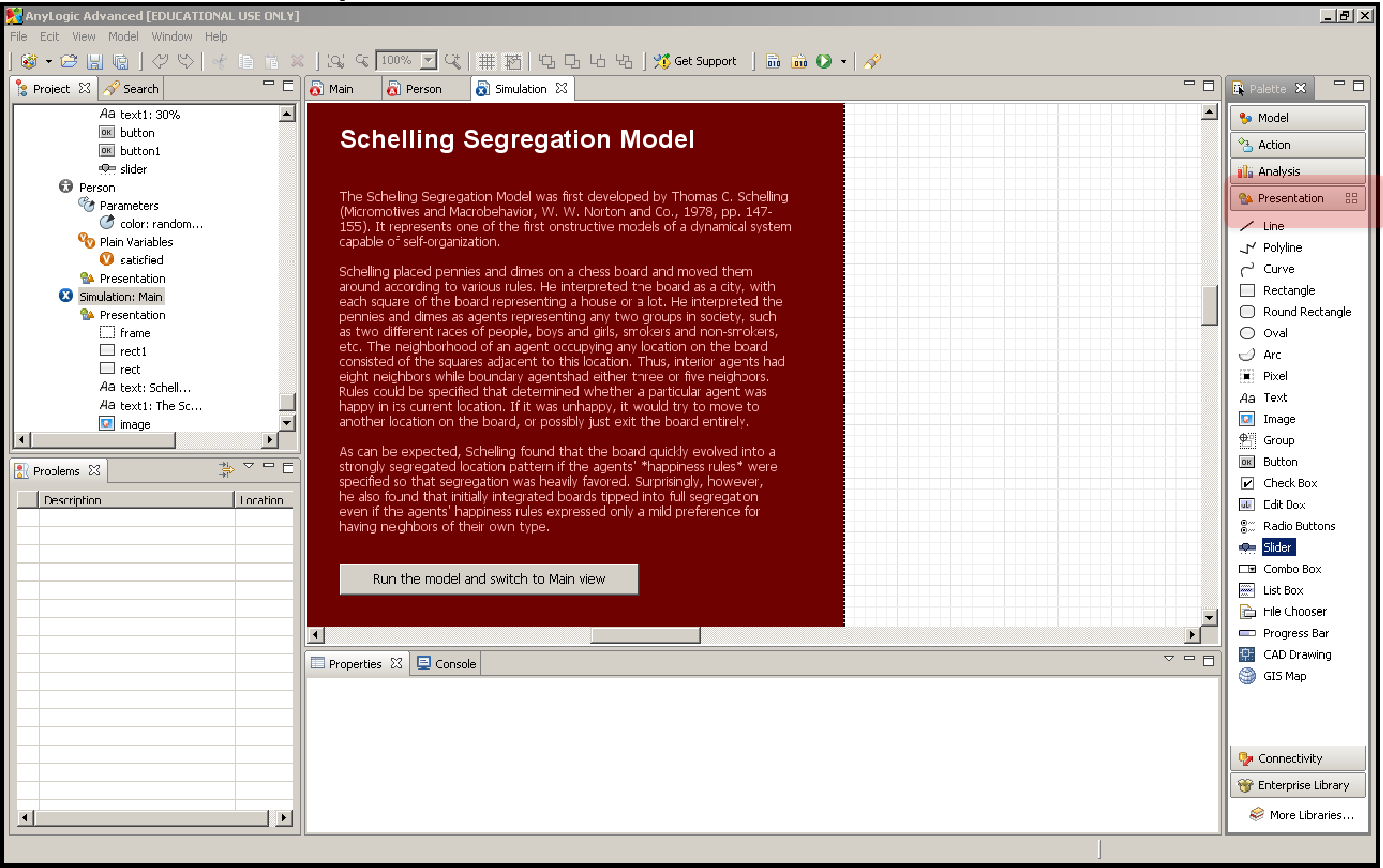
Experiment: Simulation Sets Parameter Assumptions



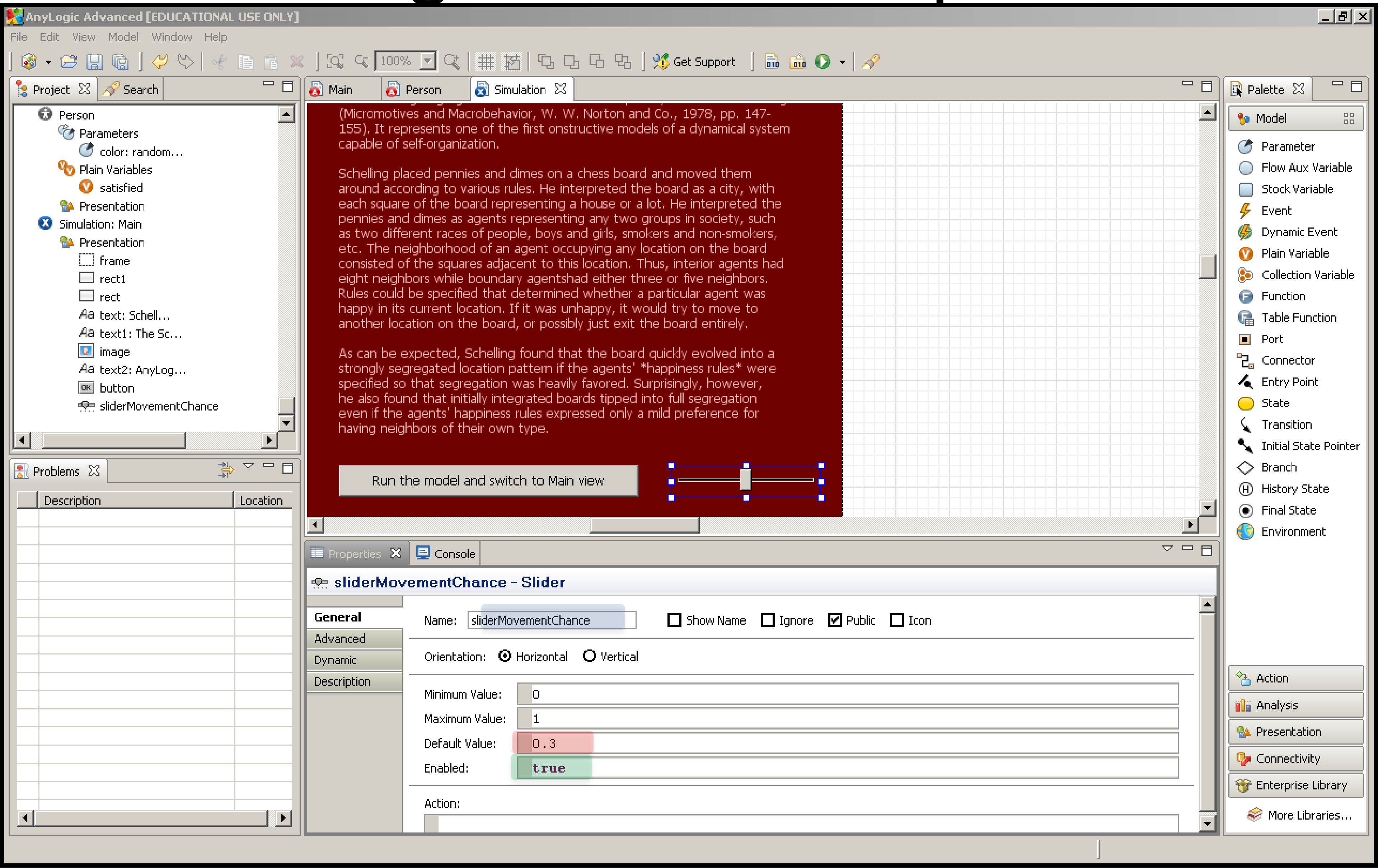
Add a Parameter to Main



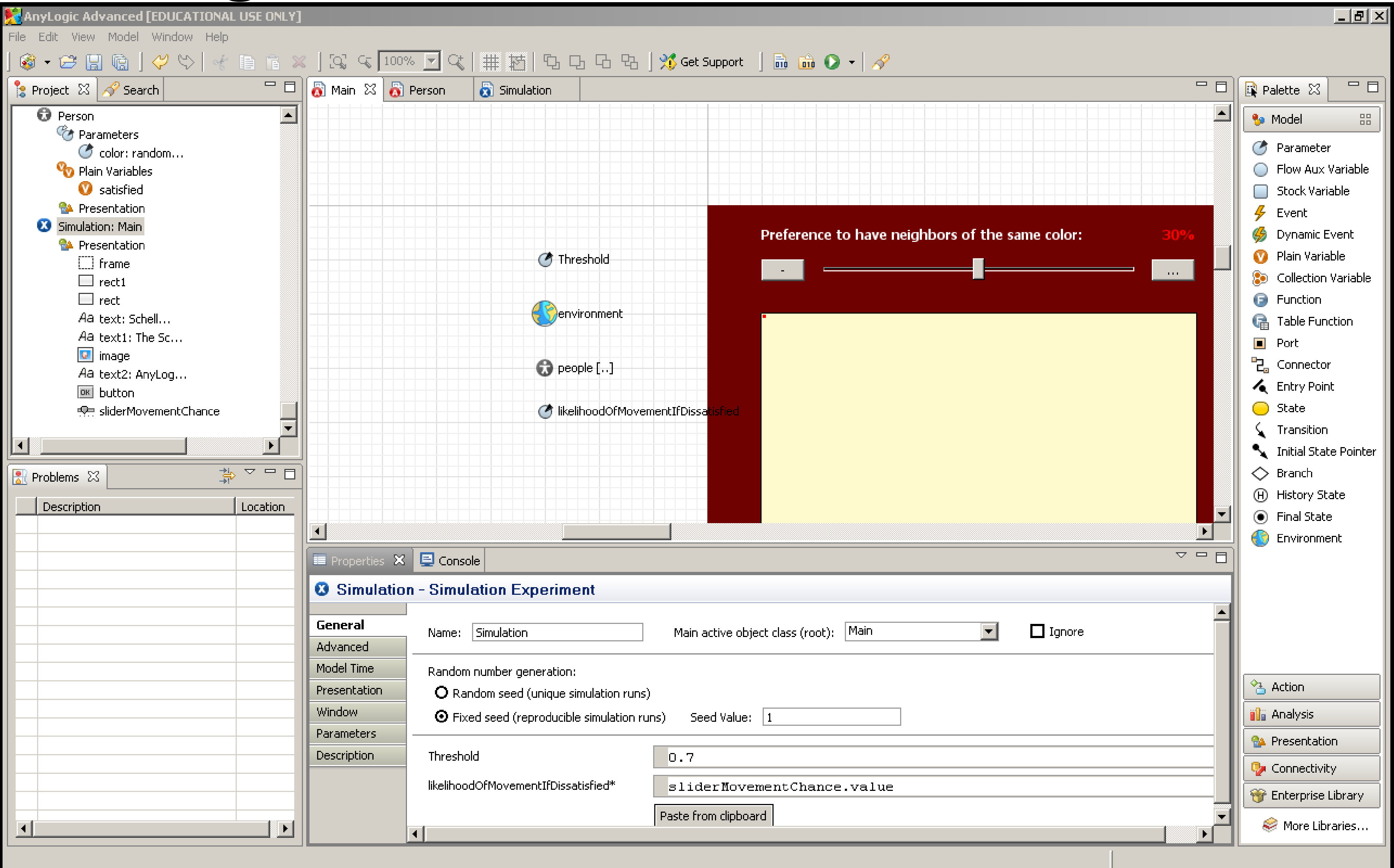
Experiment: Add a Slider!



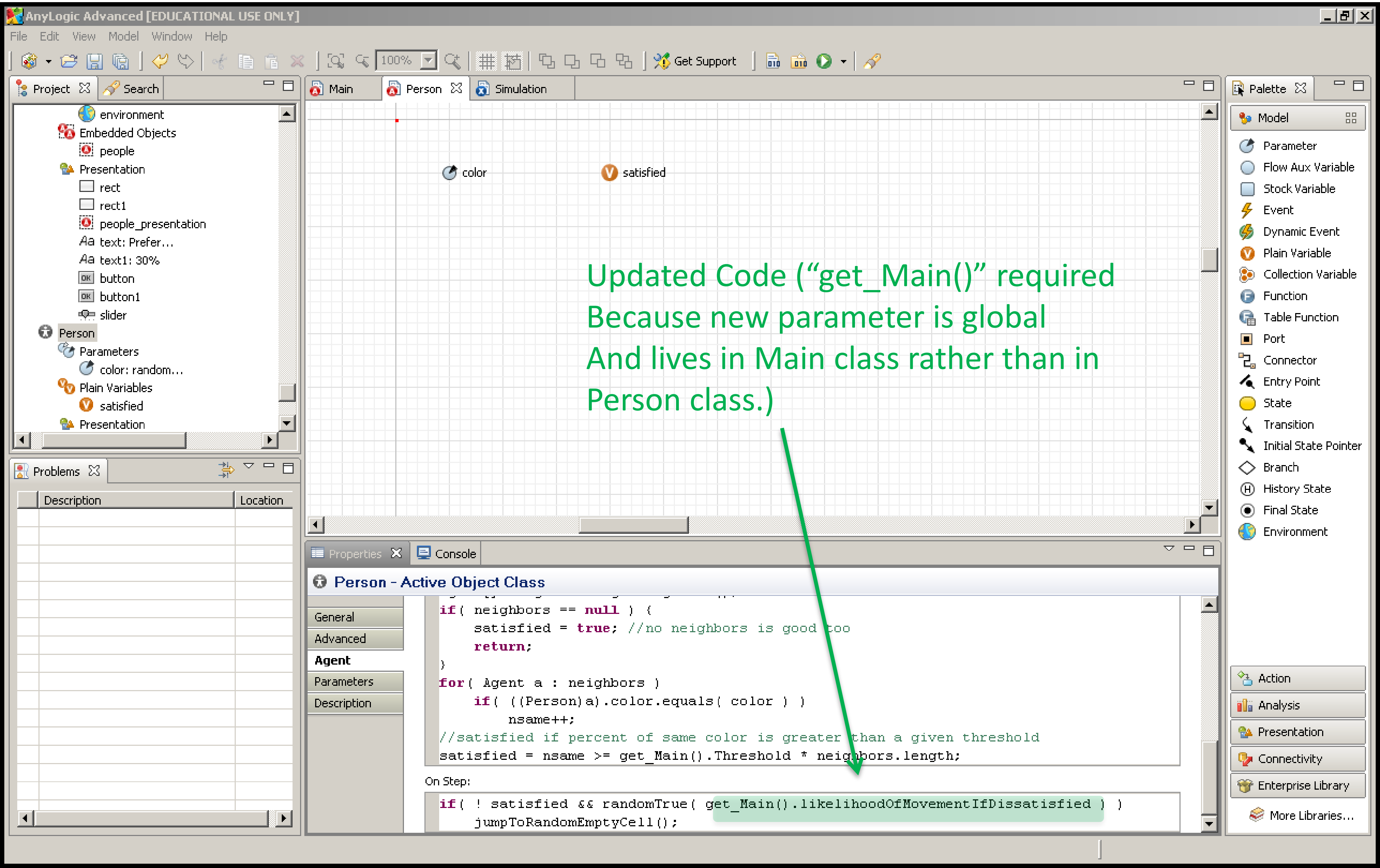
Setting the Slider Properties



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()

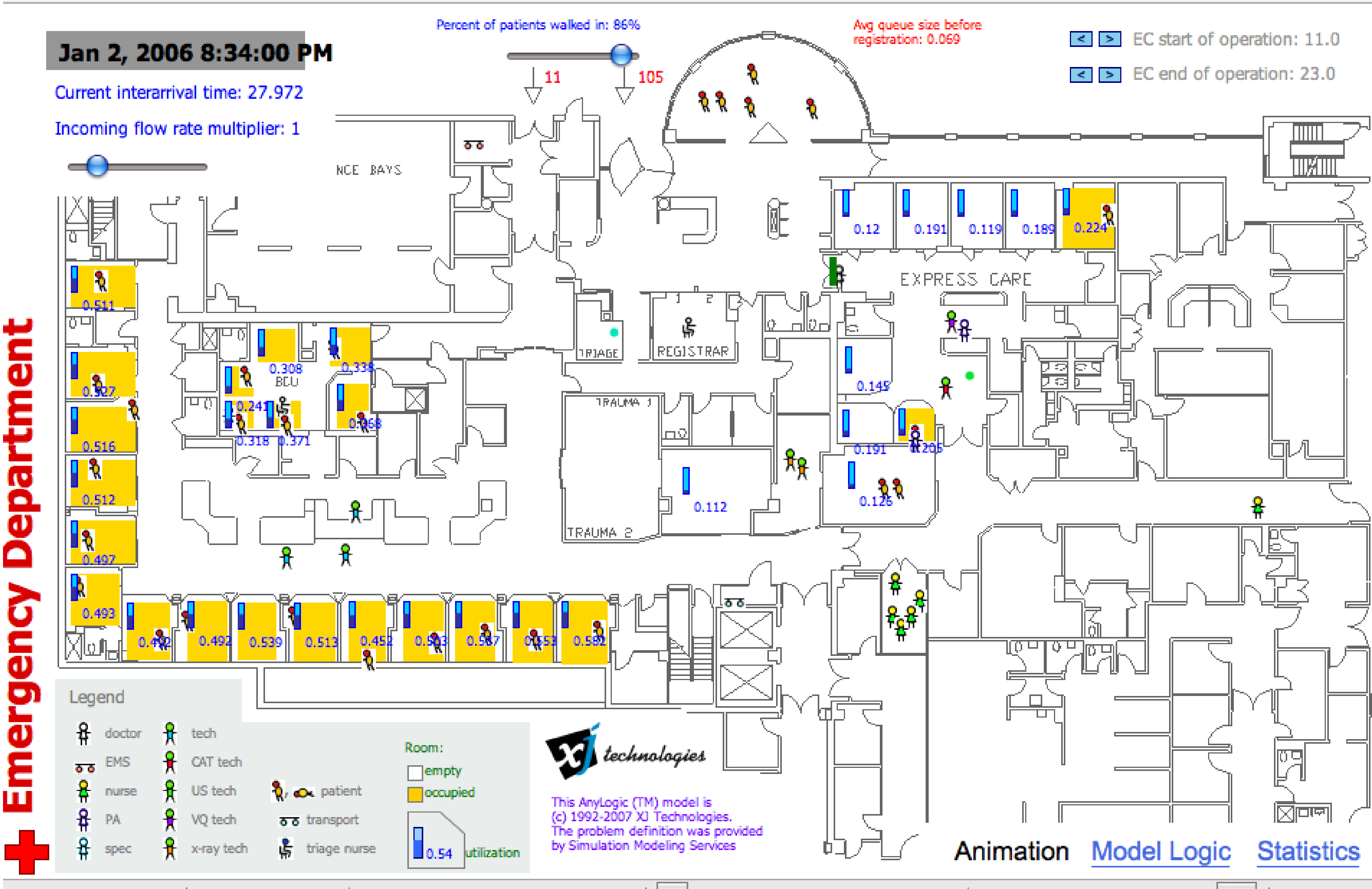
Important Distinction

- 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