

Agent-Based Modelling and Simulation in AnyLogic

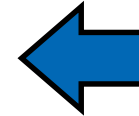
Cells and Birds

Agents in AnyLogic

Types of agents in Anylogic

- Population of agents
- A single agent
- Agent type

→ “real” ABMS



→ a “normal” model element

→ a type / Java class

Agent Elements

- Speed, position, rotation
- Custom Parameters
- Animation Shape
- Functions, variables, etc.

Agents Environment & Connections in AnyLogic

Discrete Environment

- Grid & Dimensions: columns, rows, width, height
- Neighborhood type: Moore/Euclidean
- Layout: Random, Arranged
- Network: Random, Ring lattice, Small world, Scale-free, User-defined

Continuous Environment

- Dimensions: width, height, z-height
- Layout: Random, Arranged, Ring, Spring mass, User-defined
- Network: Random, Distance-based, Ring lattice, Small world, Scale-free, User-defined

GIS Environment

- maps, routes, ...

Agents Movements in AnyLogic

Discrete Space

- Current position: row / column
- Moving in specific direction
- Jumping to target cell / random cell
- Swapping with other agent / cell

Agent.NORTHWEST	Agent.NORTH	Agent.NORTHEAST
Agent.WEST	currently occupied cell	Agent.EAST
Agent.SOUTHWEST	Agent.SOUTH	Agent.SOUTHEAST

Continuous Space

- Current position (x,y,z) and target location (x,y,z)
- Move towards specific coordinates (x,y,z)
- Move towards nearest / specific agent
- Move to node / attractor
- Speed or trip time are explicitly set

Agents Interactions in AnyLogic

General (Discrete & Continuous)

- Network determines connected agents, on startup or dynamically
- Connections can be explicitly established and cut
- *Agent[] getConnections()* – returns list of all connected agents

Discrete Space

- Neighborhood also determines connected agents
- *Agent[] getNeighbors()* – returns list of all agents in chosen neighborhood (Moore/Euclidean)

Communication

- Connected agents can access each others properties
- Agents can send messages to connected or other agents

Agents (inter-)actions can be synchronized by using steps

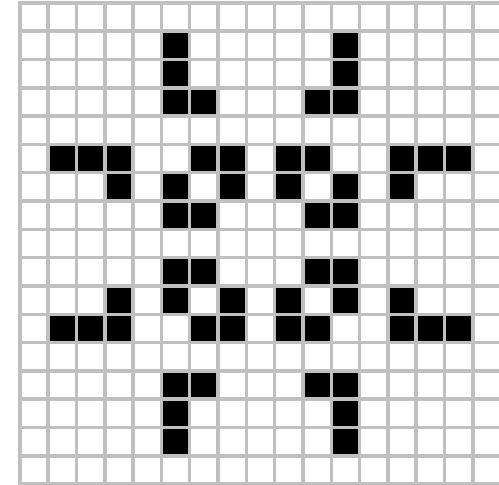
- Specific code fragments for: on before step, on step, on after step

Game of Life

Simple Examples

Game of Life

- Cellular automaton
- Each cell can be either alive or dead
- Next generation state depends on Moore-neighborhood



Rules

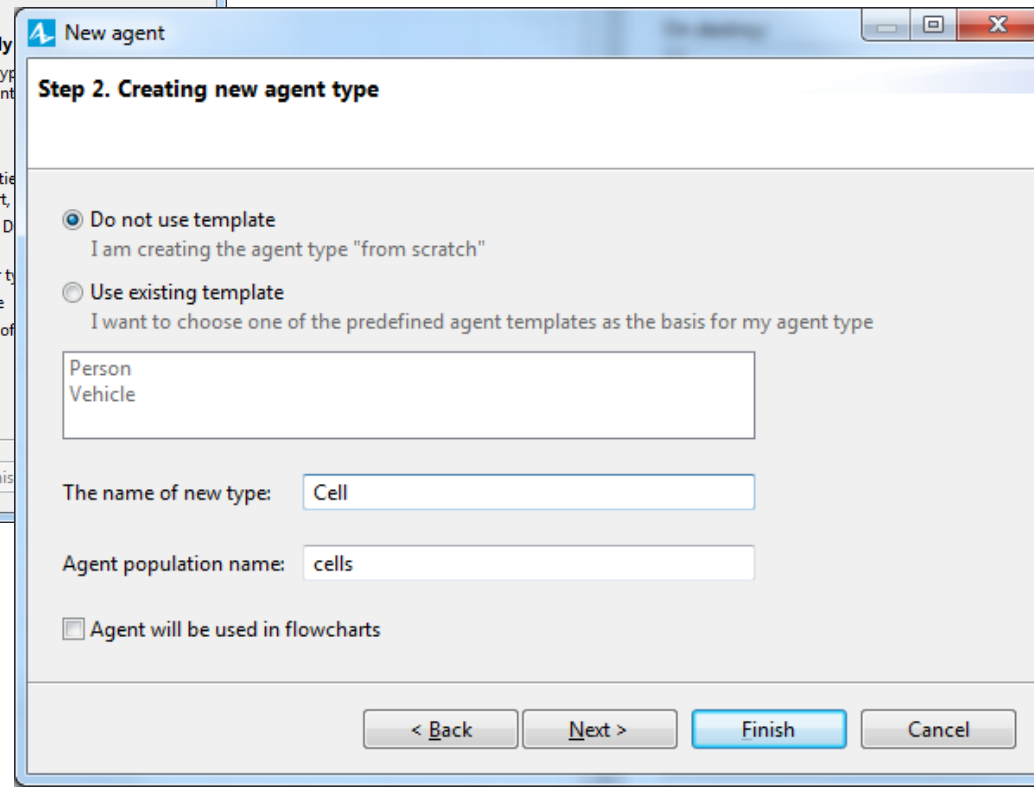
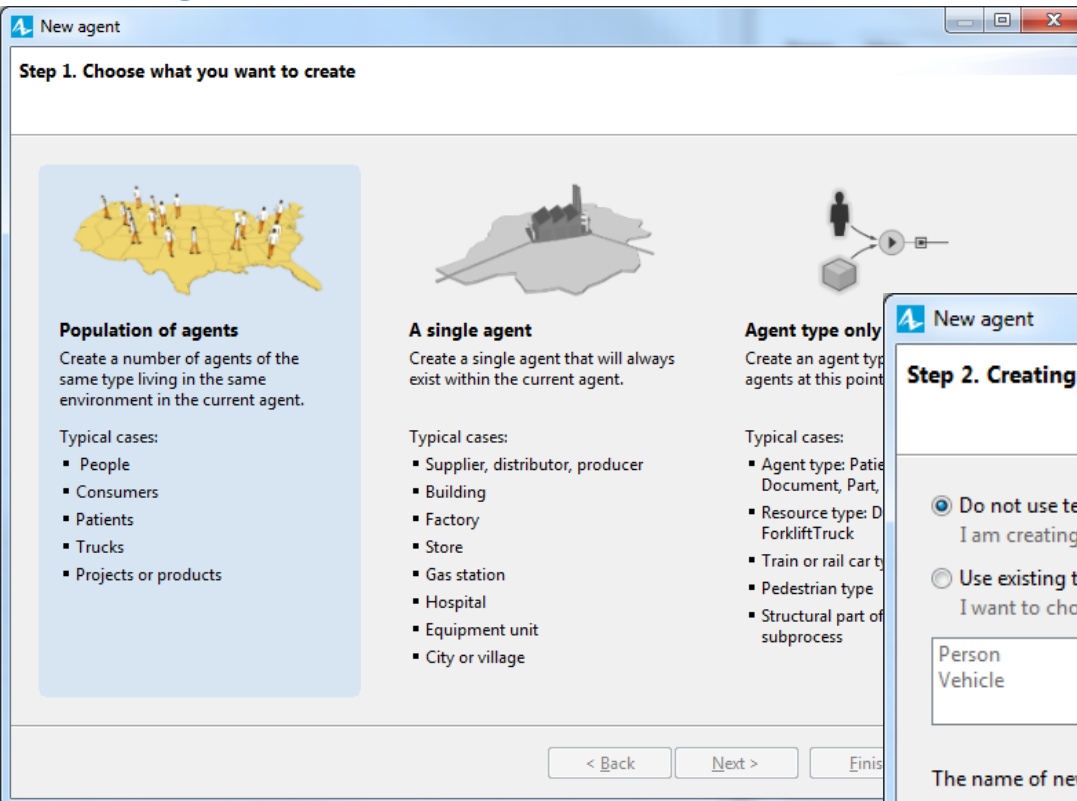
1. A dead cell with 3 live neighbors comes alive
2. A living cell with less than 2 live neighbors dies
3. A living cell with 2 or 3 live neighbors stays alive
4. A living cell with more than 3 live neighbors dies

Specifics

- Grid 50x50 → 2500 Cells

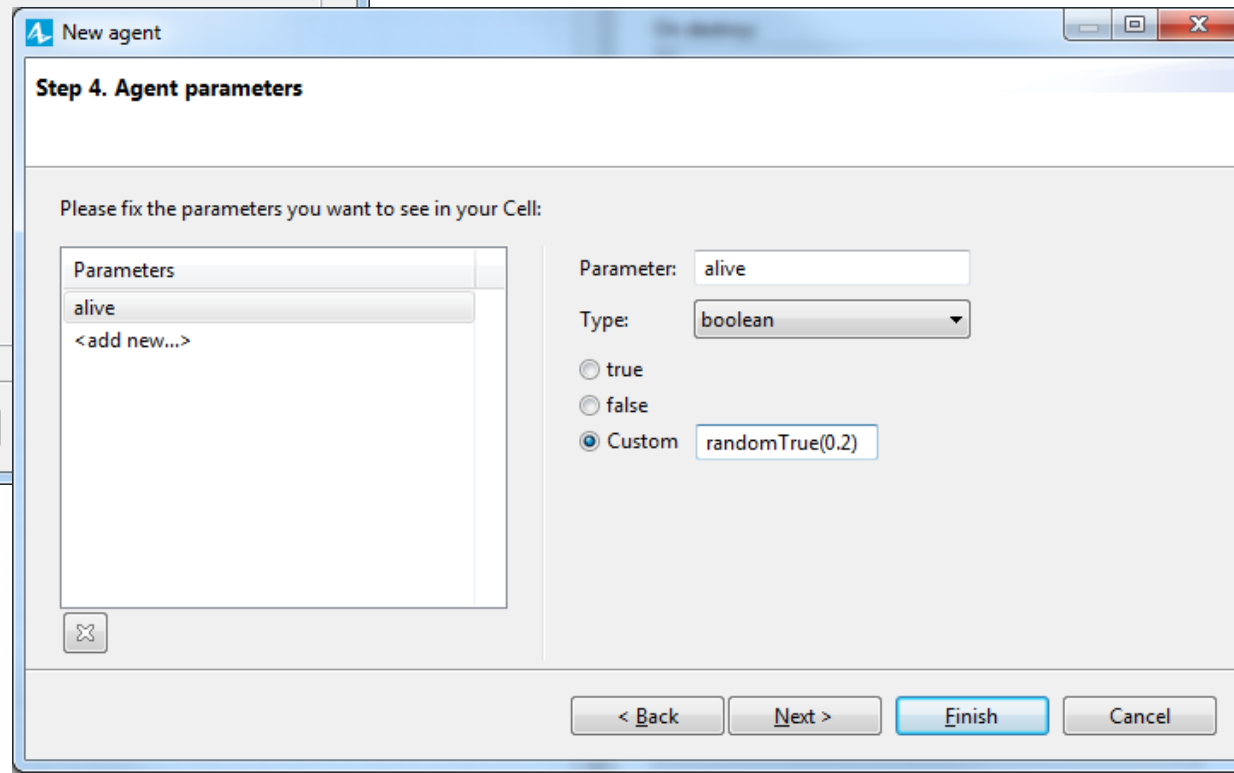
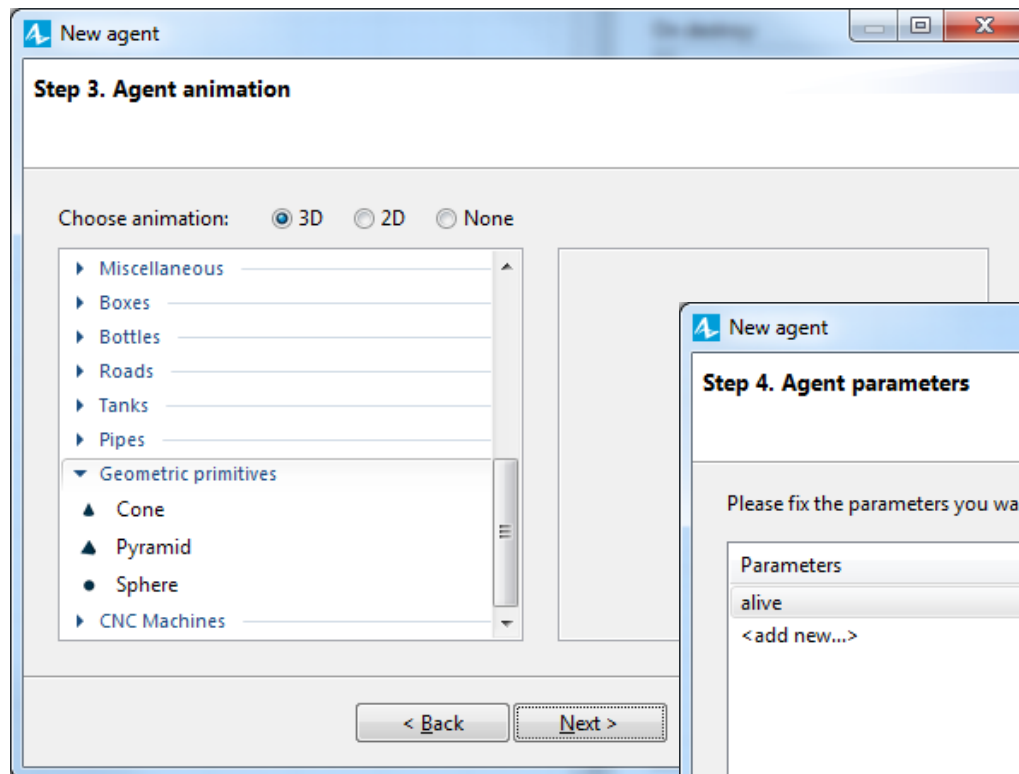
Game Of Life

Agent Type & Name



Game Of Life

Animation & Properties



Game Of Life

Population Size and Environment

New agent

Step 5. Population size

☒ Create population with agents

This is the initial population size.
You will be able to add more agents or delete any agent at runtime.

☐ Create initially empty population, I will add agents at the model runtime

< Back Next >

New agent

Step 6. Configure new environment

This agent will live in the 'Main' agent type.
The following are the environment settings.
You can always change them from the properties of Main agent type (see Space and network section)

Space type: ☐ Continuous ☐ GIS ☒ Discrete

Size: x

Cells: x

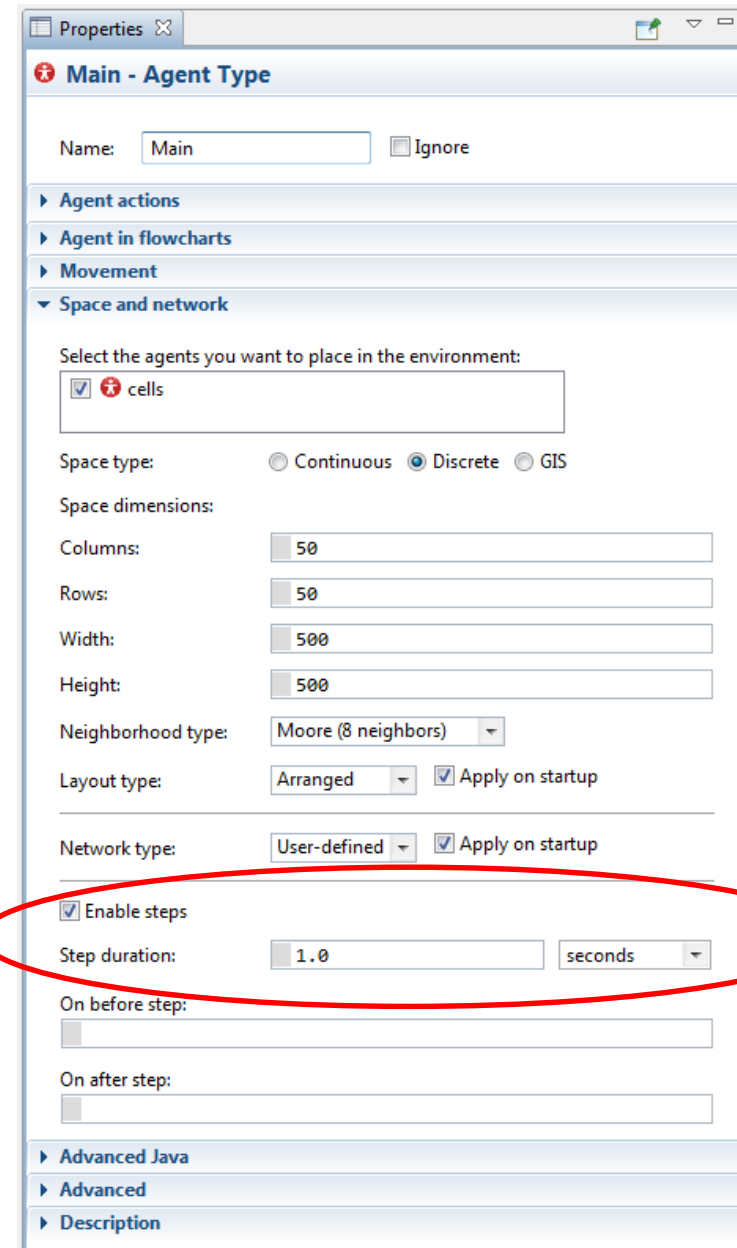
Initial location:

Neighborhood type:

< Back Next > Finish Cancel

Game Of Life

Synchronize cell generations




Properties

Main - Agent Type

Name: ☐ Ignore

- Agent actions
- Agent in flowcharts
- Movement
- Space and network

Select the agents you want to place in the environment:

☒  cells

Space type: ☐ Continuous ☒ Discrete ☐ GIS

Space dimensions:

Columns:

Rows:

Width:

Height:

Neighborhood type:

Layout type: ☒ Apply on startup

Network type: ☒ Apply on startup

☒ Enable steps

Step duration:

On before step:

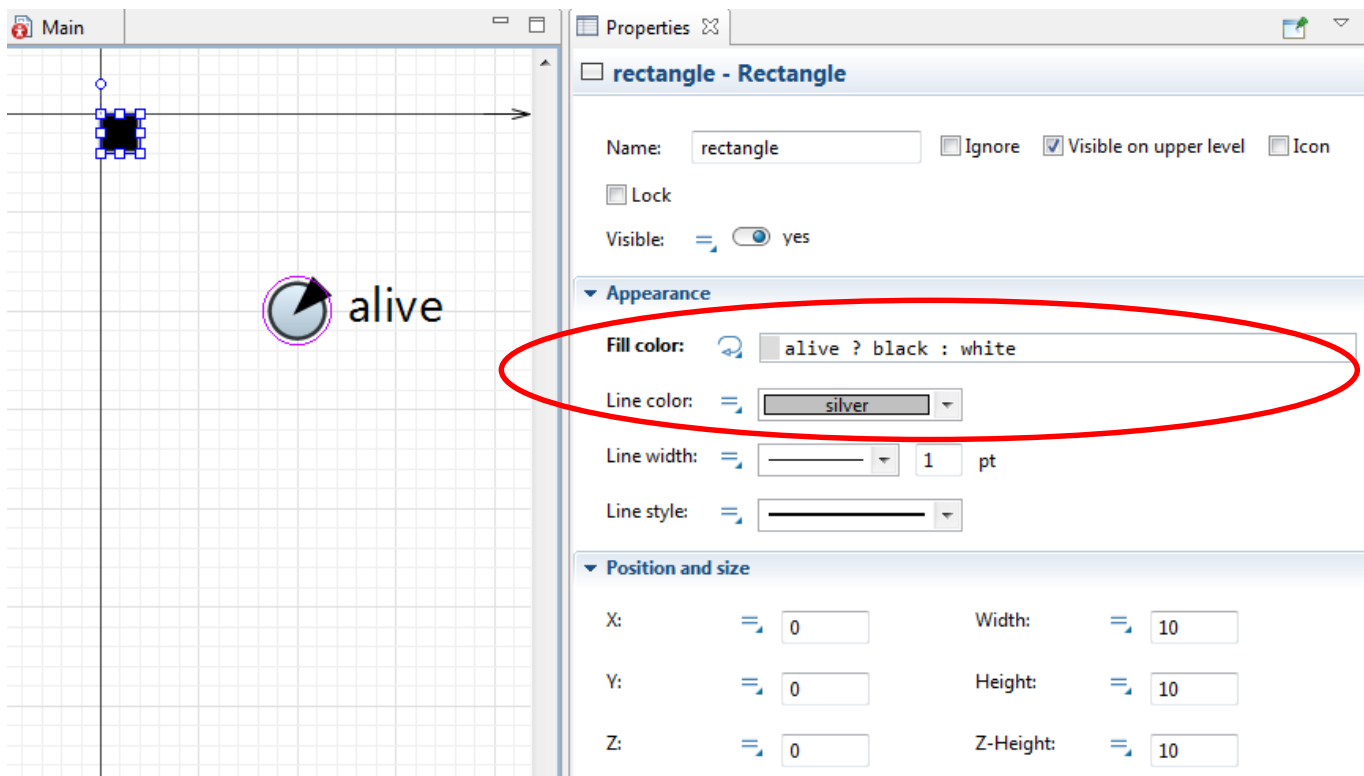
On after step:

- Advanced Java
- Advanced
- Description

Game Of Life

Agents graphical representation

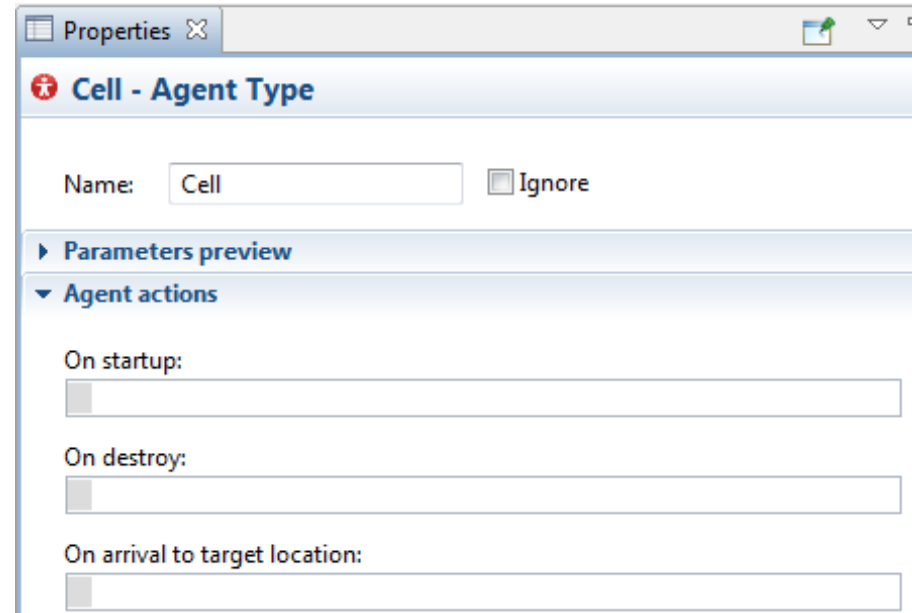
- Rectangle with dynamic fill color 10x10 pixels
- Line color results in regular grid



Game Of Life

Dynamic behavior

1. A dead cell with 3 live neighbors comes alive
2. A living cell with less than 2 live neighbors dies
3. A living cell with 2 or 3 live neighbors stays alive
4. A living cell with more than 3 live neighbors dies



The screenshot shows a 'Properties' window for a 'Cell - Agent Type'. It includes a 'Name' field set to 'Cell' and an 'Ignore' checkbox. Below are sections for 'Parameters preview' and 'Agent actions'. The 'Agent actions' section contains three fields: 'On startup:', 'On destroy:', and 'On arrival to target location:', each with a small square icon to its left.

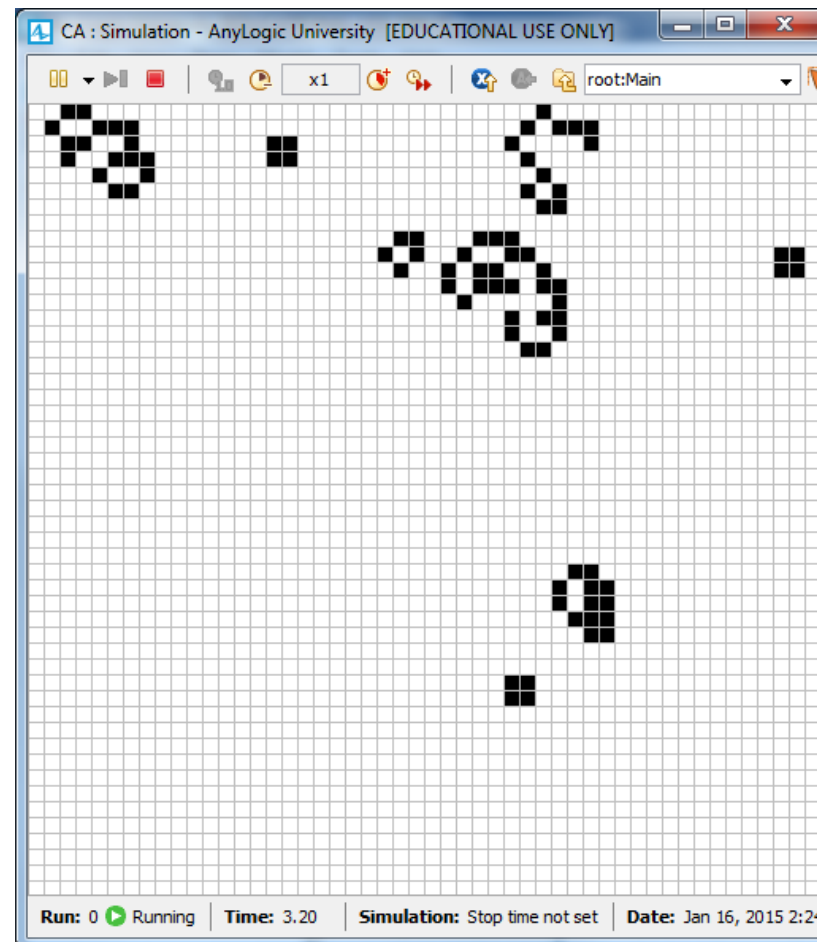
On step:

```
//count live neighbors
int liveNeighbors = 0;
for(Agent a : getNeighbors())
    if(((Cell)a).alive) liveNeighbors++;
//rule 1
if(!alive && liveNeighbors == 3)
    alive = true;
//rule 2 & 4
if(alive && (liveNeighbors < 2 || liveNeighbors > 3))
    alive = false;
```

s > 3))

Game Of Life

Resulting dynamic behavior



Game Of Life

Possible questions to solve at home ...

- Parameterize initial number of living cells
- Display development of number of living cells per generation
- Enable manual toggling of cells

Modeling a Bird Flocking Behavior

Simple Examples

Flocking behavior of birds

- Continuous space and movement
- Birds adapt their flight pattern to other birds in their vicinity
- Results in complex, seemingly coordinated flight patterns → flocks

Rules

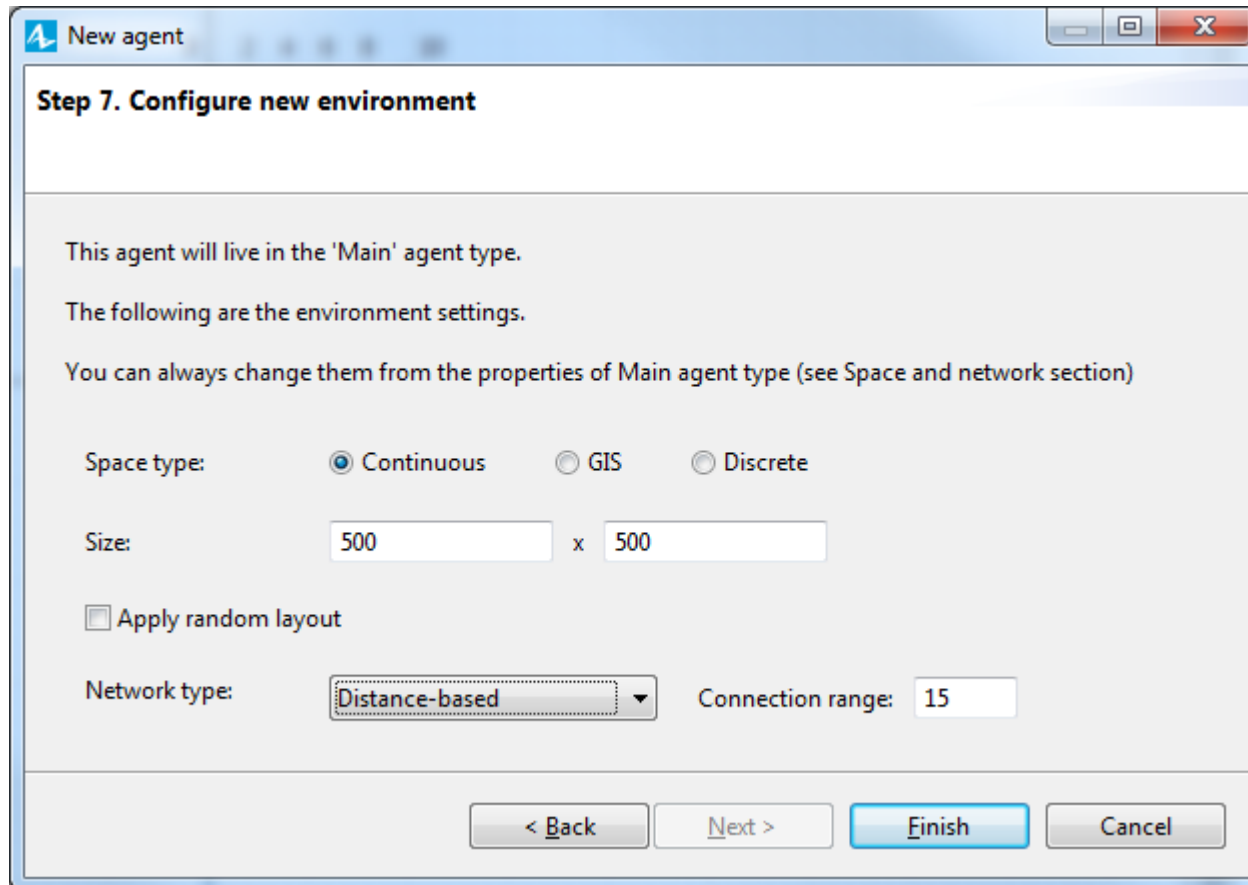
1. Separation – avoid crowding neighbors (short range repulsion)
2. Alignment – steer towards average heading of neighbors
3. Cohesion – steer towards average position of neighbors (long range attraction)



Flocking Birds

Initial properties

- Continuous environment, random layout, distance based network



New agent

Step 7. Configure new environment

This agent will live in the 'Main' agent type.

The following are the environment settings.

You can always change them from the properties of Main agent type (see Space and network section)

Space type: ☒ Continuous ☐ GIS ☐ Discrete

Size: x

☐ Apply random layout

Network type: Connection range:

< Back Next > Finish Cancel

Flocking Birds

Layout


- User defined

Network

- Distance based
- Applied in every step

▼ Space and network

Select the agents you want to place in the environment:

☒  myBirds

Space type: ☒ Continuous ☐ Discrete ☐ GIS

Space dimensions:

Width:

Height:

Z-Height:

Layout type: ☒ Apply on startup

Network type: ☐ Apply on startup

Connection range:

☒ Enable steps

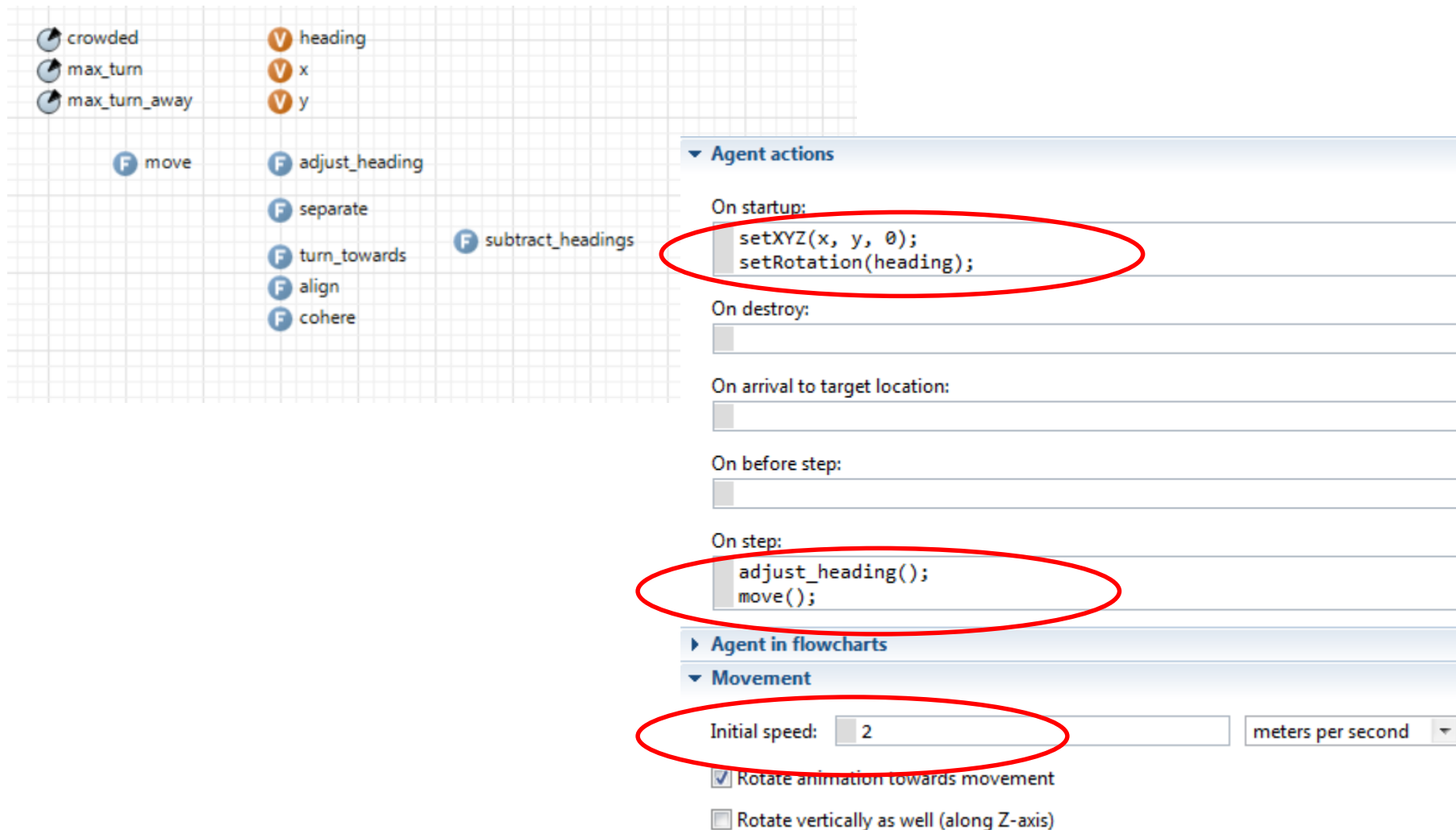
Step duration:

On before step:

On after step:

Flocking Birds

Agent Properties and Dynamics



The screenshot shows a configuration interface for a simulation. On the left, there is a grid of properties and actions. The properties include 'crowded', 'max_turn', 'max_turn_away', 'heading', 'x', and 'y'. The actions include 'move', 'adjust_heading', 'separate', 'turn_towards', 'align', 'cohere', and 'subtract_headings'. On the right, there is a panel titled 'Agent actions' with sections for 'On startup:', 'On destroy:', 'On arrival to target location:', 'On before step:', and 'On step:'. The 'On startup:' section contains the code 'setXYZ(x, y, 0);' and 'setRotation(heading);'. The 'On step:' section contains the code 'adjust_heading();' and 'move();'. Below the 'Agent actions' panel, there is a section titled 'Agent in flowcharts' with a subsection 'Movement'. The 'Movement' section has a field for 'Initial speed:' set to '2' and a unit dropdown menu set to 'meters per second'. There are also checkboxes for 'Rotate animation towards movement' (checked) and 'Rotate vertically as well (along Z-axis)' (unchecked).

crowded
max_turn
max_turn_away
move
adjust_heading
separate
turn_towards
align
cohere
subtract_headings

Agent actions

On startup:
setXYZ(x, y, 0);
setRotation(heading);

On destroy:

On arrival to target location:

On before step:

On step:
adjust_heading();
move();

Agent in flowcharts

Movement

Initial speed: 2 meters per second

☒ Rotate animation towards movement

☐ Rotate vertically as well (along Z-axis)

Flocking Birds

Variables – initialized randomly

- heading – current direction in radian ($0 \dots 2\pi$)

`heading = uniform() * 2 * Math.PI`

- x & y – current position

`x = uniform() * main.spaceWidth()`

`y = uniform() * main.spaceHeight()`

Parameters

- minimum distance between birds

`crowded = 5`

- max possible turn

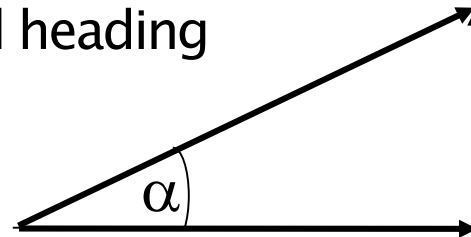
`max_turn = 0.05`

`max_turn_away = 0.02`

Flocking Birds

Move bird to new position: `void move()`

- Calculate new position
- Make environment into torus
- Modify agents position and heading



$$dy = speed * \sin \alpha$$

$$dx = speed * \cos \alpha$$

```
x += getSpeed() * cos(heading);  
y += getSpeed() * sin(heading);  
if (x > main.spaceWidth()) x -= main.spaceWidth();  
else if (x < 0) x += main.spaceWidth();  
if (y > main.spaceHeight()) y -= main.spaceHeight();  
else if (y < 0) y += main.spaceHeight();  
setXYZ(x, y, 0);  
setRotation(heading);
```

Flocking Birds

Modify heading according to rules 1–3: void adjust_heading()

- Detect nearest agent; if that one is too close, turn away
- Otherwise align and cohere with surrounding flock mates

```
if(getConnectionsNumber() == 0) return;
double closest_head = heading, double dist = 100, new_dist;
for(Agent a : getConnections()){
    new_dist = distanceTo(a);
    if(new_dist < dist){
        dist = new_dist;
        closest_head = ((Birds) a).heading;
    }
}
if(dist < crowded){ //short range repulsion
    separate(closest_head);
}else{ //long range attraction
    align();
    cohere();
}
```

Flocking Birds

Turn away from crowding flock mate

void separate(double others_heading)

- Determine difference in headings (direction & size)
- Turn further away from flock mates heading

```
double diff = subtract_headings(heading, others_heading);  
if(diff == 0) return;  
if(diff > 0) heading -= min(diff, max_turn_away);  
else heading -= max(diff, -max_turn_away);  
if(heading < 0) heading += 2*PI;  
if(heading > 2*PI) heading -= 2*PI;
```


Flocking Birds

Determine difference between two headings

double subtract_heading(double heading1, double heading2)

- Determine difference
- Normalize between $-\pi$ and π

```
double diff = heading2-heading1;  
if(diff <= -PI) diff += 2*PI;  
if(diff > PI) diff -= 2*PI;  
return diff;
```

Flocking Birds

Turn towards flock mates heading and position

void align()

- Determine average heading of surrounding flock mates
- Adjust heading in that direction

```
double avg_head = 0, avg_x = cos(heading),  
avg_y = sin(heading);  
for (Agent a : getConnections()) {  
    avg_x += cos(((Bird)a).heading);  
    avg_y += sin(((Bird)a).heading);  
}  
avg_x /= getConnectionsNumber()+1;  
avg_y /= getConnectionsNumber()+1;  
avg_head = atan2(avg_y, avg_x);  
turn_towards(avg_head);
```

Flocking Birds

Turn towards position of flock mates

void cohere()

- Determine average position of flock mates
- Adjust heading to fly there

```
double avg_pos_x = x, avg_pos_y = y;
for (Agent a : getConnections()) {
    avg_pos_x += a.getX();
    avg_pos_y += a.getY();
}
avg_pos_x /= getConnectionsNumber()+1;
avg_pos_y /= getConnectionsNumber()+1;
double pos_head = atan2(y-avg_pos_y, x-avg_pos_x);
turn_towards(pos_head);
```

Flocking Birds

Align heading with others heading

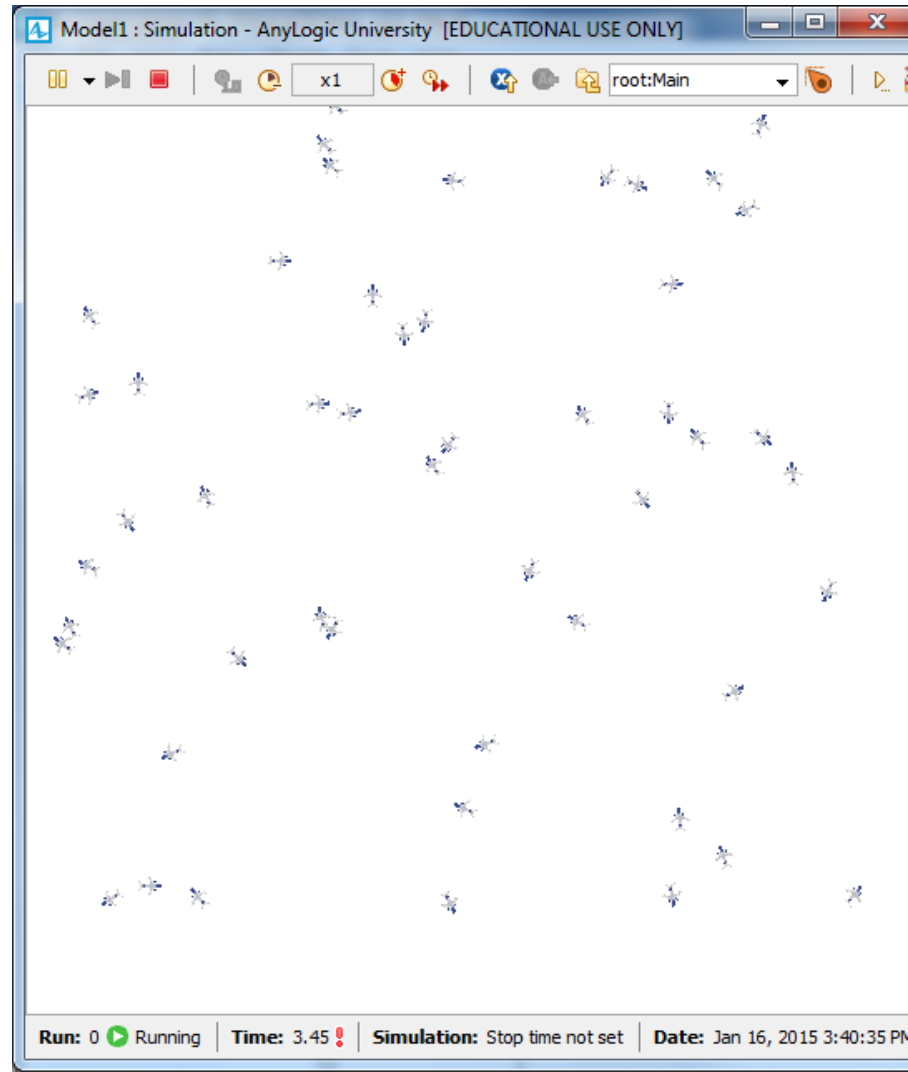
`void turn_towards(double others_heading)`

- Determine difference in headings (direction & size)
- Turn towards flock mates heading

```
double diff = subtract_headings(heading, others_heading);  
if(diff == 0) return;  
if(diff > 0) heading += min(diff,max_turn);  
else heading += max(diff,-max_turn);  
if(heading < 0) heading += 2*PI;  
if(heading >= 2*PI) heading -= 2*PI;
```

Flocking Birds

Resulting dynamic behavior



Flocking Birds

Possible questions to solve at home ...

- Divide the birds into two groups
- Create and avoid an obstacle
- Parameterize initial number of birds
- Display development of average birds heading over time
- Do replications to find patterns in the flock heading development