Events & Agent Movement over 2D Landscapes

CMPT 858

February 15, 2011

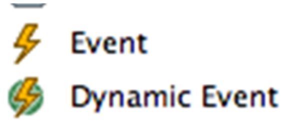
Example of Processes Associated with Fixed Timeouts

- Aging
- Tightly defined time constants associated with natural history
  - While these may be described as associated with a broad distribution (e.g. with a 1st or 2nd order delay), much of that variability may be due to heterogeneity
  - *For a given person, these may be quite specific in duration $\Rightarrow$ Can capture through a timeout*

Events in AnyLogic

Rates & Events

- *Rates* and *Timeouts* are associated with types of events in AnyLogic
- Events can also be declared explicitly from the pallette



  - Dynamic events can have multiple instances
    - Each instance can be scheduled at the same time
    - The instances disappear after event firing
  - Regular (static) events can be rescheduled, enabled/disabled, but can only have one scheduled firing at a time
- There are some subtleties with events

Event Times: Options for Event Scheduling

- Manually (via restart() – see following slides)
- When boolean condition changes (depends on *onChange* being called)
- One-time
  - Can go off at a particular time (specified as a calendar time or as a double-precision value)
- At some initial time and then cyclically beyond with set "timeout" period
  - The timeout period is set according to the time unit
  - This goes off after *exactly* the timeout time
- At a specified rate (Poisson arrivals)
  - Interarrival time is exponentially distributed!
  - Mean time between events is reciprocal of rate (i.e. 1/rate)

Event Subtleties

- Be very careful of what you count on for recomputation of rate – may think was recomputed, but hasn't been
- Event rates (and likely event timeout times) are only computed occasionally, not continuously
  - These are computed when
    - Explicitly call event methods
      - start()
      - restart()
      - onChange()
    - When event fires and requires restarting
    - (For outgoing transitions) when enter a state in a statechart
- Calling "reset" will disable a rate until re-enable (e.g. with call to *restart()*)

Agent Movement over 2D Landscapes

CMPT 858

February 10, 2011

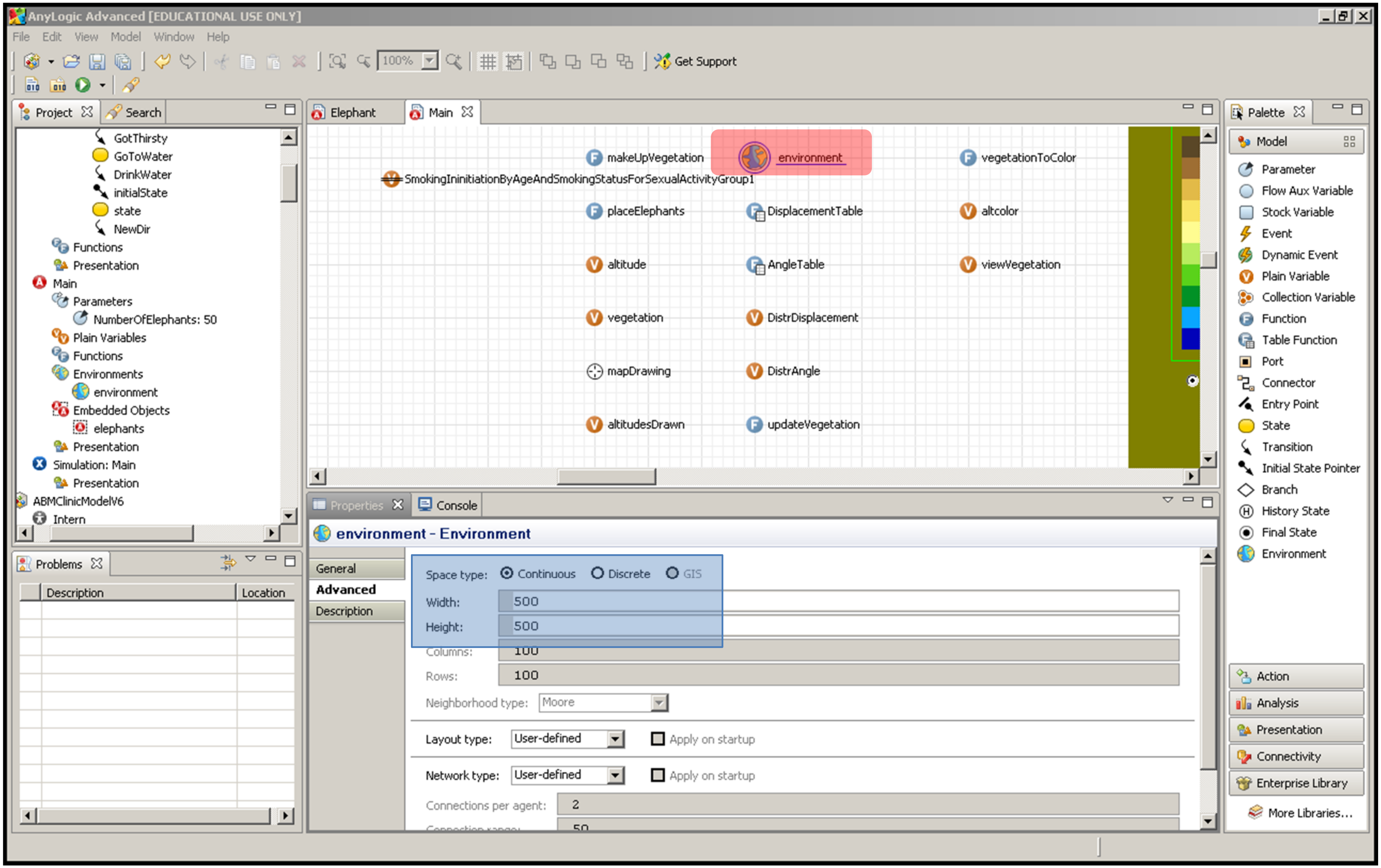
2D Landscape Movement: Two Options

- Continuous movement (e.g. Wandering elephants)
  - No physical exclusion: Agents are assumed to be small compared to landscape scale, and can pass without interfering
  - Agents move
    - In a direction
    - With some speed
- Discrete cells (e.g. Agent-based predatory prey)
  - Divided into “Columns” and “Rows”
  - Physical exclusion: Only one agent in a cell at a time
  - Agents move from cell to cell

Key Factor: 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}

Environment



environment - Environment

General | **Advanced** | Description

Space type: ● Continuous ○ Discrete ○ GIS

Width: 500

Height: 500

Columns: 100

Rows: 100

Neighborhood type: Moore

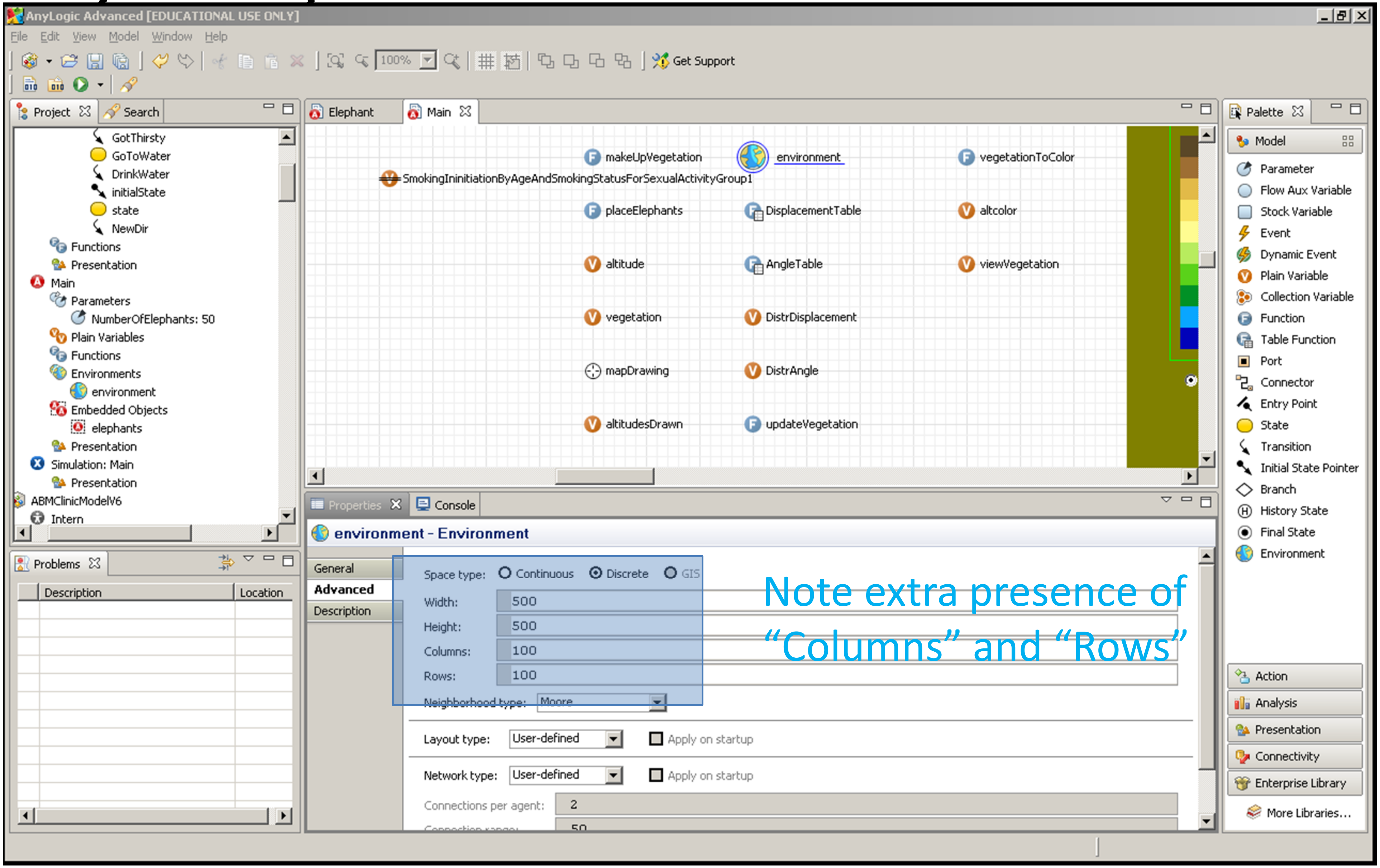
Layout type: User-defined □ Apply on startup

Network type: User-defined □ Apply on startup

Connections per agent: 2

Connection range: 50

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

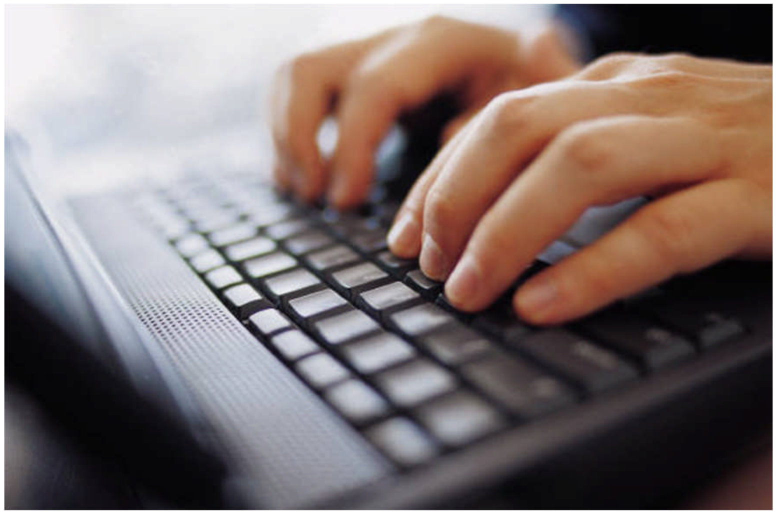
Network type: User-defined □ Apply on startup

Connections per agent: 2

Connection range: 50

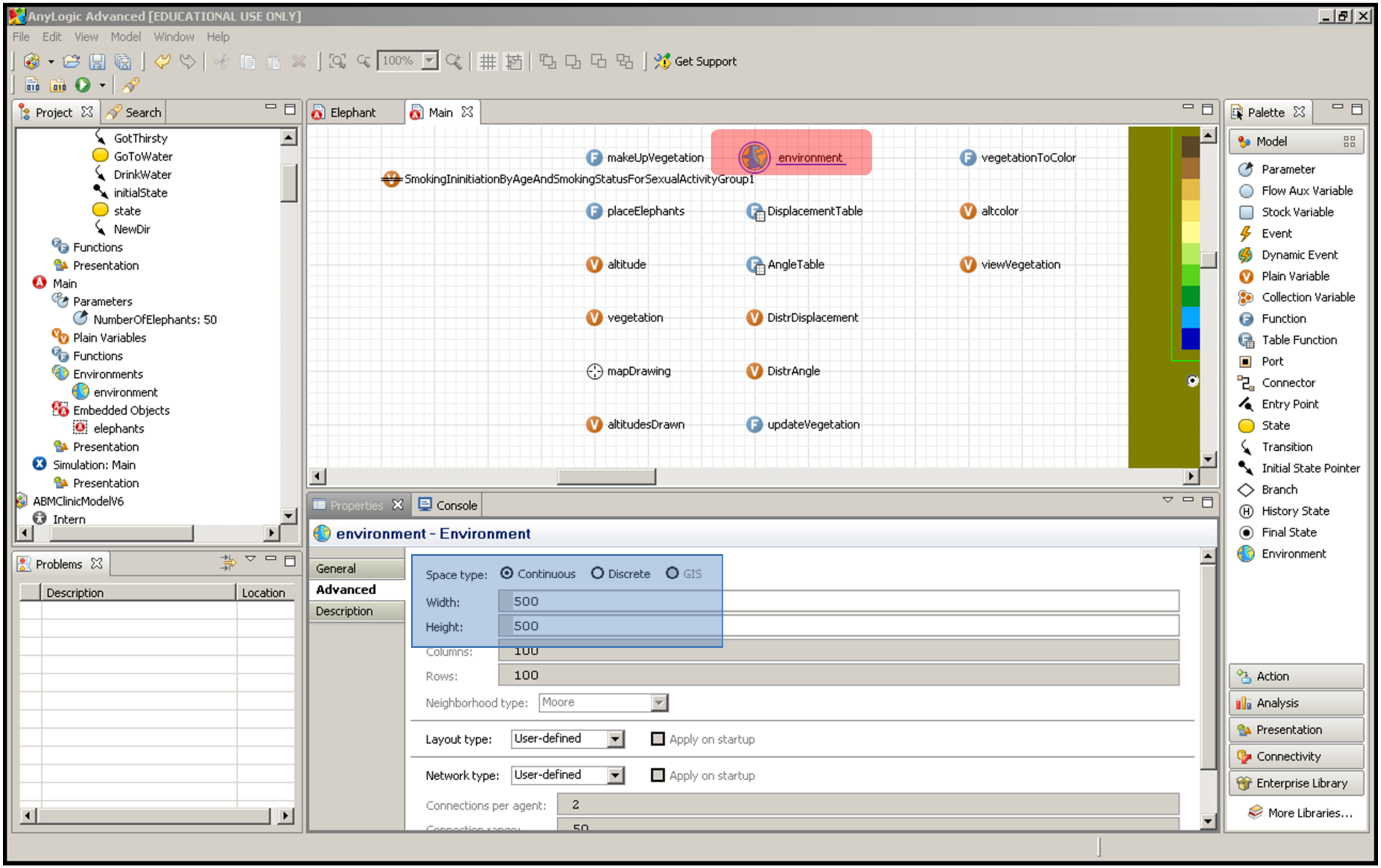
Note extra presence of "Columns" and "Rows"

Hands on Model Use Ahead



Load model: Wandering Elephants.alp

Environment



environment - Environment

Space type: ● Continuous ○ Discrete ○ GIS

Width: 500

Height: 500

Columns: 100

Rows: 100

Neighborhood type: Moore

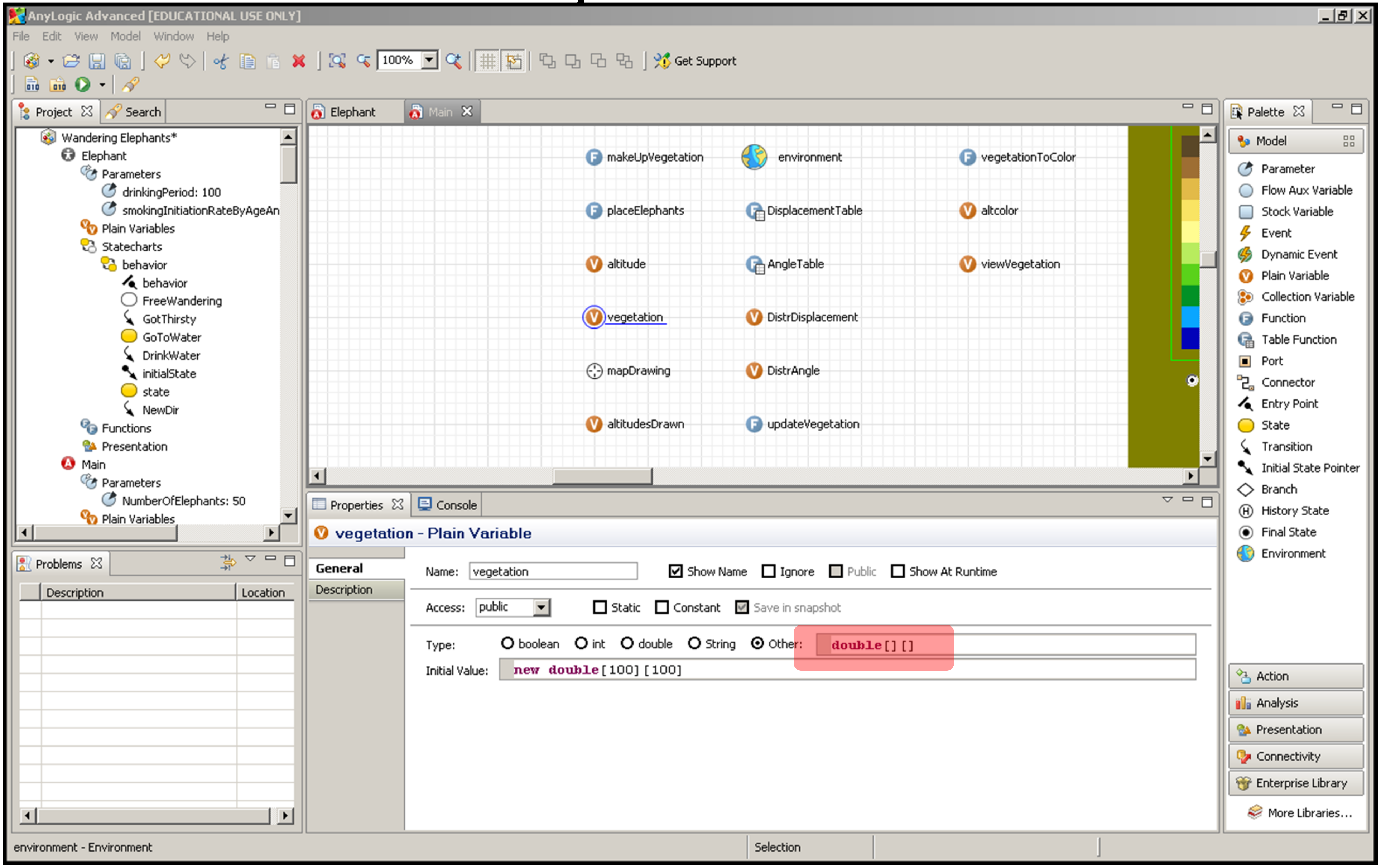
Layout type: User-defined □ Apply on startup

Network type: User-defined □ Apply on startup

Connections per agent: 2

Connection range: 50

Landscape Information



vegetation - Plain Variable

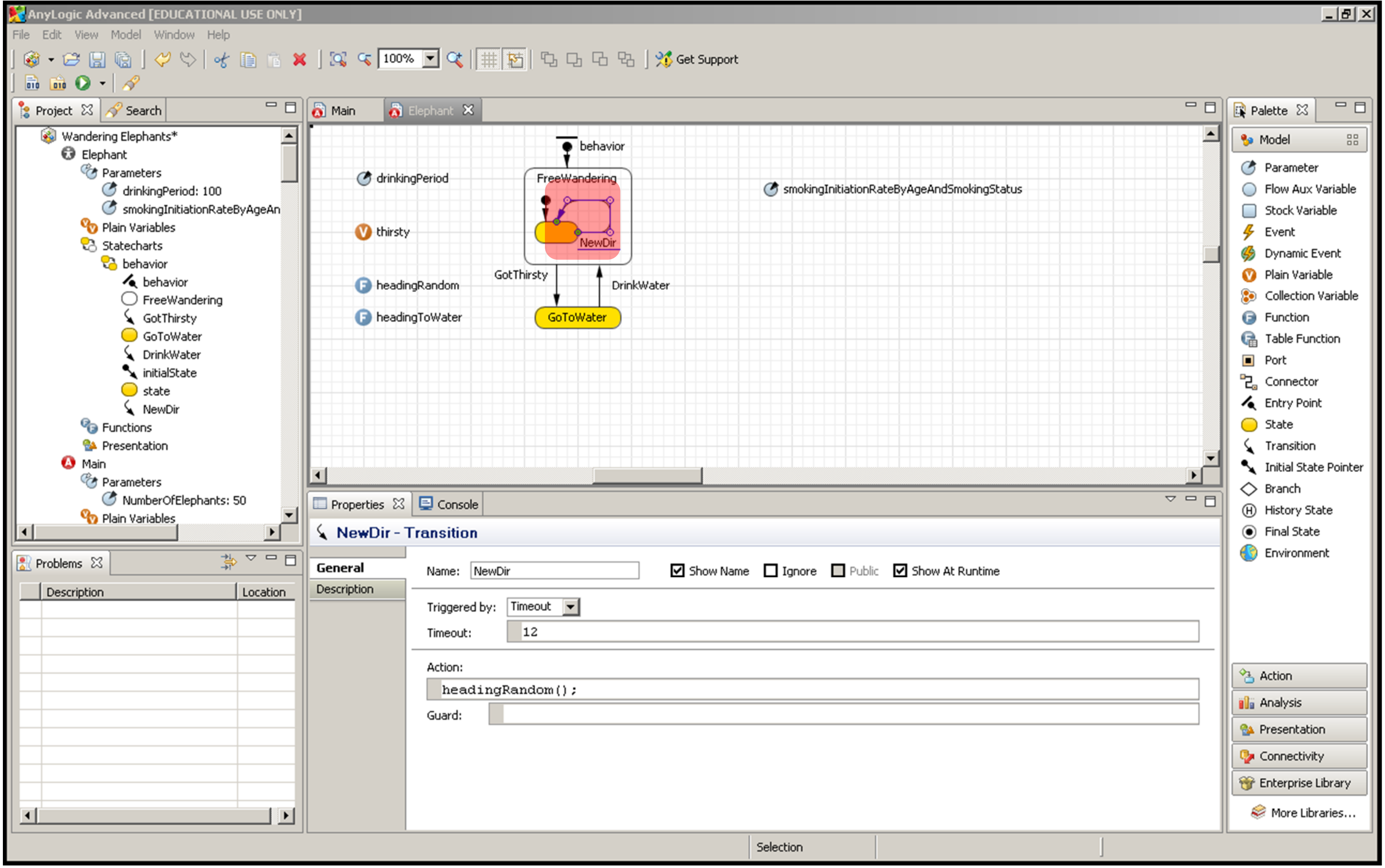
Name: vegetation ☑ Show Name □ Ignore □ Public □ Show At Runtime

Access: public □ Static □ Constant ☑ Save in snapshot

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

Initial Value: `new double[100][100]`

Agent Movement: Periodic Movement Changes



NewDir - Transition

Name: NewDir ☑ Show Name ☐ Ignore ☐ Public ☑ Show At Runtime

Triggered by: Timeout

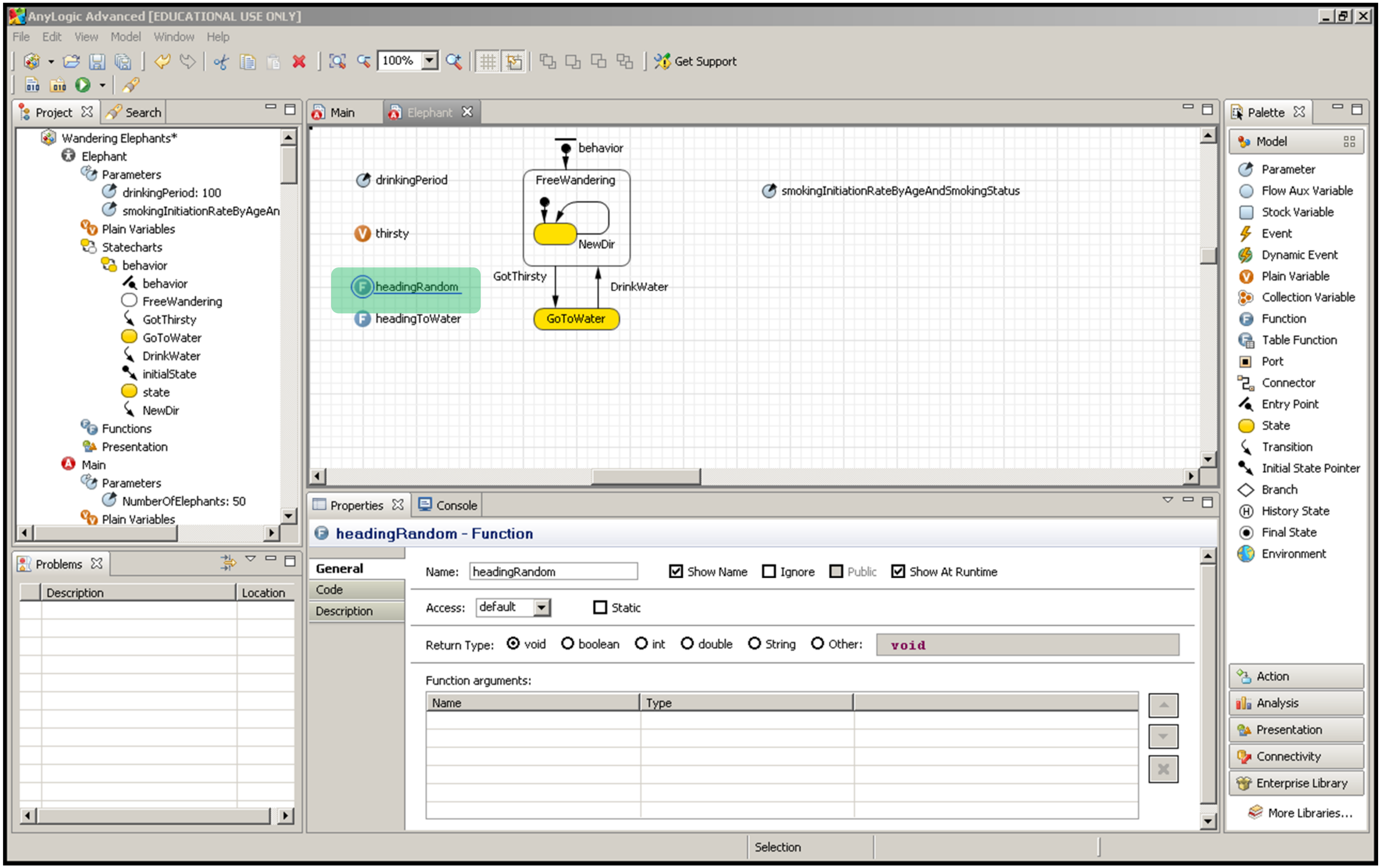
Timeout: 12

Action:

```
headingRandom();
```

Guard:

New Direction Change Function Info



headingRandom - Function

General

Name: headingRandom ☑ Show Name □ Ignore □ Public ☑ Show At Runtime

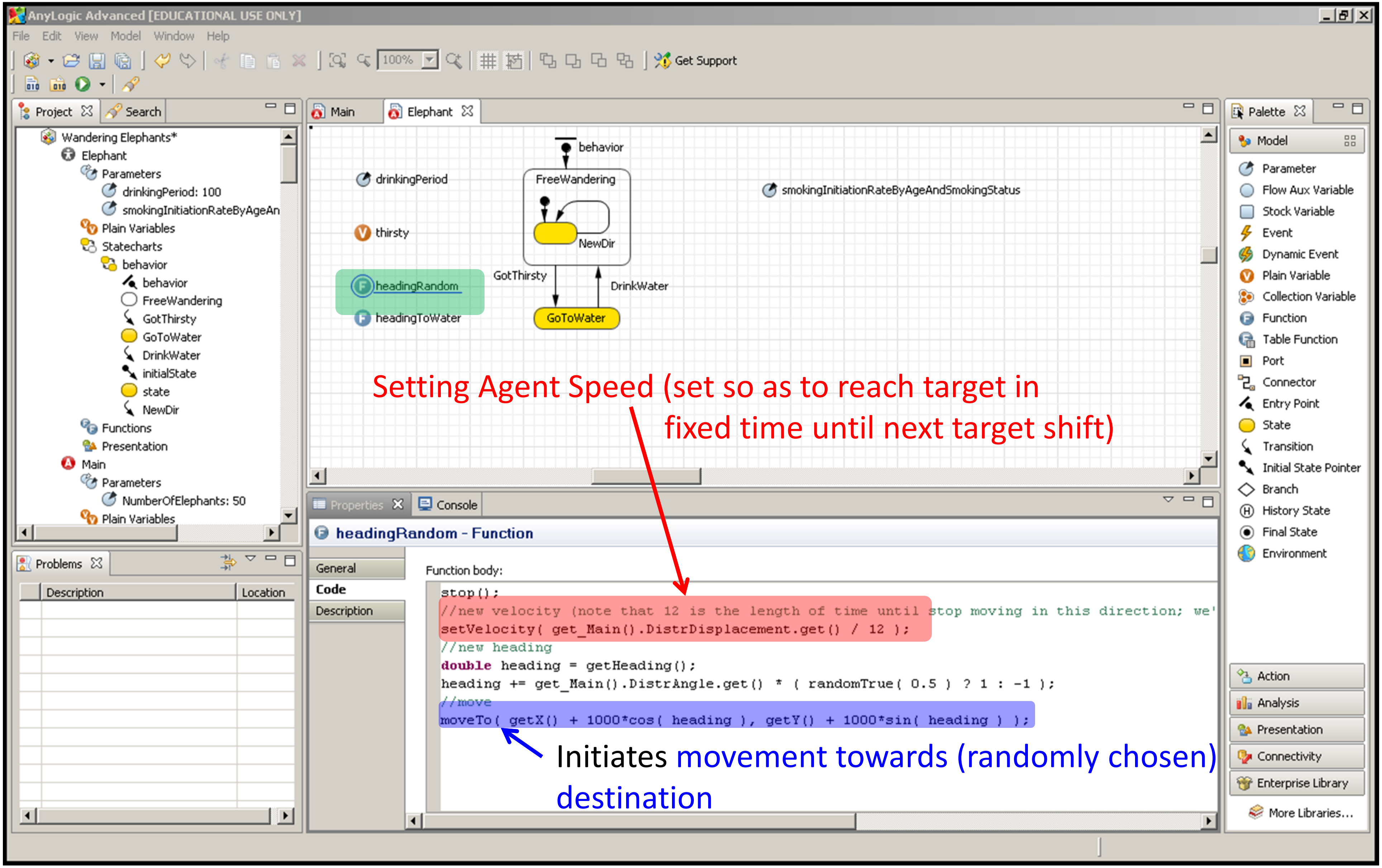
Access: default □ Static

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

Function arguments:

Name	Type	

New Direction Change: Function “Body”



headingRandom - Function

Function body:

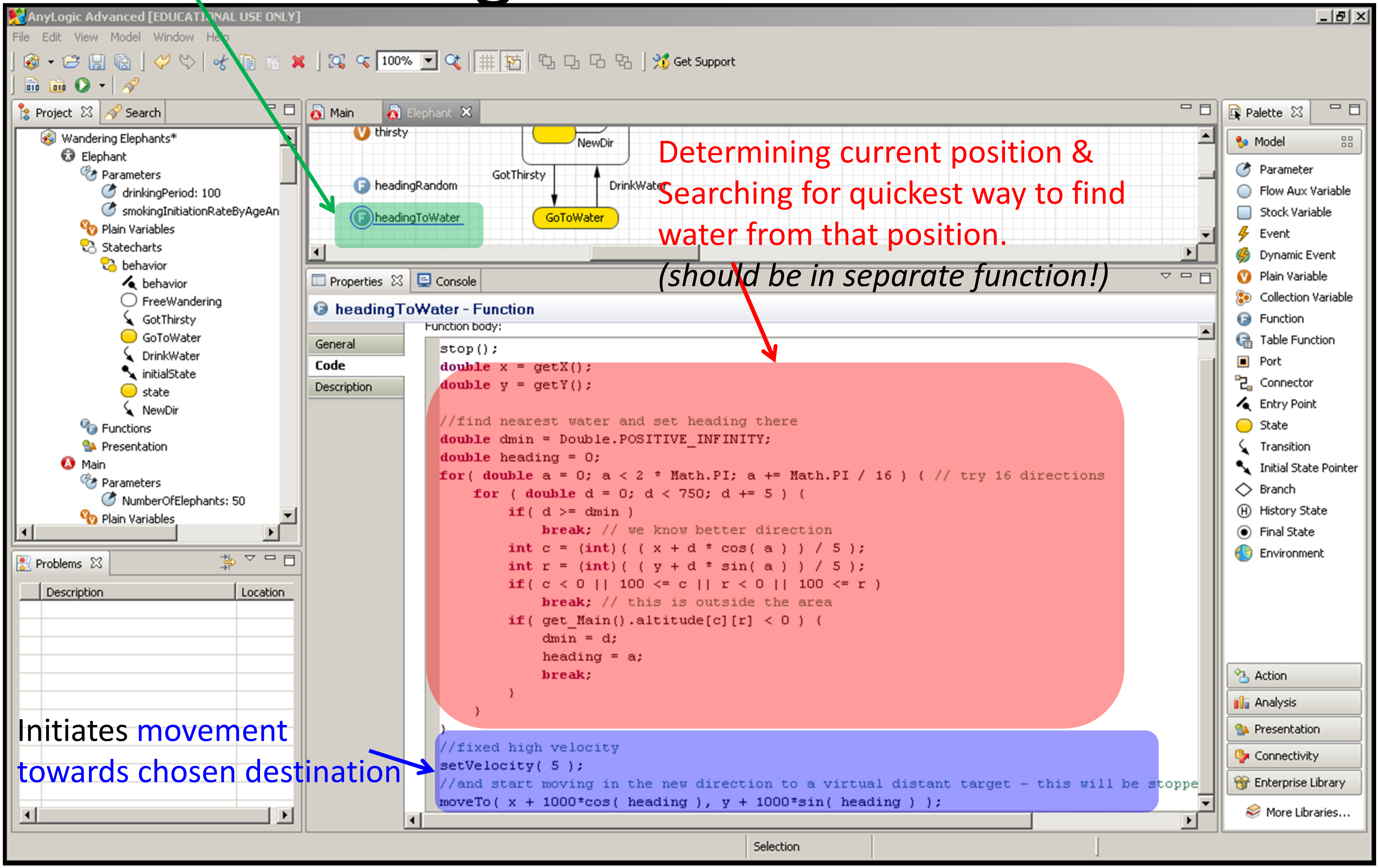
```
stop();
//new velocity (note that 12 is the length of time until stop moving in this direction; we'
setVelocity( get_Main().DistrDisplacement.get() / 12 );
//new heading
double heading = getHeading();
heading += get_Main().DistrAngle.get() * ( randomTrue( 0.5 ) ? 1 : -1 );
//move
moveTo( getX() + 1000*cos( heading ), getY() + 1000*sin( heading ) );
```

Setting Agent Speed (set so as to reach target in fixed time until next target shift)

Initiates movement towards (randomly chosen) destination

Looking at body of this function (method)

Heading Towards Resource



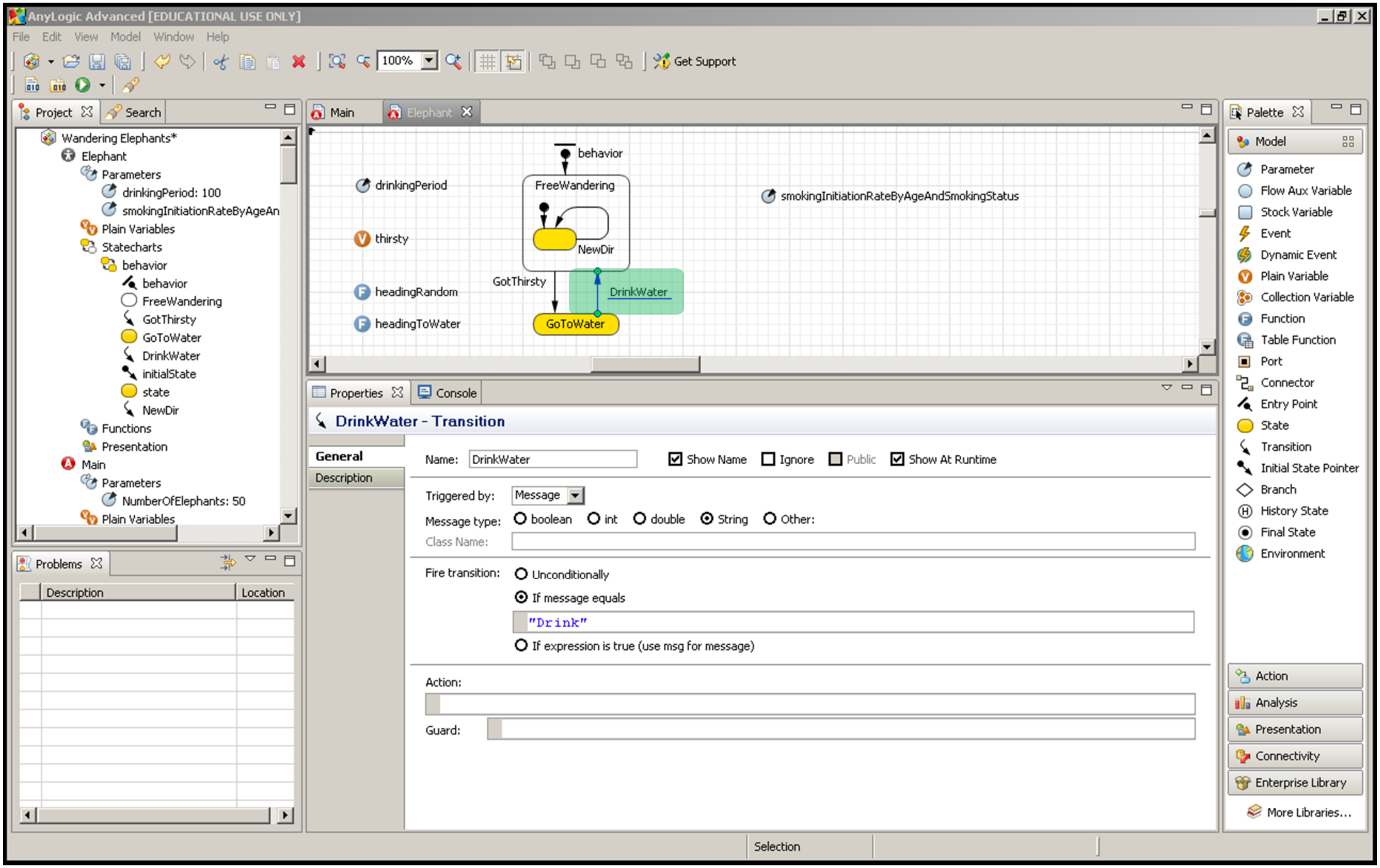
Determining current position & Searching for quickest way to find water from that position.
(should be in separate function!)

```
stop();
double x = getX();
double y = getY();

//find nearest water and set heading there
double dmin = Double.POSITIVE_INFINITY;
double heading = 0;
for( double a = 0; a < 2 * Math.PI; a += Math.PI / 16 ) { // try 16 directions
    for ( double d = 0; d < 750; d += 5 ) {
        if( d >= dmin )
            break; // we know better direction
        int c = (int)( ( x + d * cos( a ) ) / 5 );
        int r = (int)( ( y + d * sin( a ) ) / 5 );
        if( c < 0 || 100 <= c || r < 0 || 100 <= r )
            break; // this is outside the area
        if( get_Main().altitude[c][r] < 0 ) {
            dmin = d;
            heading = a;
            break;
        }
    }
}
//fixed high velocity
setVelocity( 5 );
//and start moving in the new direction to a virtual distant target - this will be stoppe
moveTo( x + 1000*cos( heading ), y + 1000*sin( heading ) );
```

Initiates movement towards chosen destination

Resumption of Wandering After Slaking Thirst



DrinkWater - Transition

General

Description

Name: DrinkWater ☑ Show Name □ Ignore □ Public ☑ Show At Runtime

Triggered by: Message

Message type: ○ boolean ○ int ○ double ● String ○ Other:

Class Name:

Fire transition: ○ Unconditionally

● If message equals

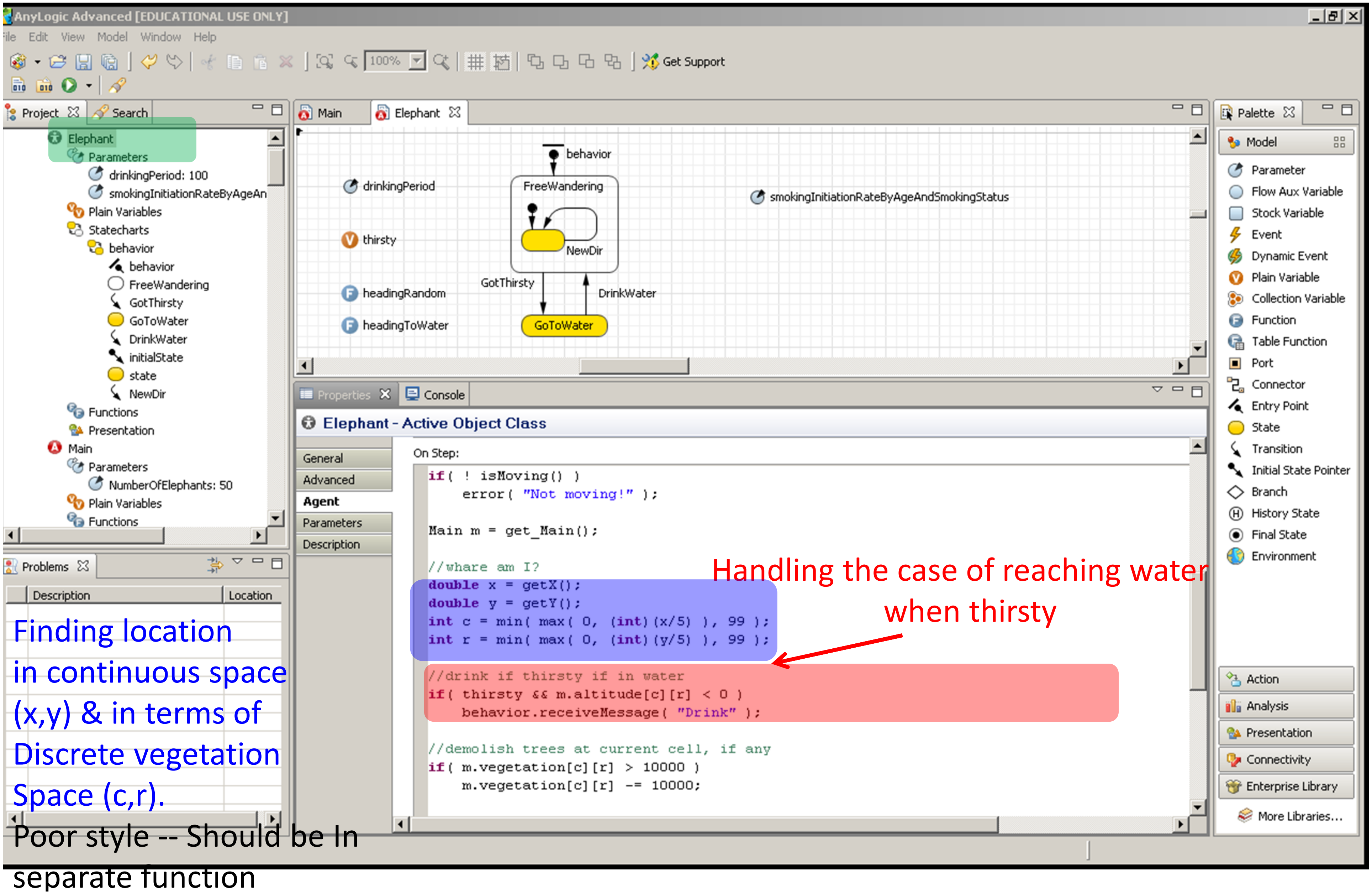
"Drink"

○ If expression is true (use msg for message)

Action:

Guard:

Handling of Movement Logic



On Step:

```
if( ! isMoving() )
    error( "Not moving!" );

Main m = get_Main();

//whare am I?
double x = getX();
double y = getY();
int c = min( max( 0, (int)(x/5) ), 99 );
int r = min( max( 0, (int)(y/5) ), 99 );

//drink if thirsty if in water
if( thirsty && m.altitude[c][r] < 0 )
    behavior.receiveMessage( "Drink" );

//demolish trees at current cell, if any
if( m.vegetation[c][r] > 10000 )
    m.vegetation[c][r] -= 10000;
```

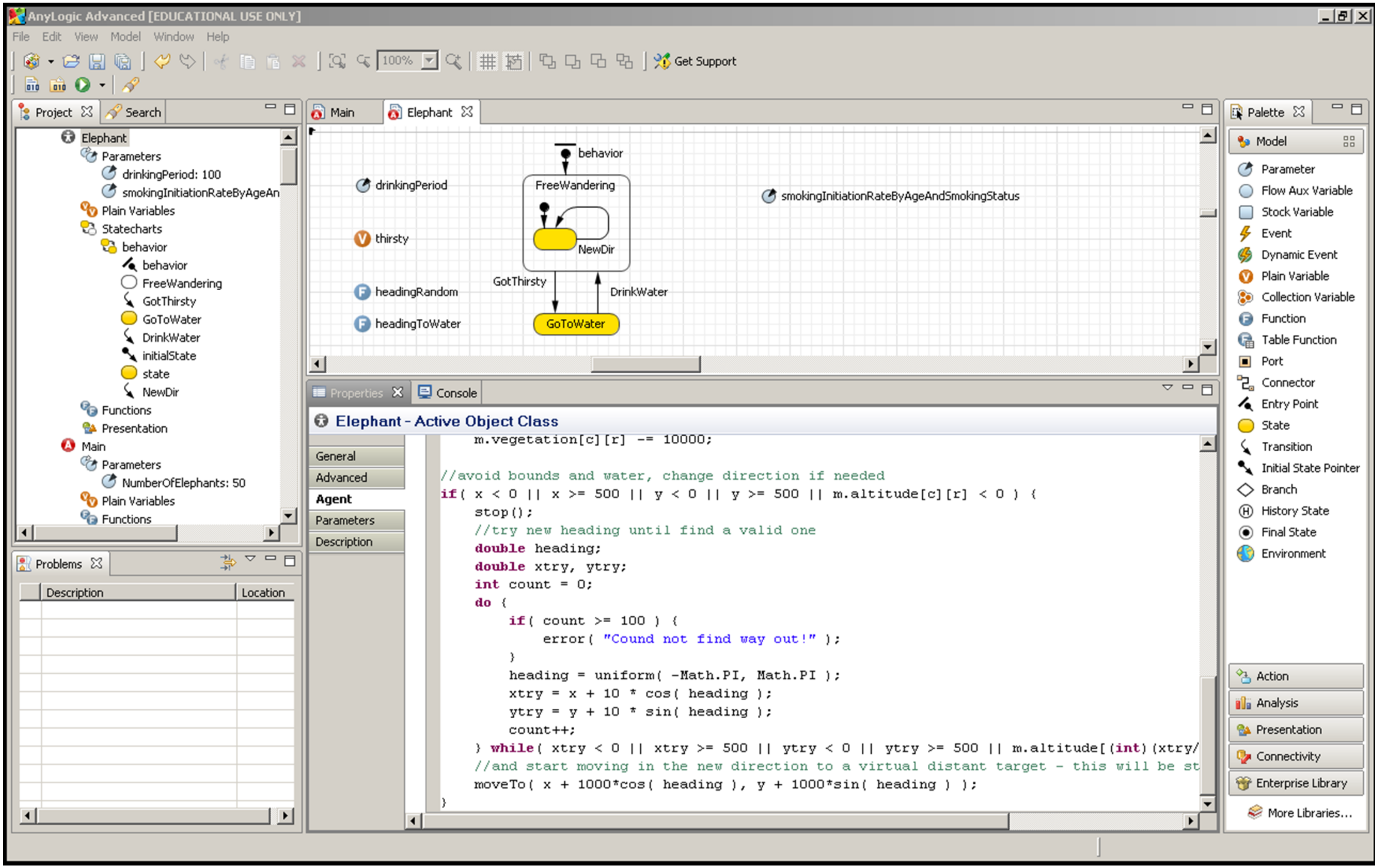
Handling the case of reaching water when thirsty

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)

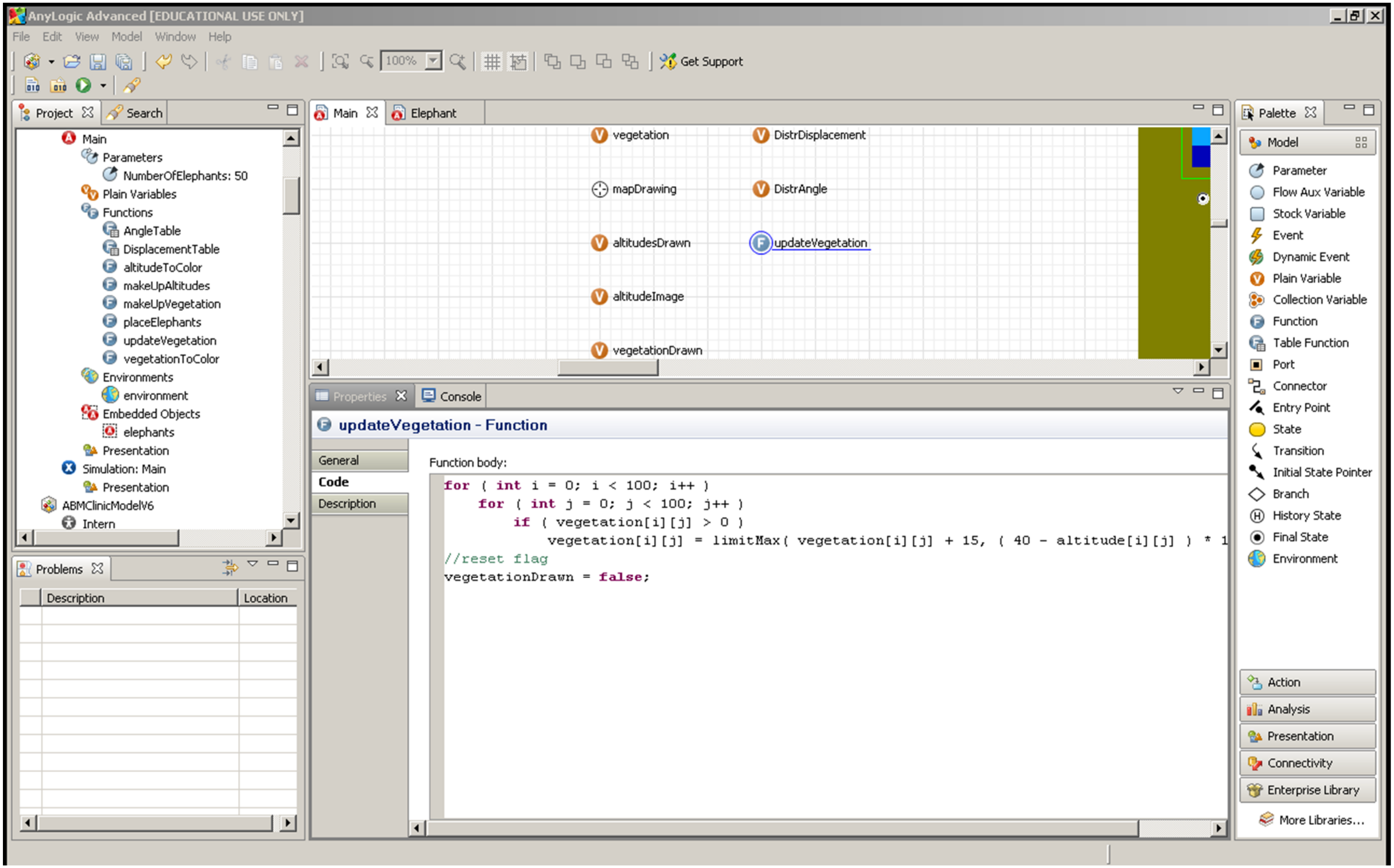
Poor Style – entire logic, conditions (checks on boundaries, whether water) & rerouting Logic should all be in separate functions from this & from each other). Remove constants



```
    m.vegetation[c][r] -= 10000;

//avoid bounds and water, change direction if needed
if( x < 0 || x >= 500 || y < 0 || y >= 500 || m.altitude[c][r] < 0 ) {
    stop();
    //try new heading until find a valid one
    double heading;
    double xtry, ytry;
    int count = 0;
    do {
        if( count >= 100 ) {
            error( "Cound not find way out!" );
        }
        heading = uniform( -Math.PI, Math.PI );
        xtry = x + 10 * cos( heading );
        ytry = y + 10 * sin( heading );
        count++;
    } while( xtry < 0 || xtry >= 500 || ytry < 0 || ytry >= 500 || m.altitude[(int)(xtry/
    //and start moving in the new direction to a virtual distant target - this will be st
    moveTo( x + 1000*cos( heading ), y + 1000*sin( heading ) );
}
```

Environment: Updating Vegetation



updateVegetation - Function

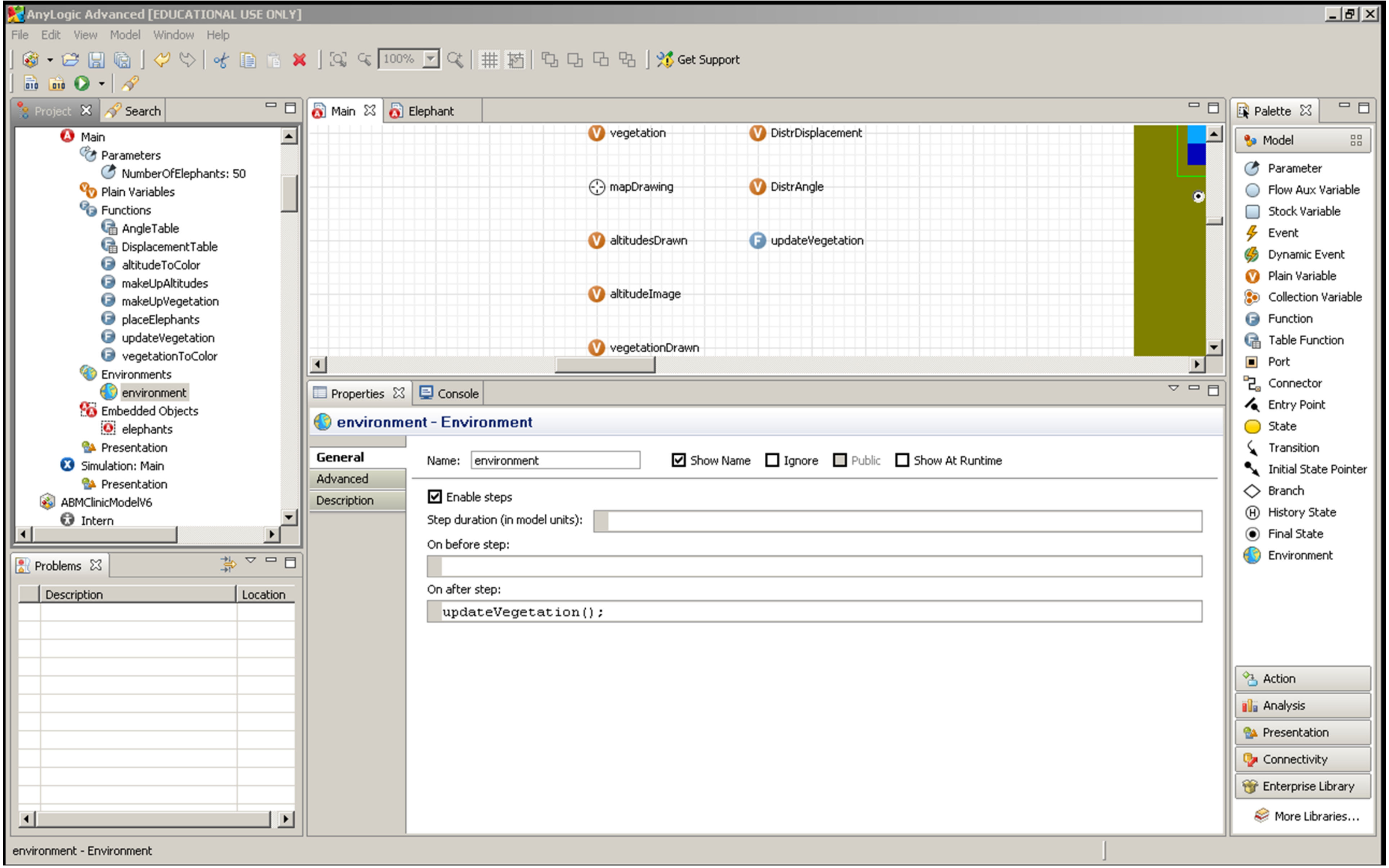
Function body:

```
for ( int i = 0; i < 100; i++ )
    for ( int j = 0; j < 100; j++ )
        if ( vegetation[i][j] > 0 )
            vegetation[i][j] = limitMax( vegetation[i][j] + 15, ( 40 - altitude[i][j] ) * 1
//reset flag
vegetationDrawn = false;
```

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

- Already covered
  - moveTo(x,y)
  - 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



environment - Environment

Name: environment ☑ Show Name □ Ignore □ Public □ Show At Runtime

☑ Enable steps

Step duration (in model units):

On before step:

On after step:

```
updateVegetation();
```