Centre for Computational Finance and Economic Agents

http://www.essex.ac.uk/ccfea/

University Of Essex, UK.

EASSS Lab Exercise: The Java Agent Based Modelling Toolkit (JABM): El Farol Bar Problem

Background Reading

Inductive Reasoning and bounded Rationality

W. B. Arthur, "Inductive reasoning and bounded rationality," *The American Economic Review*, vol. 84, no. 2, pp. 406-411, 1994. [Online]

XML Tutorial

If you are not familiar with XML files, there is an online tutorial on XML:

http://www.w3schools.com/xml/default.asp

Introduction to Dependency Injection and the Spring framework

Dependency Injection Demystified

http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework

Pre-requisites: Setting up the Eclipse IDE with the JABM Software

Step 1. Create a workspace on your J: drive

You will first need to a create folder called a workspace to hold all of your work. Create a folder on your `J:` drive called `workspace_JABM`.

Step 2. Start up the Eclipse IDE

When prompted to select a workspace, for the summer-school you should choose the folder created in the previous step.

```
J:\workspace_JABM
```

Once eclipse has started you can close the welcome screen.

Step 3. Import the JABM zip file into Eclipse

Eclipse has a feature that allows you to import existing projects from ZIP files. The file `jabm-0.8_01.zip` can be imported directly into Eclipse as an existing project. You can find this file on your EASSS pen drive, or alternatively download it from http://jabm.sourceforge.net/.

In eclipse click the 'File' button and the option 'Import…'. In the subwindow hit the 'plus' sign next to 'General' and click on 'Existing Projects into Workspace' and then hit 'Next'.

Highlight the 'Select archive file:' button. Then browse to the location of 'jabm-0.8_01.zip' and 'Open'. Hit 'Finish'.

In the eclipse 'Package Explorer' (which is by default on the left of the eclipse window) you should see two packages:

1. Jabm
2. Jabm-examples

Step 4. Viewing the example

Click on 'Window' option on taskbar at the top of eclipse window. Choose 'Open Perspective/Other' click on 'Spring' and hit OK to get the Spring perspective running.

Expand 'jabm-examples' in the package explorer on the left. Expand the config directory and double click on 'elfarolbar.xml'.

In the central pane of the eclipse window where the 'elfarolbar.xml' xml configuration is displayed. Right click in the central pane and choose 'Open With/XML Editor'.

You are now ready to proceed to the lab exercises.

Configuring simulations using JABM and Spring

We will use a framework called JABM to implement the El Farol Bar model.

JABM uses the Java Spring framework to configure our main application object `SimulationController`. `SimulationController` is a class that is already provided for you by the `jabm` library in the package `net.sourceforge.jabm`.

The spring framework provides an object factory which initialises the objects in our simulation based on the settings in an xml configuration file. It implements a design pattern called Dependency Injection. The central principle is that you should let the spring framework take care of instantiating all the instances of objects used in your simulation and setting their attributes- you should ***not*** write any code to perform object initialisation in your own classes. Thus we say that dependencies between objects are *injected* into the system instead of being created by the system itself.

Each instance of an object in the simulation is represented by an entity called a *bean*. If you browse the 'elfarolbar.xml' file you will see many different beans defined using the <bean> tag, for example:

```
<!--  The prototype used to manufacture patron agents -->
<bean id="patronAgent" scope="prototype"
```

```
        class="net.sourceforge.jabm.examples.elfarolbar.PatronAgent">
        <property name="strategy" ref="adaptivePredictionStrategy" />
        <property name="barCapacity" value="60" />
        <property name="scheduler" ref="simulationController" />
</bean>
```

The above bean definition specifies how we are going to "manufacture" the agents in our simulation. The class property of the bean definition specifies the class we are going to use to represent agents, ie: *net.sourceforge.jabm.examples.elfarolbar.PatronAgent*. If we wanted to run a simulation with a different type of agent we could substitute a different class name here.

Exercise 1

Browse through the bean definitions in the XML file in order to familiarise yourself with this notation. Open the Spring Explorer view in Eclipse by choosing the `Window` menu option then `Show View/Outline`. In the Outline window expand 'beans xmlns=http: … to get a list of beans. Notice that Eclipse understands the bean definitions and allows you to quickly find a particular bean. If you click on `patronAgent` in the Outline window the bean definition in the XML file above is found and highlighted.

A. Hold down the CTRL key and hover over the class name in the bean tag in the central window. Notice that Eclipse will open the source-code for this class if you click on it.
B. In the central window return to elfarolbar.xml and the `patronAgent` bean. Hold down the CTRL key and click on the text `adaptivePredictionStrategy`. Eclipse should take you to the corresponding bean definition for the agent's strategy. Return to the `patronAgent` bean.
C. Hold down the CTRL key and click on the text `barCapacity`. Eclipse should take you to the source-code for the `setBarCapacity()` method.

Every bean has a unique id. This allows us to refer to other beans when we are initialising a particular class. For example, in the above definition we are setting the agent's strategy based on another bean definition: *adaptivePredictionStrategy*, which refers to the id of another bean definition.

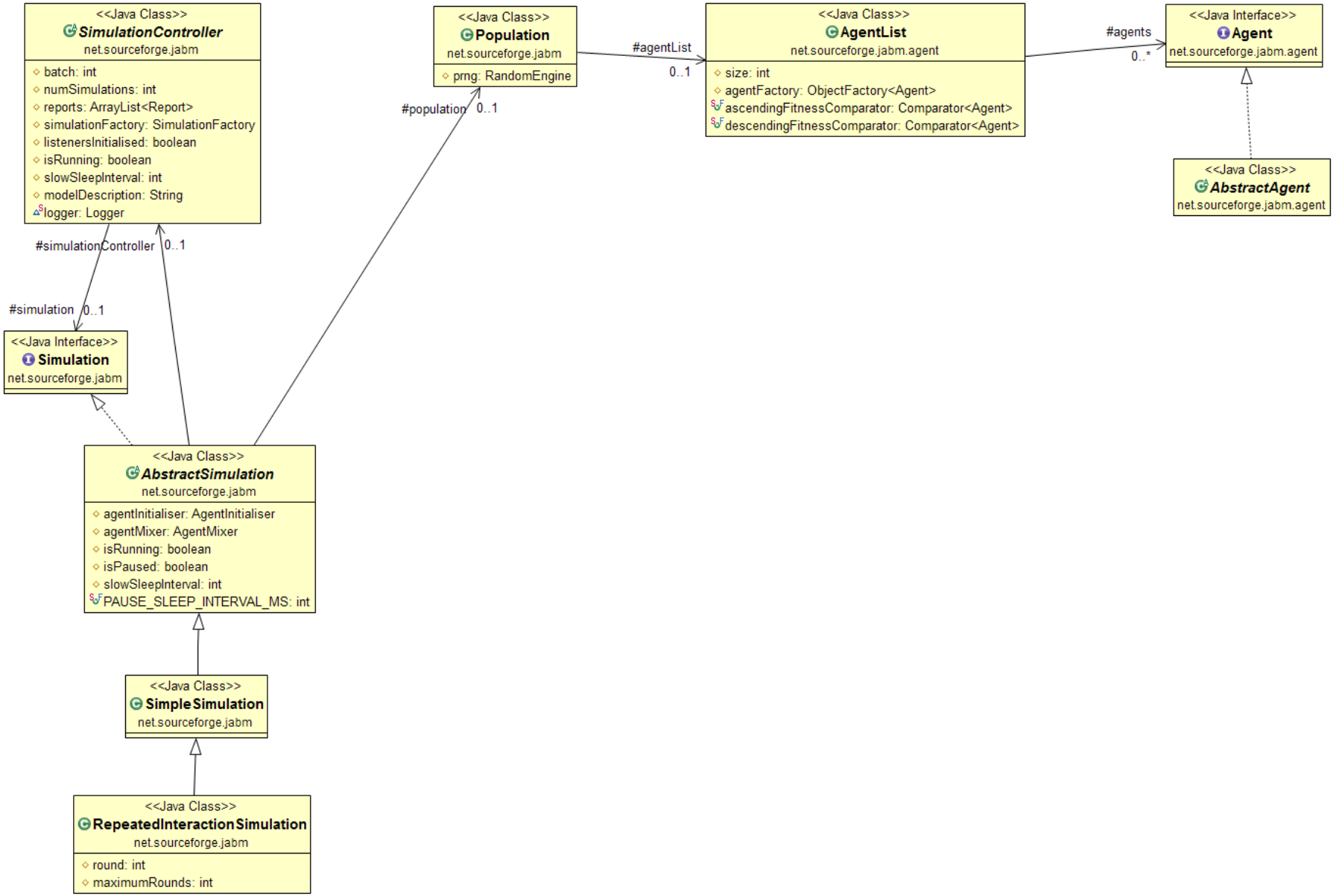
Once we have specified which class we want to use, we then specify how to configure the attributes of this class. The `property` tags refer to attributes that we can set using setters and getters. For example, the `PatronAgent` class has an attribute called `barCapacity` that can be set through the setter `setBarCapacity(int)`. When spring configures an agent it will look for a setter corresponding to the name of the attribute we are configuring and call it to initialise the object.

The `scope` attribute specifies whether or not we are allowed to instantiate multiple instances of this class. If we are only allowed to instantiate a **single** instance, then the scope should be set to `singleton`. This is the case, for example, when we configure the PRNG used in the simulation:

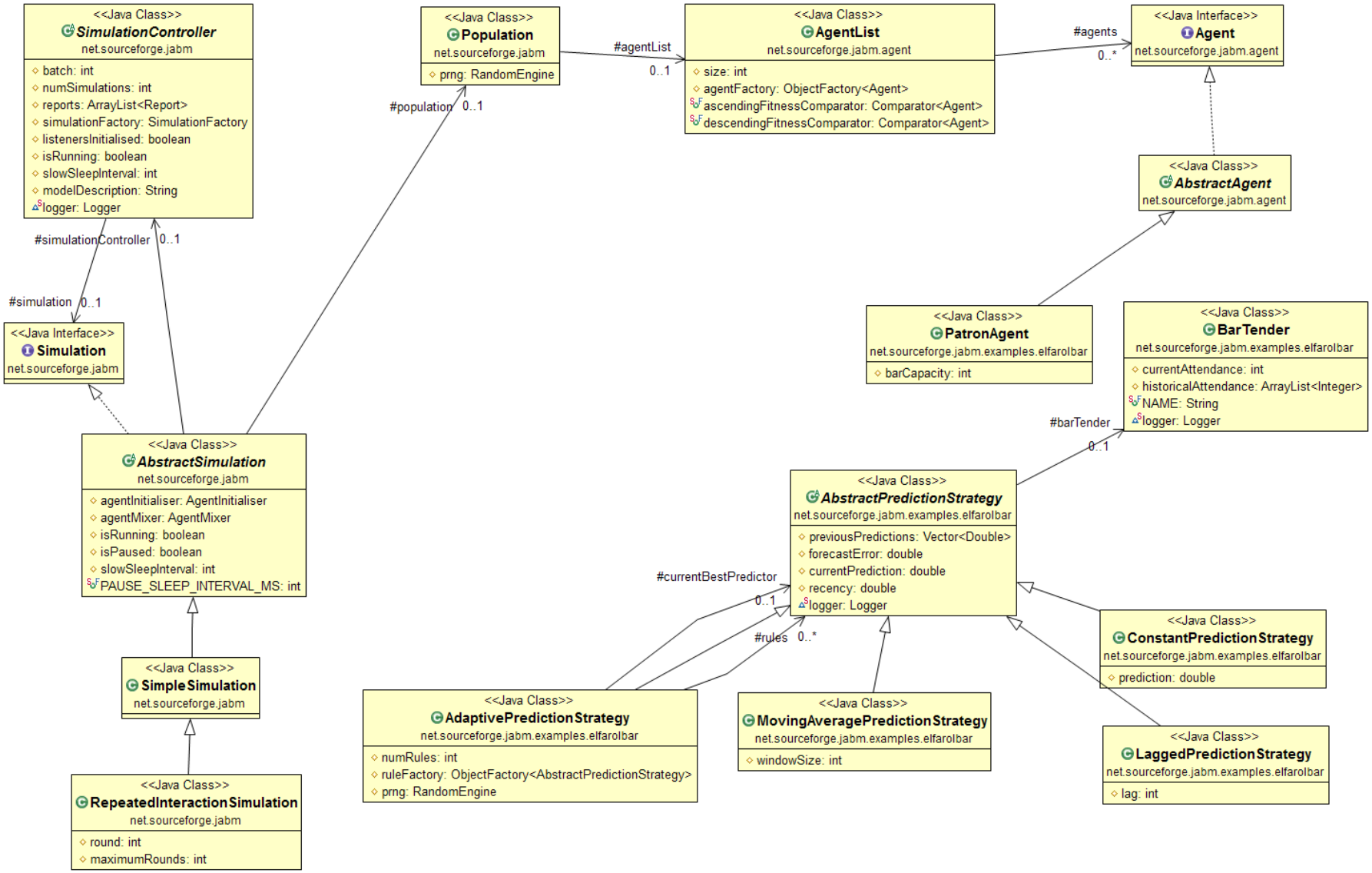
```
<bean id="prng" class="cern.jet.random.engine.MersenneTwister64"
             scope="singleton">
            <constructor-arg>
                  <bean class="java.util.Date" />
            </constructor-arg>
</bean>
```

If the scope is not specified then the default is `singleton`. The alternative is to use `scope="prototype"` as we did in the agent bean definition (since we will use **many** instances of the agent class to represent each agent in the simulation).

We will use a library called jabs that provides some generic classes useful for agent-based modelling. The classes we will use are illustrated in the following UML diagram:


<<Java Class>>
SimulationController
net.sourceforge.jabm
batch: int
numSimulations: int
reports: ArrayList<Report>
simulationFactory: SimulationFactory
listenersInitialised: boolean
isRunning: boolean
slowSleepInterval: int
modelDescription: String
logger: Logger
#simulationController
0..1
#simulation
0..1
<<Java Interface>>
Simulation
net.sourceforge.jabm
<<Java Class>>
AbstractSimulation
net.sourceforge.jabm
agentInitialiser: AgentInitialiser
agentMixer: AgentMixer
isRunning: boolean
isPaused: boolean
slowSleepInterval: int
PAUSE_SLEEP_INTERVAL_MS: int
<<Java Class>>
SimpleSimulation
net.sourceforge.jabm
<<Java Class>>
RepeatedInteractionSimulation
net.sourceforge.jabm
round: int
maximumRounds: int
<<Java Class>>
Population
net.sourceforge.jabm
prng: RandomEngine
#population
0..1
#agentList
0..1
<<Java Class>>
AgentList
net.sourceforge.jabm.agent
size: int
agentFactory: ObjectFactory<Agent>
ascendingFitnessComparator: Comparator<Agent>
descendingFitnessComparator: Comparator<Agent>
#agents
0..*
<<Java Interface>>
Agent
net.sourceforge.jabm.agent
<<Java Class>>
AbstractAgent
net.sourceforge.jabm.agent

We have extended these classes with classes which implement functionality specific to the El Farol Bar problem as follows:



We use dependency injection and the spring framework to configure this system. The top-level object is the `SimulationController`. By configuring the attributes of this class we can set up our simulation model. We do this recursively. So one of the attributes of the `SimulationController` is `simulationBeanName` which is set to `repeatedSimulation`. If you navigate to `repeatedSimulation` it has a property called `population` if you navigate to population you will see it has a property called size which is given the value 100. This bean represents the population of agents encapsulated in the class `net.sourceforge.jasa.sim.Population`. We can configure the `Population` and other simulation properties through the xml definitions until we have configured a complete simulation model right down to the attributes of individual agents.

Exercise 2

Browse all the beans in the model and familiarise yourself with the way the code works. Use CTRL click to consult the corresponding Java source code. (Note: When pressing the CTRL key the eclipse can take a little time to react).

Exercise 3

Run the simulation by creating a new Eclipse run configuration to execute the class:

```
net.sourceforge.jabm.DesktopSimulationManager
```

Click 'Run' on the menu bar and then click on 'Run Configurations …'. In the 'Project:' pane type:

jabm-examples

In the 'Main class:' pane type:
`net.sourceforge.jabm.DesktopSimulationManager`
Click on the '(x)=Arguments' tab. To tell the simulation controller which configuration file to use, specifying the following in the' VM arguments' box:

```
-Djabm.config=config/elfarolbar.xml
```

This specifies the name of the Spring beans configuration file.

Hit 'Apply' and then 'Run' at the bottom of the window. If everything is set up correctly then the simulation should run.

In the simulation window you will see two tabs –

1. Strategy execution frequency (population) tab which displays the proportion of the population adopting a particular strategy rule over time.
2. Bar attendance displays the number of patrons visiting the El Farol bar over time.

The top left hand corner holds the run, stop and pause buttons.

Exercise 4

In 'elfarolbar.xml' find the population size try to use the 'Outline' window to navigate and increase it from 100 to 150 and run the simulation. Can you see the difference it makes in the 'Bar Attendance' tab. Change the population back to 100.

Exercise 5

Create a new strategy by copying the `MovingAveragePredictionStrategy` and modifying the elfarolbar.xml bar to call your class instead of the original `MovingAveragePredictionStrategy` class.

1. In the 'Package Explorer' navigate to 'jabm-examples/src/net.sourceforge.jabm.examples.elfarolbar' highlight 'net.sourceforge.jabm.examples.elfarolbar' then go to 'New' on the taskbar of the eclipse window and click on 'Class'. In the 'Name:' pane enter out new class name `NewMovingAveragePredictionStrategy`. In this window the 'Source folder: pane should read 'jabm-examples/src' and the 'Package:' pane should read 'net.sourceforge.jabm.examples.elfarolbar'. Hit the 'Finish' button at the bottom of the window.
2. The `NewMovingAveragePredictionStrategy` class should appear in the middle window (if it doesn't you can double click on it in the 'Package Explorer' window). We want to copy the contents of the original `MovingAveragePredictionStrategy` class to our new class. Open `MovingAveragePredictionStrategy` by double clicking on it in the 'Package Explorer' window. Try to copy and paste the contents to our new class BEWARE THE CONTENTS ARE NOT GOING TO BE IDENTICAL! Things to remember when copying a. The class name in our new class definition is our new class name `NewMovingAveragePredictionStrategy` and this class extends `AbstractPredictionStrategy`. b. The toString method needs to change to return 'NewMovingAveragePredictionStrategy …' instead of 'MovingAveragePredictionStrategy ….'.
3. Go to 'elfarolbar.xml' we need to change every occurrence of `MovingAveragePredictionStrategy to NewMovingAveragePredictionStrategy`. The beans `strategyExecutionFrequency`, `MovingAveragePredictionStrategy` and `ruleFactory` will need to change. Note: Keep the bean ids consistent you will note that the bean ids start in the 'elfarolbar.xml' with a lowercase letter while the class names start with an uppercase letter.

Rerun the simulation, since we have not changed the, the functionality of the moving average calculation we should see no difference. We have though shown that we can amend the 'elfarolbar.xml' to build the simulation with our new class `NewMovingAveragePredictionStrategy`. In the 'Strategy execution frequency (population)' tab you will see in the list of strategies in the middle pane our new class

`NewMovingAveragePredictionStrategy` which is listed instead of the original `MovingAveragePredictionStrategy`.

Exercise 6

Amend the bean definition for the class `NewMovingAveragePredictionStrategy` (use the 'Outline' window to find it). Set the `windowSize` property to 50. (Hint: use value = "50" following other examples in the xml). Rerun the simulation.

Exercise 7

Update the bean definition for `ruleFactory` so that agents are able to use an additional 20 rules of type `NewMovingAveragePredictionStrategy` with random window sizes (that is change the `windowSize` from `value="50" back to ref="drawFromWindowSizeDistribution").` (Hint: To change the number of rules there are two places in the elfaronbar.xml that you will need to modify; bean adaptivePredictionStrategy and numRules (20 to 40) and bean ruleFactory <value>5</value> to <value>25</value> the bean id of `NewMovingAveragePredictionStrategy`). When you rerun the simulation you should find that the time series contains more randomness.

Exercise 8

Reset the number of rules used to their original values. Modify the `maximumRounds` property of the `repeatedSimulation` bean from 500 to 10000 and run the simulation. In longer runs do you see any evidence of cycling?

Exercise 9

The original paper states:

"*The predictors self-organize into an equilibrium pattern or 'ecology' in which, on average 40 percent are forecasting above 60, 60 percent below 60*" W. B. Arthur, "Inductive reasoning and bounded rationality," *The American Economic Review*, vol. 84, no. 2, pp. 406-411, 1994. [Online]

Create a new class which implements the interface net.sourceforge.jabm.report.Report to check whether this is the case in our simulation.