**Nanoverse Hello World**
*Anne Maslan*

This step-by-step tutorial will walk you through creating your first simulation in Nanoverse. After getting your first simulation running, some exercises are provided to experiment with changes to your simulation.

First, make sure you are set-up:
http://nanover.se/documentation.html

A. Create an XML file that specifies all necessary parameters for your simulation:
This simulation will have a few sections:

- <general> declares general simulation properties (e.g. project name, output path, etc.)
- <geometry> defines the shape and lattice for your simulation
- <layers> creates your arena using the specified geometry
- <cell-processes> top-down commands for agent processes (e.g. grow, divide, etc.)
- <writers> writes the output for your simulation

Below are instructions for each of the outlined sections. Instructions and background information are followed by the code block for each segment.

Remember that Nanoverse is in an alpha state – there are a number of inconsistencies and quirks in the syntax. The system is fully operational as a simulation tool, but please be patient as we smooth things out!

0. Create the XML file
   - Create the file in the same folder that contains the jeslime.jar file.
   - Name the XML file minimal.xml.

   *Background details on XML files can be found here [link to Getting your bearings].

1. General properties
   - Start by creating the <simulation> outer container. This will contain all of the parameter specifications.
   - Specify the <version> of Nanoverse. Current version is **0.6.3**. Prior to compiling your simulation, Nanoverse checks the version of the model to make sure that the code is written for the current version of the system.
   - Declare the general properties in <general>
     - The <random-seed> wildcard ***** is used to create a random initial value for a random number generator. This random number generator can be used for stochastic decisions in your simulation. You can also specify a particular number between 0 and 281,474,976,710,656. If you use the same number twice with the same simulation and on the same version of

Nanoverse, you will get exactly the same result. This can be useful if you want to reproduce an outcome.

- The <instances> correspond to the number of times your simulation will be run. As long as you used a wildcard seed, each instance will be different. We will run **1** instance.
- The <path> is specific to your computer. You must specify where you would like all of the results to be sent (e.g. **User/your_name/nanoverse_results/). Nanoverse will create this directory if it has adequate permissions. If it cannot, an error will be thrown.**
- Give your <project> a name. Call this project **demo**.
- Specify whether you would like a <date-stamp>. This creates a folder in which your simulation results for a given day will be stored. We would like a date stamp, so type **true**.
- Set the <max-step> parameter, which specifies the maximum time step. Your simulation will halt when the max time step is reached (prevents infinite loops). We will set a max time step of **1000**.

---

```
<simulation>
    <version>0.6.3</version>

    <!-- Declare general simulation properties -->
    <general>
        <random-seed>*</random-seed>
        <instances>1</instances>
        <path>/Users/your_name/nanoverse_results/</path>
        <project>demo</project>
        <date-stamp>true</date-stamp>
        <max-step>1000</max-step>
    </general>
</simulation>
```

---

2. Geometry and layers
   - Specify the shape and lattice connectivity of your arena in <geometry>.
     - For this simulation, use <shape> parameters: <class> **Hexagon** and <radius> **30**.
     - For the <lattice>, use <class> **Triangular**.
     - This means that the resulting arena for your simulation will be hexagon-shaped and will have a radius of 30 (defined in lattice units as explained in Distance Calculations [link]).
   - In <layers> → <cell-layer> → <boundary>, specify the <class> **Arena**. The arena is created from the shape and lattice specifications and is specified in the layers section where boundary conditions and layers can be specified.

---

```
  <!-- Declare geometry properties -->
```

```
<simulation>
    <geometry>
        <shape>
            <class>Hexagon</class>
            <radius>30</radius>
        </shape>
        <lattice>
            <class>Triangular</class>
        </lattice>
    </geometry>

    <layers>
        <cell-layer>
            <boundary>
                <class>Arena</class>
            </boundary>
        </cell-layer>
    </layers>
</simulation>
```

3. Declare simulation events and when they should occur. All of the top-down simulation events are specified in <processes>.

- In <tick>, specify <dt> as **1.0**. This means that the simulation clock will advance by 1 each time the simulation updates its state (see System, Logical, and Simulation Time [link]).
- <scatter> sets up the cells in the simulation. Specifically, it spreads out cells at random within a defined domain. Unless specified otherwise, the cells can be scattered anywhere within the simulation arena.
- This simulation will have two cell types that are each specified in separate <scatter> sections.
    - The number of a given cell type at the start of the simulation is specified by <max-targets>. For this simulation, we will start with **5** of the first cell type and **3** of the second cell type.
    - The <cell-descriptor> specifies the properties of the cell.
        - Both cells are of <class> **BehaviorCell**.
        - The <state> argument is a bit arcane (and will be replaced in a future version). For now, it's enough to know that you use it to set the color of the cells—we set them as **3** (green) and **2** (red) respectively.
    - The <threshold> is the cell's health needed to be able to divide. The first cell type will have a threshold of **1.0**. (This will be an optional feature in future versions.)
    - The first cell will have an <initial-health> of **200.0**. This parameter reflects the cell's potential for division. If the health is above the threshold, the cell can divide. A large threshold is used so that for this simulation, the cells are healthy enough for many divisions.
    - The second cell type does not have specifications for the <threshold> or <initial-health>, so the default values of threshold=2.0 and initial-health=1.0 are used.
    - Both cells have a user-defined behavior called reproduce. This will be triggered to tell the cell to reproduce. The behavior uses the action <expand-to> and the <target> of **all-neighbors**. When triggered to reproduce, it will reproduce once (<max> is **1**).
    - A <period> of **0** means that this is a one-time event. Periods greater than 0 reflect periodic events. (See System, Logical, and Wall-Clock Time [link]).
    - <start> specifies the time to start the given cell process.
- The <trigger> calls the user-defined reproduce behavior in each cell. This will result in cell division.
- <record> records the state of the system (as -- visualizations, metrics, etc.). The <period> specifies how often we wish to capture this information. For this simulation, set the <period> to **1**. This means that we record the system state once per iteration.
- <check-threshold-occupancy> throws a "halt event" when the system's total occupancy exceeds a specified threshold. Halt events cause the simulation to stop running, and provides information about the cause of the halt to the

visualizations and metrics. This can help in recording the outcome of simulations. Specify this <threshold> as **1.0**, meaning that the arena must be fully occupied in order to halt the simulation.

---

```
    <!-- Declare simulation events, and when they should occur -->
<simulation>
    <processes>

        <tick>
            <dt>1.0</dt>
        </tick>

        <scatter>
            <max-targets>5</max-targets>

            <cell-descriptor>
                <class>BehaviorCell</class>
                <state>3</state>
                <threshold>1.0</threshold>
                <initial-health>200.0</initial-health>

                <behaviors>
                    <reproduce>
                        <expand-to>
                            <target>
                                <class>all-neighbors</class>
                                <max>1</max>
                            </target>
                        </expand-to>
                    </reproduce>
                </behaviors>

            </cell-descriptor>

            <period>0</period>
            <start>0</start>
        </scatter>

        <scatter>
            <max-targets>3</max-targets>

            <cell-descriptor>
                <class>BehaviorCell</class>
                <state>2</state>
                <initial-health>200.0</initial-health>

                <behaviors>
                    <reproduce>
                        <expand-to>
```

```
                        <target>
                            <class>all-neighbors</class>
                            <max>1</max>
                        </target>
                    </expand-to>
                </reproduce>
            </behaviors>

        </cell-descriptor>

        <period>0</period>
        <start>0</start>
    </scatter>

    <trigger>
        <behavior>reproduce</behavior>
        <skip-vacant-sites/>
    </trigger>

    <record>
        <period>1</period>
    </record>

    <check-threshold-occupancy>
        <threshold>1.0</threshold>
    </check-threshold-occupancy>

  </processes>
</simulation>
```

4. Specify everything that should be output from your simulation in the <writers> section and end your script.

- <coordinate-indexer> specifies the mapping between index value and coordinate in a file coordinates.txt.
- <cell-state-writer> writes the state of the files to data.txt and metadata.txt. These files contain state vectors, with vector indices corresponding to coordinates.
- <time-writer> creates clock.bin, which contains the time at each state encoded in binary.
- <visualization-serializer> outputs a .png file for each state. Specify <class> as **map** with an <outline> of **0** and <edge> of **6**. The <outline> parameter determines how wide any outlines should be, and the <edge> parameter determines how long each edge should be (sets the visual scale). The <prefix> **cellState** will be the start of each .png filename.
- <census-writer> creates a file census.txt that writes out the number of each “state” as a function of time.
- <running-time-writer> writes out the runtime for each “state” to the file runtime.txt.

- <random-seed-writer> writes out the random seed used to random.txt (allows for future replication of results).
- <individual-halt-writer> writes out the halt condition to halt.txt.

```
</simulation>
    <writers>
        <progress-reporter/>
        <coordinate-indexer/>
        <cell-state-writer/>
        <time-writer />
        <visualization-serializer >
            <visualization>
                <class>map</class>
                <outline>0</outline>
                <edge>6</edge>
            </visualization>
            <prefix>cellState</prefix>
        </visualization-serializer>
        <census-writer/>
        <running-time-writer/>
        <random-seed-writer/>
        <individual-halt-writer/>
    </writers>

</simulation>
```

At this point, your final script should match minimal.xml [link].

B. Run the script:
In terminal, navigate to the folder that contains both the jeslime.jar and minimal.xml files. Type:

java –jar jeslime.jar minimal.xml

C. Exercises modifying minimal.xml:

(1) There is one thing about the file that almost certainly will not work on your computer. What is it?

(2) See if you can figure out how the <geometry> section works. What would happen if you tried to change to a different lattice connectivity? Does this make sense?

(3) Can you figure out how to change it to a rectangular arena? Look at the visualizations. Is the connectivity what you would expect for a rectangle?

(4) Try to change to a rectangular lattice. What happens to the visualizations?

(5) Try to convert the model to a 1D process. You will need to use slightly different visualization, called a "kymograph." See if you can modify the code to replace the existing "map" visualization with a kymograph. What is the relationship between these two visualizations? (Note that the kymograph may ultimately get folded into the 1D map.)

(6) Does this model have top-down and/or bottom-up processes?

(7) What do the parameters <threshold> and <initial-health> mean in the context of the cells?

(8) Right now, you get a fixed number of agents. There are two other ways to get a fixed number of agents. Can you figure out the other ones? What would you change to get a random number each of green and red agents?

(9) There is an optional parameter, associated with Process objects, called <active-sites>. See if you can figure out how to use it to restrict the placement of green cells to a specific region of the arena.

(10) Putting together the things that you read above, see if you can figure out how to get between three and five cells to swap places with their neighbors every cycle.

D. Exercise Answers:

(1) There is one thing about the file that almost certainly will not work on your computer. What is it?
    a. You must specify the path for the results output specific to your computer (e.g. /Users/your_name/…)

(2) See if you can figure out how the <geometry> section works. What would happen if you tried to change to a different lattice connectivity? Does this make sense?

Valid shape-lattice combinations:
Hexagon-Triangular
Rectangle-Rectangular
Rectangle-Triangular
Line-Linear
Cuboid-Cubic → no visualization

Specifications:

a. Specifications for different shape:

```
<shape>
        <class>Hexagon</class>
         <radius>r</radius>

        <class>Cuboid</class>
        <height>y</height>
        <width>x</width>
        <depth>x</depth>

        <class>Line</class>
        <length>l</length>

        <class>Rectangle</class>
        <width>x</width>
        <height>y</height>

</shape>
```

b. Specifications for different lattice connectivity:

```
<lattice>
        <class>Triangular</class>
        <class>Cubic</class>
        <class>Linear</class>
        <class>Rectangular</class>
</lattice>
```

(3) Can you figure out how to change it to a rectangular arena? Look at the visualizations. Is the connectivity what you would expect for a rectangle?

a. The Arena is specified by the shape and lattice from the geometry section. To create a rectangular arena, specify the shape as **Rectangle** as outlined in (2).
b. The lattice is still **Triangular**, which you'll notice in your simulation is not the connectivity you would expect for a rectangular arena. Try (4) to specify a connectivity that better corresponds with the arena shape.

(4) Try to change to a rectangular lattice. What happens to the visualizations?
a. Specify shape as **Rectangle** and lattice as **Rectangular** as detailed in (2). The connectivity is now better fitting for a rectangular arena.

(5) Try to convert the model to a 1D process. You will need to use slightly different visualization, called a "kymograph." See if you can modify the code to replace the existing "map" visualization with a kymograph. What is the relationship between these two visualizations? (Note that the kymograph may ultimately get folded into the 1D map.)
a. You must change shape to **Line** and lattice to **linear** because kymographs are 1D.
b. Change the line <class>map</class> to **<class>kymograph</class>**
c. A kymograph returns an image only on the last frame with the horizontal axis representing time, while a map shows a visualization of the system's state at each time step in a separate image.

(6) Does this model have top-down and/or bottom-up processes?
a. This model has top-down and bottom-up processes. In top-down processes, the various commands correspond to the state of the system and tell cells to perform certain functions. A bottom-up process is in the point of view of each cell, which knows its neighborhood within a given radius and decomposes the top-down commands into steps that the cell itself can perform. This bottom-up nature is reflected in the reproduce behavior details.

(7) What do the parameters <threshold> and <initial-health> mean in the context of the cells? What does <uniform-biomass-growth> do? What would happen when executing <divide-anywhere> if we got rid of <uniform-biomass-growth>?
a. The <threshold> is the health needed for division. A cell must have a value for health greater than this minimum value. The default threshold value is 2.0.
b. The <initial-health> represents a cell's potential for division. The health can be adjusted when a cell receives a benefit or penalty. When a cell divides, each daughter has half the health of the parent. The default initial-health value is 1.0.

(8) Right now, you get a fixed number of agents. There are two other ways to get a fixed number of agents. Can you figure out the other ones? What would you change to get a random number each of green and red agents?

a. Other ways to get a specific number of agents:

```
<tokens>
      <constant>3</constant>
 </tokens>
```

-OR-

```
<tokens>
     <uniform>
          <min>3</min>
          <max>3</max>
     </uniform>
 </tokens>
```

b. To get a random number each of green and red agents, replace the <tokens> code with the code below. The first option will give you a random number of agents from 3 to 7, and the second will give you a random number of agents from 1 to 4. The random is drawn from a <uniform>distribution between the two bounds.

```
<tokens>
     <uniform>
          <min>3</min>
          <max>7</max>
     </uniform>
</tokens>

<tokens>
     <uniform>
          <min>1</min>
          <max>4</max>
     </uniform>
</tokens>
```

(9) There is an optional parameter, associated with Process objects, called <active-sites>. See if you can figure out how to use it to restrict the placement of green cells to a specific region of the arena.

a. Adding the below section within the <scatter> container for the green cell will restrict the placement of green cells to a disc of radius 5 centered at the center of the arena.

```
<active-sites>
   <disc>
      <radius>5</radius>
      <offset x="0" y="0" z="0"/>
   </disc>
  </active-sites>
```

(10) Putting together the things that you read above, see if you can figure out how to get between three and five cells to swap places with their neighbors every cycle

a. In <processes>, add:

```
<general-neighbor-swap>
  <count>
    <uniform>
      <min>3</min>
      <max>5</max>
    </uniform>
  </count>
</general-neighbor-swap>
```