



MultiAgent-based Simulation

Repast Symphony

Recursive Porous Agent Simulation Toolkit

Argonne National Laboratory

University of Chicago

<http://repast.sourceforge.net/>

mirror : <https://seafile.emse.fr/d/0a454872c8/>

Amro Najjar

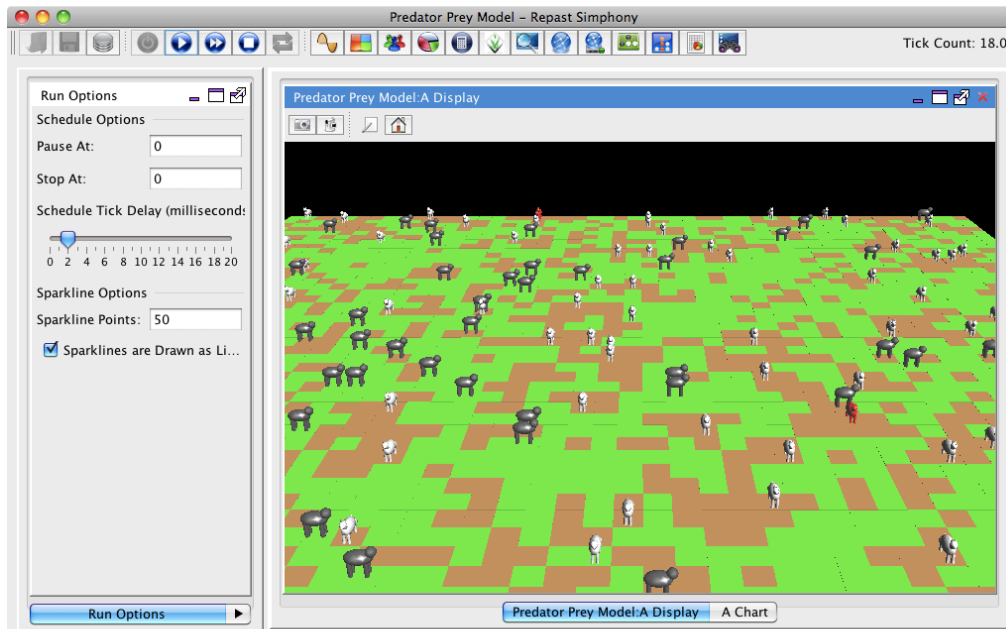
Hubert Curien Laboratory

This course is based on a course by
Mahdi Zargayouna – hamza-mahdi.zargayouna@ifsttar.fr

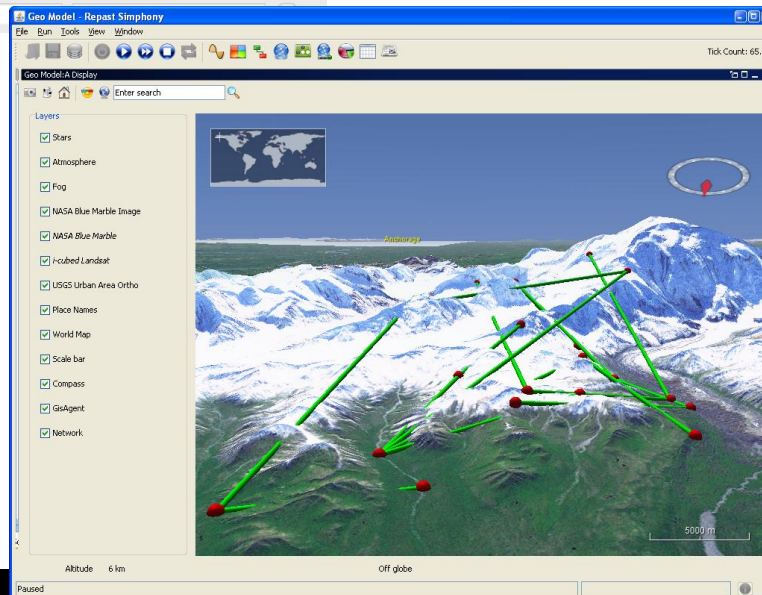
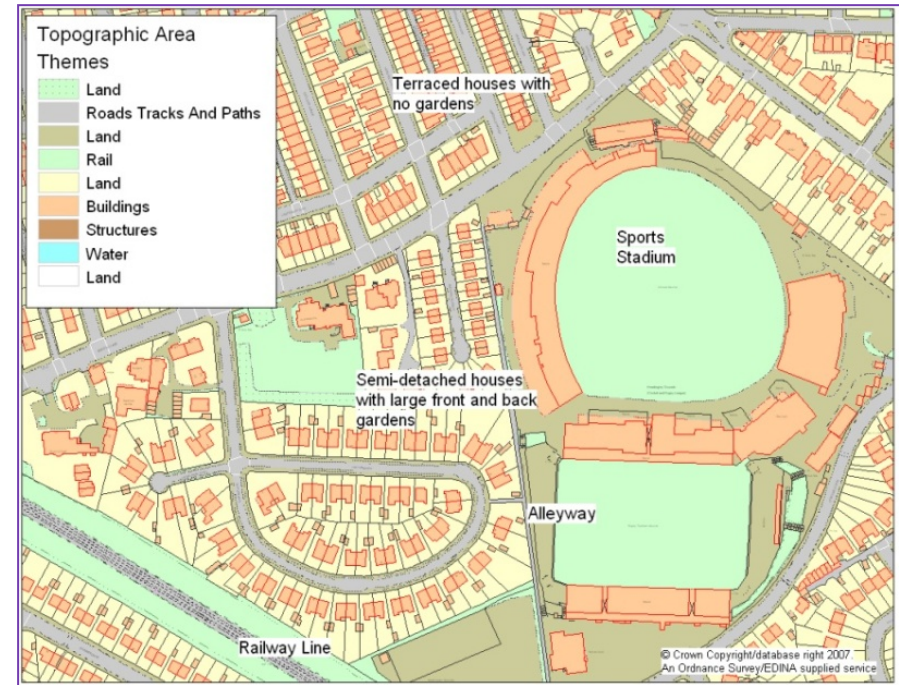
Présentation

- Repast is a multi-agent platform used to develop, test and execute multi-agent systems
- Written in Java (open-source)
- Very popular in the MAS simulation community, mainly for social and geographic simulations
- Offers an API for the design and implementation of MAS
- Offers a graphical modeling tool (specially for beginners)
 - ✓ 2D & 3D visualization
 - ✓ Grid management
 - ✓ Parameter management
 - ✓ Data observation
 - ✓ Data can be analyzed and exported from a format to another
- Availability of machine learning libraries (neural networks, regression)
- Import & Export of shapefile

Repast Illustration



Paused



Paused

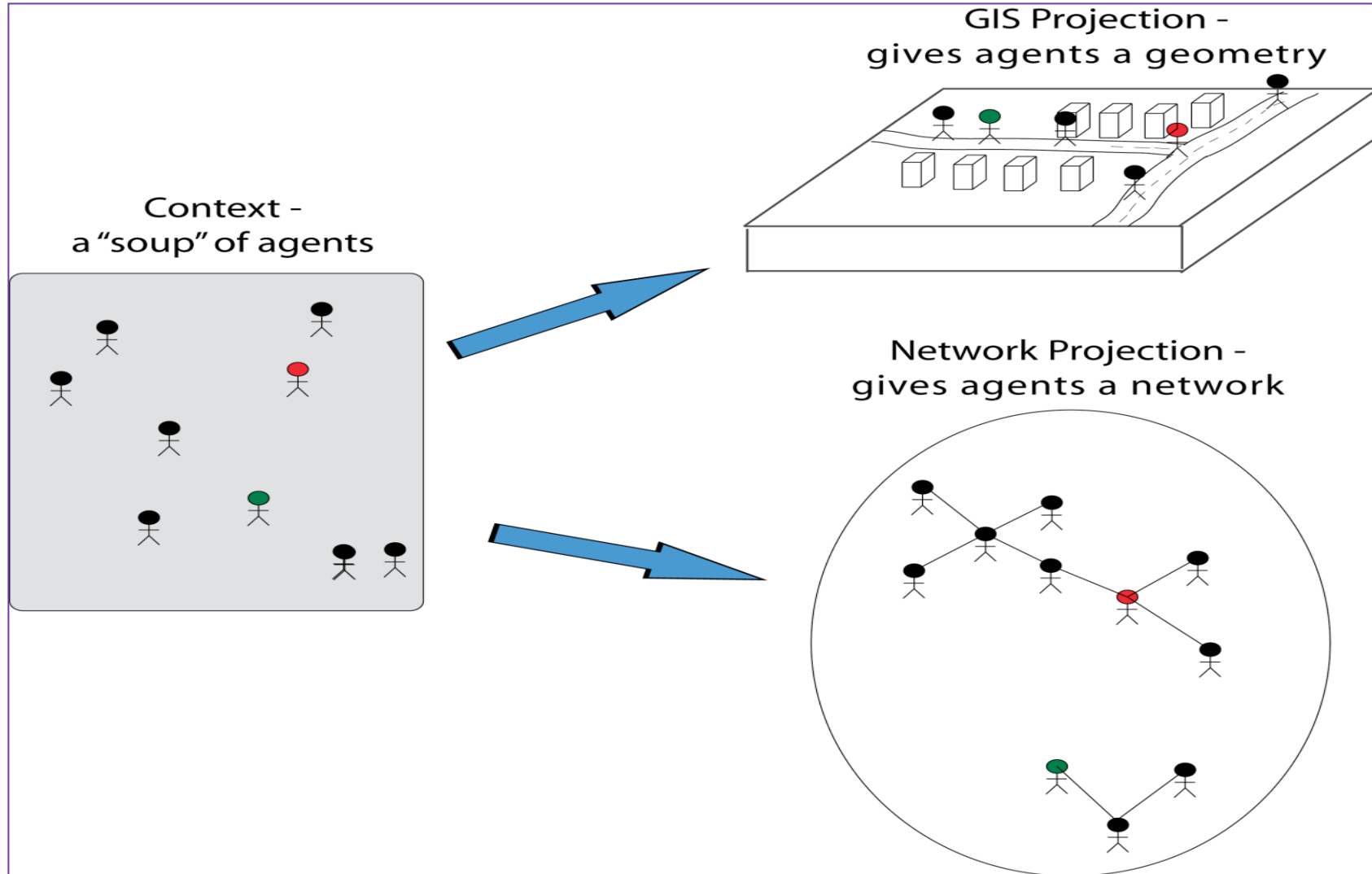
Key Notions: context

- A fundamental concept in Repast Symphony
 - ✓ Offers a data structure allowing to organize agents
 - ✓ Concretely, contexts are a collection of hierarchically nested agents each containing some of the components of the model
 - ✓ These components are java objects
 - ✓ Possibility to have other contexts
 - ✓ A component can be involved in a multiple contexts at once
 - ✓ A component involved in one context is also involved in all contexts having a parent relationship with this context
 - ✓ At least there is one *master context* that contains all the others

Key Notions: projection

- Whereas contests create a collection allowing to organize agents, projections impose a structure on these agents
- Projections allow to create a structure that define relations (can be spatial or semantic)
- A projection is attached to a particular context and is applied to all the agents belonging to this context
 - ✓ An object must belong to a context before being used in a projection
 - ✓ Multiple projections can be applied to the same context. Thus, we can have a context that contains a grid and to « geographical » spaces (e.g. two representations of the same transport network)

Contexts and Projections





Example:

Conway's Game of life

- Is a cellular automaton devised by the British mathematician John Horton Conway in 1970
- is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input
- You create an initial configuration and observing how it evolves
- Rules
 - an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead
 - Every cell interacts with its eight neighbours
 - Any live cell with fewer than two live neighbours dies (underpopulation)
 - Any live cell with two or three live neighbours lives on to the next generation
 - Any live cell with more than three live neighbours dies
 - Any dead cell with exactly three live neighbours becomes a live cell



Example: Conway's Game of life

- New Repast Symphony Project
- GameOfLife
- Perspective java
- In « GameOfLife.rs », « context.xml », create the projection« Grid »

```
<context id="GameOfLife" xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance" xsi:noNamespaceSchemaLocation="http://repast.org/scenario/  
context">  
  <projection type="grid" id="grid"></projection>  
</context>
```

- In this configuration file, we can create all the contexts, the subcontexts and projections



Create the Class Agent

```
import repast.simphony.space.grid.Grid;

public abstract class Agent {
    protected Grid<Agent> grid;
    protected boolean alive;

    public Agent(Grid<Agent> grid) {
        this.grid = grid;
        alive = true;
    }
    // modify the value of alive according to neighborhood
    public abstract void compute();

    // create complementary type of agent
    public abstract void implement();
}
```

- **We will have different agent representation based on the agent types, for this we will create two classes extending agents**
 - DeadAgent dead cells
 - AliveAgent living cells

Create A ContextBuilder

- The principal class for Repast
 - A generic class of which we specify the type of object that it contains
- ```
public class ContextCreator implements ContextBuilder<Agent> {
```

```
@Override
```

```
public Context<Agent> build(Context<Agent> context) {
```

```
...
```

```
}
```

# Create A projection

- As a general rule, *Projections* are created as follows:
  - ✓ Find their « *factory* »
  - ✓ Use the *factory* to create the *projection*
- Each *Factory* create a projection of a specific type and requires the context to which the projection is associated the name of the projection as well as some supplementary arguments
- Projection for GameOfLife

```
GridFactory gridFactory = GridFactoryFinder.createGridFactory(null);
Grid<Agent> grid = gridFactory.createGrid("grid", context,
new GridBuilderParameters<Agent>(new WrapAroundBorders(),
new SimpleGridAdder<Agent>(), false, 50, 50));
```

# Execute GameOfLife Model



- GameOfLife Model
- Build Installer for GameOfLife Model
- Batch GameOfLife Model

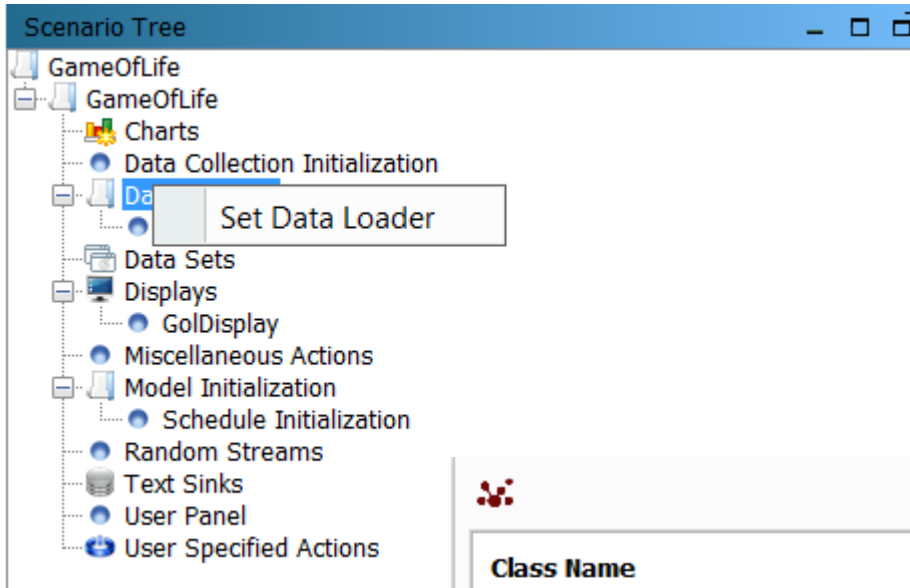
The screenshot shows the Repast Simphony application window titled "GameOfLife - Repast Simphony". The interface includes a menu bar (File, Run, Tools, Window), a toolbar with various icons, and a "Tick Count: 0.0" indicator. On the left, a "Scenario Tree" panel displays a hierarchical structure of the model components:

- GameOfLife
  - GameOfLife
    - Charts
    - Data Collection Initialization
    - Data Loaders
      - ContextCreator
    - Data Sets
    - Displays
      - GoLDisplay
    - Miscellaneous Actions
    - Model Initialization
      - Schedule Initialization
    - Random Streams
    - Text Sinks
    - User Panel
    - User Specified Actions

At the bottom of the window, a status bar displays "GameOfLife loaded".

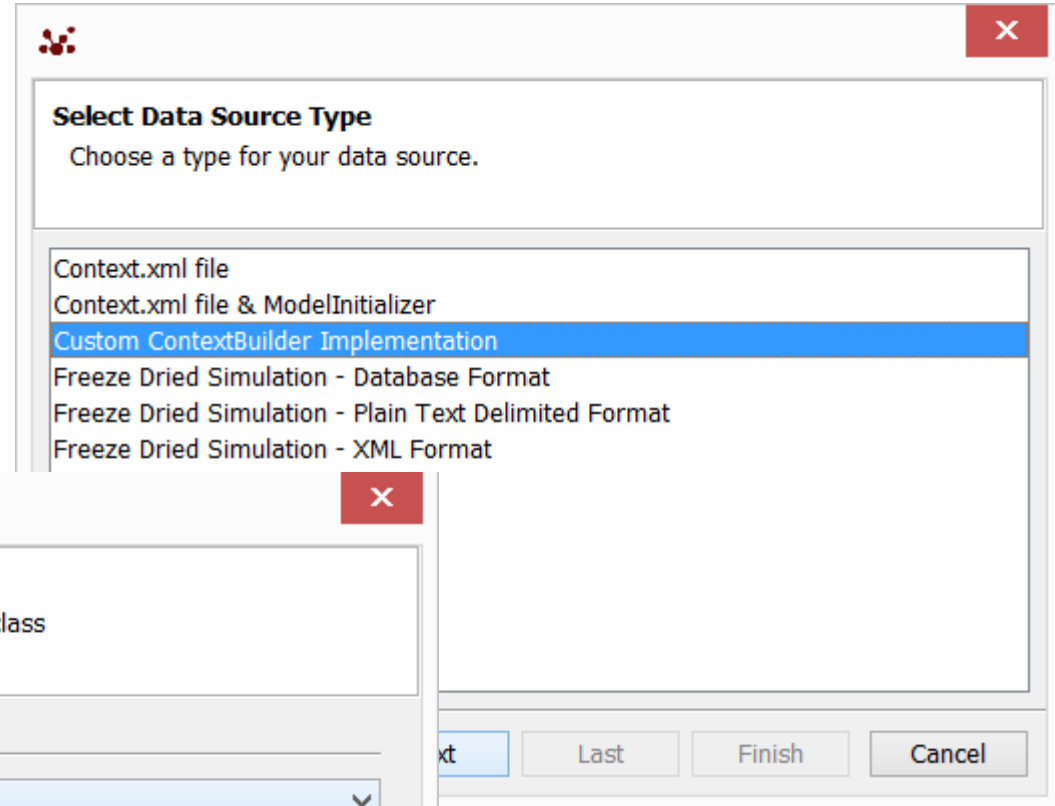
# Define the principal context

## ➤ Relancer le modèle



Scenario Tree

- GameOfLife
  - GameOfLife
    - Charts
    - Data Collection Initialization
    - Data Loader** (Set Data Loader)
    - Data Sets
    - Displays
      - GoIDisplay
    - Miscellaneous Actions
    - Model Initialization
      - Schedule Initialization
    - Random Streams
    - Text Sinks
    - User Panel
    - User Specified Actions

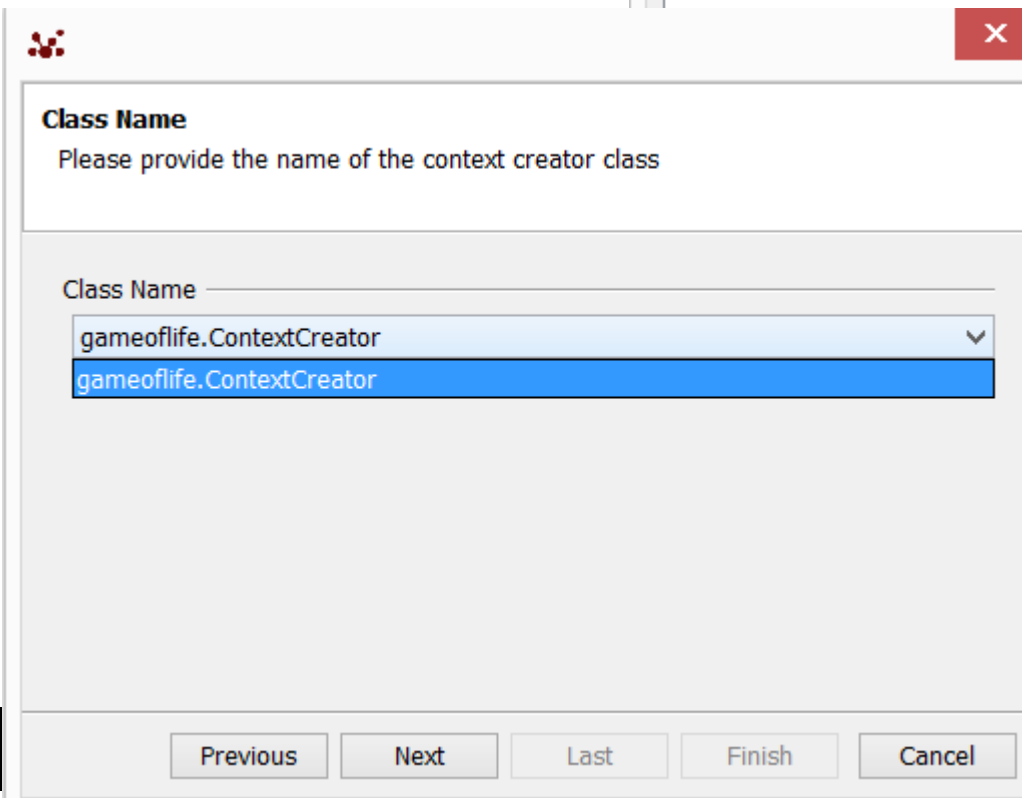


Select Data Source Type

Choose a type for your data source.

- Context.xml file
- Context.xml file & ModelInitializer
- Custom ContextBuilder Implementation**
- Freeze Dried Simulation - Database Format
- Freeze Dried Simulation - Plain Text Delimited Format
- Freeze Dried Simulation - XML Format

Buttons: Next, Last, Finish, Cancel



Class Name

Please provide the name of the context creator class

Class Name

gameoflife.ContextCreator

Buttons: Previous, Next, Last, Finish, Cancel

# Define the visualization

**Scenario Tree**

- GameOfLife
  - GameOfLife
    - Charts
    - Data Collection Initialization
    - Data Loaders
      - ContextCreator
    - Data Sets
    - Displays
      - Add Display**
    - Miscellaneous Actions
    - Model Initialization
      - Schedule Initialization
    - Random Streams
    - Text Sinks
    - User Panel
    - User Specified Actions

**Display Configuration**

**Display Details**  
Please enter the name and type of the display as well the projections the display should visualize

General

Name: A Display

Type: 2D

Projections and Value Layers

- grid

**Display Configuration**

**Agent Selection**  
Please select the agent types to display and the order in which the layers (2D) will appear

Agent Selection List:

- Agent
- ContextCreator

Selected Agent: AliveAgent

Order List:

- DeadAgent

Text Box: gameoflife.DeadAgent

Navigation: Previous, Next, Finish, Cancel

# Agents' styles

**Display Configuration**

**Agent Style**  
Please provide a style for each agent type in the model

Agents —  
AliveAgent  
DeadAgent

Agent Class:  
gameoflife.ActiveAgent

Style Class:  
repast.simphony.visualizationOGL2D.DefaultStyleOGL2D

Display B...

**2D Shape Editor**

Icon Shape and Color  
circle (selected)  
circle  
cross  
star  
square  
triangle  
X  
arrow  
Icon Size: 1.0 (Minimum), 15.0 (Maximum), 1.0 (Scaling)

Icon Preview  
Sample Label

Icon Label  
Value: (empty) Position: bottom Offset: 5.0 Precision: 3

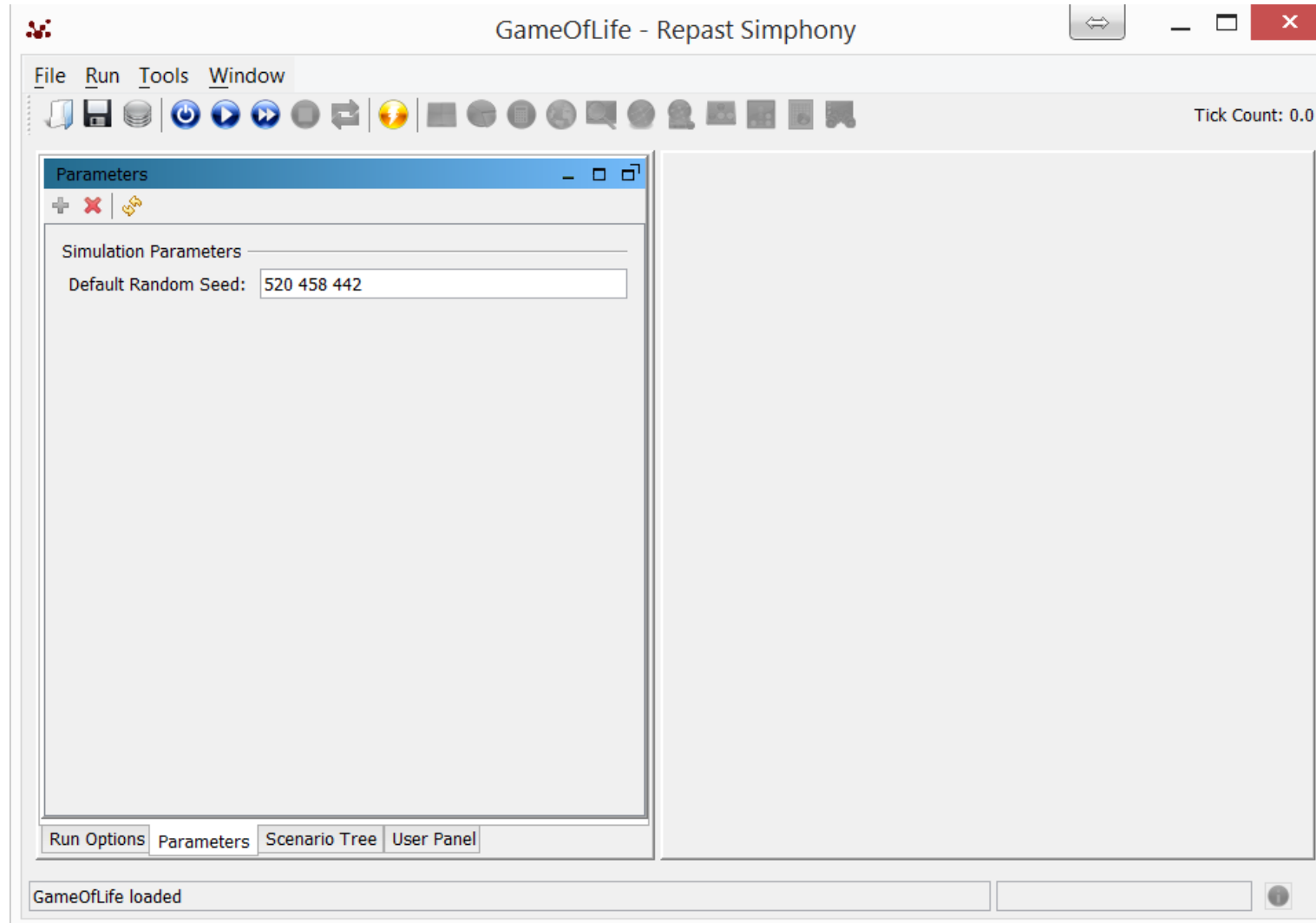
Icon Label Font  
Font: Arial Font Size: 8 Font Style: Plain Font Color: (black)

Variable Icon Color

|       | Value | Minimum | Maximum | Scaling |
|-------|-------|---------|---------|---------|
| Red   | 0.0   | 0.0     | 10.0    | 1.0     |
| Green | 0.0   | 0.0     | 10.0    | 1.0     |
| Blue  | 1.0   | 0.0     | 10.0    | 1.0     |

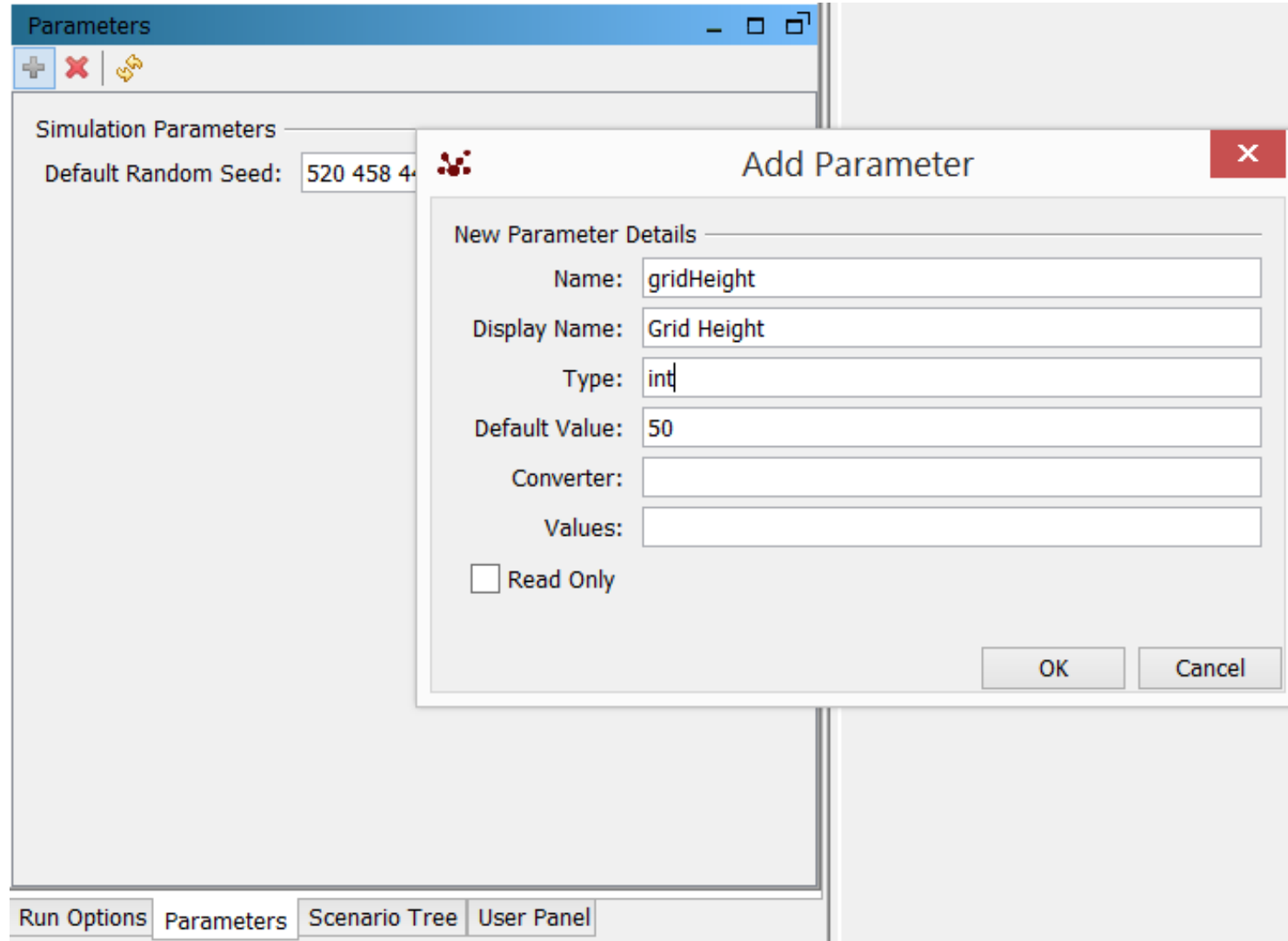
OK Cancel

# Parameterizing the Simulation





# Parameterizing the Simulation



➤ Enregistrer le modèle

# Using the parameters

```
public class ContextCreator implements ContextBuilder<Agent> {
 @Override
 public Context<Agent> build(Context<Agent> context) {
 context.setId("GameOfLife");
 int width = RunEnvironment.getInstance().getParameters().getInteger("gridWidth");

 int height = RunEnvironment.getInstance().getParameters().getInteger("gridHeight");

 GridFactory gridFactory = GridFactoryFinder.createGridFactory(null);
 Grid<Agent> grid = gridFactory.createGrid("grid", context,
 new GridBuilderParameters<Agent>(new WrapAroundBorders(),
 new SimpleGridAdder<Agent>(), false, width, height));

 for (int x = 0; x < height; x++) {
 for (int y = 0; y < width; y++) {
 boolean b = Math.random()>0.5 ? true: false;
 Agent a = b? new AliveAgent(grid): new DeadAgent(grid);
 context.add(a);
 grid.moveTo(a, x, y);
 }
 }
 return context;
 }
}
```

# Scheduling in Repast (1/3)

There exists three ways to work with Repast Scheduler

➤ Schedule an action directly:

```
// Récupérer une référence vers l'ordonnanceur
ISchedule schedule = RunEnvironment.getInstance().getCurrentSchedule();
```

```
//Spécifier que l'action devrait commencer au tick 1 et s'exécuter à chaque tick
ScheduleParameters params = ScheduleParameters.createRepeating(1, 2);
```

```
//L'agent exécute la méthode move avec les paramètres d'ordonnancement définis
//plus haut
schedule.schedule(params, myAgent, "move");
```

➤ As you can see, ScheduleParameters can be created using one of its class methods:

```
//Spécifier que l'action devrait commencer au tick 1 et s'exécuter à chaque tick
ScheduleParameters params = ScheduleParameters.createRepeating(1, 2);
//L'agent exécute la méthode move avec les paramètres d'ordonnancement définis
schedule.schedule(params, myAgent, "move", "Forward", 4);
```



# Scheduling in Repast (2/3)

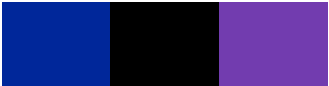
➤ Scheduling with annotations

```
@ScheduledMethod(start=1 , interval=2)
public void deliverPaper()
```

- Most of the time, objects with annotations are automatically added to the scheduler. However, if we create an object while the simulation is running, it may be not the case
- For this reason, we can use the « schedule » object to add them

```
//Ajouter les méthodes annotées de l'agent au scheduler.
scheduler.schedule(myAgent);
```

# Comportement des agents



```
public abstract class Agent {
protected Grid<Agent> grid;
protected boolean alive;
```

```
public Agent(Grid<Agent> grid2) {
this.grid = grid2;
alive = true;
}
```

```
@ScheduledMethod(start = 1, interval = 1, priority = 2)
public abstract void compute();
```

```
@ScheduledMethod(start = 1, interval = 1, priority = 1)
public abstract void implement();
```

```
}
```

# Scheduling in Repast (3/3)

## ➤ Scheduling with the « Watchers »

➤ Watchers are designed to be used to schedule in a dynamic manner

➤ A watcher allows an agent to be notified about a change of status of another agent and thus to plan an event that will take place subsequently

➤ (e.g. the agent « A » decides that if agent « B » does an action it will make a reaction)

➤ The watcher is used via an annotation:

✓ A request defining the agents to observe

✓ A request defining the condition that should be met in order to trigger and execute the action.

```
@Watch(watcheeClassName = "repast.demo.simple.SimpleHappyAgent",
 watcheeFieldNames = "happiness", query = "linked_from",
 whenToTrigger = WatcherTriggerSchedule.LATER,
 scheduleTriggerDelta = 1, scheduleTriggerPriority = 0)
```

```
public void friendChanged(SimpleHappyAgent friend) {
 if (Math.random() > .25) {
 this.setHappiness(friend.getHappiness());
 } else {
 this.setHappiness(Math.random());
 }
 System.out.println("Happiness Changed");
}
```

- watcheeClassName defines the type of agents that this object is going to watch.
- watcheeFieldName defines the attribute to be watched. When this attribute changes, this object will be notified
- query: determines (or filter) the instances of Simple happy agents to be watched
  - ✓ Here, we watch agents that are linked to us.
- whenToTrigger specifies if the action should be executed immediately (before other actions scheduled at this instant are executed) or wait for the next tick
- scheduleTriggerDelta defines how many ticks to wait before scheduling the actions
- scheduleTriggerPriority defines the priority of the action

# Agents Behavior

```
public class AliveAgent extends Agent{
public AliveAgent(Grid<Agent> grid) {
super(grid);
alive = true;}

public void compute() {
MooreQuery<Agent> query = new MooreQuery<Agent>(grid, this);
int neighbours = 0;
for (Agent o : query.query())
 if (o instanceof AliveAgent)
 neighbours++;
if (neighbours != 2 && neighbours != 3)
 alive = false;
}

public void implement() {
if (!alive) {
GridPoint gpt = grid.getLocation(this);
Context<Object> context = ContextUtils.getContext(this);
context.remove(this);
DeadAgent a = new DeadAgent(grid);
context.add(a);
grid.moveTo(a, gpt.getX(), gpt.getY());
}
}
```



# Agents Behavior

```
public class DeadAgent extends Agent{
public DeadAgent(Grid<Agent> grid) {
 super(grid);
 alive = false;
}

public void compute() {
MooreQuery<Agent> query = new MooreQuery<Agent>(grid, this);
int neighbours = 0;
for (Agent o : query.query())
 if (o instanceof AliveAgent)
 neighbours++;
if (neighbours == 3)
 alive = true;
}

public void implement() {
if (alive) {
GridPoint gpt = grid.getLocation(this);
Context<Agent> context = ContextUtils.getContext(this);
context.remove(this);
AliveAgent a = new AliveAgent(grid);
context.add(a);
grid.moveTo(a, gpt.getX(), gpt.getY());
}
}
}
```

# Executing the model

The screenshot shows a software application window titled "GameOfLife - Repast Symphony". The interface includes a menu bar with "File", "Run", "Tools", and "Window". Below the menu bar is a toolbar with various icons. The main area is divided into two panes:

- Scenario Tree:** A hierarchical tree view on the left side. The root is "GameOfLife", which contains several sub-items: "Charts", "Data Collection Initialization", "Data Loaders", "Data Sets", "Displays", "Miscellaneous Actions", "Model Initialization", "Random Streams", "Text Sinks", "User Panel", and "User Specified Actions". The "Schedule Initialization" item under "Model Initialization" is currently selected and highlighted in blue.
- GameOfLife: GoDisplay:** A large display area on the right side showing a 2D grid. The grid is composed of green and grey pixels, representing a complex, irregular pattern of cells.

At the bottom of the window, there are tabs for "Run Options", "Parameters", "Scenario Tree", and "User Panel". The "Scenario Tree" tab is currently active. In the top right corner of the window, there are window control buttons (minimize, maximize, close) and a "Tick Count: 0.0" indicator.

# Run options

The screenshot displays the 'GameOfLife - Repast Simphony' application window. The main window has a menu bar with 'File', 'Run', 'Tools', and 'Window'. Below the menu bar is a toolbar with various icons. The 'Run Options' panel is open on the left, showing the following settings:

- Schedule Options:**
  - Pause At: 0
  - Stop At: 0
  - Schedule Tick Delay: A slider set to 20, with a scale from 0 to 100.
- Sparkline Options:**
  - Sparkline Points: 50
  - Sparklines are Drawn as Line...

At the bottom of the 'Run Options' panel are four buttons: 'Run Options', 'Parameters', 'Scenario Tree', and 'User Panel'. The main simulation area, titled 'GameOfLife: Goldisplay', shows a grid of blue squares representing the state of the simulation. The status bar at the bottom left indicates 'Paused' and the top right shows 'Tick Count: 641.0'.

# Dynamic Agent Styles

```
public class AgentStyle2D extends DefaultStyleOpenGL2D {

 @Override
 public Color getColor(Object o) {
 if(RunEnvironment.getInstance().getCurrentSchedule().getTickCount()%2==0)
 return Color.RED;
 else
 return Color.BLUE;
 }
 }
}
```

# Dynamic Agent Styles

```
public class AgentStyle2D extends DefaultStyleOpenGL2D {

 @Override
 public Color getColor(Object o) {

 if (o instanceof AliveAgent)
 if(((AliveAgent) o).getAge()>=100)
 return Color.BLACK;

 if(RunEnvironment.getInstance().getCurrentSchedule().getTickCount()%2==0)
 return Color.RED;
 else
 return Color.BLUE;
 }
}
```

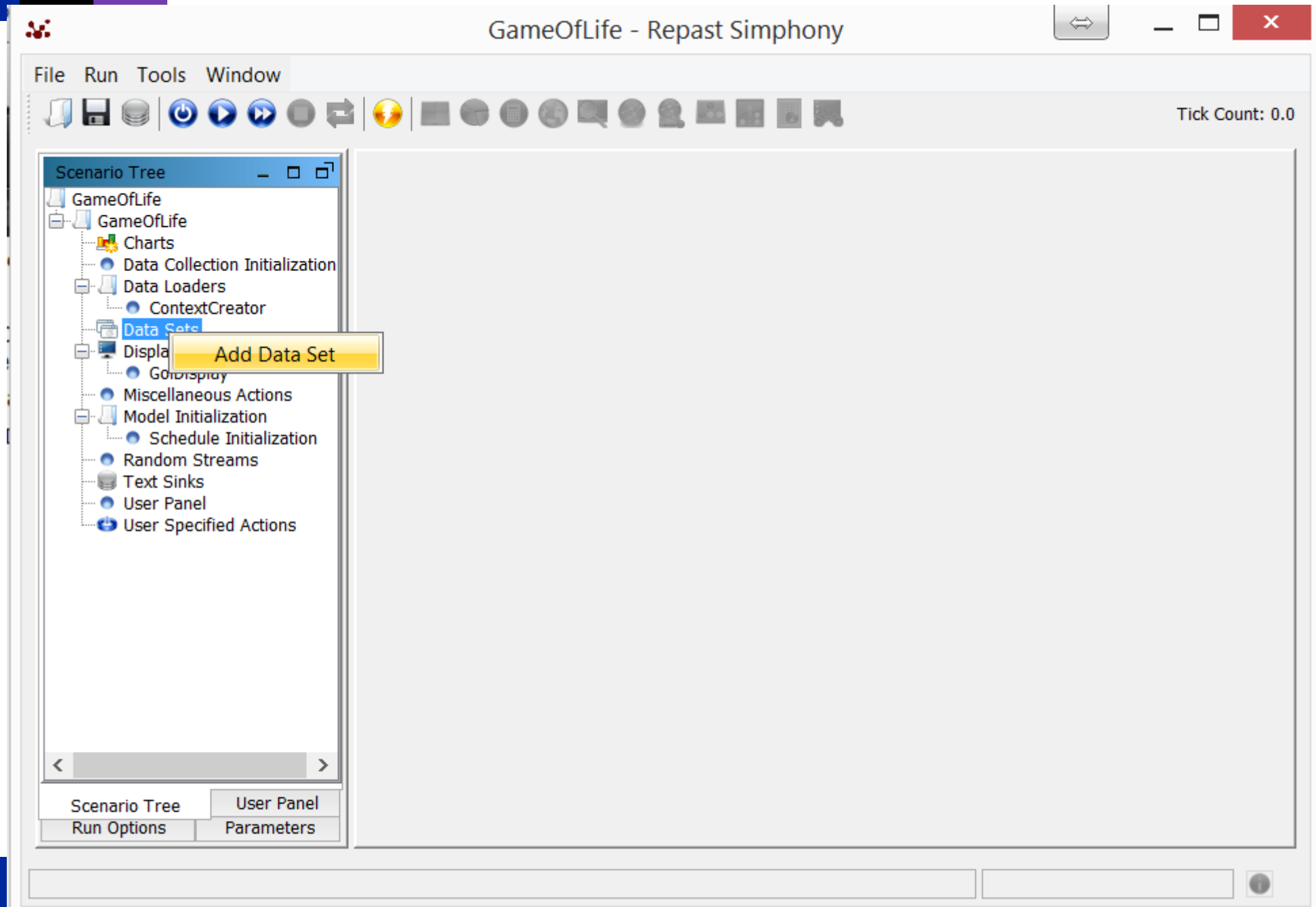
# Dynamic Agent Styles

The screenshot displays the 'Options Editor' dialog box for the 'GameOfLife - Repast Symphony' application. The dialog is titled 'Options Editor' and has a close button (X) in the top right corner. It is divided into several sections:

- General:** Contains tabs for 'General', 'Agent Selection', 'Agent Style', 'Grid Style', and 'Schedule Details'. The 'Agent Style' tab is currently selected.
- Agents:** A list of agents is shown, with 'AliveAgent' selected and highlighted in blue. 'DeadAgent' is also visible below it.
- Agent Class:** A text field containing the value 'gameoflife.AliveAgent'.
- Style Class:** A dropdown menu showing a list of style classes. The selected class is 'gameoflife.AgentStyle2D'. Other visible options include 'gameoflife.AgentStyle2D' and 'repast.simphony.visualizationOGL2D.DefaultStyleOGL2D'.

At the bottom of the dialog, there are three buttons: 'OK', 'Apply All', and 'Cancel'. The background application window shows a 'Scenario Tree' on the left with various components like 'GameOfLife', 'Charts', 'Data Collectors', etc. At the bottom of the application window, there are tabs for 'Scenario Tree', 'User Panel', 'Run Options', and 'Parameters'.

# Data Sets



# Data Observation and Export

The screenshot displays the 'GameOfLife - Repast Symphony' application window. The main interface includes a menu bar (File, Run, Tools, Window), a toolbar with various icons, and a 'Scenario Tree' on the left. The 'Scenario Tree' shows a hierarchical structure with 'Data Sets' highlighted. A 'Data Set Editor' dialog box is open in the foreground, featuring a 'General Settings' section with the instruction: 'Please enter a unique id for that data set and select the data set type.' The dialog contains two input fields: 'Data Set Id' with the value 'A Data Set' and 'Data Set Type' with a dropdown menu set to 'Aggregate'. At the bottom of the dialog are four buttons: 'Previous', 'Next', 'Finish', and 'Cancel'. The background application window also shows a 'Tick Count: 0.0' in the top right corner and a status bar at the bottom with 'Scenario Tree', 'User Panel', 'Run Options', and 'Parameters' tabs.



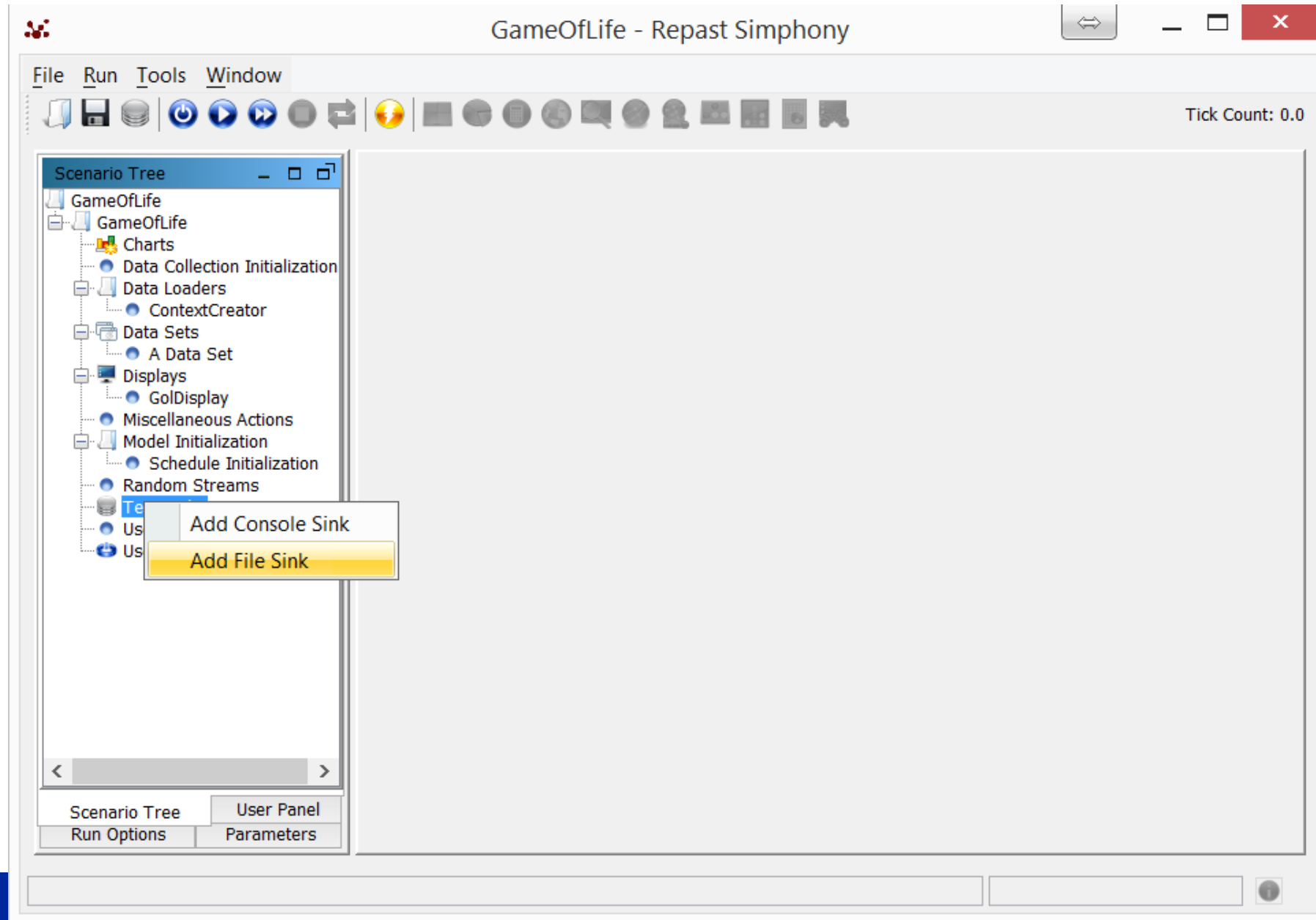
# Data Observation and Export

The screenshot displays the 'GameOfLife - Repast Simphony' application window. The main interface includes a menu bar (File, Run, Tools, Window), a toolbar with various icons, and a 'Scenario Tree' on the left. The 'Data Sets' folder in the tree is selected. A 'Data Set Editor' dialog box is open in the foreground, titled 'Data Set Editor'. It contains a 'Select Data Sources' section with the instruction 'Please choose the data sources to add to this data set.' Below this, there are three tabs: 'Standard Sources', 'Method Data Sources', and 'Custom Data Sources'. The 'Standard Sources' tab is active, showing a table with the following data:

| Source Name | Agent Type | Method | Aggregate Ope... |
|-------------|------------|--------|------------------|
| Alive count | AliveAgent | N/A    | Count            |
| Dead count  | DeadAgent  | N/A    | Count            |

Below the table are 'Add' and 'Remove' buttons. At the bottom of the dialog are 'Previous', 'Next', 'Finish', and 'Cancel' buttons. The 'Next' button is highlighted. The background application window shows a 'Tick Count: 0.0' in the top right corner and a 'Scenario Tree' panel at the bottom left with sub-panels for 'User Panel' and 'Parameters'.

# Data Observation and Export



# Data Observation and Export

The screenshot displays the 'GameOfLife - Repast Symphony' application window. The main interface includes a menu bar (File, Run, Tools, Window), a toolbar with various icons, and a 'Scenario Tree' on the left. The 'Scenario Tree' shows a hierarchical structure of components, with 'Text Sinks' highlighted. A 'File Sink Editor' dialog box is open in the foreground, titled 'File Sink Editor'. It contains the following elements:

- File Data Properties:** A section with the instruction: "Please enter the file sink's name, data set, and the data sources to record as output."
- Name:** A text input field containing "A File Sink".
- Data Properties:** A section with a dropdown menu for "Data Set ID" set to "A Data Set".
- Data Sources:** Two lists of data sources. The left list contains "Dead count" and "Alive count". The right list contains "tick" and "Alive count". Green arrows indicate the mapping from "Dead count" to "tick" and from "Alive count" to "Alive count".
- Navigation:** Buttons for "Previous", "Next", "Finish", and "Cancel" at the bottom.

At the bottom of the application window, there are tabs for "Scenario Tree", "User Panel", "Run Options", and "Parameters". The "Tick Count: 0.0" is displayed in the top right corner of the application window.

# Data Observation and Export

The screenshot displays the 'GameOfLife - Repast Simphony' application window. The main interface includes a menu bar (File, Run, Tools, Window), a toolbar with various icons, and a 'Scenario Tree' on the left. The 'Scenario Tree' lists components such as GameOfLife, Charts, Data Collection Initialization, Data Loaders, Data Sets, Displays, Model Initialization, Random Streams, Text Sinks, and User Panel. The 'Text Sinks' component is currently selected. A 'File Sink Editor' dialog box is open in the foreground, titled 'Configure File Properties'. It prompts the user to enter file and format properties. The 'File Properties' section shows the 'File Name' as 'Mahdi\Desktop\GolOutput.xls' with a 'Browse' button and a checked 'Insert Current Time into File Name' option. The 'Format Properties' section shows a 'Delimiter' of ',' and a 'Format Type' of 'Tabular'. At the bottom of the dialog are 'Previous', 'Next', 'Finish', and 'Cancel' buttons. The 'Tick Count: 0.0' is visible in the top right corner of the application window.

# Data Observation and Export

The screenshot displays the 'GameOfLife - Repast Symphony' application window. The main interface is divided into several sections:

- Scenario Tree (Left):** A hierarchical tree view showing the simulation's structure. The 'Charts' folder is expanded, listing components like 'GoTSCart', 'Data Collection Initialization', 'Data Loaders', 'Data Sets', 'Displays', 'Miscellaneous Actions', 'Model Initialization', 'Random Streams', 'Text Sinks', and 'User Panel'. Under 'Text Sinks', 'A File Sink' is selected.
- Main Display (Center):** A window titled 'GameOfLife: Goldisplay' showing a 'Dead/Alive Proportions' visualization. A 'Spreadsheet' dialog box is overlaid on this window, prompting the user to 'Select the outputter(s) to pass to Spreadsheet'. The dialog lists 'A File Sink' as the selected data source.
- Bottom Panel:** A control bar with buttons for 'Scenario Tree', 'User Panel', 'Run Options', and 'Parameters'. The status bar at the bottom indicates the simulation is 'Paused'.

The 'Spreadsheet' dialog box contains the following text:

**Select the outputter(s) to pass to Spreadsheet**  
Please select which file sinks' data you would like to send to Spreadsheet.

The list in the dialog shows:

- A File Sink

Navigation buttons at the bottom of the dialog include 'Previous', 'Next', 'Last', 'Finish', and 'Cancel'.

# Data Observation and Export

The screenshot displays the 'GameOfLife - Repast Symphony' application window. The interface includes a menu bar with 'File', 'Run', 'Tools', and 'Window'. Below the menu bar is a toolbar with various icons for file operations, execution, and visualization. The main area is divided into a 'Scenario Tree' on the left and a large empty workspace on the right. The 'Scenario Tree' shows a hierarchical structure of components, including 'GameOfLife', 'Data Sets', 'Displays', and 'Text Sinks'. A context menu is open over the 'Data Sets' folder, offering options to 'Add Histogram Chart' and 'Add Time Series Chart'. The 'Add Time Series Chart' option is highlighted. At the bottom of the window, there are tabs for 'Scenario Tree', 'User Panel', 'Run Options', and 'Parameters'. The status bar at the bottom right shows 'Tick Count: 0.0'.

GameOfLife - Repast Symphony

File Run Tools Window

Tick Count: 0.0

Scenario Tree

- GameOfLife
  - GameOfLife
    - Ch
    - Da
    - Da
    - Data Sets
      - A Data Set
    - Displays
      - GoDisplay
    - Miscellaneous Actions
    - Model Initialization
      - Schedule Initialization
    - Random Streams
    - Text Sinks
      - A File Sink
    - User Panel
    - User Specified Actions

Scenario Tree User Panel  
Run Options Parameters

IS  
anne

# Data Observation and Export

The screenshot displays the 'GameOfLife - Repast Symphony' application window. The main interface includes a menu bar (File, Run, Tools, Window), a toolbar with various icons, and a 'Scenario Tree' on the left. The 'Scenario Tree' is expanded to show the 'Charts' folder, which contains a 'GoITSCart' chart. A 'Time Series Editor' dialog box is open in the foreground, titled 'Time Series Editor'. The dialog has a 'General Settings' section with the instruction: 'Please enter a chart name and the data set to provide the chart's data'. The 'Name' field contains 'GoITSCart' and the 'Data Set' dropdown menu is set to 'A Data Set'. At the bottom of the dialog are four buttons: 'Previous', 'Next', 'Finish', and 'Cancel'. The 'Next' button is highlighted. The main window also shows a 'Tick Count: 0.0' in the top right corner and a status bar at the bottom with 'Scenario Tree', 'User Panel', 'Run Options', and 'Parameters' tabs.

# Data Observation and Export

The screenshot displays the 'GameOfLife - Repast Symphony' application window. The main interface includes a menu bar (File, Run, Tools, Window), a toolbar with various icons, and a 'Scenario Tree' on the left. The 'Scenario Tree' shows a hierarchical structure of components, with 'Charts' selected. A 'Time Series Editor' dialog box is open in the foreground, titled 'Configure Chart Data Properties'. The dialog contains a table with the following data:

| ID                                              | Label       | Color |
|-------------------------------------------------|-------------|-------|
| <input checked="" type="checkbox"/> Alive count | Alive count | Blue  |
| <input checked="" type="checkbox"/> Dead count  | Dead count  | Red   |

At the bottom of the dialog, there are four buttons: 'Previous', 'Next', 'Finish', and 'Cancel'. The 'Next' button is highlighted. The background application window shows a 'Tick Count: 0.0' in the top right corner and a status bar at the bottom with 'Scenario Tree', 'User Panel', 'Run Options', and 'Parameters' tabs.



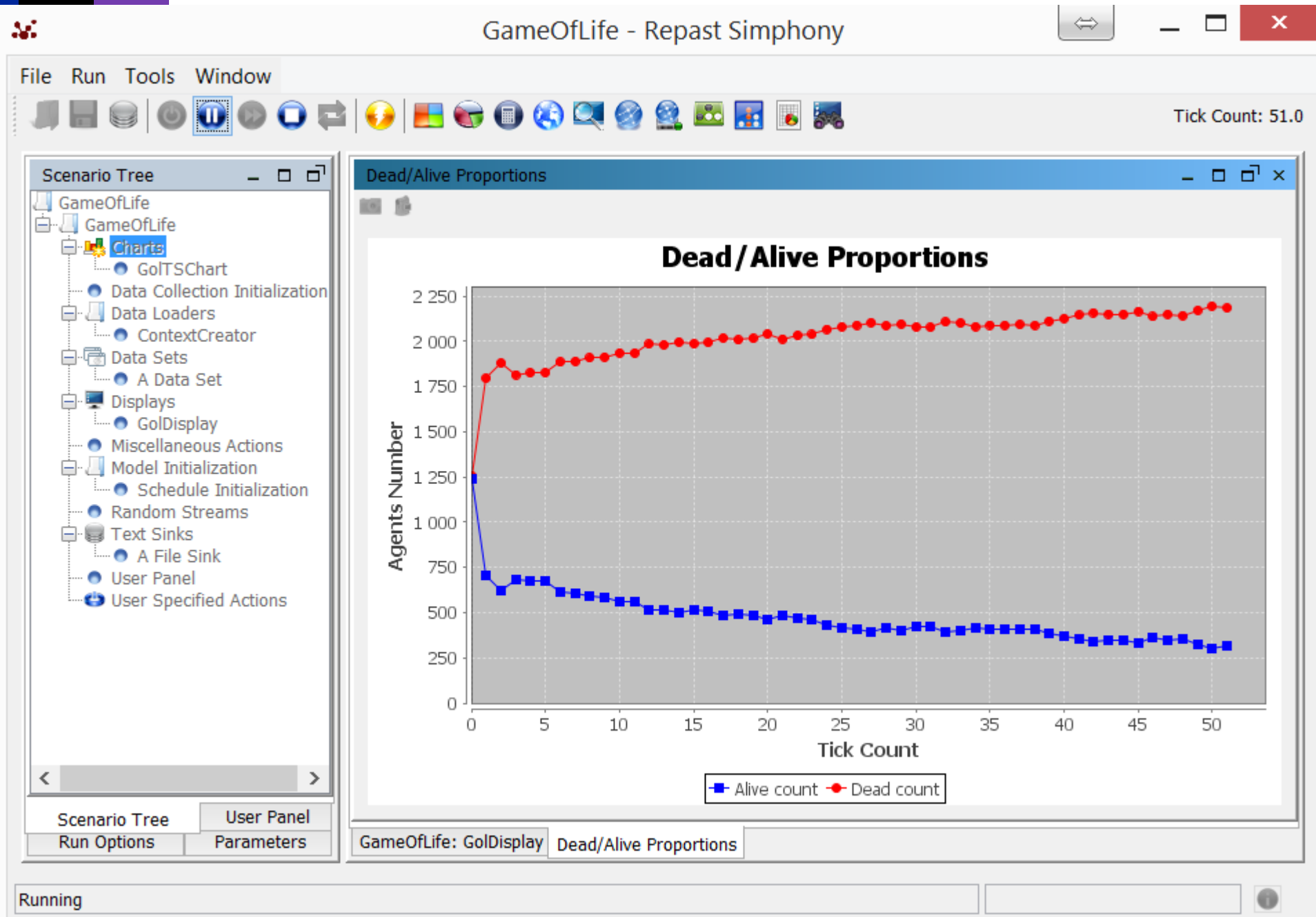
# Data Observation and Export

The screenshot displays the 'GameOfLife - Repast Symphony' application window. The main interface includes a menu bar (File, Run, Tools, Window), a toolbar with various icons, and a 'Scenario Tree' on the left. The 'Scenario Tree' shows a hierarchical structure of components, with 'Charts' selected. A 'Time Series Editor' dialog box is open, titled 'Configure Chart Properties'. The dialog contains the following fields and options:

- Title:** Dead/Alive Proportions
- X-Axis:** Tick Count
- Y-Axis:** Agents Number
- Plot Properties:**
  - Background Color: [Color Picker]
  - Show Grid Lines:
  - Grid Line Color: [Color Picker]
  - X-Axis Range: -1 [Spin Box]
  - Show Legend:

At the bottom of the dialog are four buttons: Previous, Next, Finish, and Cancel. The 'Finish' button is highlighted in blue. The main window also shows a 'Tick Count: 0.0' in the top right corner and a 'User Panel' at the bottom.

# Data Observation and Export



# Exercice

- **Create a new project where vehicules move in a continuous space and change their color when they have more than 3 vehicles around them**

- **Continuous Space:**

```
ContinuousSpaceFactory spaceFactory =
ContinuousSpaceFactoryFinder.createContinuousSpaceFactory(null);
ContinuousSpace<Voiture> space = spaceFactory.createContinuousSpace("space", context,
 new RandomCartesianAdder<Voiture>(), new
 repast.simphony.space.continuous.WrapAroundBorders(), 50, 50);
```

- **Getting the location**

- `NdPoint myPoint = space.getLocation(this);`

- **Move Action**

- Space:
  - `NdPoint otherPoint = new NdPoint(50*Math.random(), 50 *Math.random() );`
  - `double angle = SpatialMath.calcAngleFor2DMovement(space, myPoint, otherPoint);`
  - `space.moveByVector(this, 1, angle, 0);`
- Grid
  - `myPoint = space.getLocation(this);`
  - `grid.moveTo(this, (int) myPoint.getX(), (int) myPoint.getY());`