Marine Biology Simulation Case Study

Chapter 1

Experimenting with the Marine Biology Simulation Program

Last summer I was hired to modify a simulation program used by some marine biologists. On the first day, I met with my boss who told me a little bit about the program I would be working on. It was designed to help the marine biologists study fish movement in a bounded environment, such as a lake or a bay. It was working fine for them, but they wanted to make some improvements and add some new functionality.

Jamie, an experienced programmer on the team, would be available the following day to give me a "guided tour" of the actual program code. In the meantime I could experiment with the program and start figuring out what it did. I was given some instructions for running it and told where to find several data files to use with it.

[Educational prerequisites for this chapter: The first two sections of the chapter, Experimenting with the Simulation and Looking at the Data Files, have no prerequisites. The last section, Sneaking a Peek at Some Code, assumes familiarity with constructing objects, invoking methods (including the `obj.method()` *calling syntax), and* `for` *loops.]*

Experimenting with the Simulation

The first thing I did was to determine how to compile and run the existing program.

Exercise Set 1:

1. Run the marine biology simulation program with two different data files, `fish.dat` and `manyFish.dat`, to see how the simulation works. You will need to find out from your teacher how to compile and run the program on your school's computers. You will also need to know where to find the `fish.dat` and `manyFish.dat` data files. These files describe the initial configuration of the fish in the environment. *[Note: In the `JavaMBS.zip` distribution file, the main method to run the Marine Biology Simulation is in the `MBSGUI` class in the `Code` folder. Select "Open environment file . . ." in the `file` menu to find and open an initial configuration file. The configuration files are in the `DataFiles` folder.]*
2. Run the marine biology simulation program twice with the same data file. Is the behavior of the fish in the environment the same both times?

Analysis Question Set 1:

1. How does the program appear to model the body of water (the "bounded environment")? What do the grid lines represent?
2. Where can fish be in the model? Can there be more than one fish in the same place at the same time?
3. Do all fish face the same direction? Does a fish's direction appear to matter?
4. Pick one fish and watch it for several steps. Does it move in every step? How far does it move? Does it always move in the same direction? Can it move in any direction? Is it more likely to move in one direction over another?

I ran the program several times with the different files and watched the behavior. I then made some notes and tried to deduce what the "rules" were in the program for fish movement. I found it difficult, though, to keep track of everything that was happening, and I wasn't sure that all my original conclusions were correct. I decided I needed to develop a strategy, to be more scientific in my tests. I started with the file called `onefish.dat`. This file, as its name implied, seemed to model a single fish in a small (7 x 5) environment. I created a table showing where the fish was at each timestep, which direction it was facing, and where it moved.

Exercise Set 2:

1. Run the marine biology simulation program with `onefish.dat` for five timesteps, keeping notes.

Timestep	Fish's Location	Fish's Direction	Did it Move?	In what Direction?	New Location	New Direction
1						
2						
3						
4						
5						

   How often does the fish move forward? Right? Left? Backward?

2. Compare your results with a classmate. Did your classmate have the same results? What conclusions can you draw about fish movement in a single run of the program and about different runs of the program? How confident are you that you have enough data?

Conclusions:

I made several conclusions based on my tests.

- The fish always moved in every timestep, but it didn't always move the same way.
- It always moved one space.
- It did not always move in the direction it was facing.
- Sometimes it moved forward; sometimes it moved to the side.
- I never saw it move backward in that test run.
- When it moved to the side it always changed direction.
- After every move, its new direction was always the same as the direction it moved.
- Every time I ran the program I saw different results.

Analysis Question Set 2:

1. Did your test results show the same thing? What did others in your class discover? Are five timesteps enough to come to any conclusions?

2. What if you run the program with `onefish.dat` for 10 timesteps? 20 timesteps?

Timestep	Fish's Location	Fish's Direction	Did it Move?	In what Direction?	New Location	New Direction
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						

Looking at the Data Files

I wanted to see if I could read the data files that were providing the initial configuration of fish in the environment. I decided to start with `onefish.dat`. The contents of the file are:

```
bounded 7 5
Fish 3 2 North
```

I guessed that the "bounded" referred to the "bounded environment" my boss had mentioned, and that the 7 and 5 in the first line specified the dimensions of environment, since I had already noticed that its size was 7 x 5. I thought the second line probably determined the initial location and direction of the fish, especially since I noticed that every time I ran the program with this file the fish started out facing the top of the screen.

Analysis Question Set 3:

1. If the numbers "3 2" in `onefish.dat` refer to the initial location of the fish in the environment, how are locations in an environment numbered?

Exercise Set 3:

1. Look over another of the data files and then run the program using it. Are the size of the environment and the initial locations and directions of the fish what you expected based on the analysis of `onefish.dat`?
2. Create a data file of your own and then run the program using it. Is the initial configuration of the environment what you would have expected?

Sneaking a Peek at Some Code

As I was compiling and running the program, I noticed two files called `SimpleMBSDemo1.java` and `SimpleMBSDemo2.java`. I decided to run the first one and discovered that it was a simpler version of the simulation program that always ran for 15 timesteps. It wasn't as interesting as the full version of the program, but the filename said "Simple" so I thought that I might be able to read it and understand it even before meeting with Jamie. I decided to look at the code.

```
public class SimpleMBSDemo1
{
    // Specify number of rows and columns in environment.
    private static final int ENV_ROWS = 10;   // rows in environment
    private static final int ENV_COLS = 10;   // columns in environment

    // Specify how many timesteps to run the simulation.
    private static final int NUM_STEPS = 15;  // number of timesteps

    // Specify the time delay for each step
    private static final int DELAY = 1000;    // delay in milliseconds

    /** Start the Marine Biology Simulation program.
     *  The String arguments (args) are not used in this application.
     **/
    public static void main(String[] args)
    {
        // Construct an empty environment and several fish in the
        // context of that environment.
        BoundedEnv env = new BoundedEnv(ENV_ROWS, ENV_COLS);
        Fish f1 = new Fish(env, new Location(2, 2));
        Fish f2 = new Fish(env, new Location(2, 3));
        Fish f3 = new Fish(env, new Location(5, 8));

        // Construct an object that knows how to draw the environment
        // with a delay; display the initial configuration of the
        // environment.
        SimpleMBSDisplay display = new SimpleMBSDisplay(env, DELAY);
        display.showEnv();

        // Run the simulation for the specified number of steps.
        for ( int i = 0; i < NUM_STEPS; i++ )
        {
            f1.act();
            f2.act();
            f3.act();
            display.showEnv();
        }
    }
}
```

I noticed that the first few lines create named constants (`ENV_ROWS`, `ENV_COLS`, `NUM_STEPS`, and `DELAY`) that can be used later in the class. However, it was the `main` method that was most interesting to me, because I knew that this is where the program execution starts. I saw that the first thing it does is to create a `BoundedEnv` object. I figured that must represent the body of water — the "bounded environment". The program then creates three `Fish` objects, passing each one the environment and a `Location` object. I couldn't remember what the initial positions of the fish had been when I ran the program, but it seemed reasonable that these `Location` objects were determining the initial positions. (I decided to rerun the program to verify this.) I wasn't as sure about why the new fish were being passed the `Environment` object, and made a note to ask Jamie when we met.

Next, the program creates an object called `display` of a class called `SimpleMBSDisplay`. This seemed to be the object that was displaying the environment on the screen. I noticed that the new `SimpleMBSDisplay` object is also passed the environment when it is constructed, but this made sense to me if its job is to display the environment. I assumed that the call

```
display.showEnv();
```

actually does the displaying. The `SimpleMBSDisplay` constructor is also passed the `DELAY` constant which, according to the comment, seemed to be the time delay between timesteps that gives the user time to see what is happening.

Finally, the main method contains a loop in which the three fish are told to "act" and then `display` is asked to show the environment again. I wondered, does "act" mean "move"?

Now I was curious about what the difference was between `SimpleMBSDemo1.java` and `SimpleMBSDemo2.java`. I ran `SimpleMBSDemo2.java`, but it seemed to do pretty much what `SimpleMBSDemo1.java` had done. (It was a little difficult to know for sure, since the behavior of every run of the program was different anyway.) So I decided to compare the code in the two files. The only differences are at the end of the two programs.

From SimpleMBSDemo1:

```
// Construct an object that knows how to draw the environment with
// a delay; display the initial configuration of the environment.
SimpleMBSDisplay display = new SimpleMBSDisplay(env, DELAY);
display.showEnv();

// Run the simulation for the specified number of steps.
for ( int i = 0; i < NUM_STEPS; i++ )
{
    f1.act();
    f2.act();
    f3.act();
    display.showEnv();
}
```

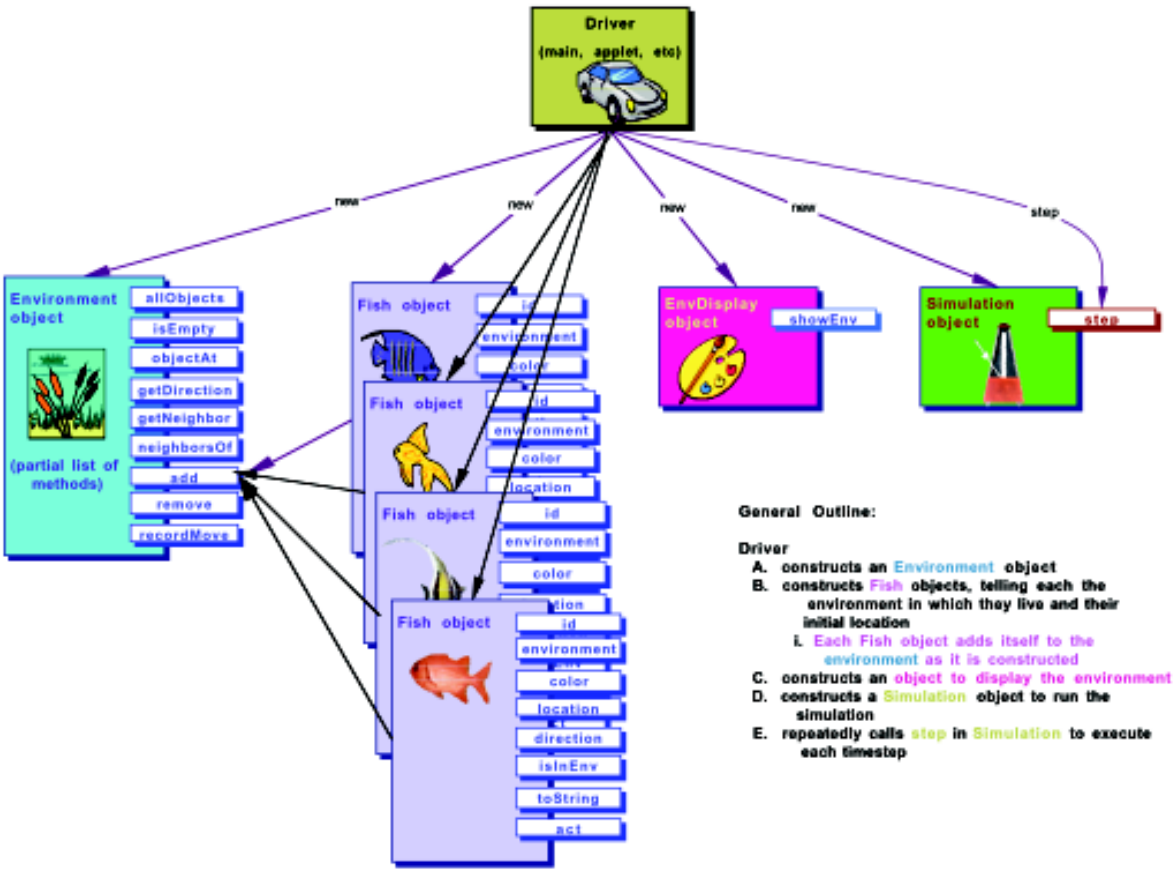
From SimpleMBSDemo2:

```
// Construct an object that knows how to draw the environment with
// a delay.
SimpleMBSDisplay display = new SimpleMBSDisplay(env, DELAY);

// Construct the simulation object. It needs to have the environment
// and the object that can draw the environment.
Simulation sim = new Simulation(env, display);

// Run the simulation for the specified number of steps.
for ( int i = 0; i < NUM_STEPS; i++ )
{
    sim.step();
}
```

I found the code for `SimpleMBSDemo2` harder to understand, and I was glad that I had looked at `SimpleMBSDemo1` first. First of all, `SimpleMBSDemo2` constructs the display object, `display`, but never asks it to show the environment. Secondly, even though `SimpleMBSDemo2` constructs three fish just as `SimpleMBSDemo1` did, it never asks them to act. Instead, it constructs an object of a new `Simulation` class, and in the loop asks it to "step." I noticed that the program passes the environment and the display object to the `Simulation` constructor, so I assumed that the `Simulation` object must ask the display object to show the environment when the `Simulation` object is constructed. I also assumed that its `step` method must ask the fish to act, although I wasn't sure how since the program never passes the fish to the simulation.

Later in the summer I created the picture below. It illustrates the behavior of a *driver* in the marine biology simulation, such as the main method in `SimpleMBSDemo2`. (A driver is a main method, applet, or graphical user interface that "drives" the rest of the program.)

The Driver



At this point I felt that I had learned about as much about the marine biology simulation program as I could without Jamie's "guided tour." I had a basic idea about how the program was working and I was ready to start exploring some of the code (like the mysterious Simulation class!) in more detail.