Agent-Oriented Programming: Intro

Presenter:

Leigh Tesfatsion
Professor of Economics
Courtesy Professor of Mathematics
Iowa State University
Ames, Iowa 50011-1070
https://www2.econ.iastate.edu/tesfatsi/
tesfatsi@iastate.edu

26 February 2008
Outline

* What is **Object-Oriented Programming (OOP)**?
* **Agent-Oriented Programming (AOP)** vs. OOP?
* AOP via “Computational Laboratories”

**Example of a Computational Laboratory**

The **Trade Network Game (TNG)** Laboratory

https://www2.econ.iastate.edu/tesfatsi/tnghome.htm
Object-Oriented Programming (OOP)

KEY CONCEPTS:

★ **Object**
  - Methods (behaviors, functions, procedures,...)
  - Attributes (data, state information,...)
  - Access: public, private, or protected

★ **Class**

★ **Interface**

★ **Encapsulation**

★ **Inheritance (subclass, superclass)**

★ **Composition**
Object-Oriented Programming (OOP)

An **object** is a software entity containing **attributes** plus **methods** that act on these attributes.

An object controls **access** to its attributes and methods by declaring them

- **public** (accessible to all other objects);
- **private** (inaccessible to all other objects);
- or **protected** (accessible only to certain designated other objects).

A **class** is a blueprint for an object, i.e., a template used to create (“instantiate”) an object.
Class = Object Template

Class Employee

Employee Objects (Instances of Employee)

Ann  Ping  Mario  Dan
Illustration: Employee Class

```java
class EMPLOYEE {
    // Public Access:
    methods:
        getSocialSecurityNumber();
        getGender();
        getDateOfBirth();

    // Private Access:
    attributes:
        SocialSecurityNumber;
        Gender;
        DateOfBirth;
        Trustworthyness;
}
```
The public methods and public attributes of an object are called the *interface* of the object.

Objects *communicate* with each other via their public methods, i.e., by activating (“invoking”) the public methods of other objects.
In “good” OOP design, an object should only reveal to other objects what these objects need to know to interact with it.

Each class template specifies the interfaces for its instantiated objects -- it completely describes how users of these instantiated objects can interact with these instantiated objects.
Illustration: Employee Class

Class EMPLOYEE

Public Access:

Methods:

getSocialSecurityNumber( ) ;
getGender( ) ;
getDOB( ) ;

Private Access:

Attributes:

SocialSecurityNumber ;
Gender ;
DOB ;
Trustworthyness ;
Illustration: Payroll Class
(involves public methods in Employee class)

Class  PAYROLL
{
 Public Access:
 Methods:
calculateEmployeePay( );
payEmployee( );
Employee.getSocialSecurityNumber( );
Employee.getGender( );
Employee.getDateOfBirth( );

Private Access:
 Attributes:
 CurrentProfits;
 EmployeePayroll;
}
Encapsulation in conventional OOP is the process of determining which aspects of a class are not needed by other classes, and then hiding these aspects from other classes.

More precisely, encapsulation of a class is the process of dividing the class into two distinct parts:

1. (public) interface;

2. private (or protected) stuff that other classes do not need to know about.
Class Inheritance

- A class C can *inherit* the attributes and methods of another class B.
- Class C is then called the *subclass* of class B, and class B is called the *superclass* of class C.
- A subclass can also include specialized attributes and methods not present in the superclass.
Class Inheritance: Example

Superclass of Buyer and Seller

TradeBot

attribute
method
Price;
trade( );

Subclass of TradeBot

Buyer
Price = BidPrice;
trade() = buy();
calculateUtility();

Seller
Price = AskPrice;
trade() = sell();
calculateProfits();
Objects can be built -- or “composed” -- from other objects. This is called **composition**.

*Example:* A firm is composed of employees.

A **composition relationship** between objects is often termed a “Has-A” relationship. A firm “has an” employee.

An **inheritance relationship** between objects is often termed an “Is-A” relationship. A buyer “is a” trader.
AOP vs. OOP

- What is an agent?
- How does Agent-Oriented Programming (AOP) extend conventional Object-Oriented Programming (OOP)?
What is an Agent?

According to Nick Jennings (2000), an agent is an object capable of displaying...

- **(Structural) Reactivity:** Changes in internal structure in response to environmental changes

- **Social Ability:** Interaction with other agents through some form of language.

- **Pro-Activity:** Goal-directed actions.

- **Autonomy:** Some degree of control over its own actions (“self-activation”).
Key Distinction is Autonomy

- Distributed control, not simply distributed actions.

- According to Jennings, conventional objects encapsulate attributes and methods but not self-activation and localized action choice.

Autonomy means...

- Each agent effectively has its own persistent thread of control.
- Each agent decides for itself which actions to perform at what time, based in part on external environmental conditions and in part on private internal aspects (current beliefs, desires,…).
- Thus, in multi-agent systems, a potential source of uncertainty for each agent is not knowing for sure what other agents will do (called “behavioral” or “strategic” uncertainty).
Example: Worker Agent

Public Access:

// Public Methods
Protocols governing job search
Protocols governing negotiations with potential employers
Protocols governing unemployment benefits program
Methods for retrieving Worker data

Private Access:

// Private Methods
Method for calculating my expected utility assessments
Method for calculating my actual utility outcomes
Method for updating my worksite strategy (learning)
Methods for updating my methods (learning to learn)

// Private Attributes
Data about myself (my history, utility fct., current wealth...)
Data recorded about external world (employer behaviors,...)
Addresses for potential employers (permits communication)
AOP via Computational Laboratories

- **Computational Laboratory** = Computational framework for the study of complex system behaviors by means of controlled and replicable experiments.

- **Graphical User Interface (GUI)** permits experimentation by users with no programming background.

- **Modular/extensible form** permits framework capabilities to be changed/extended by users who have programming background.
Example: The Trade Network Game Lab (TNG) Lab

- Evolution of trade networks among strategically interacting traders (buyers, sellers, and dealers)
- Traders are instantiated as “tradebots,” i.e., as autonomous software entities with internal attributes and methods.
- Tradebots engage in event-driven communication
- Tradebots evolve their trade methods over time, starting from initially random trade methods
TNG Lab Architecture

- **Four-Layer Architecture:**
  - SimBioSys (C++ class framework)
  - TNG/SimBioSys (extension classes)
  - TNG/COM (permits interactive display)
  - TNG Lab (graphical user interface)

- Downloadable as Freeware in a Zip file that includes an automatic installation wizard

  https://www2.econ.iastate.edu/tesfatsi/tnghome.htm
TNG Lab 4-Layer Architecture

(TcFadzean, Stewart, and Tesfatsion, IEEE-TEC, 2001)
SimBioSys (McFadzean, 1995)

- Simulation toolkit
- C++ class library
- Designed for artificial life simulations (populations of autonomous interacting agents evolving in a virtual spatial world)
TNG/SimBioSys (McFadzean/Tesfatsion 1997)
Each Tradebot has...

- **Internalized social norms** (market protocols) taken as given
- **Internally stored state data** that can change through experiences
- An **internal trade method (personality)** that the tradebot evolves over time in an attempt to increase its profit (net payoff).
TNG Flow Diagram

- **INITIALIZATION**

- **LOOP** Through $T_{Max}$ Trade Cycles
  - **Trade Cycle:**
    - Search for Trade Partners;
    - Interactions with Trade Partners;
    - Update Expectations about Trade Partners.

- **EVOLUTION STEP (Update Trade Methods)**

- **LOOP** Through $T_{Max}$ Trade Cycles . . .
TNG Settings Screen

Genetic Algorithm:
- Generations: 50
- Trade Cycles: 150
- Seed: 19
- Mutation Rate: 0.005

Payoffs:
- Initial Expected: 1.4
- Refusal: -0.5
- Inactive: 0
- Experience Gain: 0
- Both Co-op: 1.4
- Both Defect: -0.6
- Temptation: 3.4
- Sucker: -1.6

Trade Network Game:
- Buyer: 12, % Elite: 0.67
- Seller: 12, % Elite: 0.67
- Dealer: 0, % Elite: 0.67

FSM:
- States: 16
- Memory: 1

Quotas:
- Buyer: 1
- Seller: 1
### TNG Results Screen

#### Table of Results

<table>
<thead>
<tr>
<th>Generation</th>
<th>Buyer Average</th>
<th>Buyer Minimum</th>
<th>Buyer Maximum</th>
<th>Buyer Std Dev</th>
<th>Seller Average</th>
<th>Seller Minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>+1.1713</td>
<td>-0.1307</td>
<td>+1.3233</td>
<td>+0.3930</td>
<td>+1.2902</td>
<td>+0.2160</td>
</tr>
<tr>
<td>38</td>
<td>+0.8433</td>
<td>+0.7000</td>
<td>+0.9500</td>
<td>+0.0700</td>
<td>+1.3639</td>
<td>+0.9667</td>
</tr>
<tr>
<td>39</td>
<td>+1.3542</td>
<td>+1.3333</td>
<td>+1.3633</td>
<td>+0.0098</td>
<td>+1.4000</td>
<td>+1.4000</td>
</tr>
<tr>
<td>40</td>
<td>+1.1678</td>
<td>+0.7033</td>
<td>+1.2700</td>
<td>+0.1451</td>
<td>+1.3706</td>
<td>+1.0467</td>
</tr>
<tr>
<td>41</td>
<td>+1.3386</td>
<td>+1.2800</td>
<td>+1.3633</td>
<td>+0.0232</td>
<td>+1.4000</td>
<td>+1.4000</td>
</tr>
<tr>
<td>42</td>
<td>+1.3581</td>
<td>+1.3433</td>
<td>+1.3633</td>
<td>+0.0069</td>
<td>+1.4000</td>
<td>+1.4000</td>
</tr>
<tr>
<td>43</td>
<td>+1.2500</td>
<td>+1.1267</td>
<td>+1.2967</td>
<td>+0.0453</td>
<td>+1.3972</td>
<td>+1.3667</td>
</tr>
<tr>
<td>44</td>
<td>+1.3400</td>
<td>+1.3000</td>
<td>+1.3567</td>
<td>+0.0156</td>
<td>+1.4000</td>
<td>+1.4000</td>
</tr>
<tr>
<td>45</td>
<td>+1.2511</td>
<td>+0.6633</td>
<td>+1.3467</td>
<td>+0.1793</td>
<td>+1.3550</td>
<td>+0.8600</td>
</tr>
<tr>
<td>46</td>
<td>+1.1322</td>
<td>-0.1407</td>
<td>+1.2833</td>
<td>+0.3845</td>
<td>+1.3058</td>
<td>+0.3893</td>
</tr>
<tr>
<td>47</td>
<td>+1.3503</td>
<td>+1.3300</td>
<td>+1.3633</td>
<td>+0.0103</td>
<td>+1.4000</td>
<td>+1.4000</td>
</tr>
<tr>
<td>48</td>
<td>+1.3514</td>
<td>+1.3267</td>
<td>+1.3633</td>
<td>+0.0131</td>
<td>+1.4000</td>
<td>+1.4000</td>
</tr>
<tr>
<td>49</td>
<td>+1.0989</td>
<td>-0.1393</td>
<td>+1.3700</td>
<td>+0.5511</td>
<td>+1.1514</td>
<td>+0.4493</td>
</tr>
<tr>
<td>50</td>
<td>+1.3475</td>
<td>+1.3233</td>
<td>+1.3633</td>
<td>+0.0140</td>
<td>+1.4000</td>
<td>+1.4000</td>
</tr>
</tbody>
</table>
TNG Chart Screen

![image of TNG Chart Screen]
TNG Network Animation Screen

- File
- Edit
- View
- Help

50/50 gens

150/150 cycles

Animation: Stop
TNG “Interaction Physics” Screen

- **File**, **Edit**, **View**, and **Help** menu options are available at the top.
- **Run**, **Pause**, and **End** buttons are located at the top right.
- The screen includes options for **50/50 gens** and **150/150 cycles**.
- **Settings**, **Results**, **Chart**, **Animation**, and **Physics** tabs are visible.
- The **Physics** section has options for **Springs** and **Repulsion**.
  - **Springs** include **Latched** and **Recurrent** with **Length** and **Strength** values.
  - **Repulsion** includes **Boundary** and **Trader** with a **Friction** value.
- **Network Settings** include a **Frequency Threshold** with **Latched** and **Transient** values.
Related Online Resources

- ACE/CAS General Software and Toolkits
  https://www2.econ.iastate.edu/tesfatsi/acecode.htm

- ACE/CAS Computational Laboratories
  https://www2.econ.iastate.edu/tesfatsi/acedemos.htm

- Research Area: Development and Use of Computational Laboratories
  https://www2.econ.iastate.edu/tesfatsi/acomplab.htm

- TNG Lab Home Page
  https://www2.econ.iastate.edu/tesfatsi/tnghome.htm