BSim Modelling Exercises

Simulations in BSim are written using the Java programming language. We don't have time to teach you the whole of this language, so instead, we will walk you through some examples to give you an idea of the steps required to create a simulation. By the end of these exercises you will hopefully be familiar with creating agents (virtual bacteria), using chemical fields and customising these elements for your specific needs.

The system we will be modelling consists of two types of bacteria. The first move around the environment randomly, searching for a second type which remain stationary. To communicate, the stationary bacteria produce a chemical signal that diffuses in the environment. Once the randomly moving bacteria sense this chemical above a specific threshold they emit a signal by changing colour.

Setting Up BSim

Before getting started make sure you have a copy of the '`BSimExercises`' folder which contains everything you will need to complete the exercises. It should contain the following:

- `BSimExercises.java` – This file contains the simulation that you will build. To edit it, right hand click on the file and select the 'Edit' option.
- `BSimQuestionSheet.pdf` – This document available if you wish to copy and paste the code snippets rather than write them from scratch.
- `compile_and_run.bat` – This file will compile and run your simulation. You simply double click on it to see what your current simulation looks like.
- `Java` – This folder contains the Java programming language and is required to compile and run your code. You shouldn't need to enter this folder.
- `Libraries` – This folder holds BSim and other libraries that you will use during the exercises. You shouldn't need to enter this folder.

Your first step will be to take a look at the template Java file called `BSimExercises.java`. This contains the code required to setup a BSim simulation and comments are included (lines beginning with `//`) to explain the purpose of each part. Comments are ignored by the computer when compiling the code and are used to aid with understanding.

Before making any changes, lets check that this compiles and runs successfully. Double click on the `compile_and_run.bat` file. A black window should be displayed, this is compiling the code, and then after while another window will open containing an empty cube and a timer in the left corner. This process can take a while the first time it is run, so be patient.

1 Randomly Moving Bacteria

Now that we have the BSim environment setup and working lets create some randomly moving bacteria. BSim already comes with a built in bacterium agent which we can use for this task. To add a group of these to the simulation make the following changes to the template:

(a) First add the following lines after the bounds of our simulation are set, i.e. after the line `sim.setBound(100,100,100);`

```
// -------------------------------------------------------------------
// Create a list of bacteria and add some new bacteria to it
// -------------------------------------------------------------------
final Vector<BSimBacterium> randomBacteria = new Vector<BSimBacterium>();

randomBacteria.add(new BSimBacterium(sim, new Vector3d(10,10,90)));
randomBacteria.add(new BSimBacterium(sim, new Vector3d(70,70,70)));
randomBacteria.add(new BSimBacterium(sim, new Vector3d(90,90,10)));
```

This code defines a new list of bacteria called `randomBacteria` which we add three standard `BSimBacterium` to. Try compiling and running your new program, does anything happen? What might the 3 numbers at the end of the lines creating the bacteria represent?

(b) BSim needs to be told how to draw the objects you create. To do this for our group of bacteria we alter the drawing section of our simulation to be:

```
sim.setDrawer(new BSimP3DDrawer(sim, 800,600) {
   @Override
   public void scene(PGraphics3D p3d) {
      // -------------------------------------------------------------
      // Loop through all the randomly moving bacteria and draw
      // -------------------------------------------------------------
      for(BSimBacterium b : randomBacteria) {
         draw(b, Color.GREEN);
      }
   }
});
```

This code loops through each of the bacterium we have added and draws them as a green sphere. Try changing the colour to `YELLOW`, `ORANGE`, `PINK` or `BLUE`. When you run your simulation what else is missing?

(c) In addition to telling BSim how to draw objects you also need to describe how they should change over time. This information is contained within what is known as the ticker. This is called at each time step and lets you update the simulation as required. For us this means telling the bacteria to update their position. To do this, update the ticker as follows:

```
sim.setTicker(new BSimTicker() {
   @Override
   public void tick() {
      // -------------------------------------------------------------
      // Loop through all the randomly moving bacteria and update position
      // -------------------------------------------------------------
      for(BSimBacterium b : randomBacteria) {
         b.action();
         b.updatePosition();
      }
   }
});
```

Compile and run your updated simulation. Is everything working correctly?

(d) (Optional) You should now have 3 randomly moving bacteria. See if you create some more so that there are 10 in total and change the initial start position of each.

2 Chemical Fields

Next we focus on creating a new type of bacterium that is stationary and produces a chemical signal. Starting from the current simulation we make the following changes:

(a) First we need to gain access to the chemical fields that are built into BSim. This is done by adding the following line to start of the simulation code. You should notice that there are already quite a few BSim elements defined in the code.

```
import bsim.BSimChemicalField;
```

(b) Now that we have access to the built in chemical fields we can add them to our simulation. Add the following lines before the randomly moving bacteria are created.

```
// -------------------------------------------------------------
// Create a chemical field to act as a signal
// -------------------------------------------------------------
final double decayRate = 0.9;
final double diffusivity = 890; // (microns)^2/sec
final BSimChemicalField field = new BSimChemicalField(sim, new int[]{20,20,20},
                                                       diffusivity, decayRate);
```

This code creates a new chemical field called `field` in the environment with specific parameters for diffusivity and decay rate of the chemical. For these exercises, all parameters remain fixed. However, it is worth noting that when analysing real systems it might be necessary to perform 'sensitivity analysis'. This is where parameters are varied to see the effect they have on the simulation results.

(c) Next we need to define our stationary bacteria that emits a chemical in this field. Add the following code after the chemical field has been created.

```
// ---------------------------------------------------------------------
// Define our new type of bacterium by extending the built-in type
// ---------------------------------------------------------------------
class MyChemicalCreatorBacterium extends BSimBacterium {
   final double productionRate = 8e9; // molecules/sec
   public MyChemicalCreatorBacterium(BSim sim, Vector3d position) {
      super(sim, position);
   }
   @Override
   public void action() {
      super.action();
      field.addQuantity(position, productionRate*sim.getDt());
   }
}

// ---------------------------------------------------------------------
// Create a single stationary bacterium and change its size
// ---------------------------------------------------------------------
final MyChemicalCreatorBacterium chemicalCreatorBacterium =
               new MyChemicalCreatorBacterium(sim, new  Vector3d(50,50,50));
chemicalCreatorBacterium.setRadius(4);
```

The reason this code looks a little more complex than for the previous randomly moving bacteria is that BSim does not contain a built-in agent that does exact what we want. Thankfully,

Java allows for agents such as the existing `BSimBacterium` to be extended and customised as we wish – technically known as sub-classing. The benefit of this approach is that it allows us to inherit all existing functionality of the built-in bacterium and means only a small portion of the code needs to be changed. In relation to the new code, the first part defines our new type of bacterium and the second creates an instance that the simulation can then use.

(d) As before, we also need to tell BSim how to draw the new type of bacterium and the chemical field. We therefore update our drawing code to the following:

```
sim.setDrawer(new BSimP3DDrawer(sim, 800,600) {
   @Override
   public void scene(PGraphics3D p3d) {
      // ---------------------------------------------------------------
      // Draw both types of bacteria and the chemical field
      // ---------------------------------------------------------------
      draw(field, Color.RED, (float)(255/12.0e5));
      draw(chemicalCreatorBacterium, Color.RED);
      for(BSimBacterium b : randomBacteria) {
         draw(b, Color.GREEN);
      }
   }
});
```

(e) Finally, we need to update the ticker to let the chemical field diffuse and stationary bacterium to emit it's signalling chemical. The new ticker is as follows:

```
sim.setTicker(new BSimTicker() {
   @Override
   public void tick() {
      // ---------------------------------------------------------------
      // Update the state of the bacteria and chemical field
      // ---------------------------------------------------------------
      chemicalCreatorBacterium.action();
      field.update();
      for(BSimBacterium b : randomBacteria) {
         b.action();
         b.updatePosition();
      }
   }
});
```

Compile and run the simulation. You should notice that both types of bacteria are now visible and that a chemical concentration develops slowly around the stationary bacterium.

(f) (Optional) Experiment with different parameter values for decay rate and diffusivity of the chemical field. How do these change the final concentrations in the environment?

3 Signalling

The final part of the exercise is to allow the randomly moving bacteria to sense the chemical emitted by the stationary bacteria. To do this we will have to customise the bacteria (like for the stationary type) and allow them to detect the chemical concentration at their current location.

(a) We begin by defining our new type of bacteria that will be able to detect the chemical signal. Replace the existing randomly moving bacteria code with the following:

```
// ---------------------------------------------------------------------
// Create our signalling bacteria by extending the built-in type
// ---------------------------------------------------------------------
class MySignallingBacterium extends BSimBacterium {
   final double threshold = 1e4;
   public boolean activated = false;
   public MySignallingBacterium(BSim sim, Vector3d position) {
      super(sim, position);
   }
   @Override
   public void action() {
      super.action();
      if(field.getConc(position) > threshold)
         activated = true;
      else
         activated = false;
   }
}

// ---------------------------------------------------------------------
// Create a list of bacteria and add some new bacteria to it
// ---------------------------------------------------------------------
final Vector<MySignallingBacterium> randomBacteria =
                                      new Vector<MySignallingBacterium>();

randomBacteria.add(new MySignallingBacterium(sim, new Vector3d(10,10,90)));
randomBacteria.add(new MySignallingBacterium(sim, new Vector3d(50,50,50)));
randomBacteria.add(new MySignallingBacterium(sim, new Vector3d(90,90,10)));
```

This creates 3 new `MySignallingBacterium` agents that are able to detect the current chemical concentration, and if exceeding a threshold value updates a flag called `activated`. This flag can then be used later when drawing the bacteria to the screen.

(b) The final change is to update the drawing code so that the `activated` flag results in the bacterium being drawn in a different colour.

```
sim.setDrawer(new BSimP3DDrawer(sim, 800,600) {
   @Override
   public void scene(PGraphics3D p3d) {
      // ---------------------------------------------------------------
      // Draw stationary bacteria and chemical field
      // ---------------------------------------------------------------
      draw(field, Color.RED, (float)(255/12.0e5));
      draw(chemicalCreatorBacterium, Color.RED);
      for(MySignallingBacterium b : randomBacteria) {
         if (b.activated == true)
            draw(b, Color.BLUE);
         else
            draw(b, Color.GREEN);
      }
   }
});
```

That's it! If you run the simulation you should find that when the randomly moving bacteria get too close to the stationary bacterium, their colour changes from green to blue.

(c) (Optional) Attempt to add more randomly moving bacteria to the simulation and vary the threshold to check that the system is behaving as expected. How might a model like this be

of use during the development of a synthetic biology project? Also, how might experimental results feed into the model and vice-versa?

4 Optional Extensions

BSim offers a wealth of other features that we don't have time to go into here. Below are a few interesting additions that can be added to your exiting simulation if you have time. To find out more about the features of BSim check out the iGEM wiki at:

- `http://2009.igem.org/Team:BCCS-Bristol/BSim`

(a) (Optional) Simulations in BSim take place in a 3-dimensional world which is difficult to judge when viewed from a single perspective. Sometimes it is useful to rotate the display as the simulation runs to gain a better understanding of the whole scene. This can be easily achieved by updating the drawing code to the following:

```
BSimP3DDrawer drawer = new BSimP3DDrawer(sim, 800,600) {
   public float rot = 0.0f;
   @Override
   public void scene(PGraphics3D p3d) {
      // ---------------------------------------------------------------
      // Rotate the 3D scene
      // ---------------------------------------------------------------
      p3d.scale(0.35f);
      p3d.rotateY(rot * (float)(Math.PI/2.0));
      p3d.rotateX(rot * (float)(Math.PI/2.0));
      rot += 0.015;
      p3d.translate(-50, -50, -50);
      // ---------------------------------------------------------------
      // Draw stationary bacteria and chemical field
      // ---------------------------------------------------------------
      draw(field, Color.RED, (float)(255/12.0e5));
      draw(chemicalCreatorBacterium, Color.RED);
      for(MySignallingBacterium b : randomBacteria) {
         if (b.activated == true)
            draw(b, Color.BLUE);
         else
            draw(b, Color.GREEN);
      }
   }
};
sim.setDrawer(drawer);
```

(b) (Optional) Outputting results from a simulation is important to allow of later statistical analysis. BSim can output data in the form of text based files or a simulation movie. For this example we will generate a movie of our simulation and save it to a file. First we need to use the movie exporter from the BSim toolkit and add the following code to the beginning of the simulation.

```
import bsim.export.BSimMovExporter;
```

Next, we replace the command to preview the simulation (`sim.preview();`) with code to export the simulation to file.

```
BSimMovExporter movExporter = new BSimMovExporter(sim,drawer,"simulation.mov");
movExporter.setDt(0.03);
sim.addExporter(movExporter);
sim.setSimulationTime(20);
sim.export();
```

When running the simulation now you should notice that no preview window is displayed, but instead that time values are output to the window. Once this reaches 20, the length we specified for the output, a new file called `simulation.mov` will exist containing a movie of the simulation.