Agent Spatial Embedding in 2D Landscapes (Bonus: Discrete Time)

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

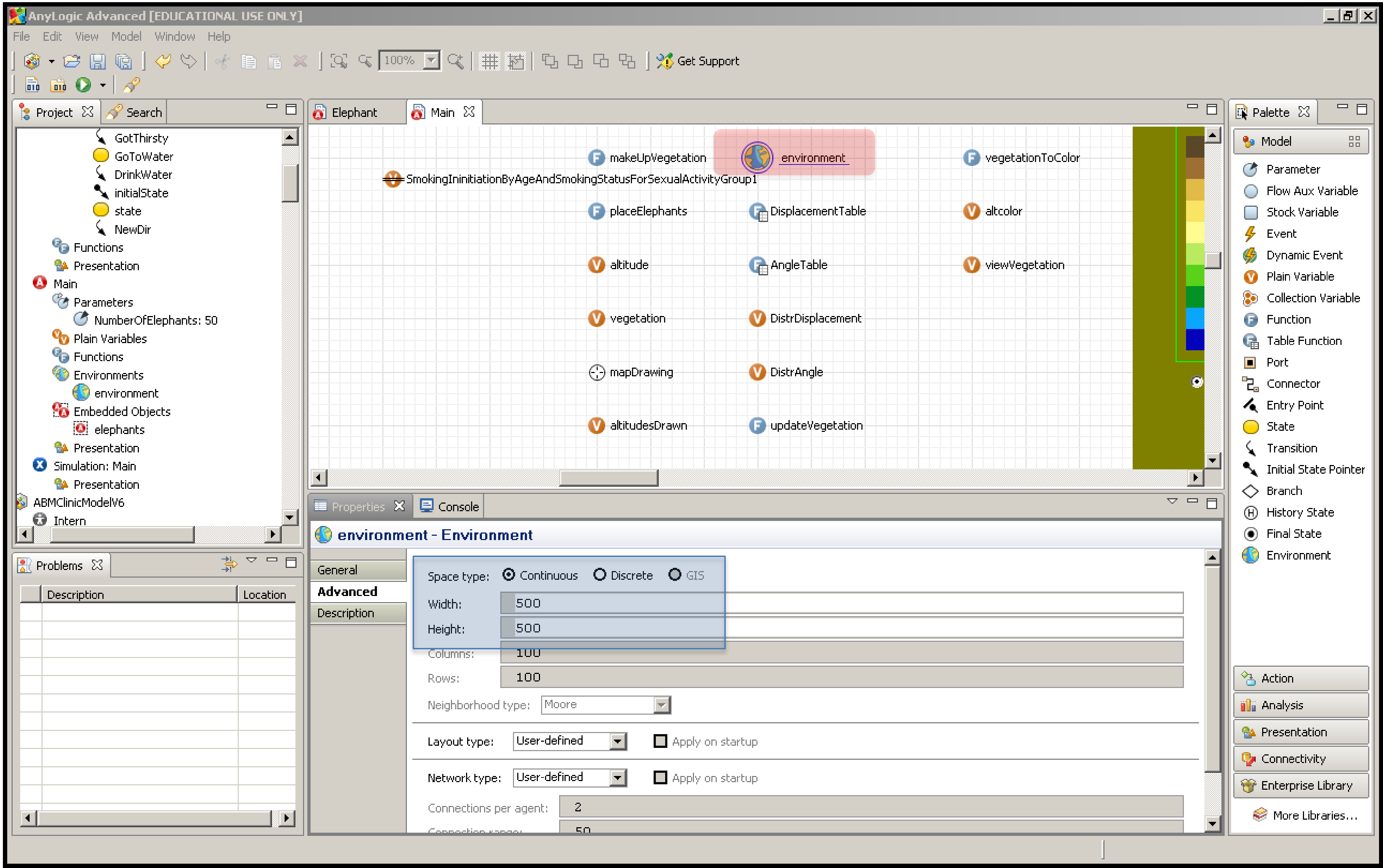
The Locus of Control: Environment

- 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



environment - Environment

Space type: ● Continuous ○ Discrete ○ GIS

Width: 500

Height: 500

Columns: 100

Rows: 100

Neighborhood type: Moore

Layout type: User-defined □ Apply on startup

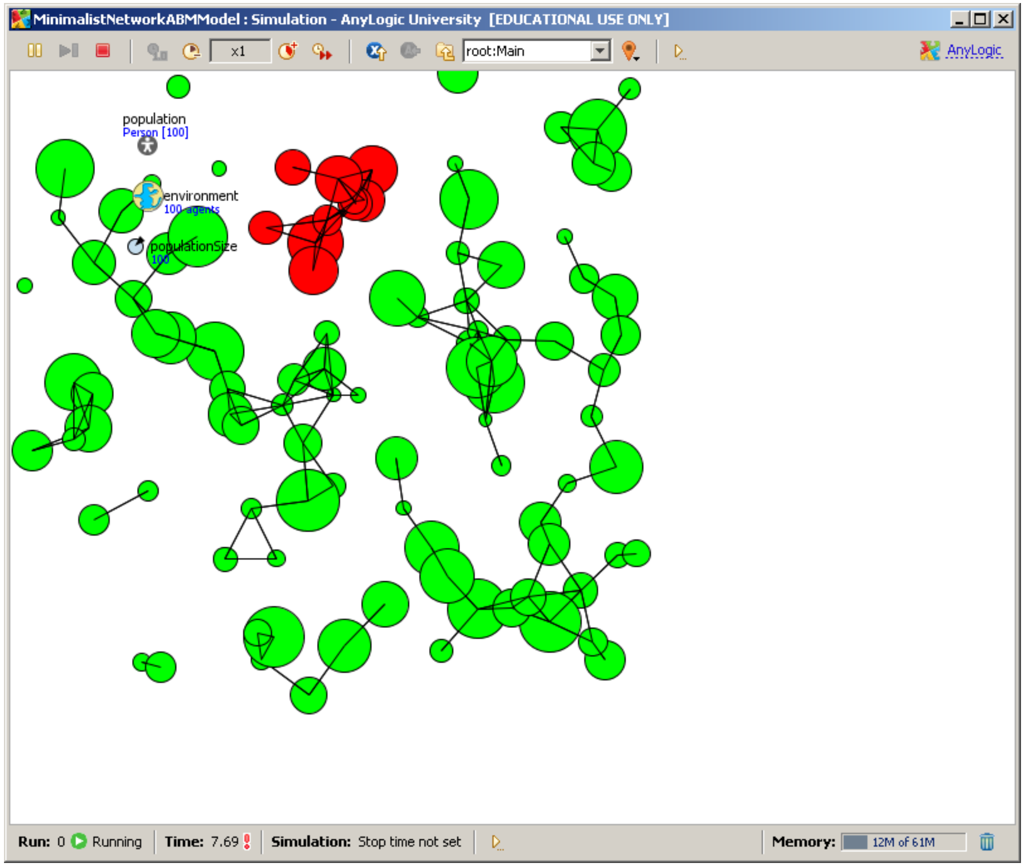
Network type: User-defined □ Apply on startup

Connections per agent: 2

Connection range: 50

Continuous Environment: Your Model

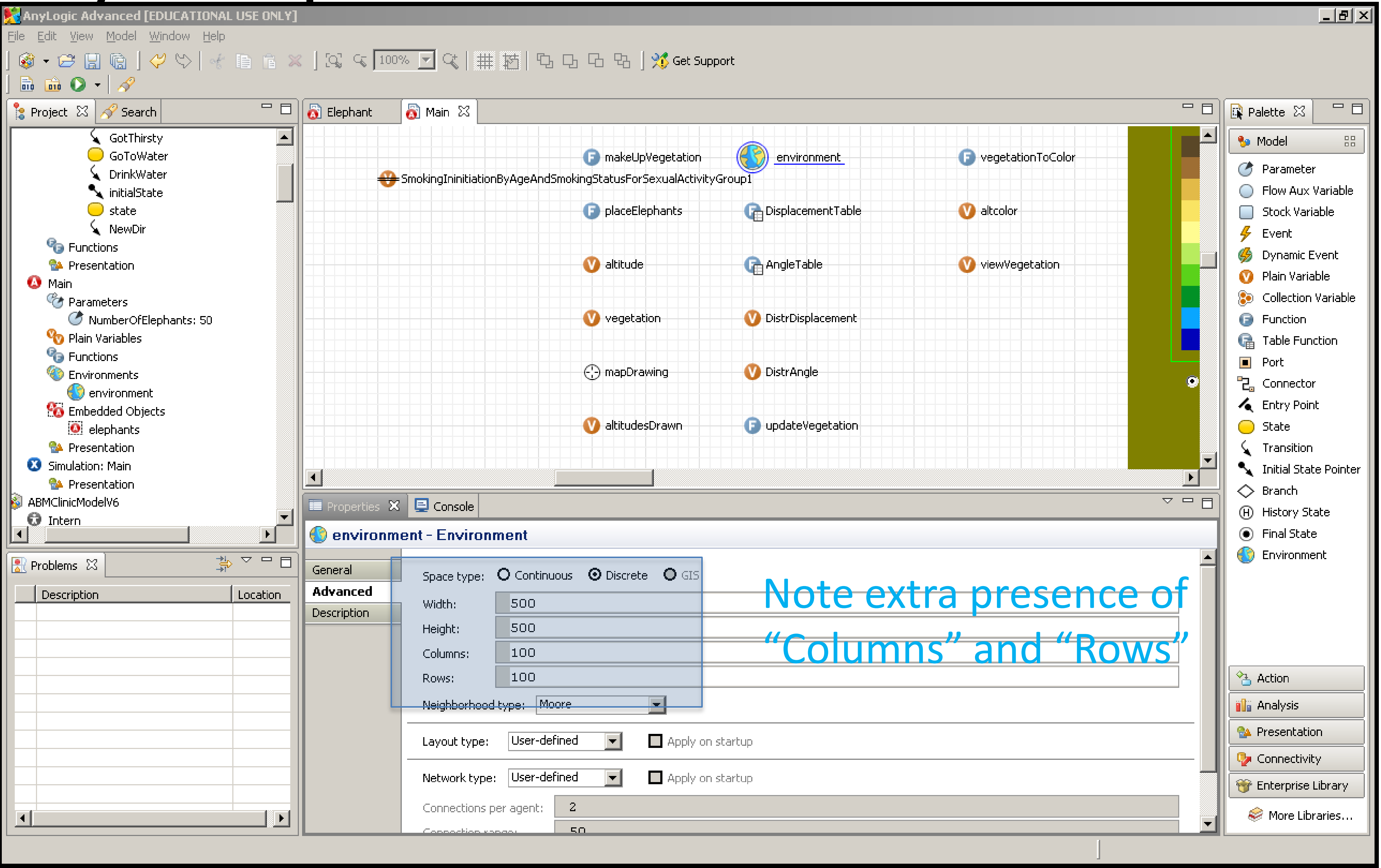
- We’ve already seen the continuous embedding in our built up model.



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



environment - Environment

Space type: ○ Continuous ● Discrete ○ GIS

Width: 500

Height: 500

Columns: 100

Rows: 100

Neighborhood type: Moore

Layout type: User-defined ☐ Apply on startup

Network type: User-defined ☐ Apply on startup

Connections per agent: 2

Connection range: 50

Note extra presence of “Columns” and “Rows”

Hands on Model Use Ahead

Load AnyLogic Sample Model: The Game of Life

The “Game” of Life: Background

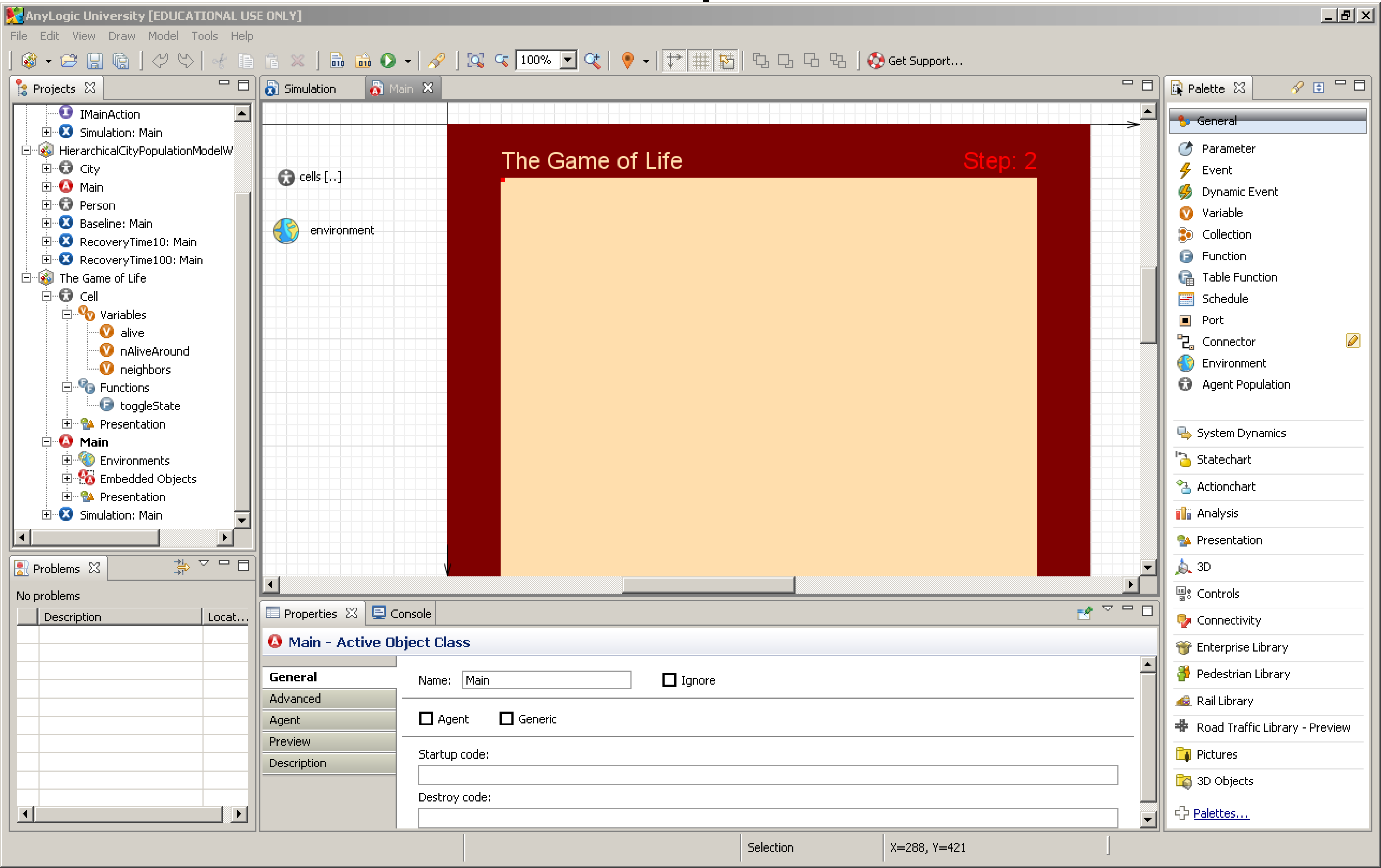
- Invented in 1970 by Mathematician Conway (modifying ideas from Von Neumann)
- Inspiration: Lifecourse of cells
  - Key dichotomy: A space contains a living element or not
  - Stylized rules for birth, death
- Cellular automaton: Uses Discrete Time (Steps) & Discrete Space (Cells) with evolving cell 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.



Main - Active Object Class

General
Advanced
Agent
Preview
Description

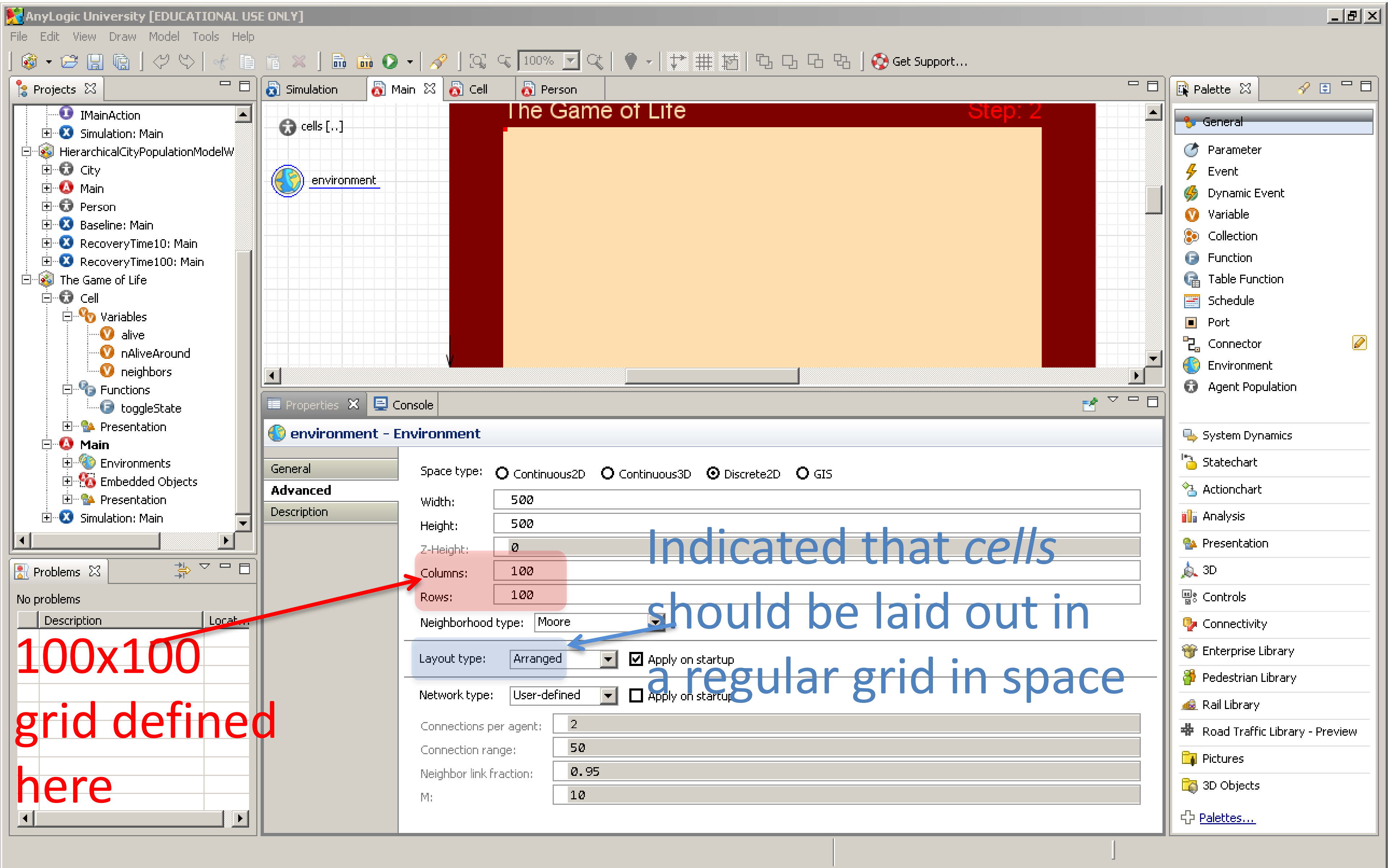
Name: Main ☐ Ignore

☐ Agent ☐ Generic

Startup code:

Destroy code:

Imposing the Regular 2D Structure



environment - Environment

General | **Advanced** | Description

Space type: ○ Continuous2D ○ Continuous3D ● Discrete2D ○ GIS

Width: 500

Height: 500

Z-Height: 0

Columns: 100

Rows: 100

Neighborhood type: Moore

Layout type: Arranged ☑ Apply on startup

Network type: User-defined □ Apply on startup

Connections per agent: 2

Connection range: 50

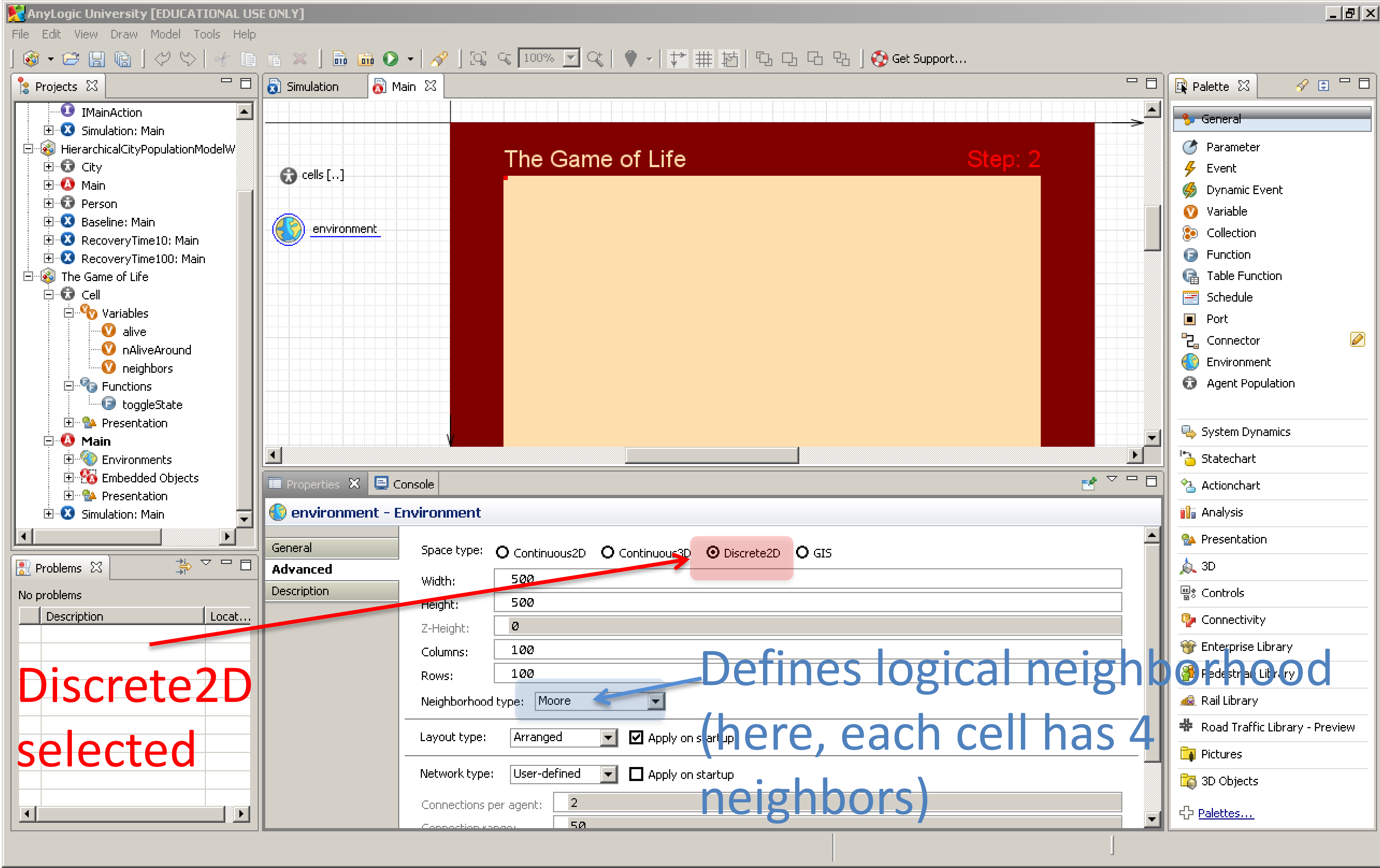
Neighbor link fraction: 0.95

M: 10

100x100 grid defined here

Indicated that *cells* should be laid out in a regular grid in space

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



environment - Environment

General
Advanced
Description

Space type: ○ Continuous2D ○ Continuous3D ● Discrete2D ○ GIS

Width: 500

Height: 500

Z-Height: 0

Columns: 100

Rows: 100

Neighborhood type: Moore

Layout type: Arranged ☑ Apply on startup

Network type: User-defined □ Apply on startup

Connections per agent: 2

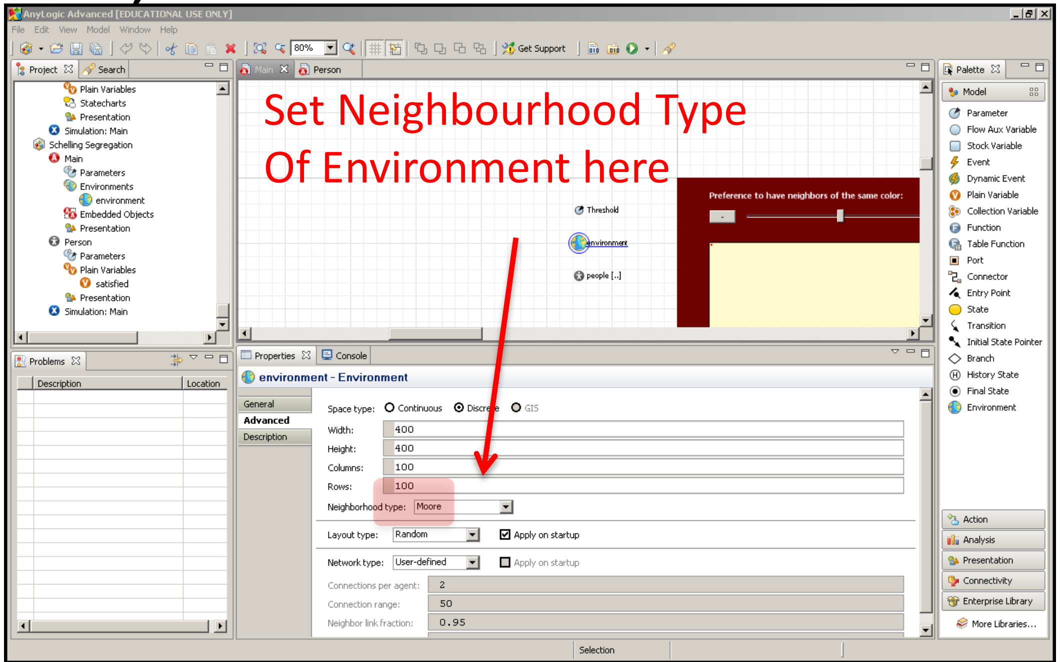
Connection range: 50

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



Set Neighbourhood Type Of Environment here

environment - Environment

Space type: ○ Continuous ● Discrete ○ GIS

Width: 400

Height: 400

Columns: 100

Rows: 100

Neighborhood type: Moore

Layout type: Random ☑ Apply on startup

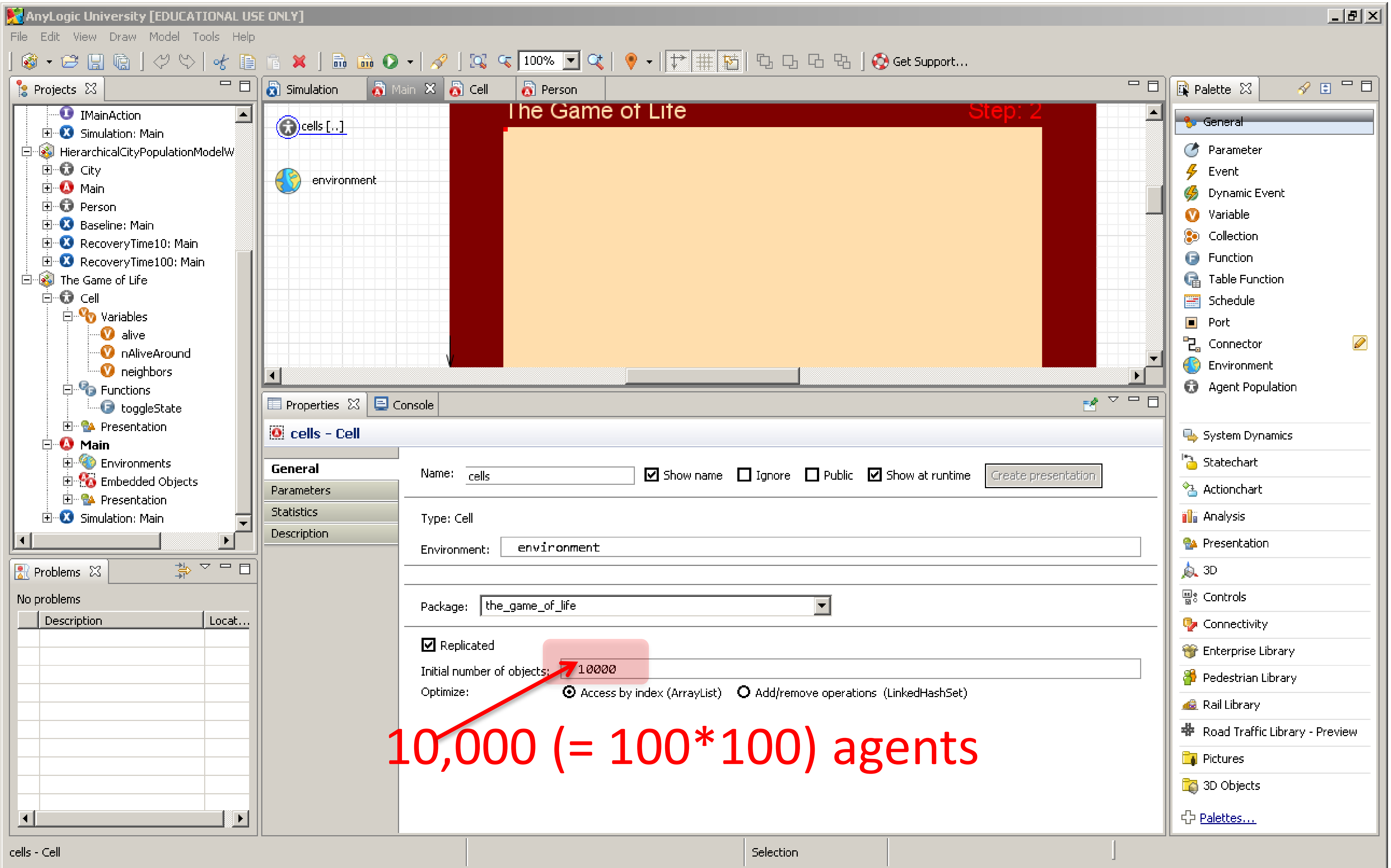
Network type: User-defined ☐ Apply on startup

Connections per agent: 2

Connection range: 50

Neighbor link fraction: 0.95

Population: One Cell Agent per Grid Point



cells - Cell

Name: cells ☑ Show name ☐ Ignore ☐ Public ☑ Show at runtime

Type: Cell

Environment: environment

Package: the_game_of_life

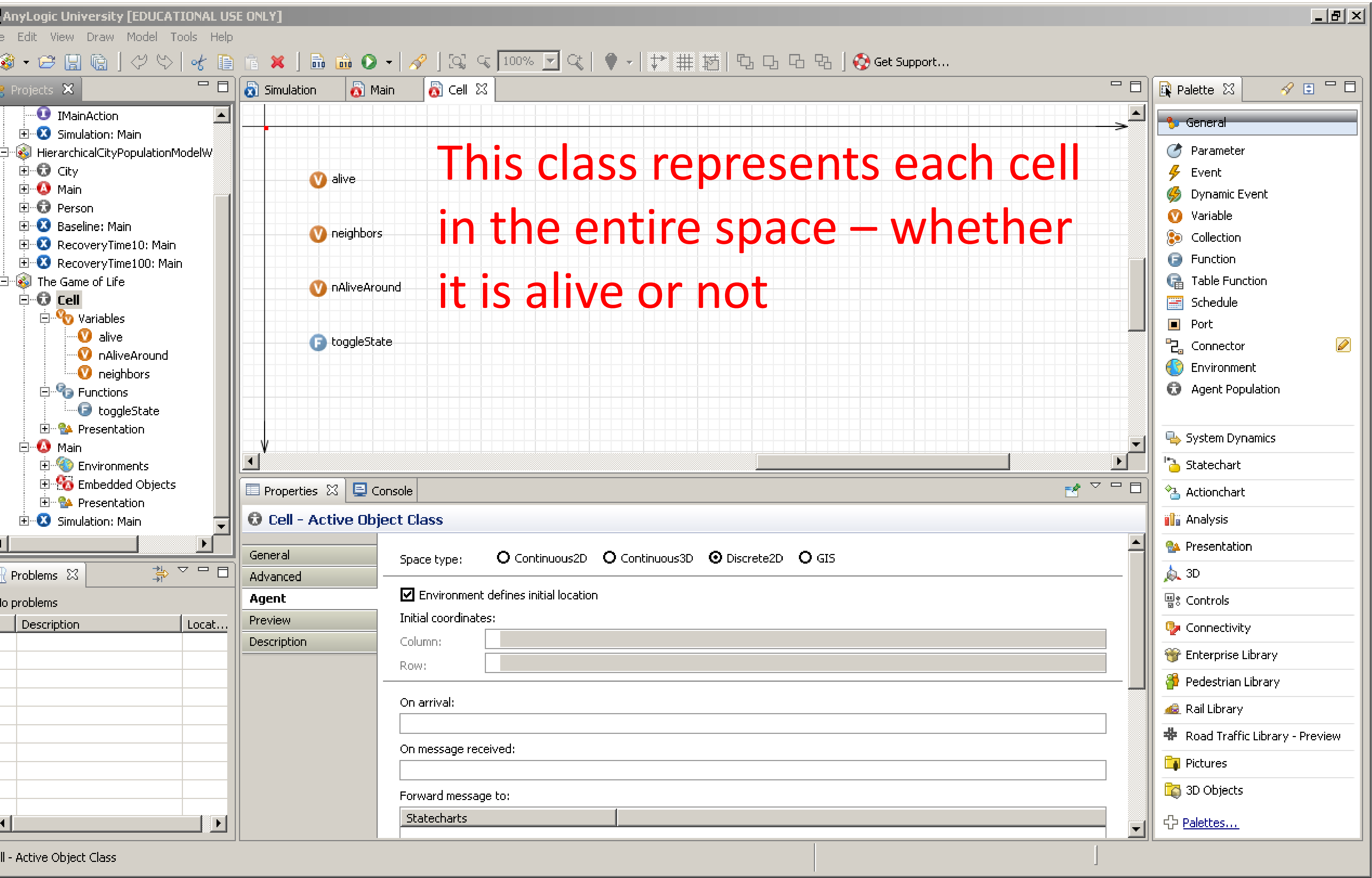
☑ Replicated

Initial number of objects: 10000

Optimize: ● Access by index (ArrayList) ○ Add/remove operations (LinkedHashSet)

10,000 (= 100*100) agents

View the “Cell” Class



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

Cell - Active Object Class

Space type: ○ Continuous2D ○ Continuous3D ● Discrete2D ○ GIS

☑ Environment defines initial location

Initial coordinates:

Column:

Row:

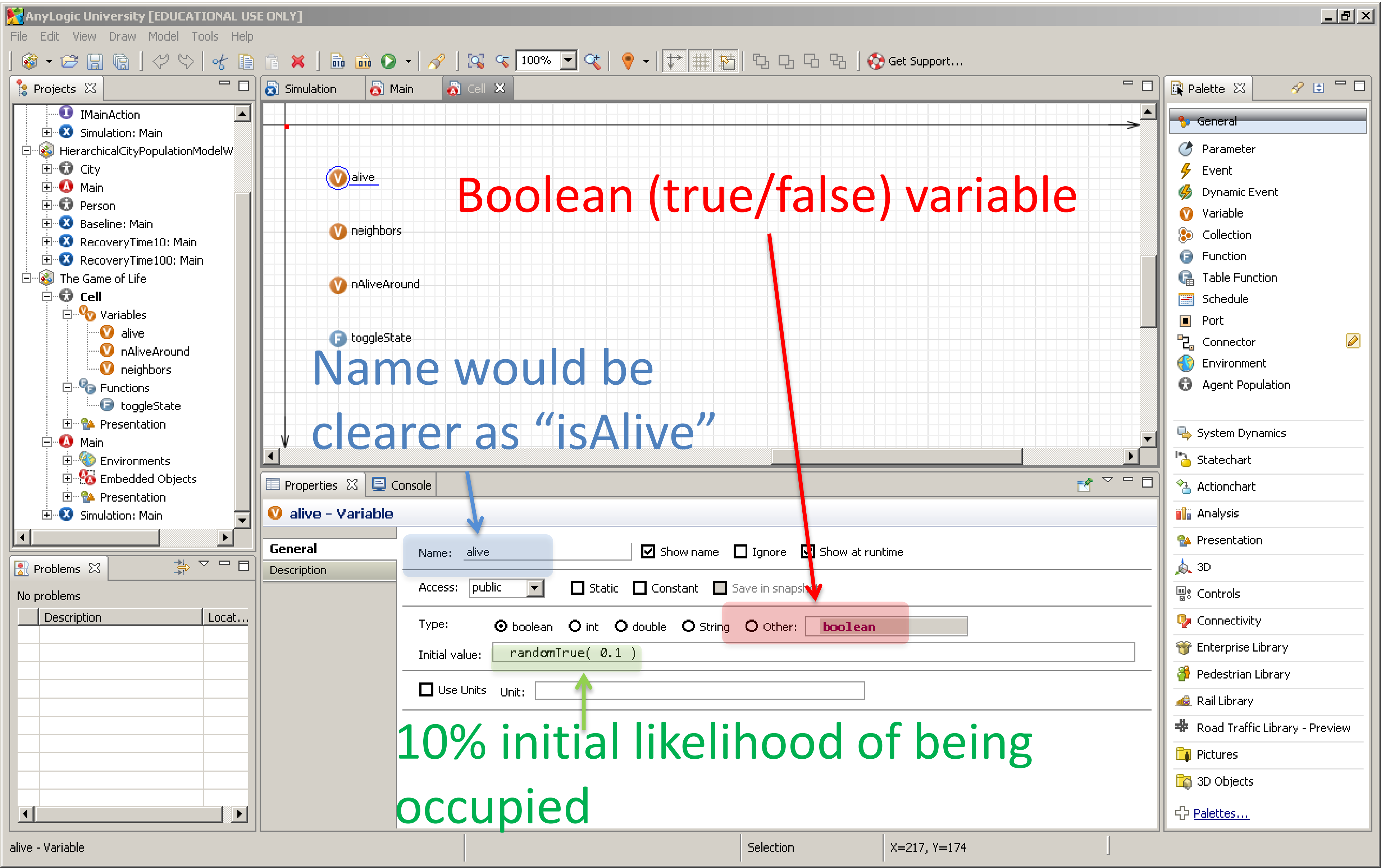
On arrival:

On message received:

Forward message to:

Statecharts

Cell Variables: “alive”



Boolean (true/false) variable

Name would be clearer as “isAlive”

alive - Variable

Name: alive ☑ Show name □ Ignore ☑ Show at runtime

Access: public □ Static □ Constant □ Save in snapsh

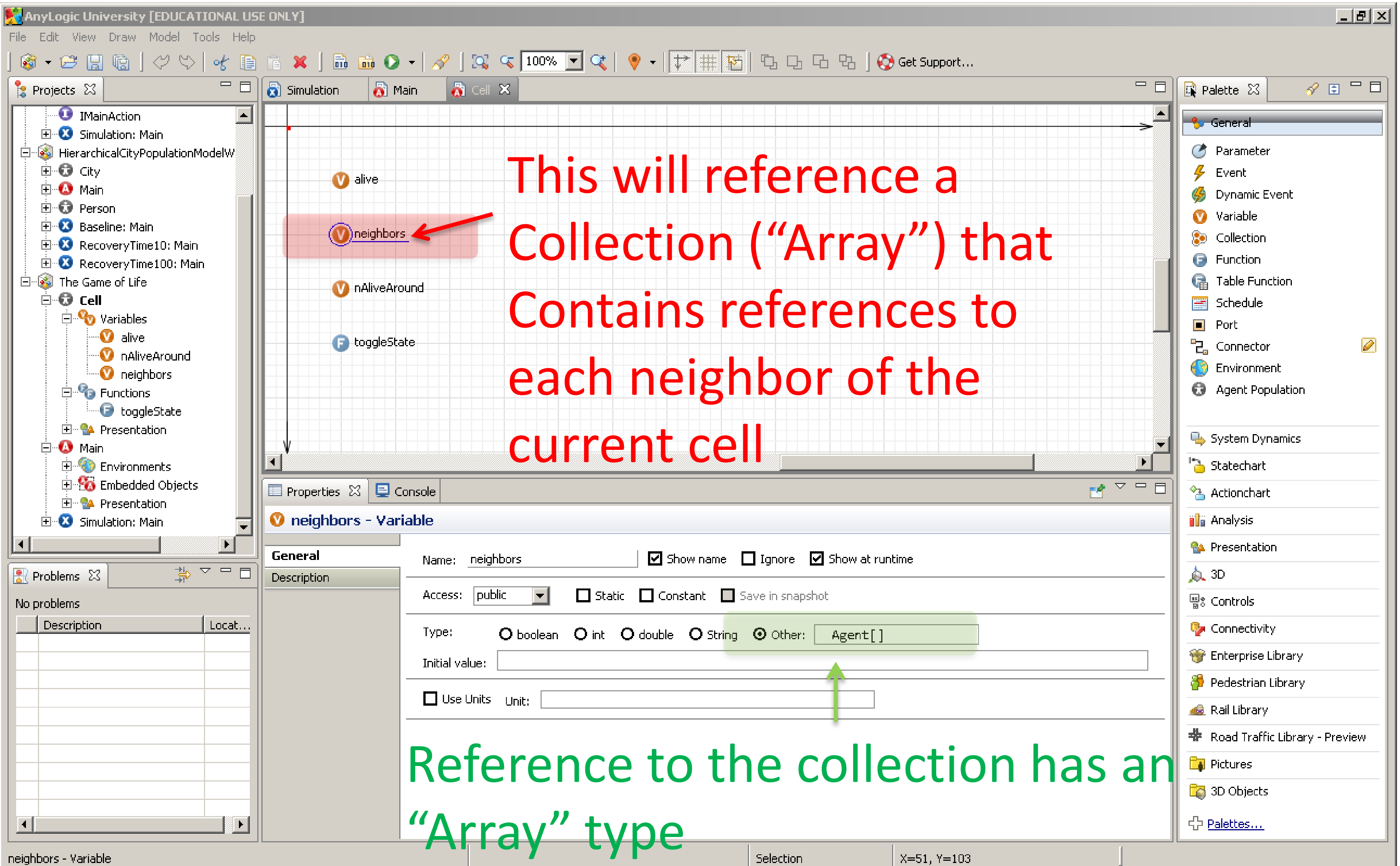
Type: ● boolean ○ int ○ double ○ String ○ Other: boolean

Initial value: `randomTrue( 0.1 )`

□ Use Units Unit:

10% initial likelihood of being occupied

Cell Variables: “neighbors”



neighbors - Variable

Name: neighbors ☑ Show name ☐ Ignore ☑ Show at runtime

Access: public ☐ Static ☐ Constant ☐ Save in snapshot

Type: ○ boolean ○ int ○ double ○ String ● Other: Agent[]

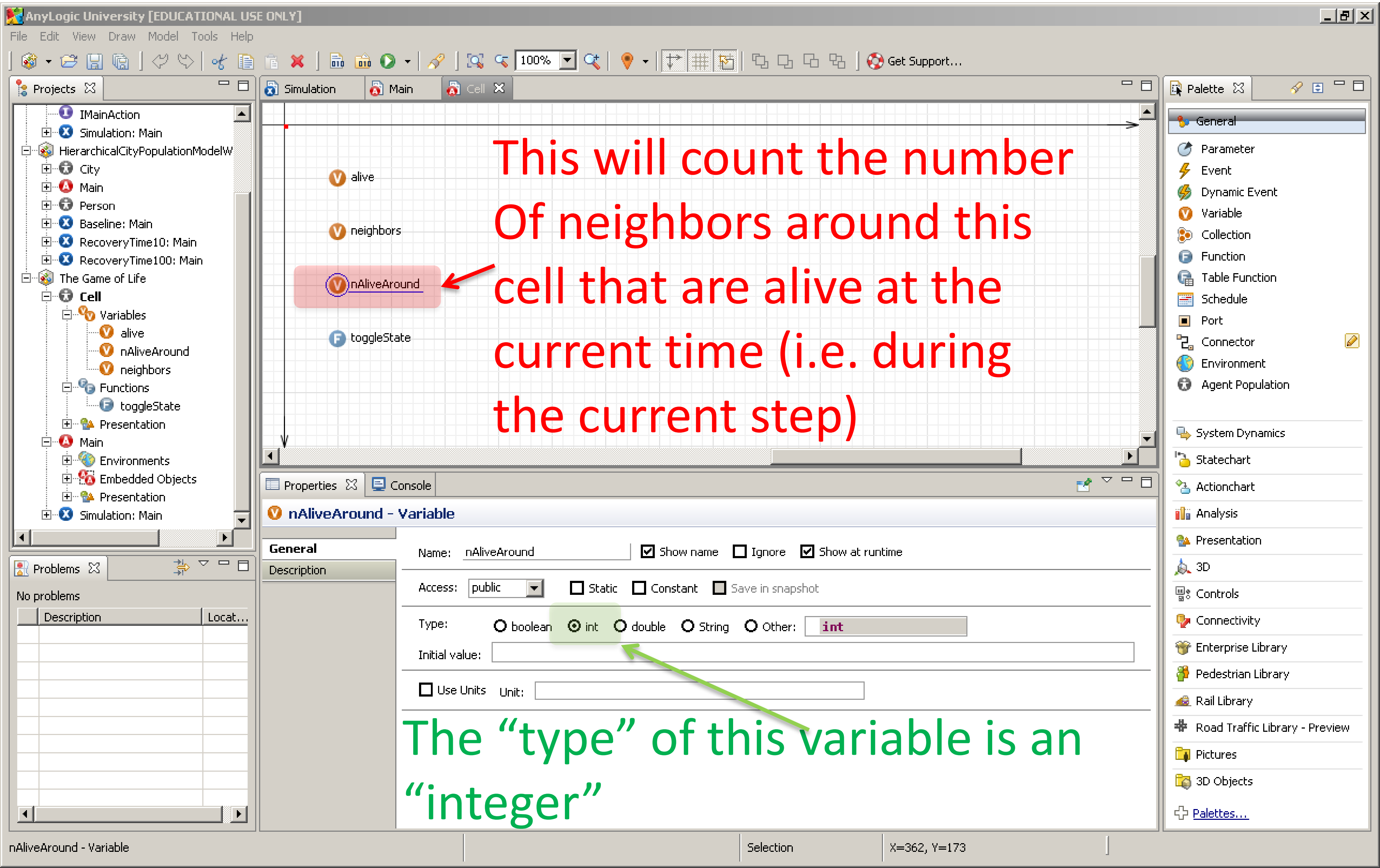
Initial value:

☐ Use Units Unit:

This will reference a Collection (“Array”) that Contains references to each neighbor of the current cell

Reference to the collection has an “Array” type

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)

nAliveAround - Variable

Name: nAliveAround ☑ Show name □ Ignore ☑ Show at runtime

Access: public □ Static □ Constant □ Save in snapshot

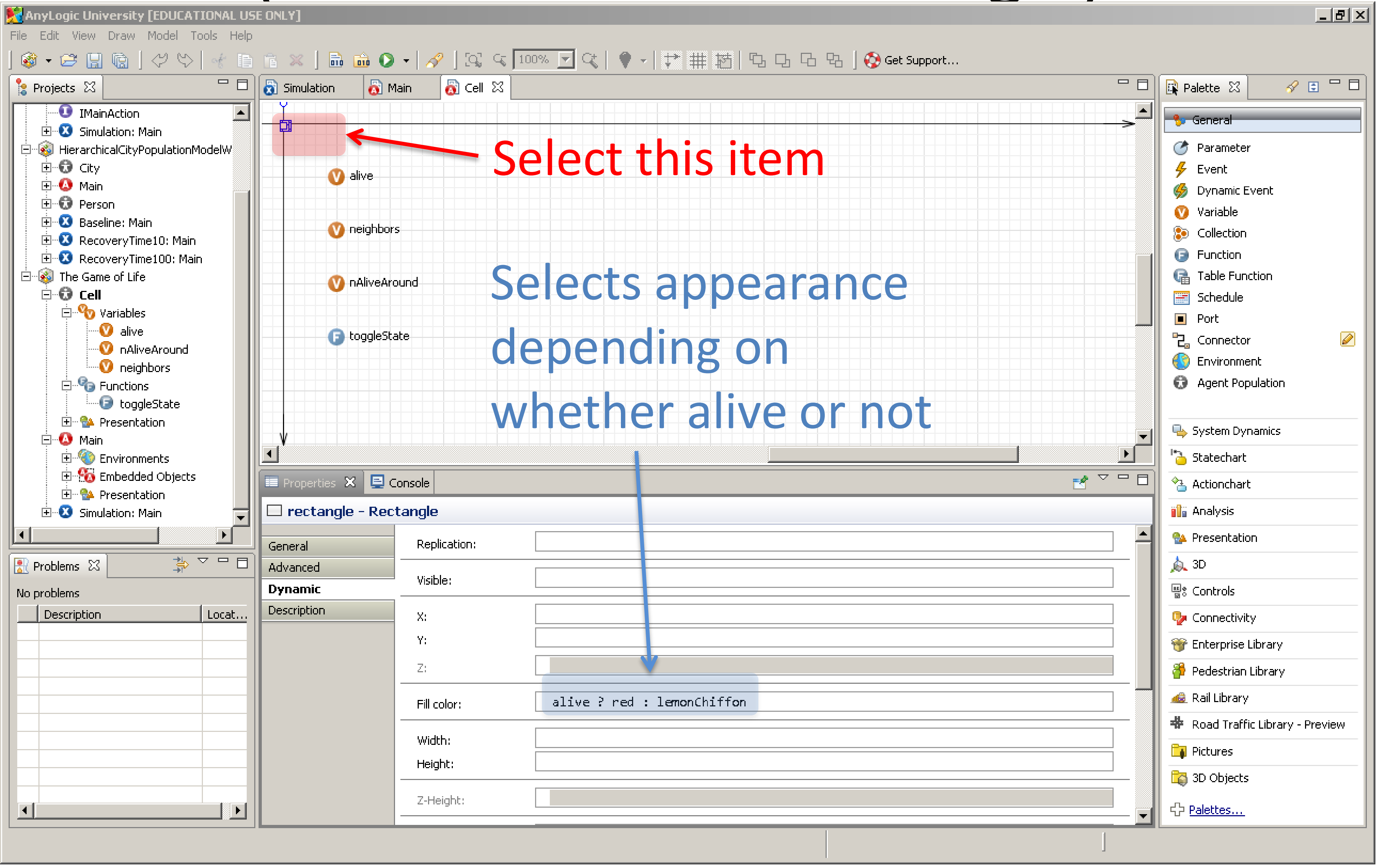
Type: ○ boolean ● int ○ double ○ String ○ Other: int

Initial value:

□ Use Units Unit:

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

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



Select this item

Selects appearance depending on whether alive or not

rectangle - Rectangle

Replication:

Visible:

X:

Y:

Z:

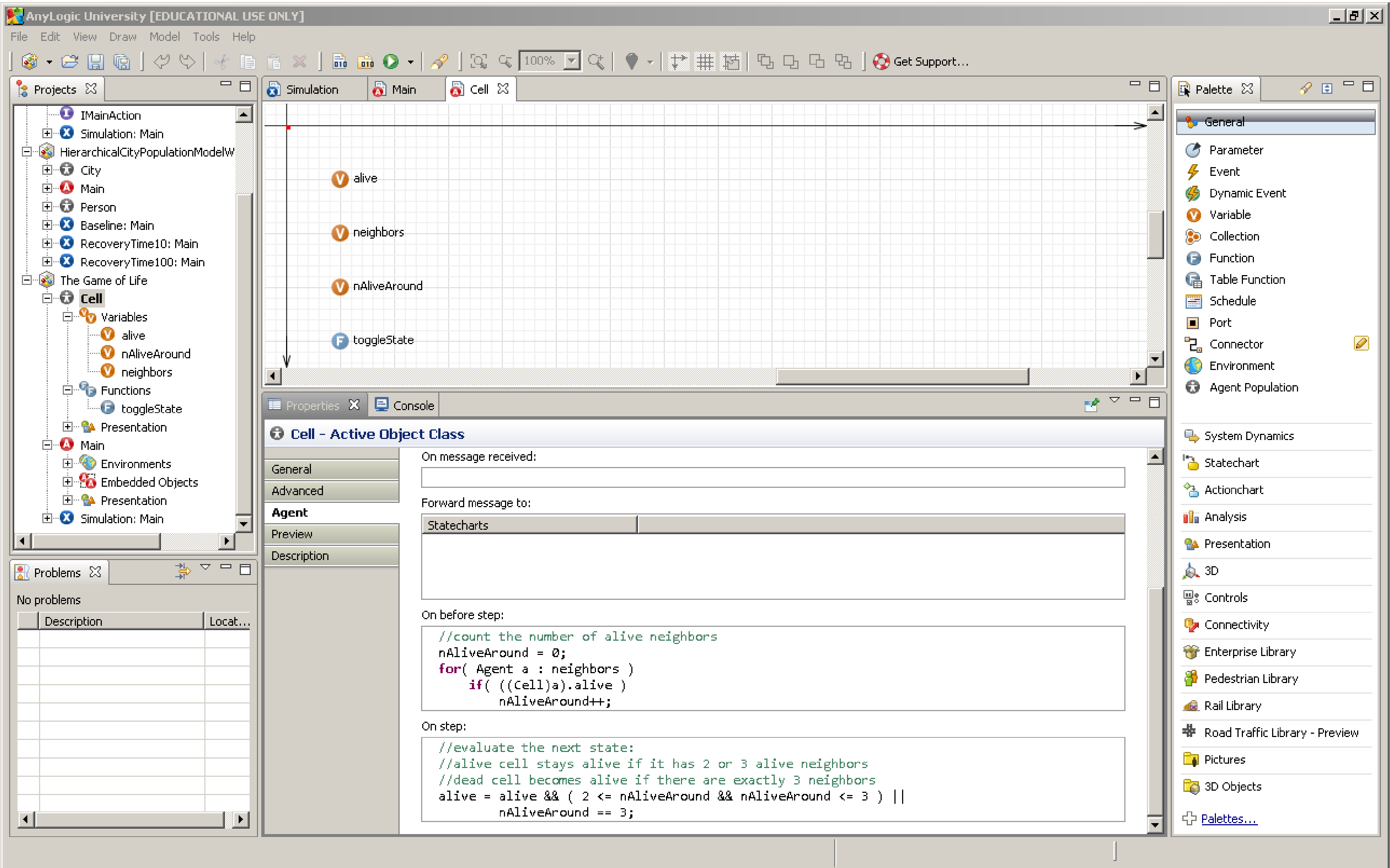
Fill color: `alive ? red : lemonChiffon`

Width:

Height:

Z-Height:

Cell Update Logic
(“Agent” Properties of “Cell”)



Cell - Active Object Class

On message received:

Forward message to:

Statecharts

On before step:

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

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;
```

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

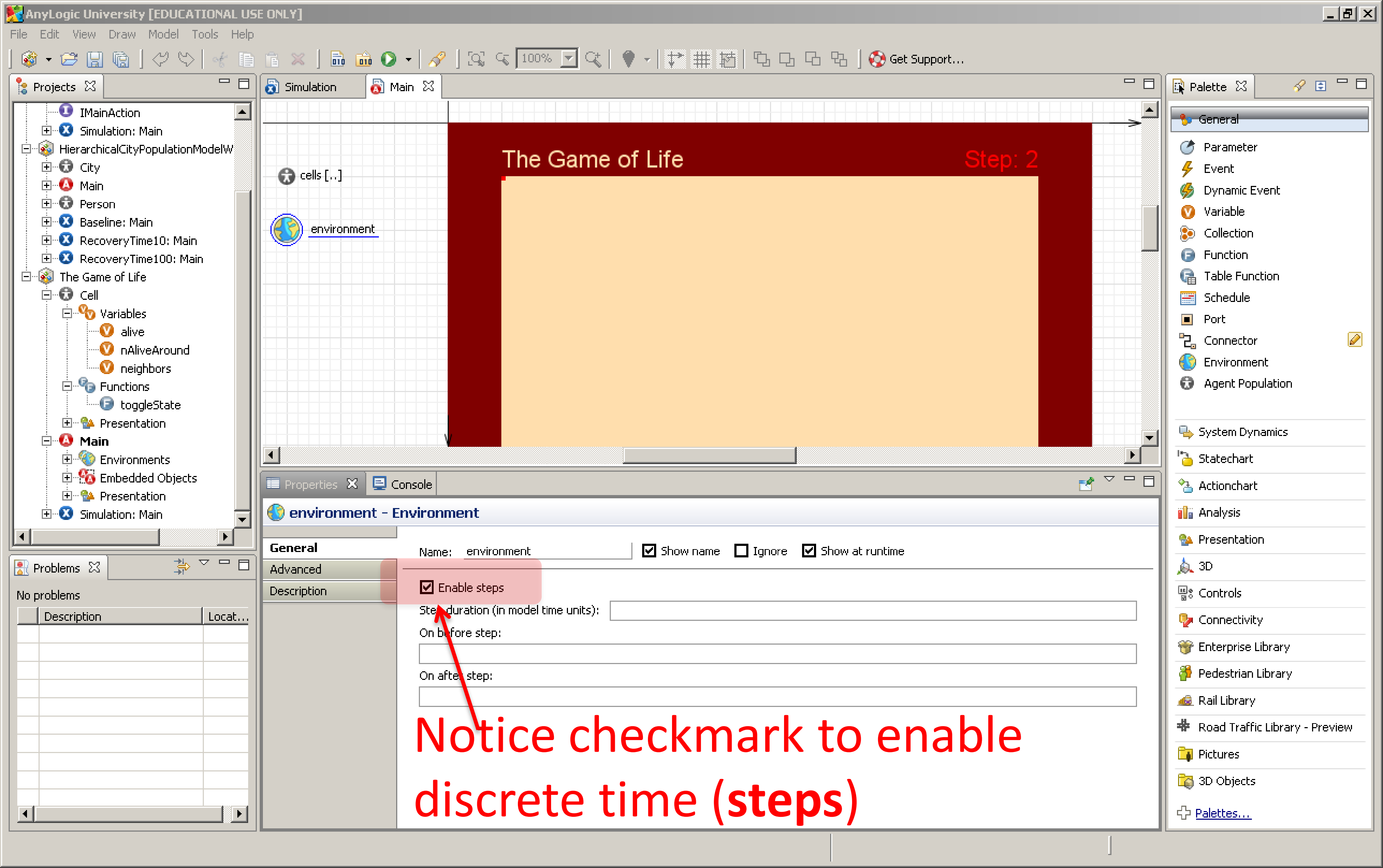
Two Key Models of Time in Anylogic: Discrete (Synchronous) Time

- 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**



environment - Environment

Name: environment ☑ Show name ☐ Ignore ☑ Show at runtime

☑ Enable steps

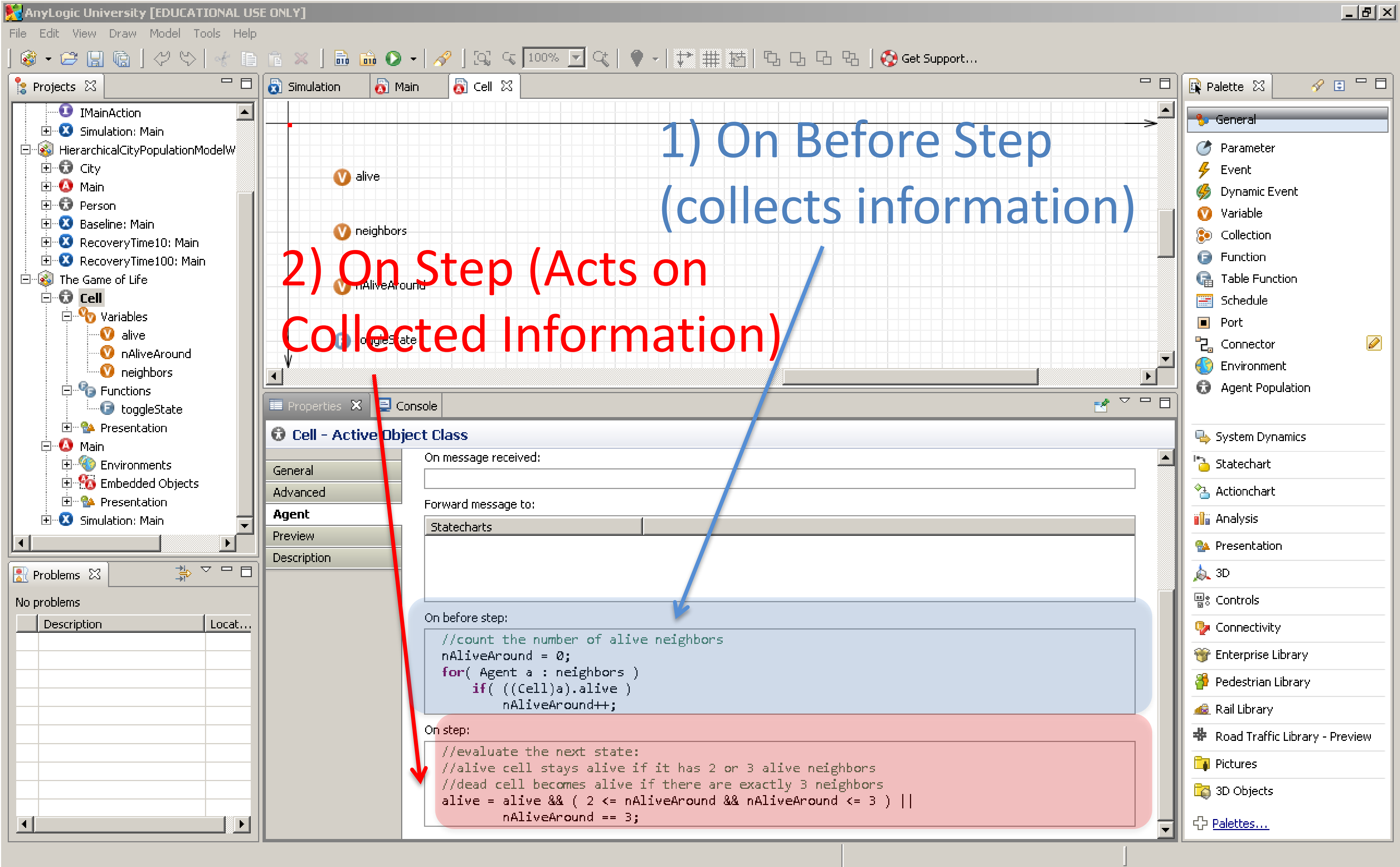
Step duration (in model time units):

On before step:

On after step:

Notice checkmark to enable discrete time (**steps**)

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



1) On Before Step (collects information)

2) On Step (Acts on Collected Information)

On message received:

Forward message to:

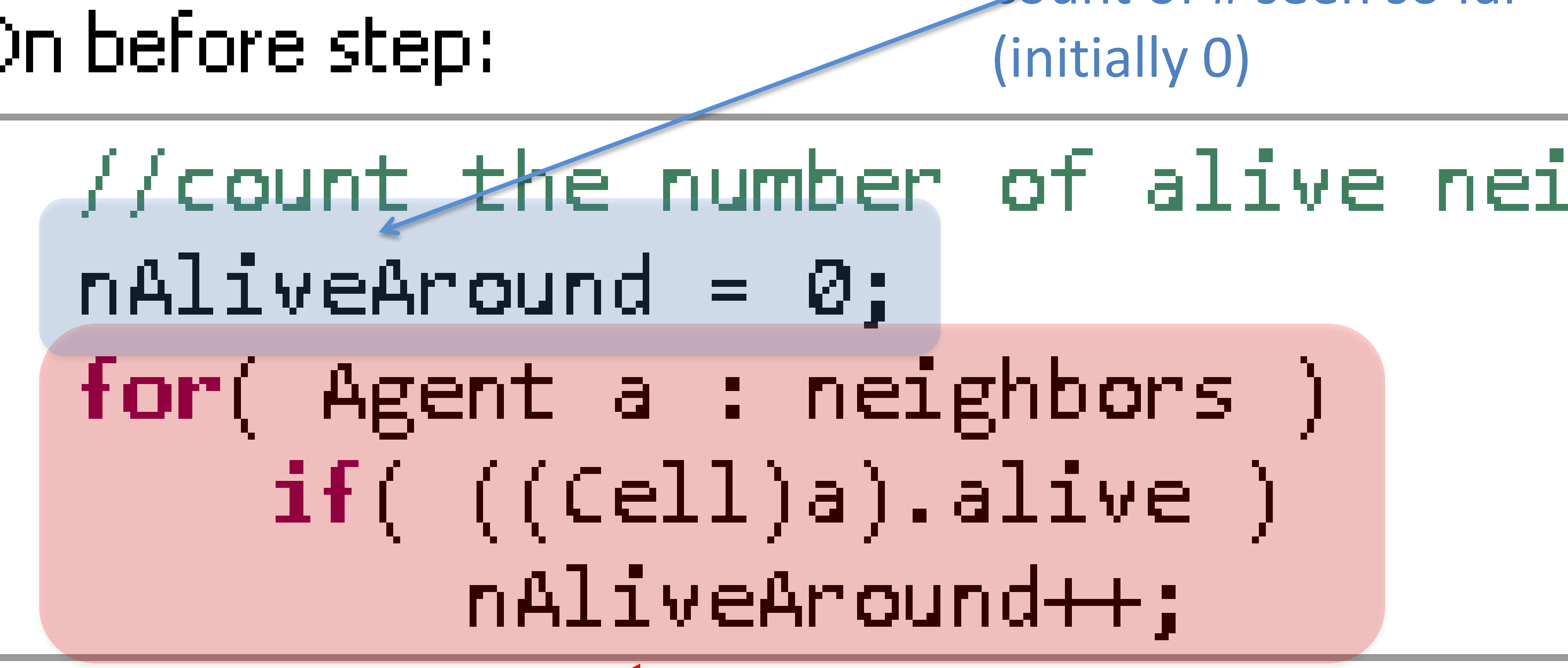
On before step:

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

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;
```

On Before Step: Collecting the Information



On before step:

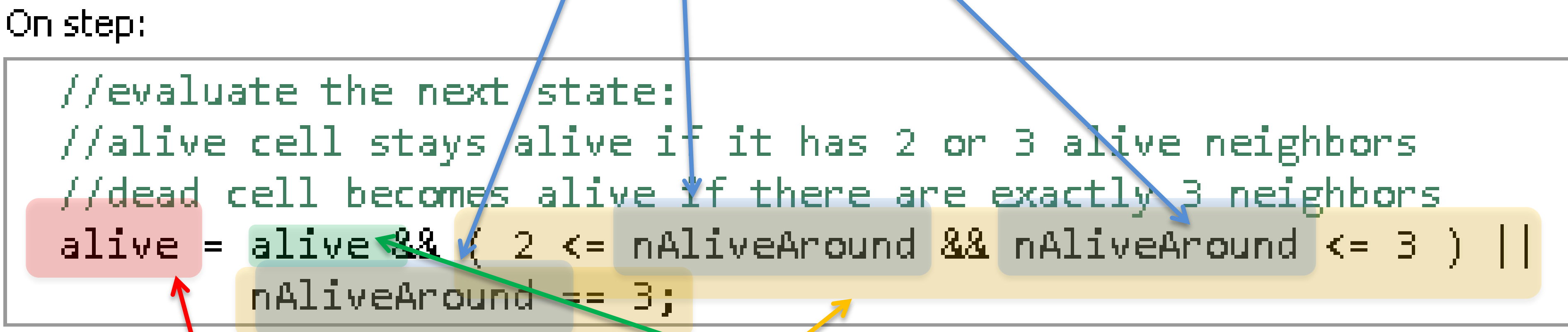
```
//count the number of alive nei
nAliveAround = 0;
for( Agent a : neighbors )
    if( ((Cell)a).alive )
        nAliveAround++;
```

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

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”

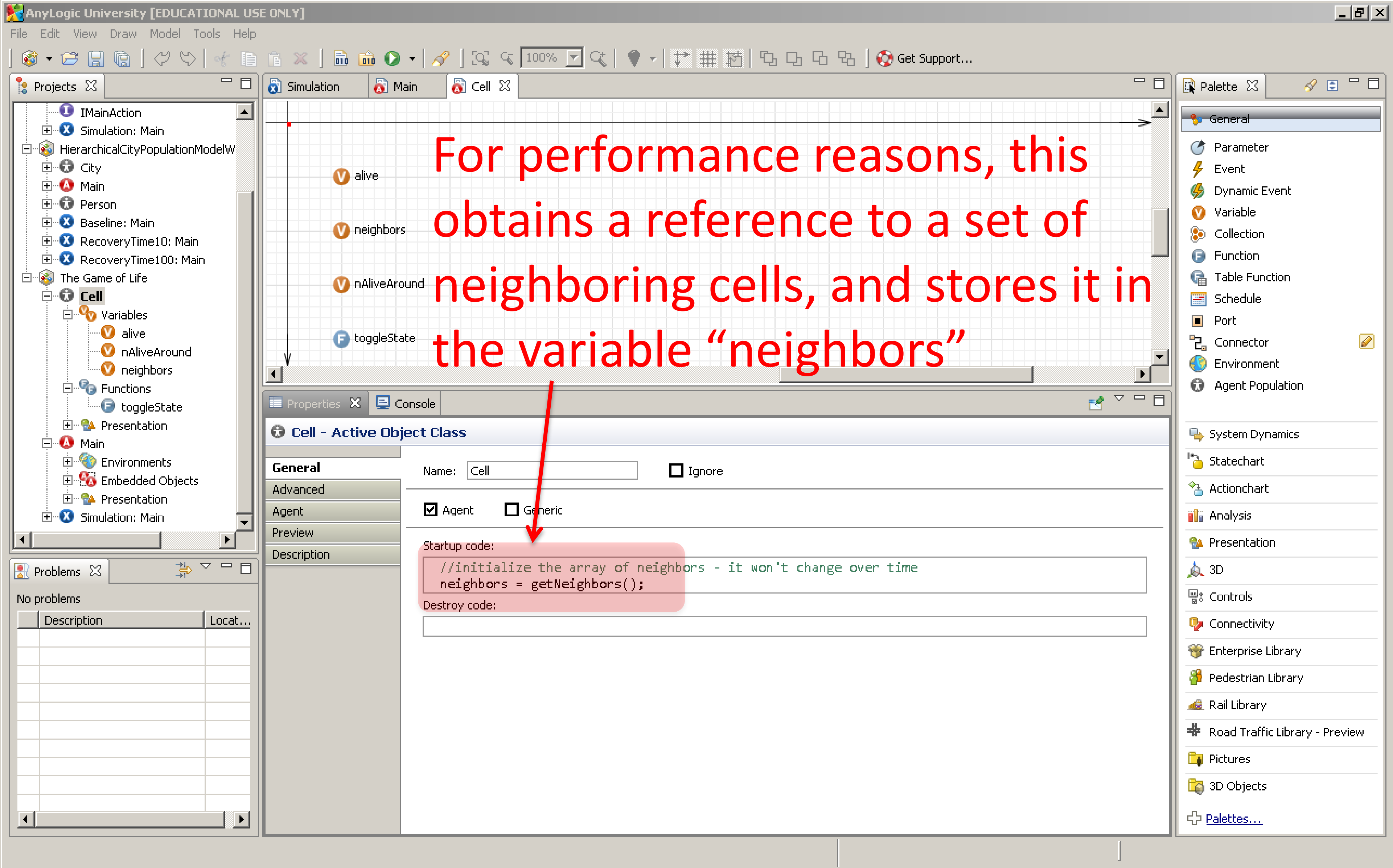


On step:

```
  //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



For performance reasons, this obtains a reference to a set of neighboring cells, and stores it in the variable “neighbors”

Cell - Active Object Class

Name: Cell □ Ignore

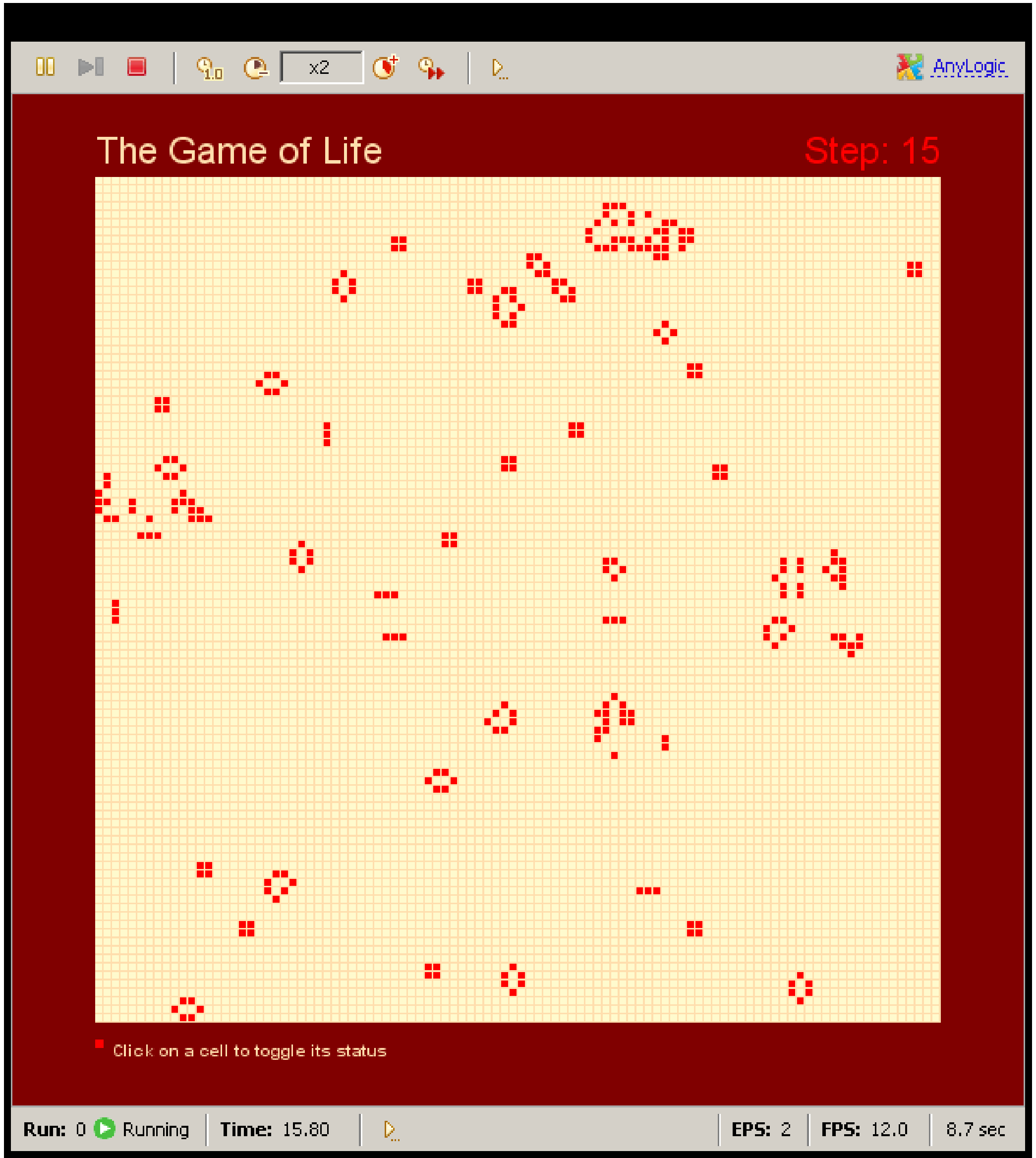
☑ Agent □ Generic

Startup code:

```
//initialize the array of neighbors - it won't change over time
neighbors = getNeighbors();
```

Destroy code:

Running the Model



The Game of Life

Step: 15

Click on a cell to toggle its status

Run: 0 Running | Time: 15.80 | EPS: 2 | FPS: 12.0 | 8.7 sec