UNA PICCOLA INTRODUZIONE A SWARM:
Listings in Java

Preparata da Marie-Edith Bissey
Dipartimento di Politiche Pubbliche e Scelte Collettive POLIS
Università del Piemonte Orientale
email: bissey@sp.unipmn.it

March 7, 2001

Contents

1 Un mercato semplice

Listing 1: La classe per il Consumer

```java
// Consumer.java

import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;

public class Consumer extends SwarmObjectImpl
    {
    // global variables for the class
    public int myBudget;
    public int myName;
    public int moneySpent;

    // constructor
    public Consumer(Zone aZone, int name, int budget)
        {
        super(aZone);
        myName=name;
        myBudget=budget;
        }

    // determines randomly whether the consumer goes to the market
    public int goToTheMarket()   // note the () when no arguments
        {
        int k;
        k=Globals.env.uniformIntRand.getIntegerWithMin$withMax(0,1);
        return k;
        }

    // determines randomly how much is spent on the market
    public int spend()
        {
        moneySpent=Globals.env.uniformIntRand.getIntegerWithMin$withMax
              (0,myBudget);
        return moneySpent;
        }

    // calculate remaining budget
    public int calculateRemainingBudget()
        {
        myBudget-=moneySpent;
        return myBudget;
        }

    // pass variables to other classes
    public int getConsumerName()
        {
        return myName;
        }

    public int getBudget()
        {
        return myBudget;
        }
    } // end of consumer class
```

Listing 2: Il modelSwarm:

```
// modelSwarm.java

import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.SymbolImpl;
import swarm.defobj.FArguments;
import swarm.defobj.FArgumentsImpl;
import swarm.defobj.FCall;
import swarm.defobj.FCallImpl;
import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.activity.FActionForEach;
import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;

public class ModelSwarm extends SwarmImpl
    {
    // global variables for the class
    public Schedule modelSchedule;
    public ActionGroup modelActions;
    public Consumer aConsumer;
    public int modelTime;

    // constructor for modelSwarm
    public ModelSwarm (Zone aZone)
        {
        super(aZone);
        modelTime=0;
        }

    // build the consumer object
    public Object buildObjects()
        {
        int budget=10;
        int name=1;

        super.buildObjects();
        aConsumer=new Consumer(getZone(),name,budget);
        return this;
        }

    // define the marketDay actions
    public Object marketDay()
        {
        int go;
        int spending;

        go=aConsumer.goToTheMarket();

        if (go==1)
            {
            spending=aConsumer.spend();
            System.out.println("This is time "+modelTime);
            System.out.println("I am consumer " + aConsumer.
                 getConsumerName() + ", I went to the market and spent
                  " + spending +".");
            System.out.println("I have " + aConsumer.
                 calculateRemainingBudget() + " of currency left.");
            }
        else
            {
            System.out.println("This is time "+modelTime);
            System.out.println("I am consumer " + aConsumer.
                getConsumerName() + ", I did not go to the market.");
            System.out.println("I have " + aConsumer.getBudget() + " of
                currency left.");
            }
        return this;
        }

    // build actions
    public Object buildActions()
        {
        // create the action group for all actions to be performed
        // at each time (it is trivial, and not necessary here, as
        // there is only one action: marketDay)
        modelActions=new ActionGroupImpl(getZone());
        try
            {
            modelActions.createActionTo$message(this,
            new Selector(getClass(),"marketDay",false));
            }
        catch (Exception e)
            {
            e.printStackTrace(System.err);
            System.exit(1);
            }

        // now schedule the actions in time
        modelSchedule=new ScheduleImpl(getZone());
        modelSchedule.at$createAction(0,modelActions);
        return this;
        }

    // activity
    public Activity activateIn(Swarm swarmContext)
        {
        super.activateIn(swarmContext);
        modelSchedule.activateIn(this);
        return this.getActivity();
        }

    }// end of class modelSwarm
```

Listing 3: Il file StartMarket

```
// StartMarket.java

import swarm.Globals; // no # but a ; at the end
import swarm.defobj.ZoneImpl;

//import "modelSwarm.java";

public class StartMarket
    {
    public static void main(String[] args) // the main function MUST be
            'void'
```

```
        {
        // declare modelSwarm
        ModelSwarm modelSwarm;
        // initialise Swarm: need the 4 strings!!!
        Globals.env.initSwarm ("market","2.1.1","bissey@sp.unipmn.it",
              args);

        // create the modelSwarm, using the class constructor
        modelSwarm=new ModelSwarm(Globals.env.globalZone);

        // get the simulation running
        Globals.env.randomGenerator.setStateFromSeed(934850934);
        modelSwarm.buildObjects();
        modelSwarm.buildActions();
        modelSwarm.activateIn(null);
        (modelSwarm.getActivity()).run();
        }
    }
```

Listing 4: Il Makefile

```
JAVA_SRC = ModelSwarm.java Consumer.java StartMarket.java

all: $(JAVA_SRC)
    $(SWARMHOME)/bin/javacswarm $(JAVA_SRC)

clean:
    -rm *.class
```

2 Il mercato con collezioni di oggetti

Listing 5: La classe per il Consumer

```
// Consumer.java

import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;
import swarm.collections.Array;
import swarm.collections.ArrayImpl;
import swarm.collections.Map;
import swarm.collections.MapImpl;

public class Consumer extends SwarmObjectImpl
    {
// global variables for the class
    public int myBudget;
    public int myMaxBudget;
    public int myName;
    public int moneySpent;
    public MapImpl mapOfSpending;
    public ArrayImpl arrayOfVisits;
```

```
// constructor (note that they have no return type)
    public Consumer(Zone aZone,int name,int maxBudget,int startBudget)
        {
        super(aZone);
        myName=name;
        myBudget=startBudget;
        myMaxBudget=maxBudget;
        }

// create the map of spending and the array of visits
    public void createMapOfSpending(Zone aZone)
        {
        mapOfSpending=new MapImpl(aZone);
        }
    public void createArrayOfVisits(Zone aZone,int size)
        {
        arrayOfVisits=new ArrayImpl(aZone,size);
        }

// this methods draws a random number between 0 and
// maxBudget to determine the budget of the consumer
    public int findBudget()
        {
        //take myBudget and add to it a random variable between
        // 0 and maxBudget.
        myBudget+=Globals.env.uniformIntRand.getIntegerWithMin$withMax
            (0,myMaxBudget);
        return myBudget;
        }

// determines randomly whether the consumer goes to the market
    public int goToTheMarket()   // note the () when no arguments
        {
        int k;
        k=Globals.env.uniformIntRand.getIntegerWithMin$withMax(0,1);
        return k;
        }

// determines randomly how much is spent on the market
    public int spend()
        {
        moneySpent=Globals.env.uniformIntRand.getIntegerWithMin$withMax
            (0,myBudget);
        return moneySpent;
        }

// calculate remaining budget
    public int calculateRemainingBudget()
        {
        myBudget–=moneySpent;
        return myBudget;
        }
```

```
// these methods are used to add elements to the map of spendings,
// and the array of visits, they also take care of casting the int
// values into Integer objects
    public void updateSpending(int key, int value)
        {
        Integer keyObject=new Integer(key);
        Integer valueObject=new Integer(value);
        mapOfSpending.at$insert(keyObject,valueObject);
        }
    public void updateVisits(int offset, int value)
        {
        Integer valueObject=new Integer(value);
        arrayOfVisits.atOffset$put(offset,valueObject);
        }
// pass variables to other classes
    public int getConsumerName()
        {
        return myName;
        }

    public int getBudget()
        {
        return myBudget;
        }

// get value at offset, in arrayOfVisits
// note: the return of a list, etc is an Object, so we need
// to cast it as and Integer when retrieving it
    public int getVisitValue(int offset)
        {
        Integer element;
        element=(Integer) arrayOfVisits.atOffset(offset);
        return element.intValue();
        }

// get value of element at key
    public int getSpendingValue(int key)
        {
        Integer keyObject;
        Integer element;
        keyObject=new Integer(key);
        element=(Integer) mapOfSpending.at(keyObject);
        return element.intValue();
        }

    } // end of consumer class
```

Listing 6: Il modelSwarm:

```
// modelSwarm.java

import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.SymbolImpl;
import swarm.defobj.FArguments;
import swarm.defobj.FArgumentsImpl;
import swarm.defobj.FCall;
import swarm.defobj.FCallImpl;
import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.activity.FActionForEach;
import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.collections.List;
import swarm.collections.ListImpl;

public class ModelSwarm extends SwarmImpl
    {
    // global variables for the class
    public Schedule modelSchedule;
    public ActionGroup modelActions;
    public int modelTime;
    public int maxTime;
    public int startBudget;
    public int maxBudget;
    public int notFinished;
    public int numberOfConsumers;
    public ListImpl listOfConsumers;

// constructor for modelSwarm
    public ModelSwarm (Zone aZone)
        {
        super(aZone);
        modelTime=0;
        maxTime=5;
        numberOfConsumers=3;
        startBudget=0; // no endowments
        maxBudget=10;
        notFinished=1;
        }

// build the consumer object
    public Object buildObjects()
        {
        int i;
        int name;

        super.buildObjects();

// create the list of consumers
        listOfConsumers=new ListImpl(getZone());

// iterate over all possible consumers
```

```
        for (i=1;i<=numberOfConsumers;++i)
            {
            Consumer aConsumer;
// name of consumer=index i
            name=i;
// create the consumers
            aConsumer=new Consumer(getZone(),name,maxBudget,startBudget
                 );
// create the mapOfSpending and the arrayOfVisits
            aConsumer.createMapOfSpending(getZone());
            aConsumer.createArrayOfVisits(getZone(),maxTime+1);
// add consumer to the list
            listOfConsumers.addFirst(aConsumer);
            }
        return this;
        }

// define the marketDay actions
    public Object marketDay()
        {
        int go;
        int spending;
        int budget;
        swarm.collections.ListIndex i=null;
        Consumer listElement;

// iterate over the list of consumers
// first create the index
        i=listOfConsumers.listBegin(getZone());
        while ((listElement=(Consumer)i.next())!=null)
            {
// update the budget of the consumer
            budget=listElement.findBudget();

// is he going to the market?
            go=listElement.goToTheMarket();

            if (go==1)
                {
                spending=listElement.spend();

// add 1 to array of visits, at position corresponding to current
// modelTime
                listElement.updateVisits(modelTime,1);
// add spending at the key modelTime in the mapOfSpending
                listElement.updateSpending(modelTime,spending);
// now print a report of the consumer's actions
                System.out.println("This is time "+modelTime);
                System.out.println("I am consumer " + listElement.
                     getConsumerName());
                System.out.println("My current budget is "+ listElement
                     .getBudget());
                System.out.println("Did I go to the market? (from array
                     ) "+ listElement.getVisitValue(modelTime));
                System.out.println("I spent (from map) "+ listElement.
                      getSpendingValue(modelTime));
                System.out.println("I have " + listElement.
                      calculateRemainingBudget() + " of currency left.")
                      ;
                }
            else
                {
// add 0 to array of visits, at position corresponding to current
// modelTime
                listElement.updateVisits(modelTime,0);
// add 0 to map of spending, at position corresponding to current
// modelTime
                listElement.updateSpending(modelTime,0);
// now print a report of the consumer's actions
                System.out.println("This is time "+modelTime);
                System.out.println("I am consumer " + listElement.
                      getConsumerName());
                System.out.println("My current budget is "+ listElement
                      .getBudget());
                System.out.println("Did I go to the market? (from array
                      ) "+ listElement.getVisitValue(modelTime));
                System.out.println("I have " + listElement.getBudget()
                       + " of currency left.");
                }
            }// end of iteration of list of consumers
// it is good practice to drop unused objects like indexes when they
// are no longer needed
            i.drop();
        return this;
        }

// at the end of each period, modelTime needs to increase by 1
    public void increaseTime()
        {
        ++modelTime;
        }

// the program should stop if it has run long enough
// in this case, if modelTime>maxTime
    public void checkToStop()
        {
        if (modelTime<=maxTime)
            {
            notFinished=1;
            }
        else
            {
            notFinished=0;
            this.getActivity().terminate();
            }
        }

// build actions
```

```
    public Object buildActions()
        {
// create the action group for all actions to be performed at each time
// (it is trivial, and not necessary here, as there is only one action:
// marketDay)
        modelActions=new ActionGroupImpl(getZone());
        try
            {
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"marketDay",false));
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"increaseTime",false));
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"checkToStop",false));
            }
        catch (Exception e)
            {
            e.printStackTrace(System.err);
            System.exit(1);
            }
// now schedule the actions in time
        modelSchedule=new ScheduleImpl(getZone(),1);
        modelSchedule.at$createAction(0,modelActions);
        return this;
        }

// activity
    public Activity activateIn(Swarm swarmContext)
        {
        super.activateIn(swarmContext);
        modelSchedule.activateIn(this);
        return this.getActivity();
        }

    }// end of class modelSwarm
```

Listing 7: Il file StartMarket

```
// StartMarket.java

import swarm.Globals; // no # but a ; at the end
import swarm.defobj.ZoneImpl;

//import "modelSwarm.java";

public class StartMarket
    {
    public static void main(String[] args) // the main function MUST be
          'void'
        {
        // declare modelSwarm
        ModelSwarm modelSwarm;
        // initialise Swarm: need the 4 strings!!!
        Globals.env.initSwarm ("market","2.1.1","bissey@sp.unipmn.it",
            args);

        // create the modelSwarm, using the class constructor
        modelSwarm=new ModelSwarm(Globals.env.globalZone);

        // get the simulation running
        Globals.env.randomGenerator.setStateFromSeed(100000);
        modelSwarm.buildObjects();
        modelSwarm.buildActions();
        modelSwarm.activateIn(null);
        (modelSwarm.getActivity()).run();
        }
    }
```

Listing 8: Il Makefile

```
JAVA_SRC = ModelSwarm.java Consumer.java StartMarket.java

all: $(JAVA_SRC)
    $(SWARMHOME)/bin/javacswarm $(JAVA_SRC)

clean:
    -rm *.class
```

3 Importare parametri da files

Listing 9: La classe per il Consumer

```
// Consumer.java

import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;
import swarm.collections.Array;
import swarm.collections.ArrayImpl;
import swarm.collections.Map;
import swarm.collections.MapImpl;

public class Consumer extends SwarmObjectImpl
    {
// global variables for the class
    public int myBudget;
    public int myMaxBudget;
    public int myName;
    public int moneySpent;
    public MapImpl mapOfSpending;
    public ArrayImpl arrayOfVisits;

// constructor (note that they have no return type)
    public Consumer(Zone aZone, int name, int maxBudget, int startBudget)
        {
        super(aZone);
```

```
            myName=name;
            myBudget=startBudget;
            myMaxBudget=maxBudget;
            }

// create the map of spending and the array of visits
      public void createMapOfSpending(Zone aZone)
            {
            mapOfSpending=new MapImpl(aZone);
            }
      public void createArrayOfVisits(Zone aZone,int size)
            {
            arrayOfVisits=new ArrayImpl(aZone,size);
            }

// this methods draws a random number between 0 and
// maxBudget to determine the budget of the consumer
      public int findBudget()
            {
            //take myBudget and add to it a random variable between
            // 0 and maxBudget.
            myBudget+=Globals.env.uniformIntRand.getIntegerWithMin$withMax
                  (0,myMaxBudget);
            return myBudget;
            }

// determines randomly whether the consumer goes to the market
      public int goToTheMarket()  // note the () when no arguments
            {
            int k;
            k=Globals.env.uniformIntRand.getIntegerWithMin$withMax(0,1);
            return k;
            }

// determines randomly how much is spent on the market
      public int spend()
            {
            moneySpent=Globals.env.uniformIntRand.getIntegerWithMin$withMax
                  (0,myBudget);
            return moneySpent;
            }

// calculate remaining budget
      public int calculateRemainingBudget()
            {
            myBudget−=moneySpent;
            return myBudget;
            }


// these methods are used to add elements to the map of spendings, and
   the
// array of visits, they also take care of casting the int values into
// Integer objects
```

```
    public void updateSpending(int key, int value)
        {
        Integer keyObject=new Integer(key);
        Integer valueObject=new Integer(value);
        mapOfSpending.at$insert(keyObject,valueObject);
        }
    public void updateVisits(int offset, int value)
        {
        Integer valueObject=new Integer(value);
        arrayOfVisits.atOffset$put(offset,valueObject);
        }
// pass variables to other classes
    public int getConsumerName()
        {
        return myName;
        }

    public int getBudget()
        {
        return myBudget;
        }

// get value at offset, in arrayOfVisits
// note: the return of a list, etc is an Object, so we need
// to cast it as and Integer when retrieving it
    public int getVisitValue(int offset)
        {
        Integer element;
        element=(Integer) arrayOfVisits.atOffset(offset);
        return element.intValue();
        }

// get value of element at key
    public int getSpendingValue(int key)
        {
        Integer keyObject;
        Integer element;
        keyObject=new Integer(key);
        element=(Integer) mapOfSpending.at(keyObject);
        return element.intValue();
        }

    } // end of consumer class
```

Listing 10: Il modelSwarm:

```
// modelSwarm.java

import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.SymbolImpl;
import swarm.defobj.FArguments;
import swarm.defobj.FArgumentsImpl;
import swarm.defobj.FCall;
import swarm.defobj.FCallImpl;
import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.activity.FActionForEach;
import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.collections.List;
import swarm.collections.ListImpl;

public class ModelSwarm extends SwarmImpl
    {
    // global variables for the class
    public Schedule modelSchedule;
    public ActionGroup modelActions;
    public int modelTime;
    public int maxTime;
    public int startBudget;
    public int maxBudget;
    public int notFinished;
    public int numberOfConsumers;
    public ListImpl listOfConsumers;


// constructor for modelSwarm
    public ModelSwarm (Zone aZone)
        {
        super(aZone);
        }

// build the consumer object
    public Object buildObjects()
        {
        int i;
        int name;

        super.buildObjects();

// create the list of consumers
        listOfConsumers=new ListImpl(getZone());

// iterate over all possible consumers
        for (i=1;i<=numberOfConsumers;++i)
            {
            Consumer aConsumer;
// name of consumer=index i
            name=i;
// create the consumers
            aConsumer=new Consumer(getZone(),name,maxBudget,startBudget
                );
// create the mapOfSpending and the arrayOfVisits
```

```
            aConsumer.createMapOfSpending(getZone());
            aConsumer.createArrayOfVisits(getZone(),maxTime+1);
// add consumer to the list
            listOfConsumers.addFirst(aConsumer);
            }
        return this;
        }

// define the marketDay actions
    public Object marketDay()
        {
        int go;
        int spending;
        int budget;
        swarm.collections.ListIndex i=null;
        Consumer listElement;

// iterate over the list of consumers
// first create the index
        i=listOfConsumers.listBegin(getZone());
        while ((listElement=(Consumer)i.next())!=null)
            {
// update the budget of the consumer
            budget=listElement.findBudget();

// is he going to the market?
            go=listElement.goToTheMarket();

            if (go==1)
                {
                spending=listElement.spend();

// add 1 to array of visits, at position corresponding to current
// modelTime
                listElement.updateVisits(modelTime,1);
// add spending at the key modelTime in the mapOfSpending
                listElement.updateSpending(modelTime,spending);
// now print a report of the consumer's actions
                System.out.println("This is time "+modelTime);
                System.out.println("I am consumer " + listElement.
                     getConsumerName());
                System.out.println("My current budget is "+ listElement
                     .getBudget());
                System.out.println("Did I go to the market? (from array
                     ) "+ listElement.getVisitValue(modelTime));
                System.out.println("I spent (from map) "+ listElement.
                     getSpendingValue(modelTime));
                System.out.println("I have " + listElement.
                     calculateRemainingBudget() + " of currency left.")
                     ;
                }
            else
                {
// add 0 to array of visits, at position corresponding to current
// modelTime
                listElement.updateVisits(modelTime,0);
// add 0 to map of spending, at position corresponding to current
// modelTime
                listElement.updateSpending(modelTime,0);
// now print a report of the consumer's actions
                System.out.println("This is time "+modelTime);
                System.out.println("I am consumer " + listElement.
                     getConsumerName());
                System.out.println("My current budget is "+ listElement
                     .getBudget());
                System.out.println("Did I go to the market? (from array
                     ) "+ listElement.getVisitValue(modelTime));
                System.out.println("I have " + listElement.getBudget()
                      + " of currency left.");
                }
            }// end of iteration of list of consumers
// it is good practice to drop unused objects like indexes when they
// are no longer needed
            i.drop();
        return this;
        }

// at the end of each period, modelTime needs to increase by 1
    public void increaseTime()
        {
        ++modelTime;
        }

// the program should stop if it has run long enough
// in this case, if modelTime>maxTime
    public void checkToStop()
        {
        if (modelTime<=maxTime)
            {
            notFinished=1;
            }
        else
            {
            notFinished=0;
            this.getActivity().terminate();
            }
        }

// build actions
    public Object buildActions()
        {
// create the action group for all actions to be performed at each time
// (it is trivial, and not necessary here, as there is only one action:
// marketDay)
        modelActions=new ActionGroupImpl(getZone());
        try
            {
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"marketDay",false));
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"increaseTime",false));
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"checkToStop",false));
            }
        catch (Exception e)
            {
            e.printStackTrace(System.err);
            System.exit(1);
            }

// now schedule the actions in time
        modelSchedule=new ScheduleImpl(getZone(),1);
        modelSchedule.at$createAction(0,modelActions);
        return this;
        }

// activity
    public Activity activateIn(Swarm swarmContext)
        {
        super.activateIn(swarmContext);
        modelSchedule.activateIn(this);
        return this.getActivity();
        }

    }// end of class modelSwarm
```

Listing 11: Il file StartMarket

```
// StartMarket.java

import swarm.Globals; // no # but a ; at the end
import swarm.defobj.ZoneImpl;
import swarm.defobj.LispArchiverImpl;

//import ”modelSwarm.java”;

public class StartMarket
    {
    public static void main(String[] args) // the main function MUST be
          ‘void’
        {
// declare modelSwarm
        ModelSwarm modelSwarm;

        LispArchiverImpl archiver;

// initialise Swarm: need the 4 strings!!!
        Globals.env.initSwarm ("market","2.1.1","bissey@sp.unipmn.it",
             args);

// create the archiver object
        archiver=new LispArchiverImpl(Globals.env.globalZone,"
            parameters.scm");

// import the parameters (note the casting as ModelSwarm). The archiver
// will also use automatically the constructor for the ModelSwarm
        modelSwarm=(ModelSwarm)archiver.getWithZone$key(Globals.env.
            globalZone,"modelSwarm");

// get the simulation running
        Globals.env.randomGenerator.setStateFromSeed(100000);
        modelSwarm.buildObjects();
        modelSwarm.buildActions();
        modelSwarm.activateIn(null);
        (modelSwarm.getActivity()).run();
        }
    }
```

Listing 12: Il Makefile

```
JAVA_SRC = ModelSwarm.java Consumer.java StartMarket.java

all: $(JAVA_SRC)
    $(SWARMHOME)/bin/javacswarm $(JAVA_SRC)

clean:
    -rm *.class
```

Listing 13: Il file parameters.scm

```
(list
    (cons 'modelSwarm
        (make-instance 'ModelSwarm
                    #:modelTime 0
                    #:maxTime 5
                    #:numberOfConsumers 3
                    #:startBudget 0 ;no endowment
                    #:maxBudget 10
                    #:notFinished 1 ))
)
```

4 L'interfaccia grafica

Listing 14: La classe per il Consumer

```
// Consumer.java

import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;
import swarm.collections.Array;
import swarm.collections.ArrayImpl;
import swarm.collections.Map;
import swarm.collections.MapImpl;
```

```
public class Consumer extends SwarmObjectImpl
    {
// global variables for the class
    public int myBudget;
    public int myMaxBudget;
    public int myName;
    public int moneySpent;
    public int currentTime;
    public MapImpl mapOfSpending;
    public ArrayImpl arrayOfVisits;

// constructor (note that they have no return type)
    public Consumer(Zone aZone,int name,int maxBudget,int startBudget)
        {
        super(aZone);
        myName=name;
        myBudget=startBudget;
        myMaxBudget=maxBudget;
        }

// create the map of spending and the array of visits
    public void createMapOfSpending(Zone aZone)
        {
        mapOfSpending=new MapImpl(aZone);
        }
    public void createArrayOfVisits(Zone aZone,int size)
        {
        arrayOfVisits=new ArrayImpl(aZone,size);
        }

// this methods draws a random number between 0 and
// maxBudget to determine the budget of the consumer
    public int findBudget()
        {
        //take myBudget and add to it a random variable between
        // 0 and maxBudget.
        myBudget+=Globals.env.uniformIntRand.getIntegerWithMin$withMax
             (0,myMaxBudget);
        return myBudget;
        }

// determines randomly whether the consumer goes to the market
    public int goToTheMarket()   // note the () when no arguments
        {
        int k;
        k=Globals.env.uniformIntRand.getIntegerWithMin$withMax(0,1);
        return k;
        }

// determines randomly how much is spent on the market
    public int spend()
        {
```

```
        moneySpent=Globals.env.uniformIntRand.getIntegerWithMin$withMax
              (0,myBudget);
        return moneySpent;
        }

// calculate remaining budget
    public int calculateRemainingBudget()
        {
        myBudget–=moneySpent;
        return myBudget;
        }


// these methods are used to add elements to the map of spendings, and
     the
// array of visits, they also take care of casting the int values into
// Integer objects
    public void updateSpending(int key, int value)
        {
        Integer keyObject=new Integer(key);
        Integer valueObject=new Integer(value);
        mapOfSpending.at$insert(keyObject,valueObject);
        }
    public void updateVisits(int offset, int value)
        {
        // note: set the currentTime to offset, which corresponds
        // to modelTime
        currentTime=offset;
        // update arrayOfVisits.
        Integer valueObject=new Integer(value);
        arrayOfVisits.atOffset$put(offset,valueObject);
        }
// pass variables to other classes
    public int getConsumerName()
        {
        return myName;
        }

    public int getBudget()
        {
        return myBudget;
        }

// get value at offset, in arrayOfVisits
// note: the return of a list, etc is an Object, so we need
// to cast it as and Integer when retrieving it
    public int getVisitValue(int offset)
        {
        Integer element;
        element=(Integer) arrayOfVisits.atOffset(offset);
        return element.intValue();
        }

// get value of element at key
```

```
    public int getSpendingValue(int key)
        {
        Integer keyObject;
        Integer element;
        keyObject=new Integer(key);
        element=(Integer) mapOfSpending.at(keyObject);
        return element.intValue();
        }

// get visit (for graphs)
    public int getVisit()
        {
        return this.getVisitValue(currentTime);
        }
// getspending (for graphs)
    public int getSpending()
        {
        return this.getSpendingValue(currentTime);
        }

    } // end of consumer class
```

Listing 15: Il modelSwarm:

```
// modelSwarm.java

import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.SymbolImpl;
import swarm.defobj.FArguments;
import swarm.defobj.FArgumentsImpl;
import swarm.defobj.FCall;
import swarm.defobj.FCallImpl;
import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.activity.FActionForEach;
import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.collections.List;
import swarm.collections.ListImpl;
import swarm.simtoolsgui.ProbeDisplayImpl;
import swarm.objectbase.EmptyProbeMapImpl;

public class ModelSwarm extends SwarmImpl
    {
    // global variables for the class
    public Schedule modelSchedule;
    public ActionGroup modelActions;
    public int modelTime;
    public int maxTime;
    public int startBudget;
    public int maxBudget;
    public int notFinished;
    public int numberOfConsumers;
    public ListImpl listOfConsumers;


// constructor for modelSwarm
    public ModelSwarm (Zone aZone)
        {
        super(aZone);
        EmptyProbeMapImpl modelProbeMap=new EmptyProbeMapImpl(getZone()
            ,getClass());
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("maxTime",getClass()));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("numberOfConsumers",getClass()
            ));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("startBudget",getClass()));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("maxBudget",getClass()));
        Globals.env.probeLibrary.setProbeMap$For(modelProbeMap,
            getClass());
        }

// build the consumer object
    public Object buildObjects()
        {
        int i;
        int name;

        super.buildObjects();

// create the list of consumers
        listOfConsumers=new ListImpl(getZone());

// iterate over all possible consumers
        for (i=1;i<=numberOfConsumers;++i)
            {
            Consumer aConsumer;
// name of consumer=index i
            name=i;
// create the consumers
            aConsumer=new Consumer(getZone(),name,maxBudget,startBudget
                );
// create the mapOfSpending and the arrayOfVisits
            aConsumer.createMapOfSpending(getZone());
            aConsumer.createArrayOfVisits(getZone(),maxTime+1);
// add consumer to the list
            listOfConsumers.addFirst(aConsumer);
            }
        return this;
        }
```

```
// define the marketDay actions
   public Object marketDay()
       {
       int go;
       int spending;
       int budget;
       swarm.collections.ListIndex i=null;
       Consumer listElement;

// iterate over the list of consumers
// first create the index
       i=listOfConsumers.listBegin(getZone());
       while ((listElement=(Consumer)i.next())!=null)
           {
// update the budget of the consumer
           budget=listElement.findBudget();

// is he going to the market?
           go=listElement.goToTheMarket();

           if (go==1)
               {
               spending=listElement.spend();

// add 1 to array of visits, at position corresponding to current
// modelTime
               listElement.updateVisits(modelTime,1);
// add spending at the key modelTime in the mapOfSpending
               listElement.updateSpending(modelTime,spending);
// now print a report of the consumer's actions
               System.out.println("This is time "+modelTime);
               System.out.println("I am consumer " + listElement.
                    getConsumerName());
               System.out.println("My current budget is "+ listElement
                    .getBudget());
               System.out.println("Did I go to the market? (from array
                    ) "+ listElement.getVisitValue(modelTime));
               System.out.println("I spent (from map) "+ listElement.
                    getSpendingValue(modelTime));
               System.out.println("I have " + listElement.
                    calculateRemainingBudget() + " of currency left.")
                    ;
               }
           else
               {
// add 0 to array of visits, at position corresponding to current
// modelTime
               listElement.updateVisits(modelTime,0);
// add 0 to map of spending, at position corresponding to current
// modelTime
               listElement.updateSpending(modelTime,0);
// now print a report of the consumer's actions
               System.out.println("This is time "+modelTime);
```

```
                System.out.println("I am consumer " + listElement.
                      getConsumerName());
                System.out.println("My current budget is "+ listElement
                      .getBudget());
                System.out.println("Did I go to the market? (from array
                      ) "+ listElement.getVisitValue(modelTime));
                System.out.println("I have " + listElement.getBudget()
                       + " of currency left.");
                }
            }// end of iteration of list of consumers
// it is good practice to drop unused objects like indexes when they
// are no longer needed
            i.drop();
        return this;
        }

// at the end of each period, modelTime needs to increase by 1
    public void increaseTime()
        {
        ++modelTime;
        }

// the program should stop if it has run long enough
// in this case, if modelTime>maxTime
// return 0 or 1 for observer
    public int checkToStop()
        {
        if (modelTime<=maxTime)
            {
            notFinished=1;
            return 0;
            }
        else
            {
            notFinished=0;
            return 1;
            }
        }

// build actions
    public Object buildActions()
        {
// create the action group for all actions to be performed at each time
// (it is trivial, and not necessary here, as there is only one action:
// marketDay)
        modelActions=new ActionGroupImpl(getZone());
        try
            {
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"marketDay",false));
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"increaseTime",false));
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"checkToStop",false));
            }
        catch (Exception e)
            {
            e.printStackTrace(System.err);
            System.exit(1);
            }

// now schedule the actions in time
        modelSchedule=new ScheduleImpl(getZone(),2);
        modelSchedule.at$createAction(0,modelActions);
        return this;
        }

// activity
    public Activity activateIn(Swarm swarmContext)
        {
        super.activateIn(swarmContext);
        modelSchedule.activateIn(this);
        return this.getActivity();
        }

// methods to pass objects to other classes
    public ListImpl getListOfConsumers()
        {
        return (ListImpl)listOfConsumers;
        }

    public Consumer getConsumer(int name)
        {
// note that elements are entered using the addFirst methods
// so the first element is the latest agent created
        return (Consumer)listOfConsumers.atOffset(listOfConsumers.
             getCount()-name);
        }

    public int getCurrentTime()
        {
        return modelTime;
        }
    }// end of class modelSwarm
```

Listing 16: L'ObserverSwarm:

```
// ObserverSwarm.java
import swarm.Globals; // no # but a ; at the end
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.ZoneImpl;
import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.objectbase.Swarm;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;
import swarm.gui.Colormap;
import swarm.gui.ColormapImpl;
import swarm.gui.ZoomRaster;
import swarm.gui.ZoomRasterImpl;
import swarm.analysis.EZGraph;
import swarm.analysis.EZGraphImpl;
import swarm.simtoolsgui.GUISwarm;
import swarm.simtoolsgui.GUISwarmImpl;
import swarm.defobj.LispArchiverImpl;

public class ObserverSwarm extends GUISwarmImpl
    {
// declare variables
    public int displayFrequency;
    public int displayConsumerName;

    public ActionGroup displayActions;
    public Schedule displaySchedule;

    public ModelSwarm modelSwarm;

    public EZGraphImpl spendingGraph;
    public EZGraphImpl consumerGraph;

// constructor
    public ObserverSwarm(Zone aZone)
        {
        super(aZone);
// create probe map
        EmptyProbeMapImpl probeMap=new EmptyProbeMapImpl(getZone(),
            getClass());
        probeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("displayFrequency",getClass())
            );
        probeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("displayConsumerName",getClass
            ()));
        Globals.env.probeLibrary.setProbeMap$For(probeMap, getClass());
        }// end of constructor

// build Objects
    public Object buildObjects()
        {
        LispArchiverImpl archiver;

        super.buildObjects();
// create the archiver object
        archiver=new LispArchiverImpl(getZone(),"parameters.scm");
// import the parameters (note the casting as ModelSwarm). The archiver
// will also use automatically the constructor for the ModelSwarm
        modelSwarm=(ModelSwarm)archiver.getWithZone$key(getZone(),"
            modelSwarm");
// create probe displays
        Globals.env.createArchivedProbeDisplay(modelSwarm,"modelSwarm")
            ;
        Globals.env.createArchivedProbeDisplay(this, "observerSwarm");
// set control panel to stop
        getControlPanel().setStateStopped();
// build models objects
        modelSwarm.buildObjects();
// create graphs
// spendingGraph shows the average, total and minimum spending
// of consumers during the game
// note the ease of use of the constructor:
//public EZGraphImpl(Zone aZone,
//                  String aTitle,
//                  String xl,
//                  String yl,
//                  String windowGeometryRecordName)
        spendingGraph=new EZGraphImpl(getZone(),"Agents' spending",
                "Timex2","Spending","spendingGraph");
// create the average,total,min sequences (remember the try/catch
     blocks
// necessary with selectors)
        try
            {
            spendingGraph.
                 createAverageSequence$withFeedFrom$andSelector
                ("Average spending",modelSwarm.getListOfConsumers(),
                new Selector(Class.forName("Consumer"), "getSpending",
                        false));
            spendingGraph.createTotalSequence$withFeedFrom$andSelector
                ("Total spending",modelSwarm.getListOfConsumers(),
                new Selector(Class.forName("Consumer"), "getSpending",
                        false));
            spendingGraph.createMinSequence$withFeedFrom$andSelector
                ("Minimum spending",modelSwarm.getListOfConsumers(),
                new Selector(Class.forName("Consumer"), "getSpending",
                        false));
            }
        catch (Exception e)
            {
            System.err.println ("Exception in creating spendingGraph: "
                          + e.getMessage ());
            }
// consumerGraph shows the frequentation and spending on the market
// of a consumer (determined by displayConsumerName).
        consumerGraph=new EZGraphImpl(getZone(),"A consumer",
                "Timex2","Visit/Spending","consumerGraph");
// create the sequences (remember the try/catch blocks
// necessary with selectors)
        try
            {
            consumerGraph.createSequence$withFeedFrom$andSelector
                ("Went to the market",modelSwarm.getConsumer(
                      displayConsumerName),
                new Selector(Class.forName("Consumer"), "getVisit",
                        false));
            consumerGraph.createSequence$withFeedFrom$andSelector
                ("Spent",modelSwarm.getConsumer(displayConsumerName),
                new Selector(Class.forName("Consumer"), "getSpending",
                        false));
            }
        catch (Exception e)
            {
            System.err.println ("Exception in creating consumerGraph: "
                          + e.getMessage ());
            }
        return this;
        }// end of observer buildObjects

    public Object buildActions()
        {
        super.buildActions();
        modelSwarm.buildActions();
// define displayActions
        displayActions=new ActionGroupImpl(getZone());
        try
            {
            displayActions.createActionTo$message(spendingGraph,
                new Selector(spendingGraph.getClass(),"step",true));
            displayActions.createActionTo$message(consumerGraph,
                new Selector(consumerGraph.getClass(),"step",true));
            displayActions.createActionTo$message(this,
                new Selector(getClass(),"observerCheckToStop",false));
            displayActions.createActionTo$message(getActionCache(),
                new Selector(getActionCache().getClass(),"doTkEvents",
                      true));           }
        catch (Exception e)
            {
            e.printStackTrace(System.err);
            System.exit(1);
            }
// now schedule the actions in time
        displaySchedule=new ScheduleImpl(getZone(),2);
        displaySchedule.at$createAction(1,displayActions);
        return this;
        }// end of observer buildActions

// activity
    public Activity activateIn(Swarm swarmContext)
        {
        super.activateIn(swarmContext);
        modelSwarm.activateIn(this);
        displaySchedule.activateIn(this);
        return this.getActivity();
        }

// observerCheckToStop
```

```
    public Object observerCheckToStop()
        {
        if(modelSwarm.checkToStop()==1)
            {
            getControlPanel().setStateStopped();
            }
        return this;
        }

    }
```

Listing 17: Il file StartMarket

```
// StartMarket.java

import swarm.Globals; // no # but a ; at the end
import swarm.defobj.ZoneImpl;
import swarm.simtoolsgui.GUISwarmImpl;
import swarm.defobj.LispArchiverImpl;

public class StartMarket
    {
    public static void main(String[] args) // the main function MUST be
          'void'
        {
        LispArchiverImpl archiver;

// initialise Swarm: need the 4 strings!!!
        Globals.env.initSwarm ("market","2.1.1","bissey@sp.unipmn.it",
             args);

// get the simulation running
        archiver=new LispArchiverImpl(Globals.env.globalZone,"
             parameters.scm");
// import the parameters (note the casting as ModelSwarm). The archiver
// will also use automatically the constructor for the ModelSwarm
        ObserverSwarm observerSwarm=(ObserverSwarm)archiver.
             getWithZone$key
                (Globals.env.globalZone,"observerSwarm");
        Globals.env.setWindowGeometryRecordName (observerSwarm, "
             observerSwarm");
        observerSwarm.buildObjects();
        observerSwarm.buildActions();
        observerSwarm.activateIn(null);
        observerSwarm.go();
        }
    }
```

Listing 18: Il Makefile

```
JAVA_SRC = ObserverSwarm.java ModelSwarm.java Consumer.java StartMarket
      .java

all: $(JAVA_SRC)
```

```
    $(SWARMHOME)/bin/javacswarm $(JAVA_SRC)

clean:
    -rm *.class
```

5 Rappresentare agenti nello spazio

Listing 19: La classe per il Consumer

```
// Consumer.java

import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;
import swarm.collections.Array;
import swarm.collections.ArrayImpl;
import swarm.collections.Map;
import swarm.collections.MapImpl;
import swarm.space.Object2dDisplay;
import swarm.space.Object2dDisplayImpl;
import swarm.space.Value2dDisplayImpl;
import swarm.gui.ZoomRasterImpl;
import swarm.gui.ZoomRaster;

public class Consumer extends SwarmObjectImpl
    {
// global variables for the class
    public int myBudget;
    public int myMaxBudget;
    public int myName;
    public int moneySpent;
    public int currentTime;
    public int marketGoer;
    public int positionX;
    public int positionY;
    public MapImpl mapOfSpending;
    public ArrayImpl arrayOfVisits;

// constructor (note that they have no return type)
    public Consumer(Zone aZone,int name,int maxBudget,int startBudget,
          int goer)
        {
        super(aZone);
        myName=name;
        myBudget=startBudget;
        myMaxBudget=maxBudget;
        marketGoer=goer;
        }

// methods to interact with the raster
    public void setPositionX$Y(int x, int y)
        {
```

```
        positionX=x;
        positionY=y;
        }

    public Object drawSelfOn(ZoomRasterImpl raster)
        {
        raster.drawPointX$Y$Color(positionX,positionY,this.
            getStrategyColor());
        return this;
        }

    public byte getStrategyColor()
        {
        byte color=0;
        if (marketGoer==0)
            {
            color=3;
            }
        else if (marketGoer==1)
            {
            color=1;
        }
        else
            {
            System.out.println("wrong marketGoer value");
            System.exit(0);
            }
        return color;
        }

// create the map of spending and the array of visits
    public void createMapOfSpending(Zone aZone)
        {
        mapOfSpending=new MapImpl(aZone);
        }
    public void createArrayOfVisits(Zone aZone,int size)
        {
        arrayOfVisits=new ArrayImpl(aZone,size);
        }

// this methods draws a random number between 0 and
// maxBudget to determine the budget of the consumer
    public int findBudget()
        {
        //take myBudget and add to it a random variable between
        // 0 and maxBudget.
        myBudget+=Globals.env.uniformIntRand.getIntegerWithMin$withMax
            (0,myMaxBudget);
        return myBudget;
        }

// determines randomly whether the consumer goes to the market
    public int goToTheMarket()  // note the () when no arguments
        {
```

```
        int k;
        k=Globals.env.uniformIntRand.getIntegerWithMin$withMax(0,1);
        marketGoer=k;
        return k;
        }

// determines randomly how much is spent on the market
    public int spend()
        {
        moneySpent=Globals.env.uniformIntRand.getIntegerWithMin$withMax
             (0,myBudget);
        return moneySpent;
        }

// calculate remaining budget
    public int calculateRemainingBudget()
        {
        myBudget-=moneySpent;
        return myBudget;
        }


// these methods are used to add elements to the map of spendings,
// and the array of visits, they also take care of casting the int
// values into Integer objects
    public void updateSpending(int key, int value)
        {
        Integer keyObject=new Integer(key);
        Integer valueObject=new Integer(value);
        mapOfSpending.at$insert(keyObject,valueObject);
        }
    public void updateVisits(int offset, int value)
        {
        // note: set the currentTime to offset, which corresponds
        // to modelTime
        currentTime=offset;
        // update arrayOfVisits.
        Integer valueObject=new Integer(value);
        arrayOfVisits.atOffset$put(offset,valueObject);
        }
// pass variables to other classes
    public int getConsumerName()
        {
        return myName;
        }

    public int getBudget()
        {
        return myBudget;
        }

// get value at offset, in arrayOfVisits
// note: the return of a list, etc is an Object, so we need
// to cast it as and Integer when retrieving it
```

```
    public int getVisitValue(int offset)
        {
        Integer element;
        element=(Integer) arrayOfVisits.atOffset(offset);
        return element.intValue();
        }

// get value of element at key
    public int getSpendingValue(int key)
        {
        Integer keyObject;
        Integer element;
        keyObject=new Integer(key);
        element=(Integer) mapOfSpending.at(keyObject);
        return element.intValue();
        }

// get visit (for graphs)
    public int getVisit()
        {
        return this.getVisitValue(currentTime);
        }
// getspending (for graphs)
    public int getSpending()
        {
        return this.getSpendingValue(currentTime);
        }
// get positions in space
    public int getPositionX()
        {
        return positionX;
        }
    public int getPositionY()
        {
        return positionY;
        }
    } // end of consumer class
```

Listing 20: Il modelSwarm:

```
// modelSwarm.java

import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.SymbolImpl;
import swarm.defobj.FArguments;
import swarm.defobj.FArgumentsImpl;
import swarm.defobj.FCall;
import swarm.defobj.FCallImpl;
import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.activity.FActionForEach;
import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.collections.List;
import swarm.collections.ListImpl;
import swarm.simtoolsgui.ProbeDisplayImpl;
import swarm.objectbase.EmptyProbeMapImpl;
import swarm.space.Object2dDisplayImpl;
import swarm.space.Grid2dImpl;
import swarm.space.Discrete2dImpl;

public class ModelSwarm extends SwarmImpl
    {
    // global variables for the class
    public Schedule modelSchedule;
    public ActionGroup modelActions;
    public int modelTime;
    public int maxTime;
    public int startBudget;
    public int maxBudget;
    public int notFinished;
    public int numberOfConsumers;
    public ListImpl listOfConsumers;
    public int worldWidth;
    public int worldHeight;
    public int sizeOfMarket;
    public int xMin;
    public int xMax;
    public int yMin;
    public int yMax;
    public Grid2dImpl world;
    public Discrete2dImpl market;

// constructor for modelSwarm
    public ModelSwarm (Zone aZone)
        {
        super(aZone);
        EmptyProbeMapImpl modelProbeMap=new EmptyProbeMapImpl(getZone()
            ,getClass());
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("maxTime",getClass()));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("numberOfConsumers",getClass()
            ));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("startBudget",getClass()));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("maxBudget",getClass()));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("worldWidth",getClass()));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("worldHeight",getClass()));
        Globals.env.probeLibrary.setProbeMap$For(modelProbeMap,
              getClass());
        }

// build the model objects
    public Object buildObjects()
        {
        int i;
        int x;
        int y;
        int name;
        EmptyProbeMapImpl consumerProbe;
        super.buildObjects();
// set position of market in world (+/- centre)
        xMin=(worldWidth-sizeOfMarket)/2;
        xMax=xMin+sizeOfMarket;
        yMin=(worldHeight-sizeOfMarket)/2;
        yMax=yMin+sizeOfMarket;
// initialise the world as a grid2d and fill with null objects
        world=new Grid2dImpl(getZone(),worldWidth,worldHeight);
        world.fillWithObject(null);
// initialise market as a discrete2d and fill the relevant part in
     yellow
        market=new Discrete2dImpl(getZone(),worldWidth,worldHeight);
        for (y=yMin;y<=yMax;++y)
            {
            for (x=xMin;x<=xMax;++x)
                {
                market.putValue$atX$Y(2,x,y);
                }
            }
// create the list of consumers
        listOfConsumers=new ListImpl(getZone());

// iterate over all possible consumers
        for (i=1;i<=numberOfConsumers;++i)
            {
            Consumer aConsumer;
// name of consumer=index i
            name=i;
// create the consumers
            aConsumer=new Consumer(getZone(),name,maxBudget,startBudget
                 ,0);
// set position to negative values
            aConsumer.setPositionX$Y(-999,-999);
// create the mapOfSpending and the arrayOfVisits
            aConsumer.createMapOfSpending(getZone());
            aConsumer.createArrayOfVisits(getZone(),maxTime+1);
// position in world
            this.findPositionInWorld$For$ExcludeMarket(world,aConsumer
                 ,1);
// probe for consumer
            consumerProbe=new EmptyProbeMapImpl(getZone(),aConsumer.
                 getClass());
            consumerProbe.addProbe(Globals.env.probeLibrary.
                getProbeForVariable$inClass("myName",aConsumer.
                getClass()));
            Globals.env.probeLibrary.setProbeMap$For(consumerProbe,
                aConsumer.getClass());
// add consumer to the list
            listOfConsumers.addFirst(aConsumer);
            }
        return this;
        }

// find position in world: puts the consumer in the world, excluding
// the market space if the last argument is 1, or only in the market
// space if the last argument is 0.
// First, it checks for the actual coordinates of the consumer
// object: if they are not negative (meaning that the consumer
// already is on the space), they start by putting a null object
// at this position. Then it looks for a random position for the
// consumer object, which has positive coordinates, is in the required
// part of the space (inside or outside market), and in an empty spot.
    public Object findPositionInWorld$For$ExcludeMarket(Grid2dImpl
         aWorld,Consumer aGuy,int exclude)
        {
        int trialX=-999;
        int trialY=-999;
// put null if consumers are already in the space
        if ((aGuy.getPositionX()>=0)&&(aGuy.getPositionY()>=0))
            {
            aWorld.putObject$atX$Y(null,aGuy.getPositionX(),aGuy.
                getPositionY());
            }
// put consumers randomly in the space
// first, if exclude is 1, outside the market part
        if (exclude==1)
            {
            while ( ((trialX<0)&&(trialY<0)) ||
                ((trialX>=xMin)&&(trialX<=yMin)&&(trialY>=yMin)&&(
                    trialY<=yMax)) ||
                (world.getObjectAtX$Y(trialX,trialY)!=null) )
                {
                trialX=Globals.env.uniformIntRand.
                    getIntegerWithMin$withMax(0,worldWidth-1);
                trialY=Globals.env.uniformIntRand.
                    getIntegerWithMin$withMax(0,worldHeight-1);
                }
            }
// then, if exclude is 0, inside the market part
        else if (exclude==0)
            {
            while ( ((trialX<0)&&(trialY<0)) ||
                (world.getObjectAtX$Y(trialX,trialY)!=null) )
                {
                trialX=Globals.env.uniformIntRand.
                    getIntegerWithMin$withMax(xMin,xMax);
                trialY=Globals.env.uniformIntRand.
```

```
                      getIntegerWithMin$withMax(yMin,yMax);
                }
            }
        else
            {
            System.out.println("Wrong value for exclude");
            System.exit(1);
            }
        aGuy.setPositionX$Y(trialX,trialY);
        world.putObject$atX$Y(aGuy,trialX,trialY);
        return this;
        }

// define the marketDay actions
    public Object marketDay()
        {
        int go;
        int spending;
        int budget;
        swarm.collections.ListIndex i=null;
        Consumer listElement;

// iterate over the list of consumers
// first create the index
        i=listOfConsumers.listBegin(getZone());
        while ((listElement=(Consumer)i.next())!=null)
            {
// update the budget of the consumer
            budget=listElement.findBudget();

// is he going to the market?
            go=listElement.goToTheMarket();

            if (go==1)
                {
                spending=listElement.spend();

// add 1 to array of visits, at position corresponding to current
// modelTime
                listElement.updateVisits(modelTime,1);
// add spending at the key modelTime in the mapOfSpending
                listElement.updateSpending(modelTime,spending);
// change position in space to inside the market (this will also
// change the colour of the consumer
                this.findPositionInWorld$For$ExcludeMarket(world,
                     listElement,0);
// now print a report of the consumer's actions
                System.out.println("This is time "+modelTime);
                System.out.println("I am consumer " + listElement.
                     getConsumerName());
                System.out.println("My current budget is "+ listElement
                     .getBudget());
                System.out.println("Did I go to the market? (from array
                     ) "+ listElement.getVisitValue(modelTime));
                System.out.println("I spent (from map) "+ listElement.
                     getSpendingValue(modelTime));
                System.out.println("I have " + listElement.
                     calculateRemainingBudget() + " of currency left.")
                     ;
                }
            else
                {
// add 0 to array of visits, at position corresponding to current
// modelTime
                listElement.updateVisits(modelTime,0);
// add 0 to map of spending, at position corresponding to current
// modelTime
                listElement.updateSpending(modelTime,0);
// change position in space to outside the market
                this.findPositionInWorld$For$ExcludeMarket(world,
                     listElement,1);
// now print a report of the consumer's actions
                System.out.println("This is time "+modelTime);
                System.out.println("I am consumer " + listElement.
                     getConsumerName());
                System.out.println("My current budget is "+ listElement
                     .getBudget());
                System.out.println("Did I go to the market? (from array
                     ) "+ listElement.getVisitValue(modelTime));
                System.out.println("I have " + listElement.getBudget()
                      + " of currency left.");
                }
            }// end of iteration of list of consumers
// it is good practice to drop unused objects like indexes when they
// are no longer needed
            i.drop();
        return this;
        }

// at the end of each period, modelTime needs to increase by 1
    public void increaseTime()
        {
        ++modelTime;
        }

// the program should stop if it has run long enough
// in this case, if modelTime>maxTime
// return 0 or 1 for observer
    public int checkToStop()
        {
        if (modelTime<=maxTime)
            {
            notFinished=1;
            return 0;
            }
        else
            {
            notFinished=0;
            return 1;
            }
        }

// build actions
    public Object buildActions()
        {
// create the action group for all actions to be performed at each time
// (it is trivial, and not necessary here, as there is only one action:
// marketDay)
        modelActions=new ActionGroupImpl(getZone());
        try
            {
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"marketDay",false));
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"increaseTime",false));
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"checkToStop",false));
            }
        catch (Exception e)
            {
            e.printStackTrace(System.err);
            System.exit(1);
            }

// now schedule the actions in time
        modelSchedule=new ScheduleImpl(getZone(),2);
        modelSchedule.at$createAction(0,modelActions);
        return this;
        }

// activity
    public Activity activateIn(Swarm swarmContext)
        {
        super.activateIn(swarmContext);
        modelSchedule.activateIn(this);
        return this.getActivity();
        }

// methods to pass objects to other classes
    public ListImpl getListOfConsumers()
        {
        return (ListImpl)listOfConsumers;
        }

    public Consumer getConsumer(int name)
        {
// note that elements are entered using the addFirst methods
// so the first element is the latest agent created
        return (Consumer)listOfConsumers.atOffset(listOfConsumers.
             getCount()-name);
        }
    public int getCurrentTime()
        {
        return modelTime;
        }

    public int getWorldWidth()
        {
        return worldWidth;
        }
    public int getWorldHeight()
        {
        return worldHeight;
        }
    public Grid2dImpl getWorld()
        {
        return world;
        }
    public Discrete2dImpl getMarket()
        {
        return market;
        }
    } // end of class modelSwarm
```

Listing 21: L'ObserverSwarm:

```
// ObserverSwarm.java
import swarm.Globals; // no # but a ; at the end
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.ZoneImpl;
import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.objectbase.Swarm;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;
import swarm.gui.Colormap;
import swarm.gui.ColormapImpl;
import swarm.gui.ZoomRaster;
import swarm.gui.ZoomRasterImpl;
import swarm.analysis.EZGraph;
import swarm.analysis.EZGraphImpl;
import swarm.simtoolsgui.GUISwarm;
import swarm.simtoolsgui.GUISwarmImpl;
import swarm.defobj.LispArchiverImpl;
import swarm.space.Object2dDisplay;
import swarm.space.Object2dDisplayImpl;
import swarm.space.Value2dDisplayImpl;
import swarm.gui.ZoomRasterImpl;
import swarm.gui.ZoomRaster;
import swarm.gui.Colormap;
import swarm.gui.ColormapImpl;
import swarm.defobj.Zone;
import swarm.Selector;

public class ObserverSwarm extends GUISwarmImpl
    {
// declare variables
    public int displayFrequency;
    public int displayConsumerName;

    public ActionGroup displayActions;
    public Schedule displaySchedule;

    public ModelSwarm modelSwarm;

    public EZGraphImpl spendingGraph;
    public EZGraphImpl consumerGraph;

    public ColormapImpl colorMap;
    public ZoomRasterImpl worldRaster;
    public Object2dDisplayImpl worldDisplay;
    public Value2dDisplayImpl marketDisplay;
    public int zoomFactor;

// constructor
    public ObserverSwarm(Zone aZone)
        {
        super(aZone);
// create probe map
        EmptyProbeMapImpl probeMap=new EmptyProbeMapImpl(getZone(),
             getClass());
        probeMap.addProbe(Globals.env.probeLibrary.
             getProbeForVariable$inClass("displayFrequency",getClass())
             );
        probeMap.addProbe(Globals.env.probeLibrary.
             getProbeForVariable$inClass("displayConsumerName",getClass
             ()));
        Globals.env.probeLibrary.setProbeMap$For(probeMap, getClass());
        }// end of constructor

// build Objects
    public Object buildObjects()
        {
        LispArchiverImpl archiver;

        super.buildObjects();
// create the archiver object
        archiver=new LispArchiverImpl(getZone(),"parameters.scm");
// import the parameters (note the casting as ModelSwarm). The archiver
// will also use automatically the constructor for the ModelSwarm
        modelSwarm=(ModelSwarm)archiver.getWithZone$key(getZone(),"
             modelSwarm");
// create probe displays
        Globals.env.createArchivedProbeDisplay(modelSwarm,"modelSwarm")
```

```
            ;
        Globals.env.createArchivedProbeDisplay(this, "observerSwarm");
// set control panel to stop
        getControlPanel().setStateStopped();
// build models objects
        modelSwarm.buildObjects();
// create graphs
// spendingGraph shows the average, total and minimum spending
// of consumers during the game
// note the ease of use of the constructor:
//public EZGraphImpl(Zone aZone,
//                   String aTitle,
//                   String xl,
//                   String yl,
//                   String windowGeometryRecordName)
        spendingGraph=new EZGraphImpl(getZone(),"Agents’ spending",
                "Timex2","Spending","spendingGraph");
// create the average,total,min sequences (remember the try/catch
     blocks
// necessary with selectors)
        try
            {
            spendingGraph.
                 createAverageSequence$withFeedFrom$andSelector
                ("Average spending",modelSwarm.getListOfConsumers(),
                new Selector(Class.forName("Consumer"), "getSpending",
                        false));
            spendingGraph.createTotalSequence$withFeedFrom$andSelector
                ("Total spending",modelSwarm.getListOfConsumers(),
                new Selector(Class.forName("Consumer"), "getSpending",
                        false));
            spendingGraph.createMinSequence$withFeedFrom$andSelector
                ("Minimum spending",modelSwarm.getListOfConsumers(),
                new Selector(Class.forName("Consumer"), "getSpending",
                        false));
            }
        catch (Exception e)
            {
            System.err.println ("Exception in creating spendingGraph: "
                              + e.getMessage ());
            }
// consumerGraph shows the frequentation and spending on the market
// of a consumer (determined by displayConsumerName).
        consumerGraph=new EZGraphImpl(getZone(),"A consumer",
                "Timex2","Visit/Spending","consumerGraph");
// create the sequences (remember the try/catch blocks
// necessary with selectors)
        try
            {
            consumerGraph.createSequence$withFeedFrom$andSelector
                ("Went to the market",modelSwarm.getConsumer(
                      displayConsumerName),
                new Selector(Class.forName("Consumer"), "getVisit",
                        false));
            consumerGraph.createSequence$withFeedFrom$andSelector
                ("Spent",modelSwarm.getConsumer(displayConsumerName),
                new Selector(Class.forName("Consumer"), "getSpending",
                       false));
            }
        catch (Exception e)
            {
            System.err.println ("Exception in creating consumerGraph: "
                          + e.getMessage ());
            }

// create raster objects
        zoomFactor=10;
// colorMap
        colorMap=new ColormapImpl(getZone());
        colorMap.setColor$ToName((byte)0,"black");
        colorMap.setColor$ToName((byte)1,"blue");
        colorMap.setColor$ToName((byte)2,"yellow");
        colorMap.setColor$ToName((byte)3,"red");
// world raster
        worldRaster=new ZoomRasterImpl(getZone());
        worldRaster.setColormap(colorMap);
        worldRaster.setZoomFactor(zoomFactor);
        worldRaster.setWidth$Height(modelSwarm.getWorldWidth(),
             modelSwarm.getWorldHeight());
        worldRaster.setWindowTitle("A little town");
        worldRaster.pack();
// market display
        marketDisplay=new Value2dDisplayImpl(getZone(),worldRaster,
             colorMap,modelSwarm.getMarket());
// world display
        try
            {
            worldDisplay=new Object2dDisplayImpl(getZone(),worldRaster,
                modelSwarm.getWorld(),new Selector(Class.forName("
                     Consumer"),"drawSelfOn",false));
            worldRaster.setButton$Client$Message(3,worldDisplay,
                new Selector(worldDisplay.getClass(),"makeProbeAtX$Y",
                     true));
            }
        catch(Exception e)
            {
            e.printStackTrace(System.err);
            System.exit(1);
            }
        worldDisplay.setObjectCollection(modelSwarm.getListOfConsumers
             ());
// display the world raster at the beginning
        worldRaster.erase();
        marketDisplay.display();
        worldDisplay.display();
        worldRaster.drawSelf();

        return this;
        }// end of observer buildObjects

    public Object buildActions()
        {
        super.buildActions();
        modelSwarm.buildActions();
// define displayActions
        displayActions=new ActionGroupImpl(getZone());
        try
            {
            displayActions.createActionTo$message(spendingGraph,
                new Selector(spendingGraph.getClass(),"step",true));
            displayActions.createActionTo$message(consumerGraph,
                new Selector(consumerGraph.getClass(),"step",true));
            displayActions.createActionTo$message(worldRaster,
                new Selector(worldRaster.getClass(),"erase",true));
            displayActions.createActionTo$message(marketDisplay,
                new Selector(marketDisplay.getClass(),"display",true));
            displayActions.createActionTo$message(worldDisplay,
                new Selector(worldDisplay.getClass(),"display",true));
            displayActions.createActionTo$message(worldRaster,
                new Selector(worldRaster.getClass(),"drawSelf",true));
            displayActions.createActionTo$message(this,
                new Selector(getClass(),"observerCheckToStop",false));
            displayActions.createActionTo$message(Globals.env.
                  probeDisplayManager,
                new Selector(Globals.env.probeDisplayManager.getClass()
                    ,"update",true));
            displayActions.createActionTo$message(getActionCache(),
                new Selector(getActionCache().getClass(),"doTkEvents",
                      true));
            }
        catch (Exception e)
            {
            e.printStackTrace(System.err);
            System.exit(1);
            }
System.out.println("create actions done for observer");
// now schedule the actions in time
        displaySchedule=new ScheduleImpl(getZone(),2);
        displaySchedule.at$createAction(1,displayActions);
        return this;
        }// end of observer buildActions

// activity
    public Activity activateIn(Swarm swarmContext)
        {
        super.activateIn(swarmContext);
        modelSwarm.activateIn(this);
        displaySchedule.activateIn(this);
        return this.getActivity();
        }

// observerCheckToStop
```

```
    public Object observerCheckToStop()
        {
        if(modelSwarm.checkToStop()==1)
            {
            System.out.println("THE MODEL STOPPED RUNNING");
            getControlPanel().setStateStopped();
            }
        return this;
        }

    }// end of observer class
```

6 Girare più volte la simulazione

Listing 22: La classe per il Consumer

```
// Consumer.java

import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;
import swarm.collections.Array;
import swarm.collections.ArrayImpl;
import swarm.collections.List;
import swarm.collections.ListImpl;
import swarm.space.Object2dDisplay;
import swarm.space.Object2dDisplayImpl;
import swarm.space.Value2dDisplayImpl;
import swarm.gui.ZoomRasterImpl;
import swarm.gui.ZoomRaster;

public class Consumer extends SwarmObjectImpl
    {
// global variables for the class
    public int myBudget;
    public int myMaxBudget;
    public int myName;
    public int moneySpent;
    public int currentTime;
    public int marketGoer;
    public int positionX;
    public int positionY;
    public ListImpl listOfSpending;
    public ArrayImpl arrayOfVisits;

// constructor (note that they have no return type)
    public Consumer(Zone aZone,int name,int maxBudget,int startBudget,
          int goer)
        {
        super(aZone);
        myName=name;
        myBudget=startBudget;
        myMaxBudget=maxBudget;
        marketGoer=goer;
        }

// methods to interact with the raster
    public void setPositionX$Y(int x, int y)
        {
        positionX=x;
        positionY=y;
        }

    public Object drawSelfOn(ZoomRasterImpl raster)
        {
        raster.drawPointX$Y$Color(positionX,positionY,this.
              getStrategyColor());
        return this;
        }

    public byte getStrategyColor()
        {
        byte color=0;
        if (marketGoer==0)
            {
            color=3;
            }
        else if (marketGoer==1)
            {
            color=1;
        }
        else
            {
            System.out.println("wrong marketGoer value");
            System.exit(0);
            }
        return color;
        }

// create the list of spending and the array of visits
    public void createListOfSpending(Zone aZone)
        {
        listOfSpending=new ListImpl(aZone);
        }
    public void createArrayOfVisits(Zone aZone,int size)
        {
        arrayOfVisits=new ArrayImpl(aZone,size);
        }

// this methods draws a random number between 0 and
// maxBudget to determine the budget of the consumer
    public int findBudget()
        {
        //take myBudget and add to it a random variable between
        // 0 and maxBudget.
        myBudget+=Globals.env.uniformIntRand.getIntegerWithMin$withMax
            (0,myMaxBudget);
        return myBudget;
        }

// determines randomly whether the consumer goes to the market
    public int goToTheMarket()  // note the () when no arguments
        {
        int k;
        k=Globals.env.uniformIntRand.getIntegerWithMin$withMax(0,1);
        marketGoer=k;
        return k;
        }

// determines randomly how much is spent on the market
    public int spend()
        {
        moneySpent=Globals.env.uniformIntRand.getIntegerWithMin$withMax
            (0,myBudget);
        return moneySpent;
        }

// calculate remaining budget
    public int calculateRemainingBudget()
        {
        myBudget-=moneySpent;
        return myBudget;
        }


// these methods are used to add elements to the list of spendings,
// and the array of visits, they also take care of casting the int
// values into Integer objects
    public void updateSpending(int value)
        {
        Integer valueObject=new Integer(value);
        listOfSpending.addFirst(valueObject);
        }
    public void updateVisits(int offset, int value)
        {
        // note: set the currentTime to offset, which corresponds
        // to modelTime
        currentTime=offset;
        // update arrayOfVisits.
        Integer valueObject=new Integer(value);
        arrayOfVisits.atOffset$put(offset,valueObject);
        }
// pass variables to other classes
    public int getConsumerName()
        {
        return myName;
        }

    public int getBudget()
```

```
        {
        return myBudget;
        }

// get value at offset, in arrayOfVisits
// note: the return of a list, etc is an Object, so we need
// to cast it as and Integer when retrieving it
    public int getVisitValue(int offset)
        {
        Integer element;
        element=(Integer) arrayOfVisits.atOffset(offset);
        return element.intValue();
        }

// get value of element at key
    public int getSpendingValue()
        {
        Integer element;
        element=(Integer) listOfSpending.getFirst();
        return element.intValue();
        }

// get visit (for graphs)
    public int getVisit()
        {
        return this.getVisitValue(currentTime);
        }
// getspending (for graphs)
    public int getSpending()
        {
        return this.getSpendingValue();
        }
// get positions in space
    public int getPositionX()
        {
        return positionX;
        }
    public int getPositionY()
        {
        return positionY;
        }
// get all spending
    public ListImpl getAllSpending()
        {
        return listOfSpending;
        }
    } // end of consumer class
```

Listing 23: Il modelSwarm:

```
// modelSwarm.java

import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.SymbolImpl;
import swarm.defobj.FArguments;
import swarm.defobj.FArgumentsImpl;
import swarm.defobj.FCall;
import swarm.defobj.FCallImpl;
import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.activity.FActionForEach;
import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.collections.List;
import swarm.collections.ListImpl;
import swarm.simtoolsgui.ProbeDisplayImpl;
import swarm.objectbase.EmptyProbeMapImpl;
import swarm.space.Object2dDisplayImpl;
import swarm.space.Grid2dImpl;
import swarm.space.Discrete2dImpl;

public class ModelSwarm extends SwarmImpl
    {
    // global variables for the class
    public Schedule modelSchedule;
    public ActionGroup modelActions;
    public int modelTime;
    public int maxTime;
    public int startBudget;
    public int maxBudget;
    public int notFinished;
    public int numberOfConsumers;
    public ListImpl listOfConsumers;
    public int worldWidth;
    public int worldHeight;
    public int sizeOfMarket;
    public int xMin;
    public int xMax;
    public int yMin;
    public int yMax;
    public Grid2dImpl world;
    public Discrete2dImpl market;

// constructor for modelSwarm
    public ModelSwarm (Zone aZone)
        {
        super(aZone);
        EmptyProbeMapImpl modelProbeMap=new EmptyProbeMapImpl(getZone()
             , getClass());
        modelProbeMap.addProbe(Globals.env.probeLibrary.
             getProbeForVariable$inClass("maxTime", getClass()));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
             getProbeForVariable$inClass("numberOfConsumers", getClass()
            ));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("startBudget",getClass()));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("maxBudget",getClass()));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("worldWidth",getClass()));
        modelProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass("worldHeight",getClass()));
        Globals.env.probeLibrary.setProbeMap$For(modelProbeMap,
            getClass());
        }

// build the model objects
    public Object buildObjects()
        {
        int i;
        int x;
        int y;
        int name;
        EmptyProbeMapImpl consumerProbe;
        super.buildObjects();
// set position of market in world (+/- centre)
        xMin=(worldWidth-sizeOfMarket)/2;
        xMax=xMin+sizeOfMarket;
        yMin=(worldHeight-sizeOfMarket)/2;
        yMax=yMin+sizeOfMarket;
// initialise the world as a grid2d and fill with null objects
        world=new Grid2dImpl(getZone(),worldWidth,worldHeight);
        world.fillWithObject(null);
// initialise market as a discrete2d and fill the relevant part in
      yellow
        market=new Discrete2dImpl(getZone(),worldWidth,worldHeight);
        for (y=yMin;y<=yMax;++y)
            {
            for (x=xMin;x<=xMax;++x)
                {
                market.putValue$atX$Y(2,x,y);
                }
            }
// create the list of consumers
        listOfConsumers=new ListImpl(getZone());

// iterate over all possible consumers
        for (i=1;i<=numberOfConsumers;++i)
            {
            Consumer aConsumer;
// name of consumer=index i
            name=i;
// create the consumers
System.out.println("starting budget: "+startBudget);
            aConsumer=new Consumer(getZone(),name,maxBudget,startBudget
                 ,0);
// set position to negative values
```

```
            aConsumer.setPositionX$Y(-999,-999);
// create the listOfSpending and the arrayOfVisits
            aConsumer.createListOfSpending(getZone());
            aConsumer.createArrayOfVisits(getZone(),maxTime+1);
// position in world
            this.findPositionInWorld$For$ExcludeMarket(world,aConsumer
                 ,1);
// probe for consumer
            consumerProbe=new EmptyProbeMapImpl(getZone(),aConsumer.
                 getClass());
            consumerProbe.addProbe(Globals.env.probeLibrary.
                 getProbeForVariable$inClass("myName",aConsumer.
                 getClass()));
            Globals.env.probeLibrary.setProbeMap$For(consumerProbe,
                 aConsumer.getClass());
// add consumer to the list
            listOfConsumers.addFirst(aConsumer);
            }
        return this;
        }

// find position in world: puts the consumer in the world, excluding
// the market space if the last argument is 1, or only in the market
// space if the last argument is 0.
// First, it checks for the actual coordinates of the consumer
// object: if they are not negative (meaning that the consumer
// already is on the space), they start by putting a null object
// at this position. Then it looks for a random position for the
// consumer object, which has positive coordinates, is in the required
// part of the space (inside or outside market), and in an empty spot.
    public Object findPositionInWorld$For$ExcludeMarket(Grid2dImpl
         aWorld,Consumer aGuy,int exclude)
        {
        int trialX=-999;
        int trialY=-999;
// put null if consumers are already in the space
        if ((aGuy.getPositionX()>=0)&&(aGuy.getPositionY()>=0))
            {
            aWorld.putObject$atX$Y(null,aGuy.getPositionX(),aGuy.
                 getPositionY());
            }
// put consumers randomly in the space
// first, if exclude is 1, outside the market part
        if (exclude==1)
            {
            while ( ((trialX<0)&&(trialY<0)) ||
                ((trialX>=xMin)&&(trialX<=yMin)&&(trialY>=yMin)&&(
                     trialY<=yMax)) ||
                (world.getObjectAtX$Y(trialX,trialY)!=null) )
                {
                trialX=Globals.env.uniformIntRand.
                     getIntegerWithMin$withMax(0,worldWidth-1);
                trialY=Globals.env.uniformIntRand.
                     getIntegerWithMin$withMax(0,worldHeight-1);
                }
            }
// then, if exclude is 0, inside the market part
        else if (exclude==0)
            {
            while ( ((trialX<0)&&(trialY<0)) ||
                (world.getObjectAtX$Y(trialX, trialY)!=null) )
                {
                trialX=Globals.env.uniformIntRand.
                     getIntegerWithMin$withMax(xMin,xMax);
                trialY=Globals.env.uniformIntRand.
                     getIntegerWithMin$withMax(yMin,yMax);
                }
            }
        else
            {
            System.out.println("Wrong value for exclude");
            System.exit(1);
            }
        aGuy.setPositionX$Y(trialX, trialY);
        world.putObject$atX$Y(aGuy, trialX, trialY);
        return this;
        }

// set the simulation parmeters
    public Object setSimulationParameters(int simStartBudget)
        {
        startBudget=simStartBudget;
        return this;
        }

// define the marketDay actions
    public Object marketDay()
        {
        int go;
        int spending;
        int budget;
        swarm.collections.ListIndex i=null;
        Consumer listElement;

// iterate over the list of consumers
// first create the index
        i=listOfConsumers.listBegin(getZone());
        while ((listElement=(Consumer)i.next())!=null)
            {
// update the budget of the consumer
            budget=listElement.findBudget();

// is he going to the market?
            go=listElement.goToTheMarket();

            if (go==1)
                {
                spending=listElement.spend();
```

```
// add 1 to array of visits, at position corresponding to current
// modelTime
                listElement.updateVisits(modelTime,1);
// add spending at the key modelTime in the listOfSpending
                listElement.updateSpending(spending);
// change position in space to inside the market (this will also
// change the colour of the consumer
                this.findPositionInWorld$For$ExcludeMarket(world,
                     listElement,0);
// now print a report of the consumer's actions
//              System.out.println("This is time "+modelTime);
//              System.out.println("I am consumer " + listElement.
     getConsumerName());
//              System.out.println("My current budget is "+ listElement
     .getBudget());
//              System.out.println("Did I go to the market? (from array
     ) "+ listElement.getVisitValue(modelTime));
//              System.out.println("I spent (from list) "+ listElement.
     getSpendingValue(modelTime));
//              System.out.println("I have " + listElement.
     calculateRemainingBudget() + " of currency left.");
                }
            else
                {
// add 0 to array of visits, at position corresponding to current
// modelTime
                listElement.updateVisits(modelTime,0);
// add 0 to list of spending, at position corresponding to current
// modelTime
                listElement.updateSpending(0);
// change position in space to outside the market
                this.findPositionInWorld$For$ExcludeMarket(world,
                     listElement,1);
// now print a report of the consumer's actions
//              System.out.println("This is time "+modelTime);
//              System.out.println("I am consumer " + listElement.
     getConsumerName());
//              System.out.println("My current budget is "+ listElement
     .getBudget());
//              System.out.println("Did I go to the market? (from array
     ) "+ listElement.getVisitValue(modelTime));
//              System.out.println("I have " + listElement.getBudget()
      + " of currency left.");
                }
            }// end of iteration of list of consumers
// it is good practice to drop unused objects like indexes when they
// are no longer needed
            i.drop();
        return this;
        }

// at the end of each period, modelTime needs to increase by 1
    public void increaseTime()
```

```
        {
        ++modelTime;
        }

// the program should stop if it has run long enough
// in this case, if modelTime>maxTime
// return 0 or 1 for observer
    public int checkToStop()
        {
        if (modelTime<=maxTime)
            {
            notFinished=1;
            return 0;
            }
        else
            {
            notFinished=0;
            return 1;
            }
        }

// build actions
    public Object buildActions()
        {
// create the action group for all actions to be performed at each time
// (it is trivial, and not necessary here, as there is only one action:
// marketDay)
        modelActions=new ActionGroupImpl(getZone());
        try
            {
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"marketDay",false));
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"increaseTime",false));
            modelActions.createActionTo$message(this,
                new Selector(getClass(),"checkToStop",false));
            }
        catch (Exception e)
            {
            e.printStackTrace(System.err);
            System.exit(1);
            }

// now schedule the actions in time
        modelSchedule=new ScheduleImpl(getZone(),2);
        modelSchedule.at$createAction(0,modelActions);
        return this;
        }

// activity
    public Activity activateIn(Swarm swarmContext)
        {
        super.activateIn(swarmContext);
        modelSchedule.activateIn(this);
        return this.getActivity();
        }

// methods to pass objects to other classes
    public ListImpl getListOfConsumers()
        {
        return (ListImpl)listOfConsumers;
        }

    public Consumer getConsumer(int name)
        {
// note that elements are entered using the addFirst methods
// so the first element is the latest agent created
        return (Consumer)listOfConsumers.atOffset(listOfConsumers.
             getCount()-name);
        }

    public int getCurrentTime()
        {
        return modelTime;
        }

    public int getWorldWidth()
        {
        return worldWidth;
        }
    public int getWorldHeight()
        {
        return worldHeight;
        }
    public Grid2dImpl getWorld()
        {
        return world;
        }
    public Discrete2dImpl getMarket()
        {
        return market;
        }
    public ListImpl getAllSpendings()
        {
        Consumer element;
        Integer spending;
        ListImpl elementSpending;
        ListImpl listOfAllSpending;
        swarm.collections.ListIndex i=null;
        swarm.collections.ListIndex j=null;
        listOfAllSpending=new ListImpl(getZone());
        // iterate through the elements of listOfConsumers
        // to create elementSpending
        i=listOfConsumers.listBegin(getZone());
        while ((element=(Consumer)i.next())!=null)
            {
            elementSpending=new ListImpl(getZone());
            elementSpending=element.getAllSpending();
            // iterate through the element of elementSpending
            // to create listOfAllSpending
            j=elementSpending.listBegin(getZone());
            while ((spending=(Integer)j.next())!=null)
                {
                listOfAllSpending.addFirst(spending);
                }
            }
        return listOfAllSpending;
        }
    }// end of class modelSwarm
```

Listing 24: L'ExperimentSwarm:

```
// experSwarm.java

import swarm.Globals; // no # but a ; at the end
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.ZoneImpl;
import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.objectbase.Swarm;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;
import swarm.simtoolsgui.GUISwarm;
import swarm.simtoolsgui.GUISwarmImpl;
import swarm.defobj.LispArchiverImpl;
import swarm.collections.Array;
import swarm.collections.ArrayImpl;
import swarm.collections.List;
import swarm.collections.ListImpl;


public class ExperSwarm extends GUISwarmImpl
    {
// declare variables
    public int minStartBudget;
    public int maxStartBudget;
    public int incStartBudget;
    public int setStartBudget;
    public ListImpl listOfAllSpending;
    public float averageSpending;
    public ModelSwarm modelSwarm;

// constructor for ExperSwarm
    public ExperSwarm(Zone aZone)
        {
        super(aZone);
        EmptyProbeMapImpl experProbeMap=new EmptyProbeMapImpl(getZone()
            ,getClass());
        experProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass
          ("minStartBudget",getClass()));
        experProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass
          ("maxStartBudget",getClass()));
        experProbeMap.addProbe(Globals.env.probeLibrary.
            getProbeForVariable$inClass
          ("incStartBudget",getClass()));
        Globals.env.probeLibrary.setProbeMap$For(experProbeMap,getClass
            ());
        }
// build objects for experSwarm
    public Object buildObjects()
        {
        super.buildObjects();

// create probe displays
        Globals.env.createArchivedProbeDisplay(this, "experSwarm");
// Allow the user to alter experiment parameters
        getControlPanel().setStateStopped();

        return this;
        }
// run the experiment
    public Object run()
        {
        LispArchiverImpl outfile;
        SimulationData simLoop;
        LispArchiverImpl archiver;
        swarm.collections.ListIndex i=null;
        int sum=0;
        Integer spendingElement;

    // start running the simulation: for each possible starting budget
        for (setStartBudget=minStartBudget;
                setStartBudget<=maxStartBudget;
                    setStartBudget+=incStartBudget)
            {
            System.out.println("This round, startBudget is "+
                setStartBudget);
        // create setup file for ModelSwarm
            simLoop=new SimulationData();
            simLoop.initPara(setStartBudget);
            outfile=new LispArchiverImpl(this,"loop.scm");
            outfile.putShallow$object("modelSwarm",simLoop);
            outfile.sync();
        // create listOfAllSpending
            listOfAllSpending=new ListImpl(getZone());
        // load the data for the modelSwarm (and create it)
            archiver=new LispArchiverImpl(getZone(),"parameters.scm");
            modelSwarm=(ModelSwarm)archiver.getWithZone$key(getZone(),"
                modelSwarm");
            archiver.drop();
        // load simulation data, which can override some of the
              parameters
        // just loaded for the modelSwarm
            archiver=new LispArchiverImpl(getZone(),"loop.scm");
            simLoop=(SimulationData)archiver.getWithZone$key(getZone()
                 ,"modelSwarm");
            simLoop.setPara(modelSwarm);
            archiver.drop();
//          simLoop.drop();
            outfile.drop();

            modelSwarm.buildObjects();
            modelSwarm.buildActions();

        // run modelSwarm until it finishes
            while (modelSwarm.checkToStop()==0)
                {
                modelSwarm.marketDay();
                modelSwarm.increaseTime();
                modelSwarm.checkToStop();
                }
        // put values in listOfAllSpending from modelSwarm
            listOfAllSpending=(ListImpl)modelSwarm.getAllSpendings();
        // get average spending to use in graph
            i=listOfAllSpending.listBegin(getZone());
            while ((spendingElement=(Integer) i.next())!=null)
                {
                sum+=spendingElement.intValue();
                }
            averageSpending=sum/listOfAllSpending.getCount();
            System.out.println("average spending: "+averageSpending);

        // MAYBE USE IT WITHIN SELECTOR?????
//          spendingGraph.addPoints(setStartBudget,averageSpending);

            }// end of loop on startBudget
        // end of experiment. draw graph and stop panel
        System.out.println("This is the end of the experiment");
//      spendingGraph.paint();
        getControlPanel().setStateStopped();
        return this;
        }

    public Activity activateIn(Swarm swarmContext)
        {
        super.activateIn(swarmContext);
        return (this.getActivity());
        }


    }
```