Agent Mobility in 2D Landscapes
(Bonus: Some UI Customization)

Nathaniel Osgood
MIT 15.879

March 23, 2012
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
Landscape Information
Agent Movement: Periodic Movement Changes
New Direction Change Function Info
New Direction Change: Function “Body”

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

Initiates movement towards (randomly chosen) destination
(Main) Defining a Custom Angle Distribution
Data for Custom Distribution
**Heading Towards Resource**

Determining current position & Searching for quickest way to find water from that position.

(should be in separate function!)

Looking at body of this function (method)

Initsiates movement towards chosen destination

```java
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 stops
moveTo(x + 1000*cos(heading), y + 1000*sin(heading));
```
Handling Agent Arrival at Destination

(Not Currently Used in this Model)

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

Transition contingent on agent arrival
Resumption of Wandering After Slaking Thirst
Handling of Movement Logic

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.
Environment: Updating Vegetation

```java
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]) * 10); //reset flag

vegetationDrawn = false;
```
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
Hands on Model Use Ahead

Load model: Schelling Segregation.alp
A Model to Examine the Emergence of Segregation
A Discrete Spatial Environment with Random Agent Positioning

Spatial Width & Height

Width & Height in Discrete Cells
Population Dependence on the Population
Slider Input Sets Parameter Value

“Threshold” parameter

Default value is that of Threshold parameter

Sets Threshold Parameter Value
Person is Assigned a Randomly Picked Color

Color is set to either red or black with equal likelihood

Person’s Visual Representation
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.
Schelling Segregation Model

The Schelling Segregation Model was first developed by Thomas C. Schelling (Microsocieties and Macrobehavior, M. Wi. Norton and Co., 1978, pp. 247-185). 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. Each neighborhood of an agent occupying any location on the board consisted of the squares adjacent to that location. Each agent 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
Add a Parameter to Main

Parameter: likelihoodOfMovementIfDissatisfied
Type: double
Default Value: 0.3

Preference to have neighbors of the same color: 30%
Experiment: Add a Slider!

Schelling Segregation Model

The Schelling Segregation Model was first developed by Thomas C. Schelling (Microsocieties and Macrobehavior, W. W. Norton and Co., 1978, pp. 447-156). It represents one of the first interactive 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 his 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 highly favored. Surprisingly, however, he also found that initially mixed 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
Setting the Slider Properties

(Micro-motives and Micro-behavior, 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 nearly 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.
Setting Value for Parameter from Slider
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.)
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
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