

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

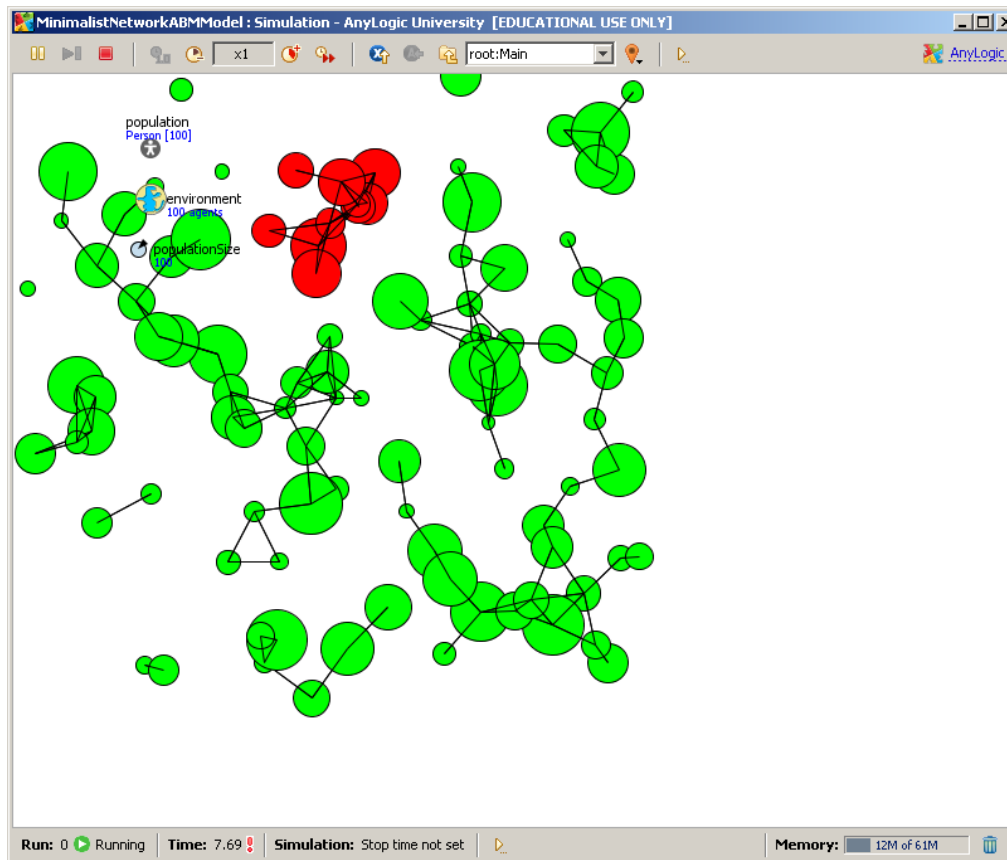
The screenshot displays the AnyLogic Advanced software interface, titled "AnyLogic Advanced [EDUCATIONAL USE ONLY]". The main workspace shows a simulation environment with a grid and various components. A pink box highlights the "environment" component. The "Properties" panel for the "environment - Environment" component 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
- Capacity per agent: 50

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

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

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 panel at the bottom is open to the 'environment - Environment' section, where the 'Space type' is set to 'Discrete'. The 'Advanced' section of the Properties panel is highlighted, showing the following settings:

Property	Value
Space type	Discrete
Width	500
Height	500
Columns	100
Rows	100
Neighborhood type	Moore

The Properties panel also shows 'Layout type' and 'Network type' both set to 'User-defined', and 'Connections per agent' set to 2. A blue text overlay on the right side of the Properties panel reads: "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.

The screenshot displays the AnyLogic University software interface. The main workspace shows a simulation titled "The Game of Life" at "Step: 2". The simulation area is a grid with a red border and a yellow center. On the left side of the workspace, there are two objects: "cells [...]" and "environment".

The left sidebar shows a project tree with the following structure:

- IMainAction
- Simulation: Main
- HierarchicalCityPopulationModelW
- City
- Main
- Person
- Baseline: Main
- RecoveryTime10: Main
- RecoveryTime100: Main
- The Game of Life
 - Cell
 - Variables
 - alive
 - nAliveAround
 - neighbors
 - Functions
 - toggleState
 - Presentation
 - Main
 - Environments
 - Embedded Objects
 - Presentation
 - Simulation: Main

The bottom panel shows the "Main - Active Object Class" properties. The "General" tab is selected, showing the following fields:

- Name: Main
- Ignore
- Agent Generic
- Startup code:
- Destroy code:

The right sidebar shows a palette of objects, including:

- General
 - Parameter
 - Event
 - Dynamic Event
 - Variable
 - Collection
 - Function
 - Table Function
 - Schedule
 - Port
 - Connector
 - Environment
 - Agent Population
- System Dynamics
- Statechart
- Actionchart
- Analysis
- Presentation
- 3D
- Controls
- Connectivity
- Enterprise Library
- Pedestrian Library
- Rail Library
- Road Traffic Library - Preview
- Pictures
- 3D Objects
- Palettes...

The bottom status bar shows "Selection" and "X=268, Y=421".

Imposing the Regular 2D Structure

The screenshot displays the AnyLogic University software interface. The main window shows a simulation titled "The Game of Life" at "Step: 2", featuring a 100x100 grid of cells. The left sidebar contains a project tree with a "Main" section expanded to show "Variables" (alive, nAliveAround, neighbors), "Functions" (toggleState), and "Presentation". The bottom-left pane shows a "Problems" table with the text "No problems".

The "environment - Environment" properties window is open, showing the following settings:

- 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

Red annotations highlight the "Columns: 100" and "Rows: 100" fields. A red arrow points from the text "100x100 grid defined here" to the "Columns" field. A blue arrow points from the text "Indicated that cells should be laid out in a regular grid in space" to the "Layout type: Arranged" dropdown.

100x100 grid defined here

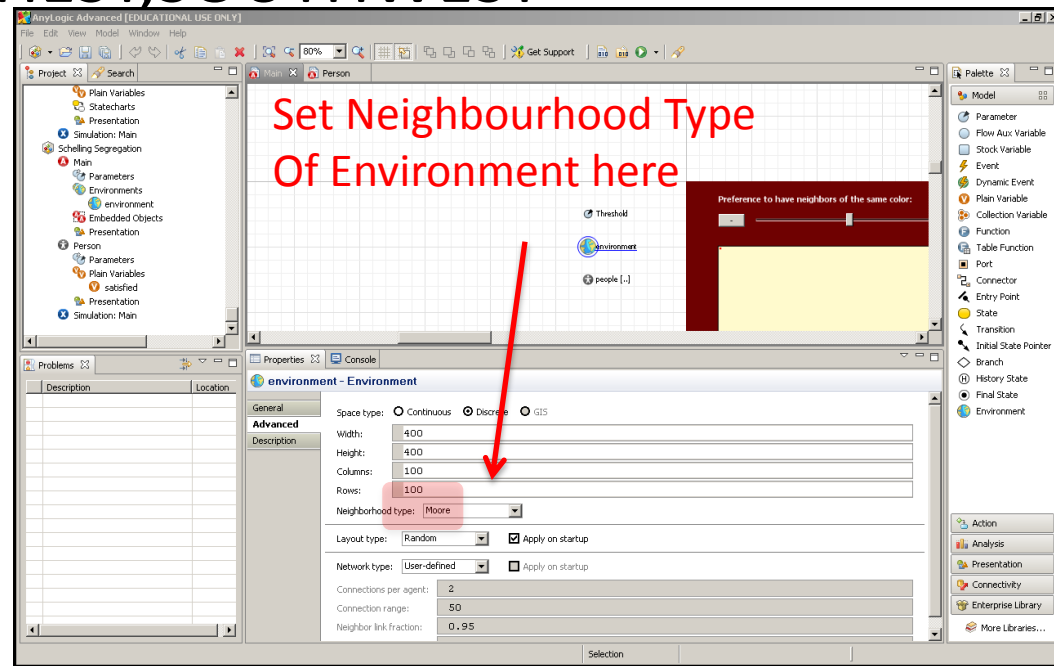
Indicated that cells should be laid out in a regular grid in space

Environment: Enabling Discrete Space (Cells)

The screenshot displays the AnyLogic University interface for a simulation titled "The Game of Life". The main workspace shows a grid with a red border and a yellow center, labeled "Step: 2". The left sidebar shows a project tree with "The Game of Life" selected, containing a "Cell" object with variables like "alive" and "neighbors". The bottom panel shows the "environment - Environment" properties window. In the "General" tab, the "Space type" is set to "Discrete2D", which is highlighted with a red box and a red arrow. A red text box in the bottom left corner says "Discrete2D selected". The "Neighborhood type" is set to "Moore", highlighted with a blue box and a blue arrow. A blue text box in the bottom right corner says "Defines logical neighborhood (here, each cell has 4 neighbors)". Other properties include Width: 500, Height: 500, Z-Height: 0, Columns: 100, Rows: 100, Layout type: Arranged, and Network type: User-defined.

Neighbourhood Models

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



Population: One Cell Agent per Grid Point

The screenshot displays the AnyLogic University interface for a simulation titled "The Game of Life". The main workspace shows a grid environment with a red border and the text "Step: 2". The left sidebar contains a project tree with the following structure:

- IMainAction
 - Simulation: Main
- HierarchicalCityPopulationModelW
 - City
 - Main
 - Person
- The Game of Life
 - Cell
 - Variables
 - alive
 - nAliveAround
 - neighbors
 - Functions
 - toggleState
 - Presentation
 - Main
 - Environments
 - Embedded Objects
 - Presentation
 - Simulation: Main

The Properties window for the "cells - Cell" agent is open, showing the following configuration:

- Name: cells
- Type: Cell
- Environment: environment
- Package: the_game_of_life
- Replicated
- Initial number of objects: 10000
- Optimize: Access by index (ArrayList) Add/remove operations (LinkedHashSet)

A red arrow points from the text "10,000 (= 100*100) agents" to the "Initial number of objects" field.

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

General

Advanced

Agent

Preview

Description

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”

The screenshot displays the AnyLogic University interface. On the left, a project tree shows the hierarchy: IMainAction > Simulation: Main > HierarchicalCityPopulationModelW > City > Main > Person > Baseline: Main > RecoveryTime10: Main > RecoveryTime100: Main > The Game of Life > Cell > Variables > alive. The main workspace shows a grid with several variables: 'alive' (highlighted with a blue circle), 'neighbors', 'nAliveAround', and 'toggleState'. A red arrow points from the text 'Boolean (true/false) variable' to the 'boolean' type selection in the properties panel. A blue arrow points from the text 'Name would be clearer as “isAlive”' to the 'Name: alive' field. A green arrow points from the text '10% initial likelihood of being occupied' to the 'Initial value: randomTrue(0.1)' field. The properties panel for 'alive - Variable' shows: Name: alive, Access: public, Type: boolean, Initial value: randomTrue(0.1).

Boolean (true/false) variable

Name would be clearer as “isAlive”

10% initial likelihood of being occupied

Cell Variables: “neighbors”

The screenshot displays the AnyLogic University interface. On the left, a project tree shows a 'Cell' model with variables 'alive', 'nAliveAround', and 'neighbors'. The main workspace shows a diagram of a cell with these variables. A red arrow points to the 'neighbors' variable in the diagram. Below the workspace, the 'Properties' panel shows the definition for the 'neighbors' variable. The 'Type' field is set to 'Agent[]', which is highlighted with a green box and a green arrow. A green text overlay at the bottom explains this reference.

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

Reference to the collection has an “Array” type

neighbors - Variable

General

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:

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”

The screenshot shows the AnyLogic University interface. On the left is a project tree with a 'Cell' object containing variables 'alive', 'nAliveAround', and 'neighbors'. The main workspace shows a grid with these variables as objects. A red box highlights 'nAliveAround', with a red arrow pointing to it from the explanatory text. Below the workspace is the 'Properties' panel for the 'nAliveAround - Variable' object. It shows the variable name, access level (public), and type (int). A green arrow points to the 'int' radio button in the 'Type' section. The right side of the interface shows a palette with various modeling elements.

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

The screenshot displays the AnyLogic software interface. On the left, a project tree shows the hierarchy: IMainAction, Simulation: Main, HierarchicalCityPopulationModelW, City, Main, Person, Baseline: Main, RecoveryTime10: Main, RecoveryTime100: Main, The Game of Life, Cell, Variables (alive, nAliveAround, neighbors), Functions (toggleState), Presentation, Main, Environments, Embedded Objects, Presentation, Simulation: Main. The main workspace shows a grid with a red rectangle at the origin (0,0). A red arrow points to this rectangle with the text "Select this item". Below the grid, the Properties window is open for a "rectangle - Rectangle". A blue arrow points to the "Fill color" field, which contains the text "alive ? red : lemonChiffon". The text "Selects appearance depending on whether alive or not" is written in blue in the center of the workspace. The right side of the interface shows a Palette with various object types like Parameter, Event, Dynamic Event, Variable, Collection, Function, Table Function, Schedule, Port, Connector, Environment, Agent Population, System Dynamics, Statechart, Actionchart, Analysis, Presentation, 3D, Controls, Connectivity, Enterprise Library, Pedestrian Library, Rail Library, Road Traffic Library - Preview, Pictures, 3D Objects, and Palettes...

Select this item

Selects appearance depending on whether alive or not

rectangle - Rectangle

General	Replication:
Advanced	Visible:
Dynamic	X:
Description	Y:
	Z:
	Fill color: alive ? red : lemonChiffon
	Width:
	Height:
	Z-Height:

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

The screenshot displays the AnyLogic University interface. On the left, a project tree shows the hierarchy: Simulation: Main > HierarchicalCityPopulationModelW > City > Main > Person > Cell. The 'Cell' object is selected, showing its variables (alive, nAliveAround, neighbors) and functions (toggleState). The main workspace shows a statechart with a vertical timeline and a horizontal axis. The statechart contains four elements: 'alive' (variable), 'neighbors' (variable), 'nAliveAround' (variable), and 'toggleState' (function). Below the workspace, the 'Properties' window for 'Cell - Active Object Class' is open. It shows the 'Agent' tab with the following logic:

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;
```

The right side of the interface shows a 'Palette' with various modeling elements like Parameter, Event, Dynamic Event, Variable, Collection, Function, Table Function, Schedule, Port, Connector, Environment, and Agent Population. The bottom left shows a 'Problems' window with no problems listed.

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

The screenshot shows the AnyLogic University interface for a simulation titled "The Game of Life". The main workspace displays a grid with a red border and a yellow center, labeled "The Game of Life" and "Step: 2". The left sidebar shows a project tree with "The Game of Life" selected, containing elements like "Cell", "Variables", "Functions", and "Presentation". The bottom panel shows the "environment - Environment" properties, with the "Enable steps" checkbox checked. A red arrow points to this checkbox, and a red text box below it says "Notice checkmark to enable discrete time (steps)".

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)

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;
```

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

The screenshot displays the AnyLogic University software interface. The main workspace shows a grid with several variables: 'alive' (orange circle), 'neighbors' (orange circle), 'nAliveAround' (orange circle), and 'toggleState' (blue circle). A red text overlay reads: "For performance reasons, this obtains a reference to a set of neighboring cells, and stores it in the variable 'neighbors'". A red arrow points from this text to the 'Startup code' field in the 'Cell - Active Object Class' properties window. The 'Startup code' field contains the following code:

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

The 'Cell - Active Object Class' properties window is open, showing the 'General' tab. The 'Name' field is set to 'Cell'. The 'Agent' checkbox is checked, and the 'Generic' checkbox is unchecked. The 'Startup code' field is highlighted with a pink background. The 'Destroy code' field is empty.

The left sidebar shows the project hierarchy, including 'Main', 'Simulation: Main', 'HierarchicalCityPopulationModelW', 'City', 'Main', 'Person', 'Baseline: Main', 'RecoveryTime10: Main', 'RecoveryTime100: Main', 'The Game of Life', 'Cell', 'Variables', 'alive', 'nAliveAround', 'neighbors', 'Functions', 'toggleState', 'Presentation', 'Main', 'Environments', 'Embedded Objects', 'Presentation', 'Simulation: Main'. The right sidebar shows the 'Palette' with various components like 'Parameter', 'Event', 'Dynamic Event', 'Variable', 'Collection', 'Function', 'Table Function', 'Schedule', 'Port', 'Connector', 'Environment', 'Agent Population', 'System Dynamics', 'Statechart', 'Actionchart', 'Analysis', 'Presentation', '3D', 'Controls', 'Connectivity', 'Enterprise Library', 'Pedestrian Library', 'Rail Library', 'Road Traffic Library - Preview', 'Pictures', '3D Objects', and 'Palettes...'. The bottom left shows the 'Problems' window with 'No problems' listed.

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

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

The screenshot displays the AnyLogic Advanced software interface, specifically the 'environment - Environment' configuration window. The interface is divided into several panels:

- Project Panel (Left):** Shows a hierarchical tree of the simulation model, including 'Main', 'Parameters', 'Plain Variables', 'Functions', 'Environments', and 'Embedded Objects'. The 'environment' environment is currently selected.
- Environment Canvas (Center):** A grid-based workspace where various environment components are placed. A pink box highlights the 'environment' component. Other components include 'makeUpVegetation', 'placeElephants', 'altitude', 'vegetation', 'mapDrawing', 'altitudesDrawn', 'DisplacementTable', 'AngleTable', 'DistrDisplacement', 'DistrAngle', 'updateVegetation', 'vegetationToColor', 'altcolor', and 'viewVegetation'. A vertical color palette is visible on the right side of the canvas.
- Properties Panel (Bottom):** Displays the configuration for the selected 'environment' component. The 'General' tab is active, 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
 - Capacity per agent: 50
- Palette (Right):** A library of components for the environment, 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, and Environment.

Landscape Information

The screenshot displays the AnyLogic Advanced software interface, which is used for modeling and simulation. The main workspace shows a diagram of a model with various components and their relationships. The 'vegetation' variable is highlighted, and its properties are shown in the bottom panel.

Model Diagram Components:

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

Properties Panel: vegetation - Plain Variable

General

Name: Show Name Ignore Public Show At Runtime

Access: Static Constant Save in snapshot

Type: boolean int double String Other: (highlighted)

Initial Value:

Right Panel: Palette

- Model
- 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
- Action
- Analysis
- Presentation
- Connectivity
- Enterprise Library
- More Libraries...

Bottom Panel: environment - Environment | Selection

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 labeled 'NewDir'. Transitions from 'FreeWandering' include 'GotThirsty' leading to a state 'GoToWater', and 'DrinkWater' leading back to 'FreeWandering'. Other elements include a state 'GoToWater', a function 'headingRandom', and a variable 'thirsty'. The 'headingRandom' function is highlighted in the diagram.

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

- Name:** headingRandom
- Show Name:**
- Ignore:**
- Public:**
- Show At Runtime:**
- Access:** default
- Static:**
- Return Type:** void (selected), boolean, int, double, String, Other
- Function arguments:** (Empty table)

Name	Type

New Direction Change: Function "Body"

The screenshot displays the AnyLogic Advanced software interface. On the left, a project tree shows the hierarchy for 'Wandering Elephants*', including 'Elephant', 'Parameters', 'Plain Variables', 'Statecharts', 'Functions', and 'Presentation'. The main workspace shows a statechart with states like 'FreeWandering' and 'GoToWater', and transitions labeled 'GotThirsty' and 'DrinkWater'. 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 arrow pointing to the `setVelocity` line with the text: "Setting Agent Speed (set so as to reach target in fixed time until next target shift)".
- A blue arrow pointing to the `moveTo` line with the text: "Initiates movement towards (randomly chosen) destination".

(Main) Defining a Custom Angle Distribution

The screenshot displays the AnyLogic software interface for defining a custom angle distribution. The main workspace shows a grid with several variables and functions, including 'altitude', 'vegetation', 'mapDrawing', 'altitudesDrawn', and 'altitudeImage'. A vertical color bar is visible on the right side of the workspace.

The 'Properties' panel for the 'DistrAngle' variable is open, showing the following configuration:

- Name:** DistrAngle
- Show name
- Ignore
- Show at runtime
- Access:** public
- Static
- Constant
- Save in snapshot
- Type:** Other: CustomDistribution
- Initial value:** new CustomDistribution(AngleTable, getDefaultRandomGenerator());
- Use Units
- Unit:** [Empty field]

The 'Palette' on the right side of the interface lists various components, including General, System Dynamics, Statechart, Actionchart, Analysis, Presentation, 3D, Controls, Connectivity, Enterprise Library, Pedestrian Library, Rail Library, Road Traffic Library - Preview, Pictures, 3D Objects, and Palettes...

Data for Custom Distribution

The screenshot displays the AnyLogic University software interface. The main workspace shows a simulation model with several variables and functions. The 'AngleTable' function is selected, and its properties are shown in the 'Properties' panel. The 'Table Data' section includes a table with the following data:

Argument	Value
0	0
120	0.118
150	0.134
180	0.217
30	0.22
60	0.175
90	0.133

To the right of the table is a line graph showing the distribution data. The x-axis represents the argument values (0, 120, 150, 180, 30, 60, 90) and the y-axis represents the value (0 to 0.4). The graph shows a red line connecting the data points, with a peak at 180 degrees.

The 'Properties' panel for 'AngleTable - Table Function' includes the following settings:

- Name: AngleTable
- Access: public
- Static:
- Interpolation: Linear
- Out of Range: Error

The 'General' tab is selected, and the 'Table Data' section is expanded. The 'Table Data' section contains a table with the following data:

Argument	Value
0	0
120	0.118
150	0.134
180	0.217
30	0.22
60	0.175
90	0.133

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!)

The screenshot shows the AnyLogic Advanced software interface. On the left is a project tree for 'Wandering Elephants*'. The main workspace displays a statechart with states like 'thirsty', 'NewDir', and 'GoToWater', and transitions like 'GotThirsty' and 'DrinkWater'. The 'headingToWater' function is highlighted in green. Below the statechart, the 'headingToWater - Function' code 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 ) );
```

Initiates movement towards chosen destination

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

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a statechart for an agent named 'Elephant'. The statechart includes a state 'FreeWandering' with a self-loop labeled 'NewDir'. A transition labeled 'GotThirsty' leads from 'FreeWandering' to a state 'GoToWater'. A red arrow points from the text below to the 'On Arrival' field in the 'Elephant - Active Object Class' properties panel.

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

Elephant - Active Object Class Properties:

- General: Space type: Continuous Discrete GIS
- Advanced: Environment defines initial location
- Agent: Initial coordinates: X: Y:
- Description: Movement parameters: Velocity: Rotation:
- On Arrival:

Handling Arrival Events in Statecharts

The screenshot displays the AnyLogic software interface. The main workspace shows a statechart for a 'Person' agent. The statechart starts at an initial state and transitions to 'Susceptible'. From 'Susceptible', there are transitions to 'Infective' (triggered by 'InfectionStatechart') and 'Death'. From 'Infective', there are transitions to 'Susceptible' (triggered by 'colorForRelation') and 'Death'. The 'Death' state is a final state. The statechart is annotated with a red arrow pointing to the 'InfectionStatechart' transition, with the text 'Transition contingent on agent arrival'.

The Properties window shows the 'transition - Transition' properties. The 'Triggered by' dropdown is set to 'Message'. The 'Message type' is 'int'. The 'Class name' is 'Message'. The 'Fire transition' dropdown is set to 'Agent arrival'. The 'Action' and 'Guard' fields are empty.

The left sidebar shows the project structure, including 'ABMModelWithBirthDeath', 'Main', 'Person', 'Parameters', 'Variables', 'Collections', 'Statecharts', 'Functions', 'Presentation', 'IAgentAction', 'IAgentSpecification', 'IMainAction', 'Simulation: Main', 'HierarchicalCityPopulationModelW', 'City', 'The Game of Life', 'Cell', and 'Variables'.

The right sidebar shows the 'Palette' with various components like 'Parameter', 'Event', 'Dynamic Event', 'Variable', 'Function', 'Table Function', 'Schedule', 'Port', 'Connector', 'Environment', 'Agent Population', 'System Dynamics', 'Statechart', 'Actionchart', 'Analysis', 'Presentation', '3D', 'Controls', 'Connectivity', 'Enterprise Library', 'Pedestrian Library', 'Rail Library', 'Road Traffic Library -', 'Pictures', '3D Objects', and 'Palettes...'.

The bottom status bar shows the time '9:41 AM' and the date '3/23/2012'.

Resumption of Wandering After Slaking Thirst

The screenshot displays the AnyLogic Advanced software interface, showing a statechart for an elephant's behavior. The main workspace contains a statechart with the following elements:

- States:** `FreeWandering` (yellow circle), `GoToWater` (yellow circle).
- Transitions:** `DrinkWater` (green arrow), `GotThirsty` (black arrow).
- Initial State:** `FreeWandering`.
- Transitions:** `FreeWandering` has a self-loop labeled `NewDir`. A transition labeled `GotThirsty` leads from `FreeWandering` to `GoToWater`. A transition labeled `DrinkWater` leads from `GoToWater` back to `FreeWandering`.

The **DrinkWater - Transition** properties window is open, showing the following configuration:

- 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 If expression is true (use msg for message)
- Message:** `"drink"`
- Action:**
- Guard:**

The left sidebar shows the project structure for `Wandering Elephants*`, including `Elephant`, `Parameters`, `Plain Variables`, `Statecharts`, `behavior`, `FreeWandering`, `GotThirsty`, `GoToWater`, `DrinkWater`, `initialState`, `state`, `NewDir`, `Functions`, `Presentation`, `Main`, `Parameters`, `NumberOfElephants: 50`, and `Plain Variables`.

The bottom status bar shows **Selection**.

Handling of Movement Logic

The screenshot displays the AnyLogic Advanced interface for an 'Elephant' model. On the left, a project tree shows the 'Elephant' object with its parameters, statecharts, and functions. The main workspace shows a statechart with a 'FreeWandering' state and a 'GoToWater' state. The 'GoToWater' state is triggered by the 'GotThirsty' event and leads to the 'DrinkWater' event, which then returns to the 'FreeWandering' state. The 'GoToWater' state also has a self-loop labeled 'NewDir'.

The 'Elephant - Active Object Class' properties window shows the following code:

```
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;
```

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

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a statechart for an elephant's behavior. 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 with a self-loop. The interface includes a project tree on the left, a palette on the right, and a console window at the bottom showing the Java code for the '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. The main workspace shows a grid with several variables and functions, including 'vegetation', 'DistrDisplacement', 'mapDrawing', 'DistrAngle', 'altitudesDrawn', 'updateVegetation', 'altitudeImage', and 'vegetationDrawn'. The 'updateVegetation' function is highlighted with a blue circle.

The 'updateVegetation - Function' properties window is open, showing the following 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;
```

The interface also includes a Project Search pane on the left, a Palette on the right, and a Problems pane at the bottom left.

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

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a grid with several variables and functions: vegetation, DistrDisplacement, mapDrawing, DistrAngle, altitudesDrawn, updateVegetation, altitudeImage, and vegetationDrawn. A small map visualization is visible on the right side of the grid.

The left sidebar contains a project tree with the following structure:

- Main
 - Parameters
 - NumberOfElephants: 50
 - Plain Variables
 - Functions
 - AngleTable
 - DisplacementTable
 - altitudeToColor
 - makeUpAltitudes
 - makeUpVegetation
 - placeElephants
 - updateVegetation
 - vegetationToColor
 - Environments
 - environment
 - Embedded Objects
 - elephants
 - Presentation
 - Simulation: Main
 - Presentation
 - ABMClinicModelV6
 - Intern

The bottom-left pane shows a 'Problems' table with columns for 'Description' and 'Location':

Description	Location

The bottom-right pane shows the 'environment - Environment' configuration window. The 'General' tab is active, displaying the following settings:

- Name: environment
- Show Name
- Ignore
- Public
- Show At Runtime
- Enable steps
- Step duration (in model units): []
- On before step: []
- On after step: `updateVegetation();`

The right sidebar contains a 'Palette' with various model components:

- Model
 - 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
- Action
- Analysis
- Presentation
- Connectivity
- Enterprise Library
- More Libraries...



Hands on Model Use Ahead



Load model: Schelling Segregation.alp

A Model to Examine the Emergence of Segregation

The screenshot displays the AnyLogic Advanced [EDUCATIONAL USE ONLY] interface. The main workspace shows a simulation titled "Schelling Segregation" with a "Main" object class selected. The simulation area features a large yellow square representing the environment, surrounded by a dark red border. A slider control at the top of the simulation area is labeled "Preference to have neighbors of the same color:" and is set to 30%. The interface includes a menu bar (File, Edit, View, Model, Window, Help), a toolbar with various icons, and several panels:

- Project Panel:** Lists the model structure, including Plain Variables, Statecharts, Presentation, Simulation: Main, Schelling Segregation, Main, Parameters, Environments, environment, Embedded Objects, Presentation, Person, Parameters, Plain Variables, satisfied, and Simulation: Main.
- Problems Panel:** A table with columns for Description and Location.
- Properties Panel:** Shows the "Main - Active Object Class" properties, including Name (Main), Ignore checkbox, Agent/Agent Generic checkboxes, and Startup/Destroy Code fields.
- Palette Panel:** Lists various model elements 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.
- Action Panel:** Includes buttons for Action, Analysis, Presentation, Connectivity, Enterprise Library, and More Libraries...

The simulation area also shows a "Threshold" control and a "people [...]" object. The status bar at the bottom indicates "Selection".

A Discrete Spatial Environment with Random Agent Positioning

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a grid-based environment with a red box containing a slider labeled "Preference to have neighbors of the same color:" and a yellow box below it. A red arrow points to the "Spatial Width & Height" text, which is positioned above the "environment - Environment" properties panel. A blue box highlights the "Width & Height in Discrete Cells" section of the properties panel, with a blue arrow pointing to the "Width" and "Height" fields. The "environment - Environment" properties panel includes the following settings:

- 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

The interface also shows a Project tree on the left, a Palette on the right, and a Properties/Console area at the bottom.

Population Dependence on the Population

The screenshot displays the AnyLogic Advanced software interface, titled "AnyLogic Advanced [EDUCATIONAL USE ONLY]". The main workspace shows a simulation model for "Schelling Segregation" with a grid background. A red-bordered window titled "Preference to have neighbors of the same color:" is visible, containing a slider control. The "Person" entity is highlighted in the "Project" pane on the left. The "Properties" pane at the bottom shows the configuration for the "people - Person" entity.

Project Pane (Left):

- Plain Variables
- Statecharts
- Presentation
- Simulation: Main
- Schelling Segregation
 - Main
 - Parameters
 - Environments
 - environment
 - Embedded Objects
 - people
 - Presentation
 - Person
 - Parameters
 - Plain Variables
 - satisfied
 - Presentation
 - Simulation: Main

Properties Pane (Bottom):

people - Person

General

Name: Show Name Ignore Public Show At Runtime

Parameters

Type:

Package:

Environment:

Color

Replication:

Palette (Right):

- Model
 - 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
- Action
- Analysis
- Presentation
- Connectivity
- Enterprise Library
- More Libraries...

Slider Input Sets Parameter Value

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a simulation environment with a slider widget. A green arrow points from the text "Threshold" parameter to the slider widget. The slider is currently set to 30% and is labeled "Preference to have neighbors of the same color: 30%".

The Properties panel for the "slider - Slider" widget is visible at the bottom. It shows the following configuration:

- Name: slider
- Orientation: Horizontal Vertical
- Minimum Value: 0
- Maximum Value: 1
- Default Value: Threshold
- Enabled:
- Action: `Threshold = value;`

Annotations in the image include:

- A green arrow pointing to the "Threshold" parameter in the simulation environment.
- A blue arrow pointing to the "Default Value" field in the Properties panel, with the text "Default value is that of Threshold parameter".
- A red arrow pointing to the "Action" field in the Properties panel, with the text "Sets Threshold Parameter Value".

Person is Assigned a Randomly Picked Color

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a simulation model with a grid background. A red square represents the 'Person's Visual Representation', and a blue circle represents the 'color' parameter. A red arrow points from the text 'Person's Visual Representation' to the red square. A blue arrow points from the text 'Color is set to either red or black with equal likelihood' to the blue circle. The 'color - Parameter' properties window is open, showing the following details:

- Name: color
- Show Name: Show Name
- Ignore: Ignore
- Public: Public
- Show At Runtime: Show At Runtime
- Type: void (just action) boolean int double String Other: Color
- Default Value: `randomTrue(0.5) ? Color.red : Color.black`
- Dynamic: Dynamic
- Save in snapshot: Save in snapshot
- On Change:

The left sidebar shows a project tree with the following structure:

- people_presentation
 - Aa text: Prefer...
 - Aa text1: 30%
 - button
 - button1
 - slider
- Person
 - Parameters
 - color: random...
 - Plain Variables
 - satisfied
 - Presentation
 - Simulation: Main
 - Presentation
 - frame
 - rect1
 - rect
 - Aa text: Schell...

The right sidebar shows a palette of model elements, including:

- Model
 - 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
- Action
- Analysis
- Presentation
- Connectivity
- Enterprise Library
- More Libraries...

Core Segregation (Movement) Logic

AnyLogic Advanced [EDUCATIONAL USE ONLY]

File Edit View Model Window Help

100%

Project Search

Person

Person - Active Object Class

Space type: Continuous Discrete GIS

Environment defines initial location ← Person's Initial Location

Initial coordinates:

X:

Y:

Movement parameters:

Velocity:

Rotation:

On Arrival:

On Message Received:

On Before Step:

```
//calc hbow many neighbors have same color as me
int nsame = 0;
Agent[] neighbors = getNeighbors();
if( neighbors == null ) {
    satisfied = true; //no neighbors is good too
    return;
}
for( Agent a : neighbors )
    if( ((Person)a).color.equals( color ) )
        nsame++;
//satisfied if percent of same color is greater than a given threshold
satisfied = nsame >= get_Main().Threshold * neighbors.length;
```

On Step:

```
if( ! satisfied && randomTrue( 0.3 ) )
    jumpToRandomEmptyCell();
```

Count neighbors
Sharing same colour
(should be in diff.
Function).

Only satisfied if fraction of
surrounding individuals
Sharing color exceeds
threshold

if dissatisfied,
30% chance of moving

Model

- 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

Action

Analysis

Presentation

Connectivity

Enterprise Library

More Libraries...

Experiment: Simulation Sets

Parameter Assumptions

The screenshot displays the AnyLogic Advanced software interface. The main workspace is a dark red area containing the following text:

Schelling Segregation Model

The Schelling Segregation Model was first developed by Thomas C. Schelling (Micromotives and Macrobehavior, W. W. Norton and Co., 1978, pp. 147-155). It represents one of the first constructive models of a dynamical system capable of self-organization.

Schelling placed pennies and dimes on a chess board and moved them around according to various rules. He interpreted the board as a city, with each square of the board representing a house or a lot. He interpreted the pennies and dimes as agents representing any two groups in society, such as two different races of people, boys and girls, smokers and non-smokers, etc. The neighborhood of an agent occupying any location on the board consisted of the squares adjacent to this location. Thus, interior agents had eight neighbors while boundary agents had either three or five neighbors. Rules could be specified that determined whether a particular agent was happy in its current location. If it was unhappy, it would try to move to another location on the board, or possibly just exit the board entirely.

As can be expected, Schelling found that the board quickly evolved into a strongly segregated location pattern if the agents' "happiness rules" were specified so that segregation was heavily favored. Surprisingly, however, he also found that initially integrated boards tipped into full segregation even if the agents' happiness rules expressed only a mild preference for having neighbors of their own type.

Run the model and switch to Main view

The interface includes a left sidebar with a project tree showing a hierarchy: slider, Person, Parameters (color: random...), Plain Variables (satisfied), Presentation, Simulation: Main, and Presentation (frame, rect1, rect, Aa text: Schell..., Aa text1: The Sc..., image, Aa text2: AnyLog..., button). Below the sidebar is a Problems table with columns for Description and Location. The bottom panel shows the Properties and Console tabs, with the Simulation - Simulation Experiment tab active, displaying a Threshold parameter set to 0.7. The right sidebar contains a Palette with various model components 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. At the bottom right, there are buttons for Action, Analysis, Presentation, Connectivity, Enterprise Library, and More Libraries...

Add a Parameter to Main

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a grid with several objects: a slider control for 'Threshold' set to 30%, and parameters for 'environment', 'people [...]', and 'likelihoodOfMovementIfDissatisfied'. A red box highlights the slider control.

The left sidebar contains a Project view with a tree structure including:

- Threshold: 0.7
- likelihoodOfMovementIfDissatisfied: C
- Environments
 - environment
- Embedded Objects
 - people
- Presentation
 - rect
 - rect1
 - people_presentation
 - text: Prefer...
 - text1: 30%
 - button
 - button1
 - slider
- Person
 - Parameters
 - color: random...

The bottom panel shows the Properties window for the selected parameter, 'likelihoodOfMovementIfDissatisfied - Parameter'.

likelihoodOfMovementIfDissatisfied - Parameter

General

Name: Show Name Ignore Public Show At Runtime

Type: void (just action) boolean int double String Other:

Default Value:

Dynamic Save in snapshot

On Change:

The right sidebar shows the Palette with various model elements, 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, and Environment.

Setting the Slider Properties

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a simulation model with a red background and white text. The text describes the Schelling model of segregation, mentioning the work of W. W. Norton and Co. (1978) and the behavior of agents on a chessboard. A slider control is visible at the bottom of the workspace, with a button labeled "Run the model and switch to Main view".

The Properties window is open, showing the configuration for the "sliderMovementChance - Slider". The properties are as follows:

Property	Value
Name	sliderMovementChance
Show Name	<input type="checkbox"/>
Ignore	<input type="checkbox"/>
Public	<input checked="" type="checkbox"/>
Icon	<input type="checkbox"/>
Orientation	<input checked="" type="radio"/> Horizontal <input type="radio"/> Vertical
Minimum Value	0
Maximum Value	1
Default Value	0.3
Enabled	true
Action	

The left sidebar shows the Project tree with the following structure:

- Person
 - Parameters
 - color: random...
 - Plain Variables
 - satisfied
 - Presentation
 - Simulation: Main
 - Presentation
 - frame
 - rect1
 - rect
 - Aa text: Schell...
 - Aa text1: The Sc...
 - image
 - Aa text2: AnyLog...
 - button
 - sliderMovementChance

The right sidebar shows the Palette with various model elements:

- Model
 - 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
- Action
- Analysis
- Presentation
- Connectivity
- Enterprise Library
- More Libraries...

Setting Value for Parameter from Slider

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a simulation model with a red rectangular area containing a slider. The slider is labeled "Preference to have neighbors of the same color:" and is currently set to 30%. Below the slider is a yellow rectangular area. The interface includes a menu bar (File, Edit, View, Model, Window, Help), a toolbar, and several panels: Project, Search, Palette, and Properties. The Properties panel is open to the "Simulation - Simulation Experiment" section, showing the "Threshold" property set to 0.7 and the "likelihoodOfMovementIfDissatisfied*" property set to "sliderMovementChance.value".

Simulation - Simulation Experiment

General

Name: Main active object class (root): Ignore

Advanced

Random number generation:

Random seed (unique simulation runs)

Fixed seed (reproducible simulation runs) Seed Value:

Model Time

Presentation

Window

Parameters

Description

Threshold:

likelihoodOfMovementIfDissatisfied*:

Modify Person's Behavior to Depend on New Parameter

Updated Code ("get_Main()") required
Because new parameter is global
And lives in Main class rather than in
Person class.)

```
Person - Active Object Class  
General  
Advanced  
Agent  
Parameters  
Description  
On Step:  
if( ! satisfied && randomTrue( get_Main().likelihoodOfMovementIfDissatisfied ) )  
    jumpToRandomEmptyCell();
```

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, SOUTHEAST, 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

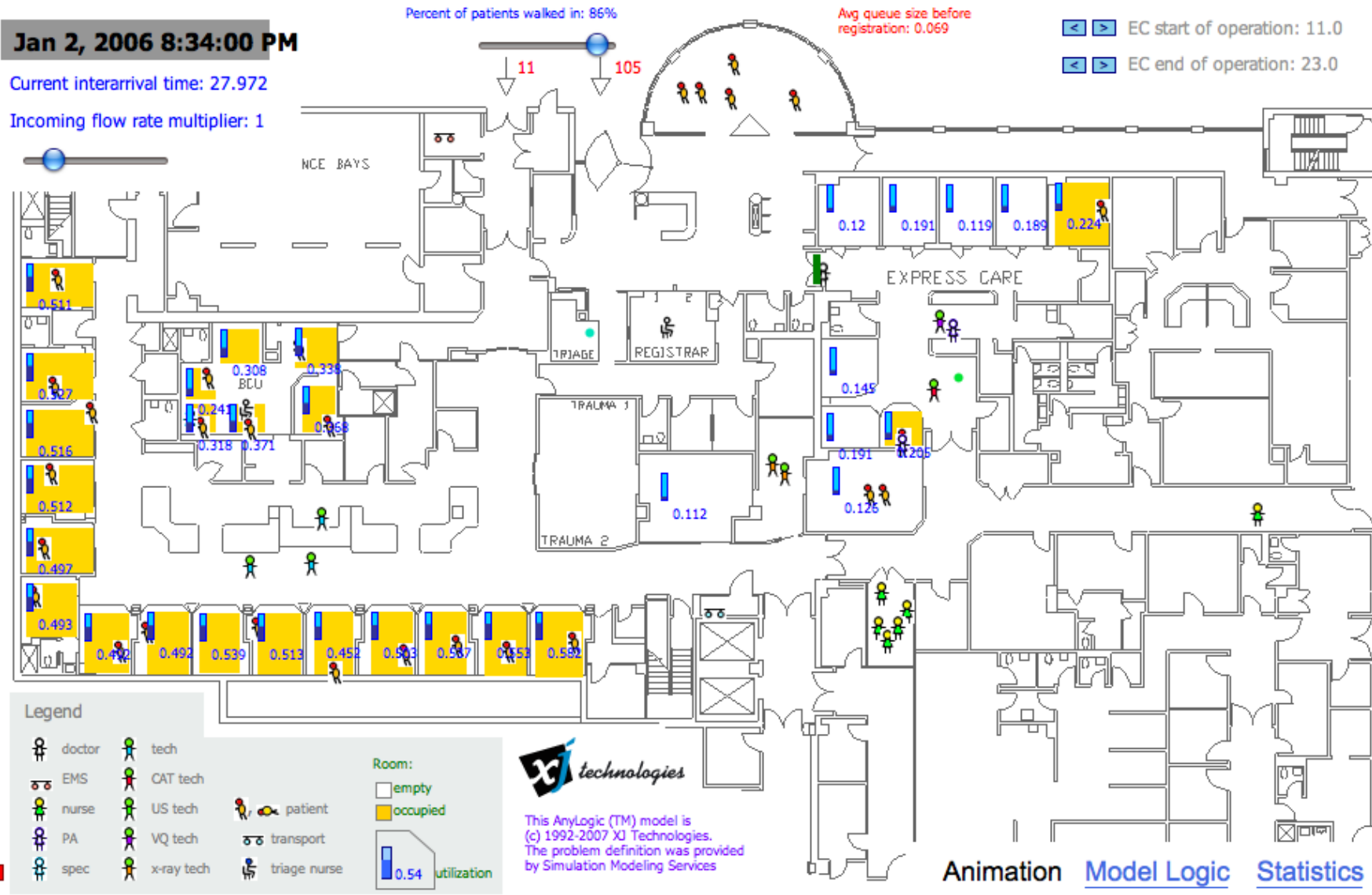
- Suppose an agent is moving in discrete 2D space and need to be concerned about moving into the same cell as another agent
- We can readily prevent this agent from moving into another cell currently occupied
- But can we prevent this agent from colliding with another agent that wishes to move into the 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

Emergency Department



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