

Events & Agent Movement  
over  
2D Landscapes

CMPT 858

February 15, 2011

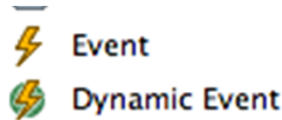
# Example of Processes Associated with Fixed Timeouts

- Aging
- Tightly defined time constants associated with natural history
  - While these may be described as associated with a broad distribution (e.g. with a 1<sup>st</sup> or 2<sup>nd</sup> order delay), much of that variability may be due to heterogeneity
  - *For a given person, these may be quite specific in duration ⇒ 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 palette



- 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.  $1/\text{rate}$ )

# 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

The screenshot displays the AnyLogic Advanced software interface, specifically the 'environment' configuration window. The main workspace shows a grid-based environment with various objects and variables. The 'environment' object is highlighted in pink. The 'Properties' window for the 'environment' object is open, showing the following settings:

- 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

The interface also includes a Project tree on the left, a Palette on the right, and a Console window at the bottom.

# By Comparison: Discrete Environment

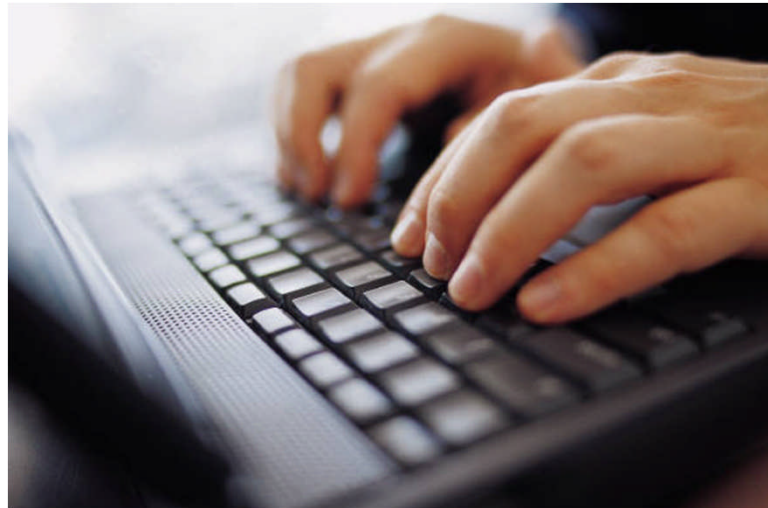
The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a grid-based environment with various objects and variables. A color palette is visible on the right side of the workspace. The Properties window at the bottom is open to the 'environment - Environment' section, showing the following settings:

- 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

A blue text box highlights the 'Columns' and 'Rows' settings, with the text: "Note extra presence of 'Columns' and 'Rows'".



## Hands on Model Use Ahead



Load model: `Wandering Elephants.alp`

# Environment

The screenshot displays the AnyLogic Advanced software interface, specifically the 'environment' configuration window. The main workspace shows a grid-based environment with various objects and variables. The 'environment' object is highlighted in pink. The 'Properties' window for the 'environment' object is open, showing the following settings:

- 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

The interface also includes a Project tree on the left, a Palette on the right, and a Console window at the bottom.

# Landscape Information

The screenshot displays the AnyLogic Advanced software interface, which is used for building simulation models. The main workspace shows a model diagram with various components:

- makeUpVegetation** (Function)
- environment** (Environment)
- vegetationToColor** (Function)
- placeElephants** (Function)
- DisplacementTable** (Table Function)
- altcolor** (Variable)
- altitude** (Variable)
- AngleTable** (Table Function)
- viewVegetation** (Variable)
- vegetation** (Variable, highlighted with a blue circle)
- DistrDisplacement** (Variable)
- mapDrawing** (Diagram)
- DistrAngle** (Variable)
- altitudesDrawn** (Variable)
- updateVegetation** (Function)

The **vegetation** variable is currently selected, and its properties are shown in the **Properties** panel:

- Name:** vegetation
- Access:** public
- Type:** double [ ] [ ] (highlighted in red)
- Initial Value:** new double [ 100 ] [ 100 ]

The **Palette** on the right side of the interface lists various model components, including:

- Parameter
- Flow Aux Variable
- Stock Variable
- Event
- Dynamic Event
- Plain Variable
- Collection Variable
- Function
- Table Function
- Port
- Connector
- Entry Point
- State
- Transition
- Initial State Pointer
- Branch
- History State
- Final State
- Environment

The **Problems** panel at the bottom left is empty, and the **Console** panel is also empty. The status bar at the bottom indicates the current selection is **environment - Environment**.

# Agent Movement: Periodic Movement Changes

The screenshot displays the AnyLogic Advanced software interface for modeling an elephant agent. The main workspace shows a statechart for the 'Elephant' agent. The statechart includes a 'FreeWandering' state with a 'NewDir' transition. The 'NewDir' transition is triggered by a 'Timeout' of 12 and performs the action 'headingRandom()'. Other transitions include 'GotThirsty' leading to 'GoToWater' and 'DrinkWater' leading back to 'FreeWandering'. The left sidebar shows the project structure, including parameters like 'drinkingPeriod: 100' and 'smokingInitiationRateByAgeAn', and statecharts like 'behavior' and 'FreeWandering'. The bottom panel shows the 'NewDir - Transition' properties, including the name 'NewDir', a checked 'Show Name' box, an unchecked 'Ignore' box, an unchecked 'Public' box, and a checked 'Show At Runtime' box. The 'Action' field contains the code 'headingRandom() ;'.

Description	Location

**NewDir - Transition**

General

Name:   Show Name  Ignore  Public  Show At Runtime

Triggered by:

Timeout:

Action:

Guard:

# New Direction Change Function Info

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a statechart diagram for an elephant's behavior. The diagram includes a state named 'FreeWandering' with a self-loop and a transition labeled 'NewDir'. Below it is a state named 'GoToWater'. Transitions between these states are labeled 'GotThirsty' and 'DrinkWater'. The diagram also shows variables like 'drinkingPeriod', 'thirsty', 'headingRandom', and 'headingToWater', and a function 'smokingInitiationRateByAgeAndSmokingStatus'.

The 'headingRandom - Function' properties window is open, showing the following details:

- Name:** headingRandom
- Access:** default
- Return Type:** void
- Function arguments:** (empty table)

Name	Type

The interface also shows a project tree on the left, a palette on the right, and a console at the bottom.



# New Direction Change: Function “Body”

The screenshot displays the AnyLogic Advanced software interface. On the left, a project tree shows the 'Elephant' model with various components like parameters, statecharts, and functions. The main workspace shows a statechart with a 'FreeWandering' state and a 'GoToWater' state. A red arrow points from the statechart to the 'headingRandom' function code editor at the bottom. The code editor shows the following code:

```
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 ) );
```

Annotations in the image include:

- A red text box: "Setting Agent Speed (set so as to reach target in fixed time until next target shift)" with a red arrow pointing to the `setVelocity` line in the code.
- A blue text box: "Initiates movement towards (randomly chosen) destination" with a blue arrow pointing to the `moveTo` line in the code.

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!)*

The screenshot shows the AnyLogic Advanced software interface. On the left is a project tree for 'Wandering Elephants\*' with an 'Elephant' object containing various parameters and functions. The main workspace displays a statechart with states like 'thirsty', 'NewDir', and 'GoToWater', and transitions 'GotThirsty' and 'DrinkWater'. The 'headingToWater' function is highlighted in green. Below the statechart, the 'headingToWater - Function' editor is open, showing the following code:

```
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 ) );
```

Annotations include a green arrow pointing to the function name in the statechart, a red arrow pointing to the code body, and a blue arrow pointing to the movement commands at the end of the code.

Initiates movement  
towards chosen destination



# Handling of Movement Logic

The screenshot displays the AnyLogic Advanced interface. On the left, a project tree shows the 'Elephant' model with various components like Parameters, Statecharts, and Functions. The main workspace shows a statechart with states 'FreeWandering' and 'GoToWater', and transitions 'GotThirsty' and 'DrinkWater'. Below the statechart, the 'Elephant - Active Object Class' code is visible. The code includes logic for finding location in continuous space and handling the case of reaching water when thirsty.

```
On Step:  
  
if( ! isMoving() )  
    error( "Not moving!" );  
  
Main m = get_Main();  
  
//where 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;
```

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

Poor style -- Should be In  
separate function

Handling the case of reaching water  
when thirsty

# 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

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a statechart for an elephant agent. The statechart includes a state named 'FreeWandering' with a self-loop labeled 'NewDir'. Transitions from 'FreeWandering' include 'GotThirsty' leading to a state 'GoToWater', and 'DrinkWater' leading back to 'FreeWandering'. The 'GoToWater' state is also shown. The interface includes a menu bar, a toolbar, a project browser on the left, a palette on the right, and a console window at the bottom.

**Elephant - Active Object Class**

```
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( "Count 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

The screenshot displays the AnyLogic Advanced software interface, which is used for building and simulating agent-based models. The main workspace shows a grid environment with several components: 'vegetation', 'DistrDisplacement', 'mapDrawing', 'DistrAngle', 'altitudesDrawn', 'updateVegetation', 'altitudeImage', and 'vegetationDrawn'. The 'updateVegetation' component is highlighted with a blue circle.

On the left side, there is a 'Project' tree view showing the model's structure, including 'Main', 'Parameters', 'Plain Variables', 'Functions', 'Environments', and 'Embedded Objects'. The 'Main' environment is currently selected.

At the bottom, the 'Properties' and 'Console' tabs are visible. The 'Properties' tab is active, showing the 'updateVegetation' function's code:

```
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;
```

On the right side, there is a 'Palette' window containing various components for the model, such as 'Parameter', 'Flow Aux Variable', 'Stock Variable', 'Event', 'Dynamic Event', 'Plain Variable', 'Collection Variable', 'Function', 'Table Function', 'Port', 'Connector', 'Entry Point', 'State', 'Transition', 'Initial State Pointer', 'Branch', 'History State', 'Final State', and 'Environment'. Below the palette are buttons for 'Action', 'Analysis', 'Presentation', 'Connectivity', and 'Enterprise Library'.

# 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

The screenshot displays the AnyLogic Advanced software interface, specifically the configuration panel for an 'environment' object. The main workspace shows a grid with several variables and functions, including 'vegetation', 'DistrDisplacement', 'mapDrawing', 'DistrAngle', 'altitudesDrawn', 'updateVegetation', 'altitudeImage', and 'vegetationDrawn'. The 'environment - Environment' panel is open, showing the following configuration:

- General:** Name: environment,  Show Name,  Ignore,  Public,  Show At Runtime
- Advanced:**  Enable steps
- Description:** Step duration (in model units): [empty field], On before step: [empty field], On after step: `updateVegetation();`

The interface also includes a Project Explorer on the left, a Properties/Console area at the bottom, and a Palette on the right with various modeling elements like Parameter, Flow Aux Variable, Stock Variable, Event, Dynamic Event, Plain Variable, Collection Variable, Function, Table Function, Port, Connector, Entry Point, State, Transition, Initial State Pointer, Branch, History State, Final State, and Environment.