Agent-Oriented Programming: Intro

Presenter:

Leigh Tesfatsion

Professor of Economics and Mathematics

Department of Economics

Iowa State University

Ames, Iowa 50011-1070

http://www.econ.iastate.edu/tesfatsi/

tesfatsi@iastate.edu

Outline

- What is Object-Oriented Programming (OOP)?
- Agent-Oriented Programming (AOP) vs. OOP
- AOP via “Computational Laboratories”
- Example: The Trade Network Game (TNG) Computational Lab

Object-Oriented Programming (OOP)

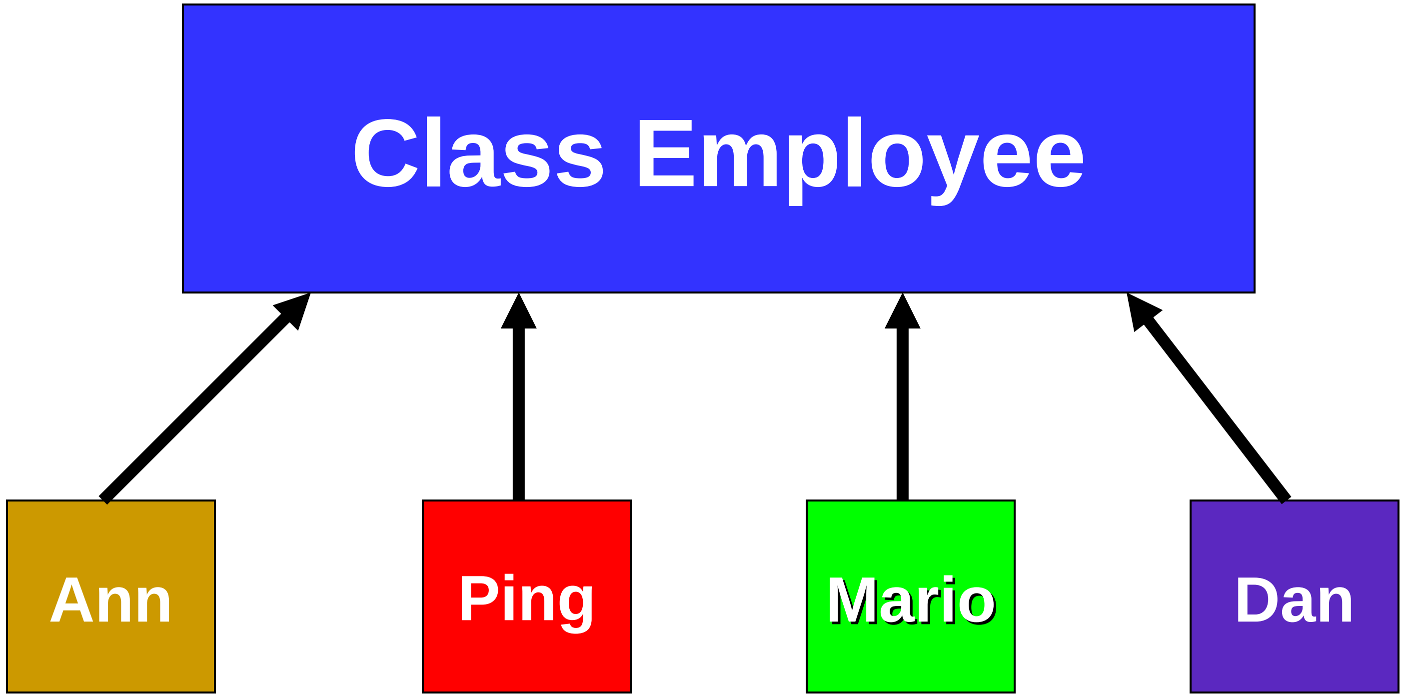
KEY CONCEPTS:

- **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



Employee Objects (Instances of Employee)

Illustration: Employee Class
(See M. Weisfeld book cited on Syllabus)

```
Class EMPLOYEE
{
  Public Access:
   Methods:
      getSocialSecurityNumber( ) ;
      getGender( ) ;
      getDateOfBirth( ) ;
  Private Access:
   Attributes:
      SocialSecurityNumber ;
      Gender ;
      DateOfBirth ;
      Trustworthyness ;
}
```

OOP … Continued

- 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.

OOP … Continued

- 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
(See Matt Weisfeld book cited on Syllabus)

Class **EMPLOYEE**
{
Public Access:

Methods:

getSocialSecurityNumber( ) ;
getGender( ) ;
getDateOfBirth( ) ;

Private Access:

Attributes:

SocialSecurityNumber ;
Gender ;
DateOfBirth ;
Trustworthyness ;
}

Illustration: Payroll Class
(invokes public methods in Employee class)

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

Private Access:
 Attributes:
  CurrentProfits;
  EmployeePayoll;
}
```

OOP … Continued

- ***Encapsulation*** is the process of determining which aspects of a class are not needed by other classes, and hiding these aspects from other classes.
- More precisely, encapsulation is the process of dividing each class of a program 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.
- The 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 that are not present in the superclass.

Class Inheritance: Example

Superclass of Buyer and Seller

TradeBot

attribute Price;
method trade( );

Subclass of TradeBot

Buyer

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

Subclass of TradeBot

Seller

Price = AskPrice ;
trade( ) = sell( ) ;
calculateProfits( ) ;

Composition vs. Inheritance

- 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 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*** just distributed actions.
- According to Jennings, conventional objects encapsulate attributes and methods but not *self*-activation and *localized* action choice.
- See N. R. Jennings, *Artificial Intelligence*, Vol. 17 (2000), pp. 277-296, for an extended discussion of this viewpoint.

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) Laboratory

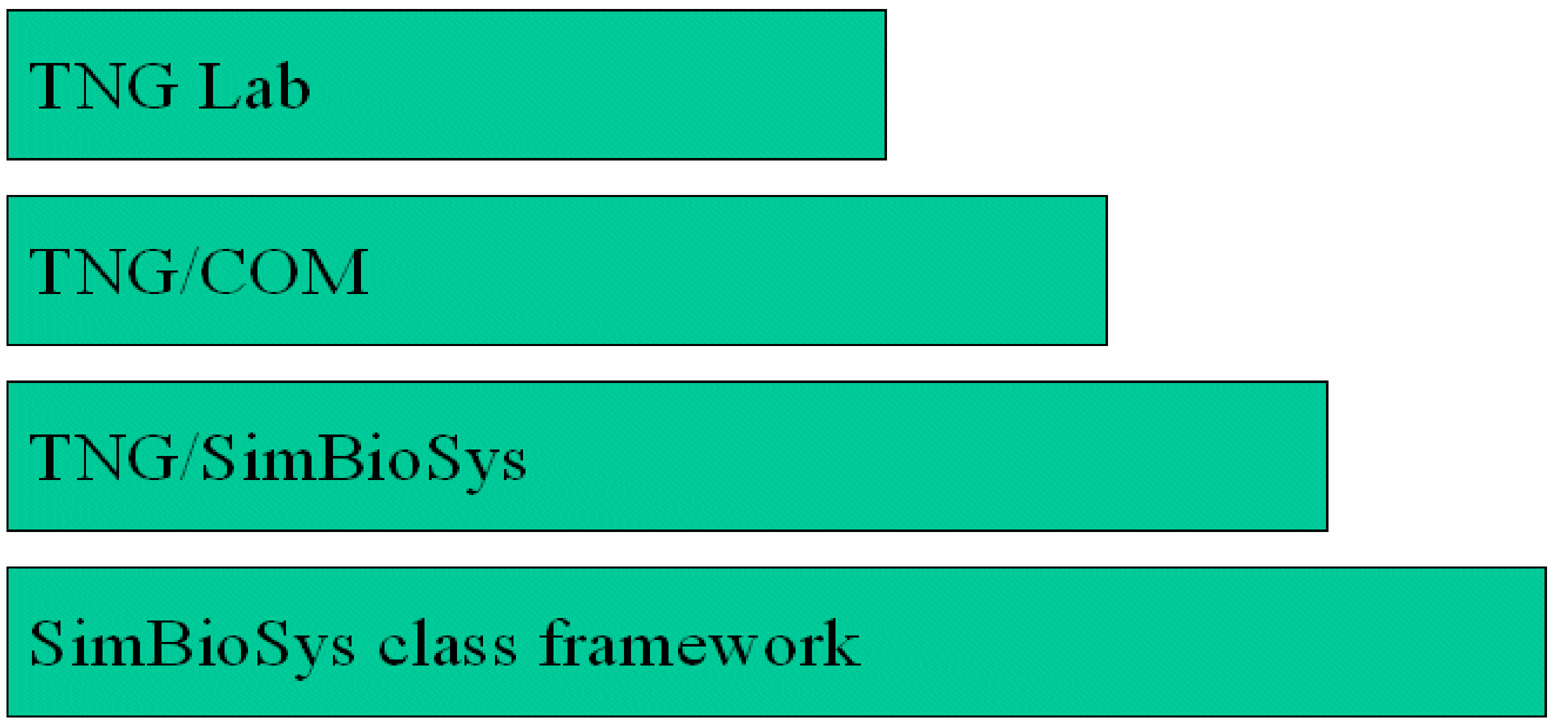
- Evolution of trade networks among strategically interacting traders (buyers, sellers, and dealers)
- Traders are instantiated as “tradebots” (autonomous software entities with internal attributes and methods)
- The tradebots engage in event-driven communication
- The tradebots evolve their trade methods 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 (Zip file includes automatic installation wizard)** www.econ.iastate.edu/ tesfatsi/tnghome.htm

TNG Lab 4-Layer Architecture

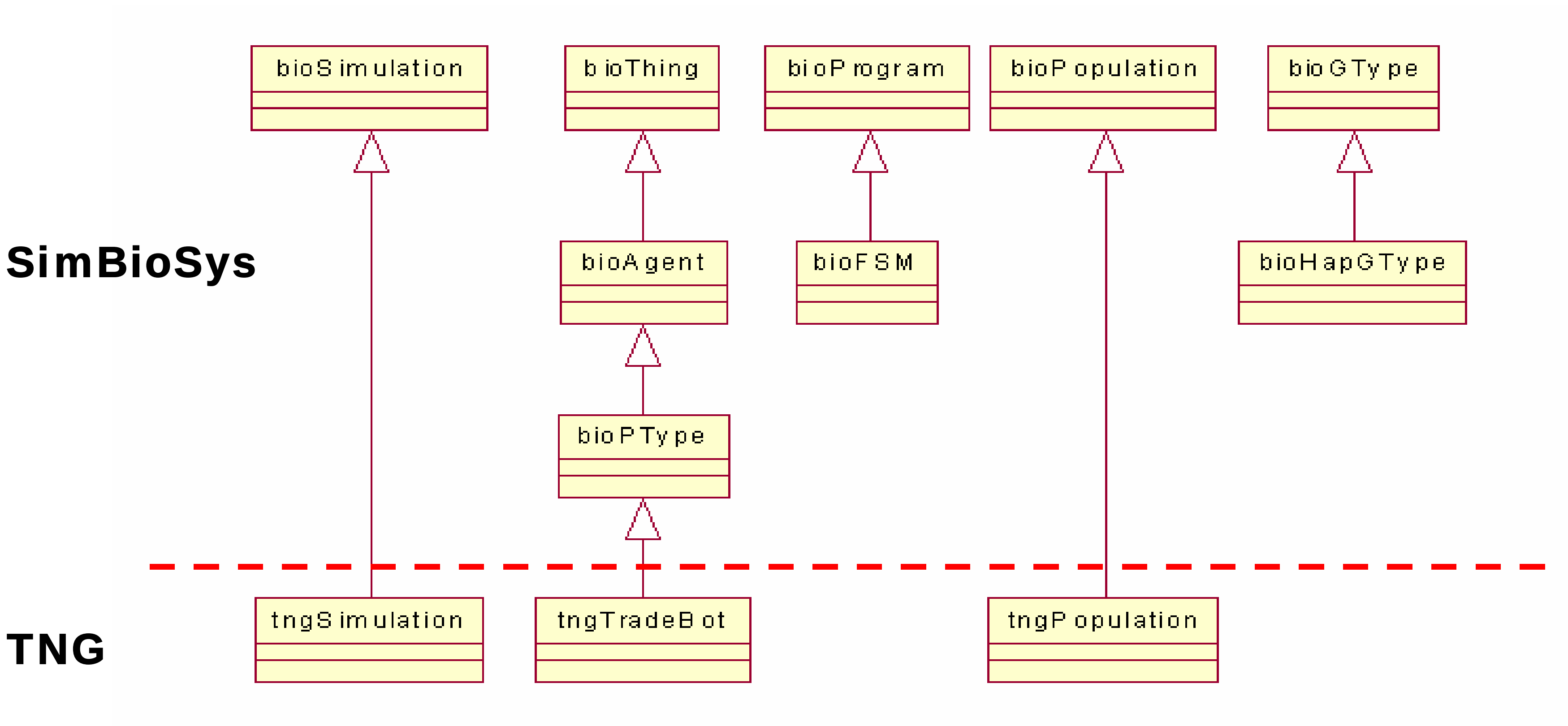
(McFadzean, Stewart, and Tesfatsion, IEEE-TEC, 2001)



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 profits

TNG Flow Diagram

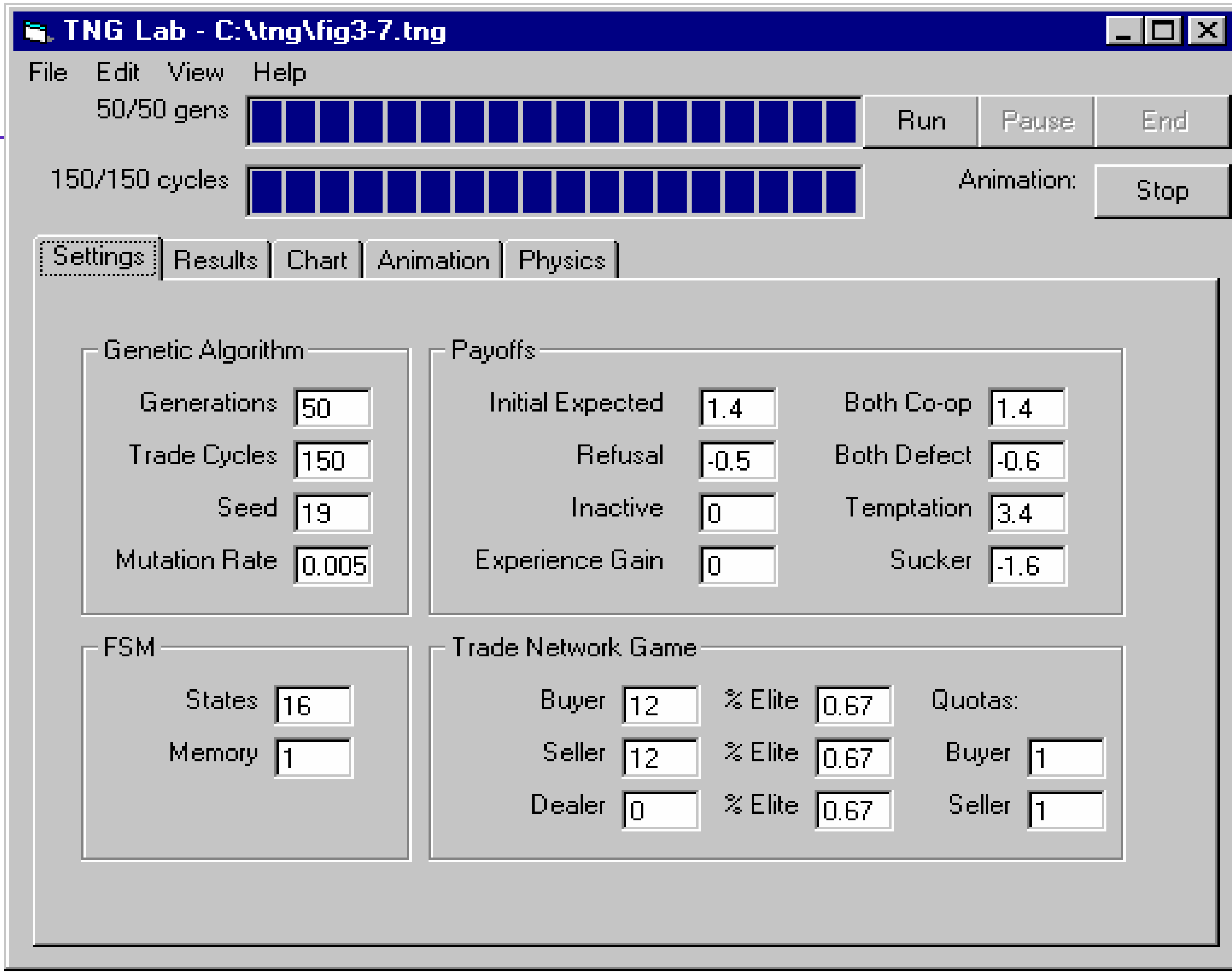
- **INITIALIZATION**
- **LOOP** Through TMax Trade Cycles
  - ***Trade Cycle:***

    Search for Trade Partners;

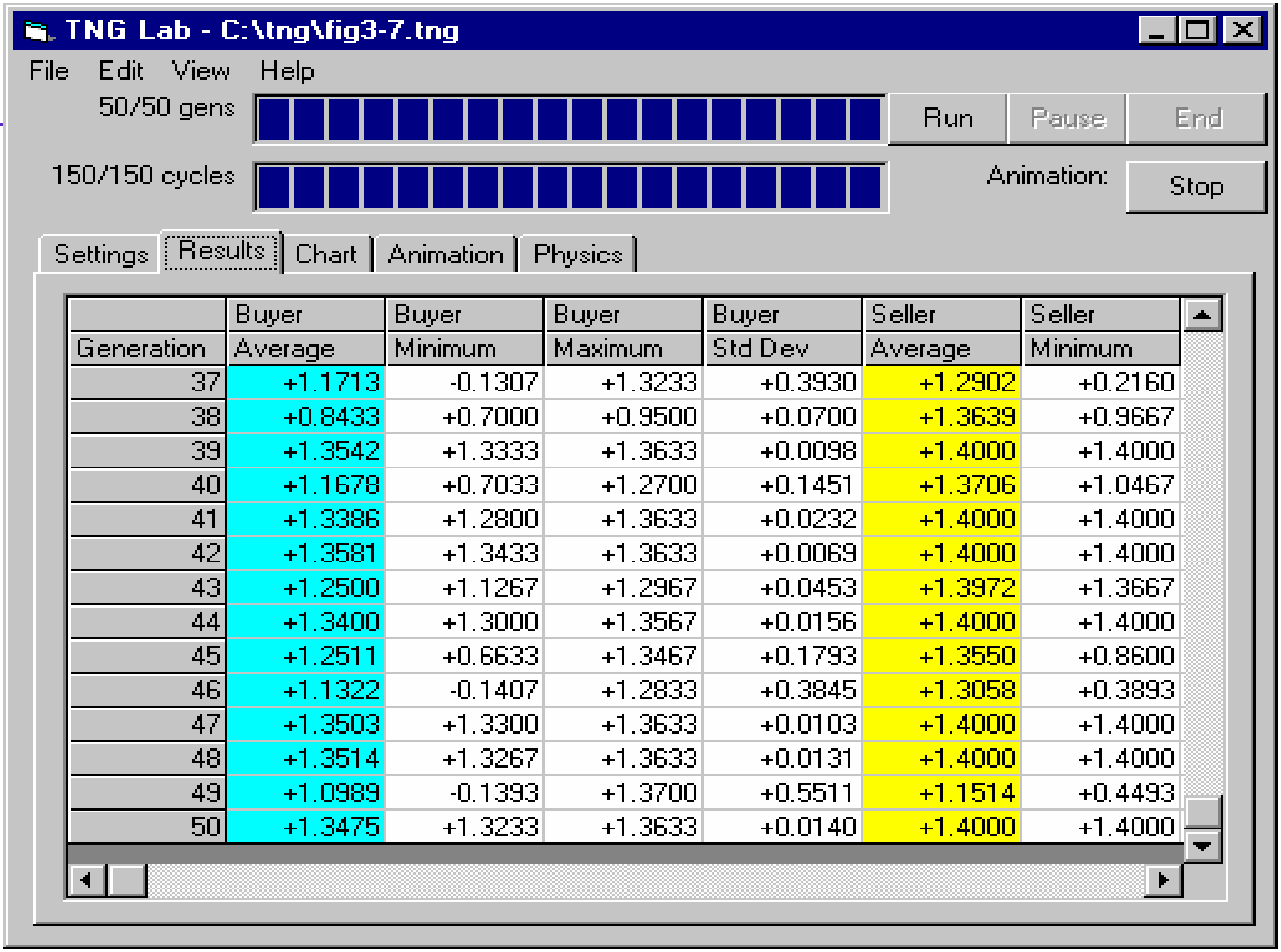
    Interactions with Trade Partners;

    Update Expectations about Trade Partners.
- **EVOLUTION STEP (Update Trade Methods)**
- **LOOP** Through Tmax Trade Cycles **. . .**

TNG Settings Screen

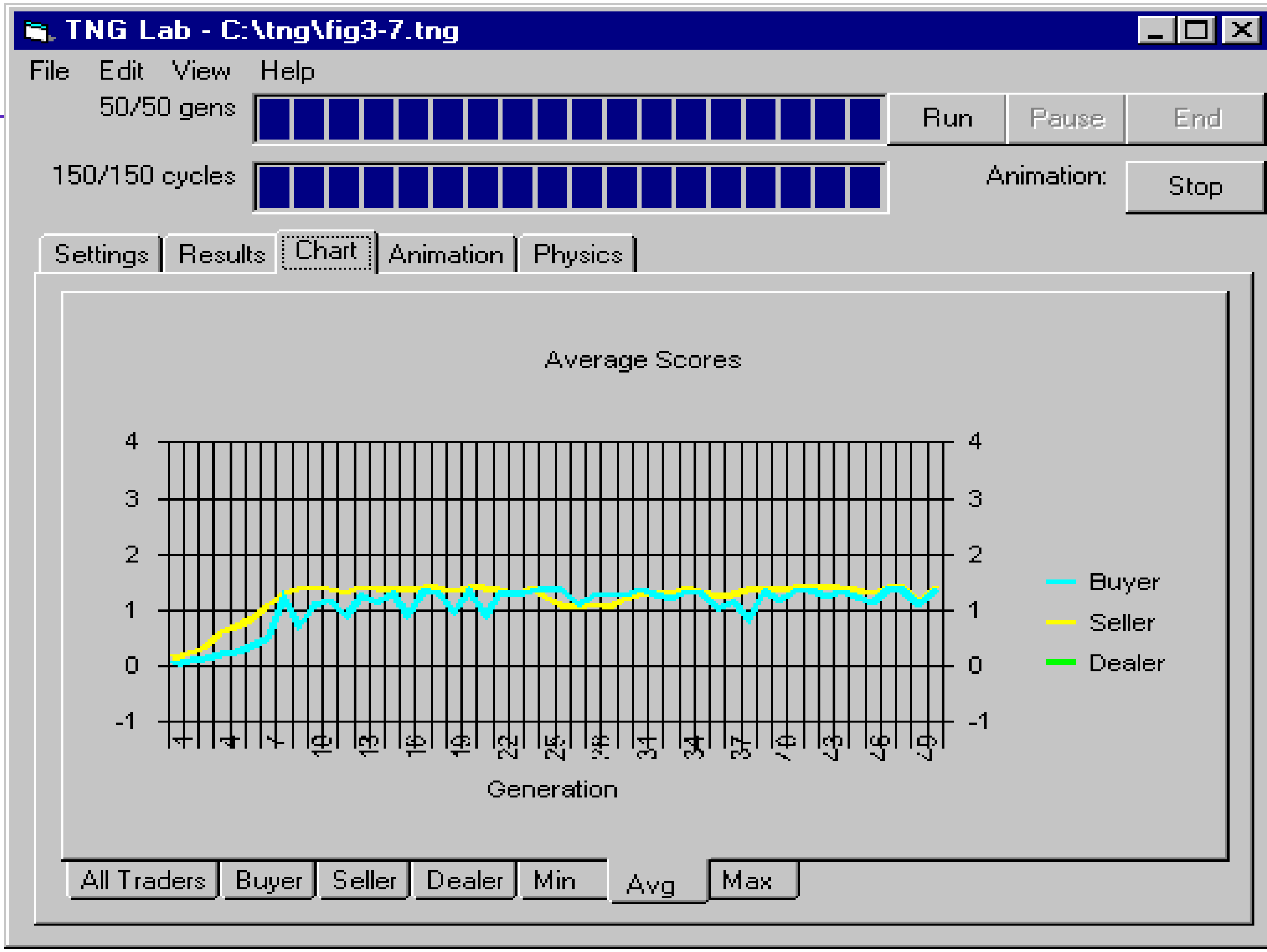


TNG Results Screen

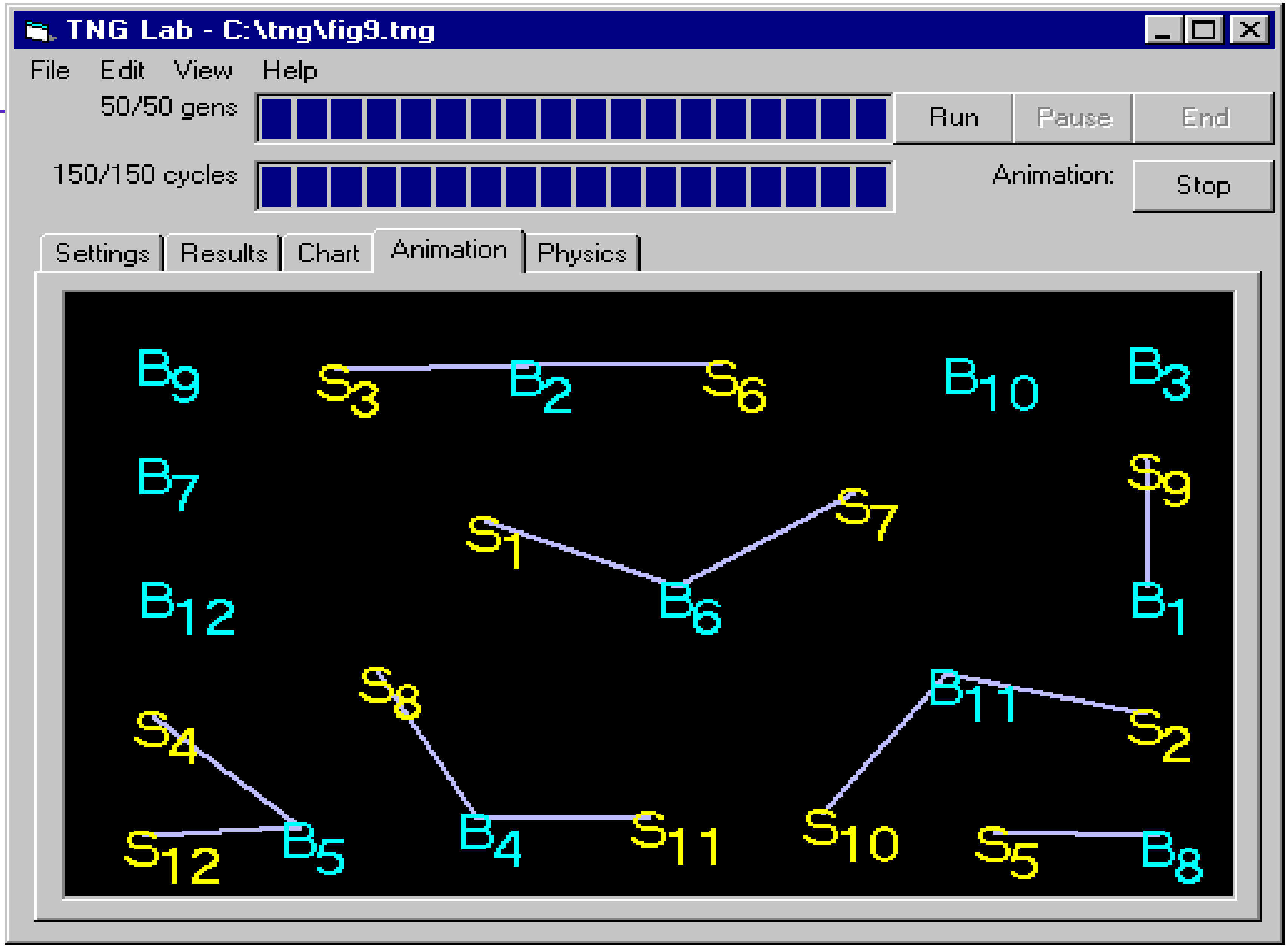


Generation	Buyer Average	Buyer Minimum	Buyer Maximum	Buyer Std Dev	Seller Average	Seller Minimum
37	+1.1713	-0.1307	+1.3233	+0.3930	+1.2902	+0.2160
38	+0.8433	+0.7000	+0.9500	+0.0700	+1.3639	+0.9667
39	+1.3542	+1.3333	+1.3633	+0.0098	+1.4000	+1.4000
40	+1.1678	+0.7033	+1.2700	+0.1451	+1.3706	+1.0467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

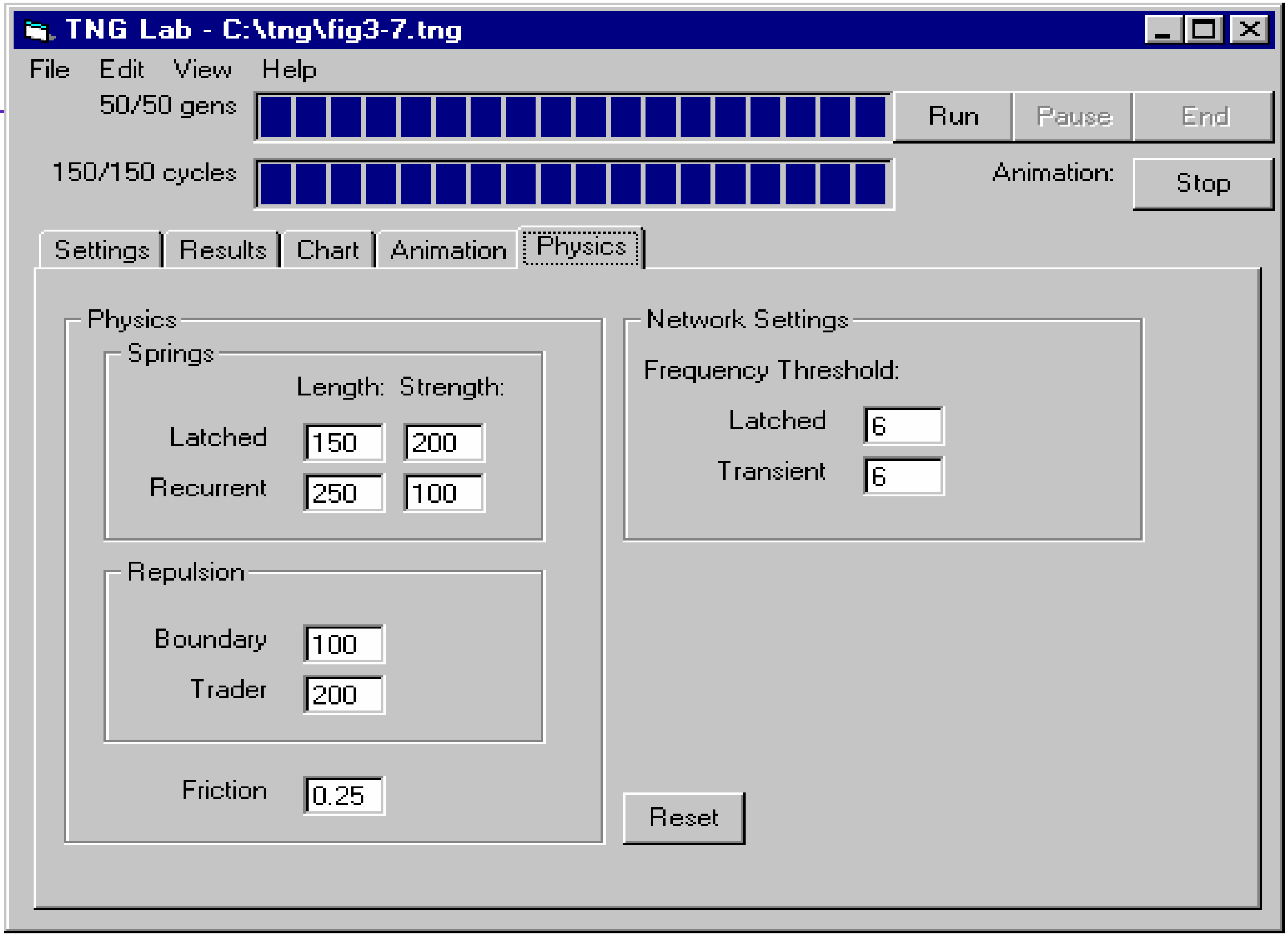
TNG Chart Screen



TNG Network Animation Screen



TNG Physics Screen



Related Online Resources

- ACE/CAS General Software and Toolkits

  www.econ.iastate.edu/tesfatsi/acecode.htm

- ACE/CAS Computational Laboratories

  www.econ.iastate.edu/tesfatsi/acedemos.htm

- Research Area: Development and Use of Computational Laboratories

  www.econ.iastate.edu/tesfatsi/acomplab.htm

- TNG Lab Home Page

  www.econ.iastate.edu/tesfatsi/tnghome.htm