

# 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

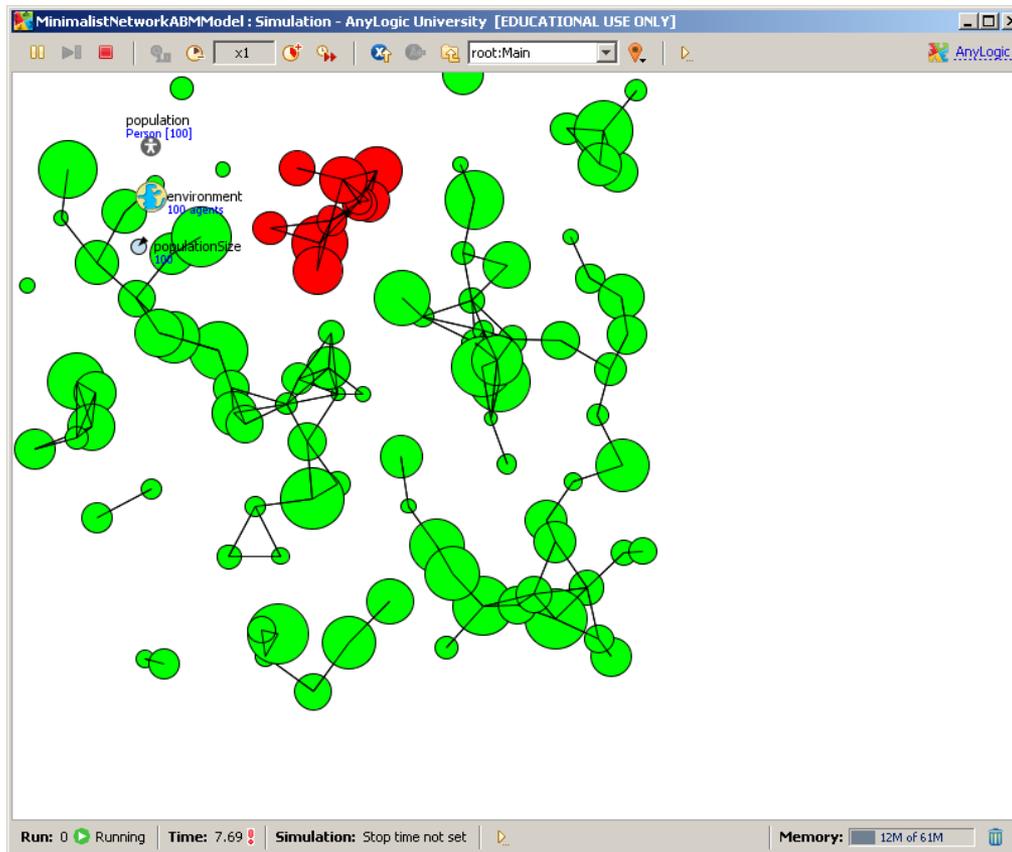
The screenshot displays the AnyLogic Advanced software interface, titled "AnyLogic Advanced [EDUCATIONAL USE ONLY]". The main workspace shows a simulation environment with a grid and various components. A pink box highlights the "environment" component. The interface includes a menu bar (File, Edit, View, Model, Window, Help), a toolbar, and several panels:

- Project Panel:** Lists the project structure, including "Main", "Parameters", "Plain Variables", "Functions", "Environments", "Embedded Objects", "Presentation", "Simulation: Main", and "ABMClinicModelV6".
- Environment Properties Panel:** Shows the configuration for the "environment - Environment" component. The "Space type" is set to "Continuous". Other settings include "Width: 500", "Height: 500", "Columns: 100", "Rows: 100", "Neighborhood type: Moore", "Layout type: User-defined", and "Network type: User-defined".
- Palette:** A vertical panel on the right side containing various simulation components 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".

The main workspace contains several components and variables, including "SmokingInitiationByAgeAndSmokingStatusForSexualActivityGroup1", "makeUpVegetation", "placeElephants", "altitude", "vegetation", "mapDrawing", "altitudesDrawn", "DisplacementTable", "AngleTable", "DistrDisplacement", "DistrAngle", "updateVegetation", "vegetationToColor", "altcolor", and "viewVegetation". A vertical color bar is visible on the right side of the workspace.

# 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, specifically the configuration for a discrete environment. 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 'Advanced' tab, 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 overlay on the right side of the Properties panel reads: "Note extra presence of 'Columns' and 'Rows'".

The interface also shows a Project tree on the left, a Palette on the right, and a Console at the bottom. The main workspace contains several objects, including a function named "SmokingInitiationByAgeAndSmokingStatusForSexualActivityGroup1" and a variable named "environment".



Hands on Model Use Ahead



Load AnyLogic Sample Model: The Game  
of Life

# The “Game” of Life: Background

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

# The Behavioral Rules of the Game of Life

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

# Open "Main" Class

## Scroll Left to See Population & Environ.

The screenshot displays the AnyLogic University software interface. The main workspace shows a simulation titled "The Game of Life" at "Step: 2". The workspace contains a red rectangular area representing the simulation environment and a yellow rectangular area representing the population. 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 status bar at the bottom indicates "Selection" and "X=288, 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", with a yellow grid on a dark red background. The left sidebar contains a project tree with "The Game of Life" selected. The bottom-left pane shows "Problems" with "No problems" listed. The bottom-right pane shows the "environment - Environment" properties, with "Columns" and "Rows" highlighted in pink. A red arrow points from the "100x100 grid defined here" text to the "Columns" field. A blue arrow points from the "Indicated that cells should be laid out in a regular grid in space" text to the "Layout type" dropdown.

100x100 grid defined here

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

environment - Environment

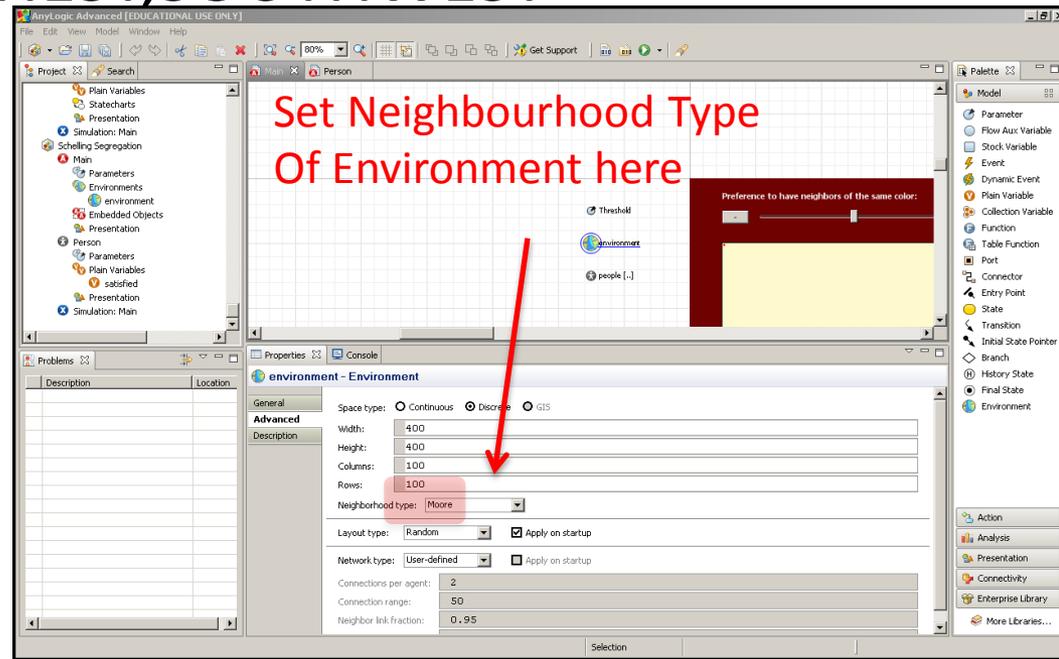
Space type:	<input type="radio"/> Continuous2D <input type="radio"/> Continuous3D <input checked="" type="radio"/> Discrete2D <input type="radio"/> GIS
Width:	500
Height:	500
Z-Height:	0
Columns:	100
Rows:	100
Neighborhood type:	Moore
Layout type:	Arranged <input checked="" type="checkbox"/> Apply on startup
Network type:	User-defined <input type="checkbox"/> Apply on startup
Connections per agent:	2
Connection range:	50
Neighbor link fraction:	0.95
M:	10

# Environment: Enabling Discrete Space (Cells)

The screenshot shows the AnyLogic software interface. The main workspace displays a simulation titled "The Game of Life" at "Step: 2", showing a grid of cells. The "environment" object is selected in the workspace. The "Properties" window for "environment - Environment" is open, showing the "General" tab. Under "Space type", the "Discrete2D" radio button is selected. A red arrow points from the text "Discrete2D selected" to this button. The "Neighborhood type" is set to "Moore", with a blue arrow pointing to it from the text "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 configuring a cell agent. The main workspace shows a grid with a yellow area and a red border, labeled "The Game of Life Step: 2". The left sidebar shows a project tree with "The Game of Life" selected, containing a "Cell" agent. The bottom panel shows the "cells - Cell" configuration window. In the "General" tab, the "Initial number of objects" is set to 10000, which is highlighted with a red box and a red arrow pointing to the text "10,000 (= 100\*100) agents" below it. Other settings include "Name: cells", "Type: Cell", "Environment: environment", and "Package: the\_game\_of\_life". The "Replicated" checkbox is checked, and "Access by index (ArrayList)" is selected for optimization.

cells - Cell

General

Name: cells  Show name  Ignore  Public  Show at runtime [Create presentation](#)

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**

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: Simulation: Main > 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.

Boolean (true/false) variable

Name would be clearer as “isAlive”

10% initial likelihood of being occupied

alive - Variable

General

Name:   Show name  Ignore  Show at runtime

Access:   Static  Constant  Save in snapshot

Type:  boolean  int  double  String  Other:

Initial value:

Use Units Unit:

# Cell Variables: “neighbors”

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

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:

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)

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' variable. The main workspace shows a grid with variables 'alive', 'neighbors', 'nAliveAround', and 'toggleState'. The 'nAliveAround' variable is highlighted in red. A red arrow points to it from the explanatory text. Below the workspace is the 'Properties' panel for the 'nAliveAround - Variable'. The 'Type' field is set to 'int', which is highlighted in green. A green arrow points to this field from the explanatory text. The 'General' tab is active, showing the variable's name, access, and type.

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

The screenshot displays the AnyLogic software interface. On the left, the 'Projects' tree shows a hierarchy including 'Cell' with sub-items 'Variables' (alive, nAliveAround, neighbors) and 'Functions' (toggleState). 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' object. A blue arrow points from the text 'Selects appearance depending on whether alive or not' to the 'Fill color' field, which contains the expression 'alive ? red : lemonChiffon'. The 'Dynamic' section of the properties window is highlighted.

**Select this item**

Selects appearance depending on whether alive or not

**rectangle - Rectangle**

General	Replication:
Advanced	Visible:
Dynamic	
Description	
	X: <input type="text"/>
	Y: <input type="text"/>
	Z: <input type="text"/>
	Fill color: <input type="text" value="alive ? red : lemonChiffon"/>
	Width: <input type="text"/>
	Height: <input type="text"/>
	Z-Height: <input type="text"/>

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

The screenshot displays the AnyLogic University interface for configuring a "Cell" agent. The main workspace shows a statechart with variables: `alive`, `neighbors`, `nAliveAround`, and a function `toggleState`. The Properties panel for the "Cell - Active Object Class" is open, showing 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 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-left corner shows a "Problems" panel with the message "No problems".

# 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 ( $\Delta t$ ) 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”



# 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 'Properties' window also shows the 'Console' tab, which is currently empty.

# 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