

Agent-based modeling with JABM and JASA

EASSS Summer School 2013

Steve Phelps and Neil Rayner

Centre for Computational Finance and Economic Agents (CCFEA)

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

July 4, 2013

- Introducing Java Auction Simulator API (JASA) and Java Agent Based Modeling (JABM)
- Overview of agent-based modelling and agent-based computational economics
- Modelling learning and adaptation in agent-based models
- Agent-based models as object models
- Configuring agent-based models using dependency-injection

JABM - Java Agent-Based Modelling Toolkit

- Arose out of work in Agent-based Computational Economics (ACE) [Tesfatsion, 2002]
- JASA <http://jasa.sourceforge.net>
- JASA used as the basis of JCAT software for the CAT tournament in TAC [Cai et al., 2009]
- More details at <http://jabm.sourceforge.net/>

Modelling adaptation

JABM is an attempt to build a library of software components that can be used for more general agent-based models, and now serves as the underlying framework used in the implementation of JASA. One of the challenges in building this framework is that, even within the trading-agent research community, there are many different approaches to modelling and analysing agents' decision making;

- 1 adaptation through *evolutionary* processes.
- 2 individual and multi-agent *reinforcement learning*, and
- 3 game-theoretic approximation based on the *empirical game-theory* methodology [Wellman, 2006],

Object-oriented modelling

- agents, events, strategies etc. modelled as Plain Old Java Objects (POJOs) [Parsons et al., 2000]
- Uses dependency-injection [Fowler, 2004] to inject parameters and random-variates.
- Allows us to express the configuration of the model, including its statistical properties, declaratively.
- This allows us to run the same model under different assumptions regarding, e.g. initial conditions and free parameter distributions without making any modifications to the code representing the model itself.

Toolkits for agent-based modelling

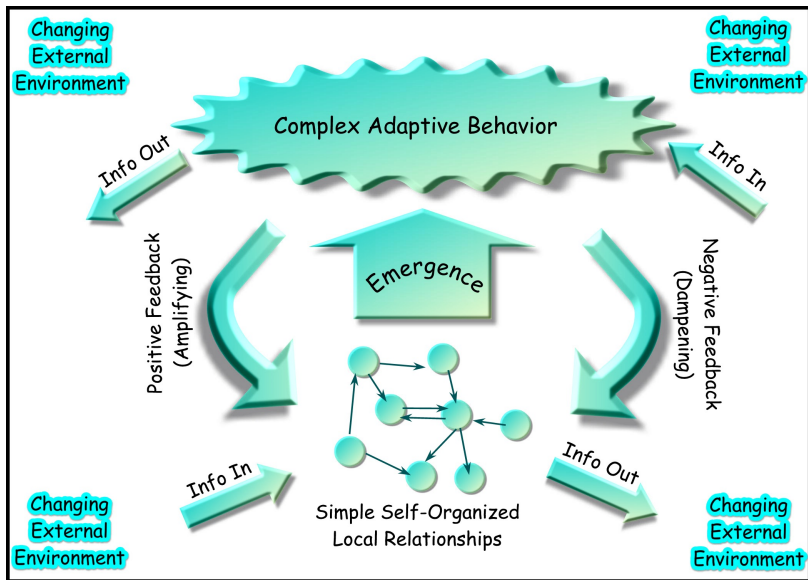
- the Swarm system [Minar et al., 1996]
- Scripting languages: NetLogo [Wilensky and Rand, 2011]
- Repast toolkit [North and Macal, 2005, Dubitzky et al., 2011]
- MASON [Luke, 2005]
- AgentSpring [Chmieliauskas et al., 2012]
- JABM [Phelps, 2012]
- Why Java? Type safety, garbage collection, libraries for numerical methods and high throughput performance [Moreira et al., 2000].

Agent-based economics (ACE)

	Agent-Based Economics	Traditional Economics
Dynamics	Open, dynamic, non-linear systems	Closed, static, linear systems in equilibrium
Agents	Modelled individually; use inductive rules of thumb to make decisions; have incomplete information; are subject to errors and biases; learn and adapt over time	Modelled collectively; use complex deductive calculations to make decisions; have complete information; make no errors and/or no biases; have no need for learning or adaptation
Networks	Explicitly model interactions between individual agents; networks of relationships change over time	Agents interact only indirectly; mean field approach
Emergence	No distinction between micro- and macroeconomics; macro patterns are emergent result of micro-level behaviours and interactions	Micro and macroeconomics remain separate disciplines
Evolution	The evolutionary process of differentiation, selection and amplification provides the system with novelty and is responsible for its growth in order and complexity	No mechanism for endogenously creating novelty, or growth in order and complexity

Beinhocker, E.D., 2007. The Origin Of Wealth: Evolution, Complexity, and the Radical Remaking of Economics, Random House Business.

Complex Adaptive Systems (CAS)



Underlying properties of CAS

- Competition and Cooperation
- Decentralization
- *Dynamic Change* in contrast to static equilibria
- *Emergent Behaviour*

It is the last two features that distinguish a CAS from a MAS.

Emergence

- Components with simple rules give rise to complex behaviour
- This behaviour is self-organising:
- Initially the behaviour appears random but then complex behaviour suddenly “emerges” with no centralised control
- In economics: Adam Smith’s invisible hand.

"... The aggregate motion of the simulated flock is created by a distributed behavioral model much like that at work in a natural flock; the birds choose their own course. Each simulated bird is implemented as an independent actor that navigates according to its local perception of the dynamic environment ... The aggregate motion of the simulated flock is the result of the dense interaction of the relatively simple behaviors of the individual simulated birds."

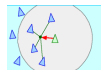
[Reynolds, 1987]

Separation



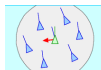
steer to avoid crowding local agents

Cohesion



steer to move toward the average local position

Alignment



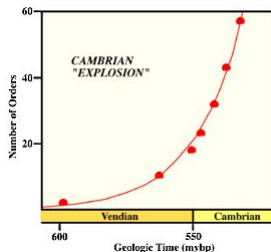
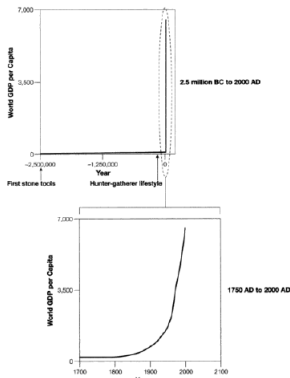
steer towards the average local heading

Empirical properties of Complex Adaptive Systems

- Self-similarity: scaling in size distributions and autocorrelations
- Example: Cauchy Distribution
- <http://mathworld.wolfram.com/CauchyDistribution.html>
- Extreme events have non-negligible probability
- Do not always have well-defined mean
- Therefore estimating “expected” value from samples can be misleading
- No “typical value” - scale free

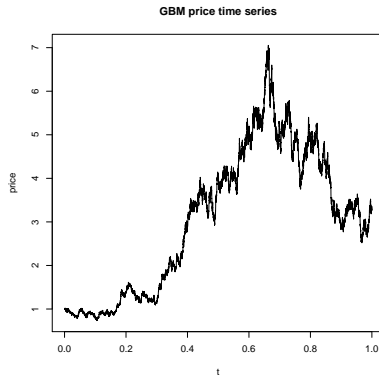
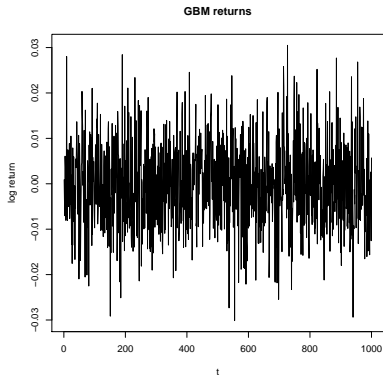
Markets as Complex Adaptive Systems

The Explosive Growth in Human Wealth

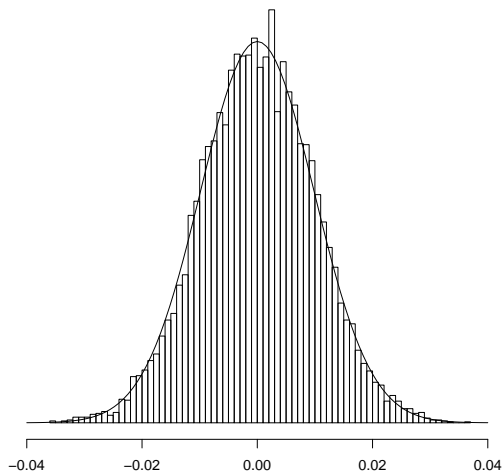


[DeLong, 1998, Kazlev, 2002]

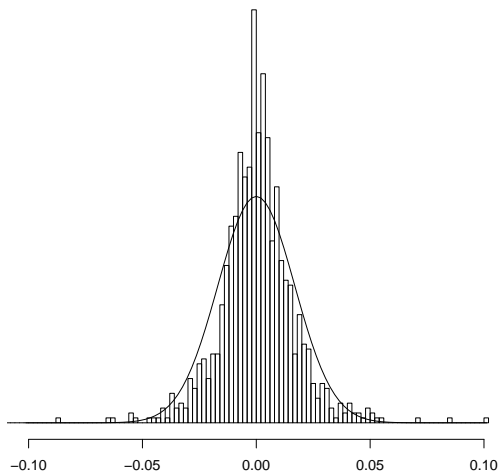
Random walk model of asset prices



Distribution of GBM returns



In reality: returns for MSFT Jan 2010 to Oct 2012

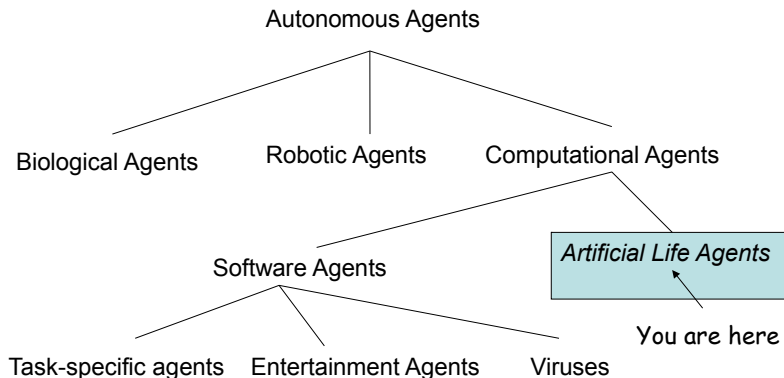


Modelling components

- Strategy
- Artifact
- Agent
- Population
- Systems
- Type
- Variety
- Interaction pattern
- Space (physical)
- Space (conceptual)
- Selection
- Success criteria or performance measure

[Axelrod and Cohen, 2001]

A Taxonomy for Autonomous Agents

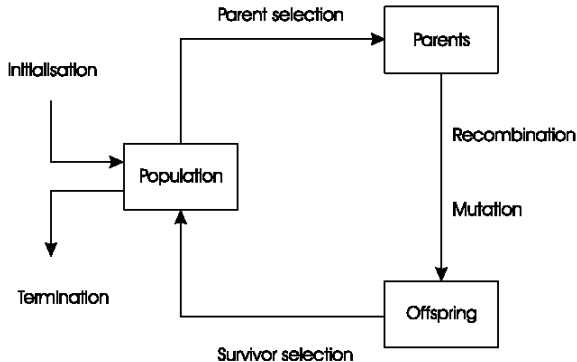


[Franklin and Graesser, 1996]

Adaptation in CAS and ACE

A key issue for research in this area is that the way that agents interact and learn can be critical in explaining certain phenomena. For example, [LeBaron and Yamamoto, 2007] introduce a model of financial markets which demonstrates that scaling behaviour in financial time series data can only be replicated when agents imitate each others' strategies. When their model is analysed under a treatment in which learning does not occur, the corresponding long-memory properties disappear. Performing such analyses requires that we are able to easily reconfigure the way in which agents interact and learn.

General Scheme of EAs



[Eiben and Smith, 2007]

Pseudo-code for typical EA

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

What are the different types of EAs

- Historically different flavours of EAs have been associated with different representations
 - Binary strings : Genetic Algorithms
 - Real-valued vectors : Evolution Strategies
 - Finite state Machines: Evolutionary Programming
 - LISP trees: Genetic Programming
- These differences are largely irrelevant, best strategy
 - choose representation to suit problem
 - choose variation operators to suit representation
- Selection operators only use fitness and so are independent of representation

n-armed bandit problem



n-armed bandit $n=4$

- n available actions
- Spend money to pull a lever (take an action)
- Stationary environment: payout (aka reward) is stochastic function of the action
- What is the best strategy to maximise our expected profit?

[Sutton and Barto, 1998]

Formal definitions

- Choose repeatedly from one of n actions; each choice is called a **play**
- After each play a_t , you get a reward r_t where $E \{r_t \mid a_t\} = Q^*(a_t)$

These are unknown **action values**

Distribution of r_t depends only on a_t

- Objective is to maximize the reward in the long term, e.g., over 1000 plays

To solve the n -armed bandit problem, you must **explore** a variety of actions and **exploit** the best of them

N-armed bandit problems in Finance & Economics

- Choosing a position:
 - Action $a=1$: take a short position at time t
 - Action $a=2$: take a long position at time t

["Stock Trading with Recurrent Reinforcement Learning" Molina 2006](#)
- Choosing a price:
 - Discretize prices:
 - Action $a=1$: Bid \$10
 - Action $a=2$: Bid \$15
 - Action $a=3$: Bid \$20

["Market Power and Efficiency in a Computational Electricity Market with Discriminatory Double-Auction Pricing" Nicolaisen et al. 2000](#)
- Choosing an exchange (the dark-pool problem):

["Censored Exploration and the Dark Pool Problem" K. Ganchev et al. 2009](#)

Estimating rewards

- Maintain an estimate of the expected reward to each action which is updated as we go along
- If our estimates were hypothetically correct then we should always pull the lever with greatest expected reward – this is called the *greedy* action
- However, if our estimates are not correct then this is not always the best thing to do

Exploration verses Exploitation

- We can help to maximise our long-term reward by forming correct estimates
- We can only refine estimates through experience (sampling)
- Therefore we need to make mistakes:
 - We need to explore the consequences of alternative actions
 - Especially important in non-stationary environments (true reward varies over time or in response to actions)
- On the other hand, we should also exploit current knowledge by playing the greedy action
- No single way of balancing exploration verses exploitation

Q values

- Value of an action is true expected reward of an action a , denoted $Q^*(a)$
- Estimate of an action denoted $Q_t(a)$
- Use sample mean to update estimates:

$$Q_t(a) = \frac{r_1 + r_2 + \dots + r_{ka}}{k_a}$$

Greedy action selection

- The action selection rule specifies the action to take as a function of the Q values
- Greedy action selection, select (one of) the greedy actions a_t^* :

$$Q_{t-1}(a_t^*) = \max_a Q_{t-1}(a)$$

Epsilon-greedy action selection

- For small $\varepsilon \in [0,1]$

Random variate $\eta_t \sim U(0,1)$

Exploit with $P(1-\varepsilon)$ $\eta_t > \varepsilon \Rightarrow \text{choose}(a_t^*)$

Explore with $P(\varepsilon)$ $\eta_t \leq \varepsilon \Rightarrow \text{choose}(a \sim U(1, n))$

How greedy?

- The optimal setting for ϵ is highly problem-specific
- Two key factors:
 - Is the problem stationary?
 - Variance of rewards:
 - Volatility or risk plays a key role

Softmax Action Selection

- Softmax action selection methods grade action probs. by estimated values.
- The most common softmax uses a Gibbs, or Boltzmann, distribution:

Choose action a on play t with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^n e^{Q_t(b)/\tau}}$$

where τ is the

“computational temperature”

Incremental update of estimates

Rather than keeping an array of all previous reward values in order to calculate action values:

$$Q_i(a) = \frac{r_1 + r_2 + \dots + r_{ka}}{k_a}$$

Instead we can update our estimates incrementally:

$$\begin{aligned} Q_{k+1} &= \frac{1}{k+1} \sum_{i=1}^{k+1} r_i \\ &= \frac{1}{k+1} \left[r_{k+1} + \sum_{i=1}^k r_i \right] \\ &= \frac{1}{k+1} [r_{k+1} + kQ_k + Q_k - Q_k] \\ &= \frac{1}{k+1} [r_{k+1} + (k+1)Q_k - Q_k] \\ &= Q_k + \frac{1}{k+1} [r_{k+1} - Q_k] \end{aligned}$$

RL as error reduction

$$Q_{k+1} = Q_k + \frac{1}{k+1} [r_{k+1} - Q_k]$$

`newEstimate = oldEstimate + stepSize * (target - oldEstimate)`

error

Constant step size: recency

$$Q_{k+1} = Q_k + \alpha[r_{k+1} - Q_k]$$

Where alpha is a **constant** in [0, 1]

This gives us a *weighted* average over previous rewards with more weight being given to recent rewards:

$$\begin{aligned} Q_k &= Q_k + \alpha[r_k - Q_{k-1}] \\ &= \alpha r_k + (1 - \alpha) Q_{k-1} \\ &= \alpha r_k + (1 - \alpha) \alpha r_{k-1} + (1 - \alpha)^2 Q_{k-2} \\ &= \alpha r_k + (1 - \alpha) \alpha r_{k-1} + (1 - \alpha)^2 \alpha r_{k-2} + \\ &\quad \dots + (1 - \alpha)^{k-1} \alpha r_1 + (1 - \alpha)^k Q_0 \\ &= (1 - \alpha)^k Q_0 + \sum_{i=1}^k \alpha (1 - \alpha)^{k-i} r_i \end{aligned}$$

A note on randomness

- Note that action selection is *probabilistic*
- This means that any agent-based model employing RL is a stochastic model
- Therefore important to use Monte-Carlo methods to assess the results of simulation (multiple-runs, confidence-intervals etc.)
- Random action selection is especially important in strategic contexts (when our environment consists of other agents)....

Multi-agent interactions

In many financial and economic applications, the outcome of our action depends not only on our action, but also the actions of other agents (the joint set of actions), eg:

a 3-armed bandit problem with two agents A and B:

	B_1	B_2	B_3
A_1	0,0	-1,+1	+1,-1
A_2	+1,-1	0,0	-1,+1
A_3	-1,+1	+1,-1	0,0

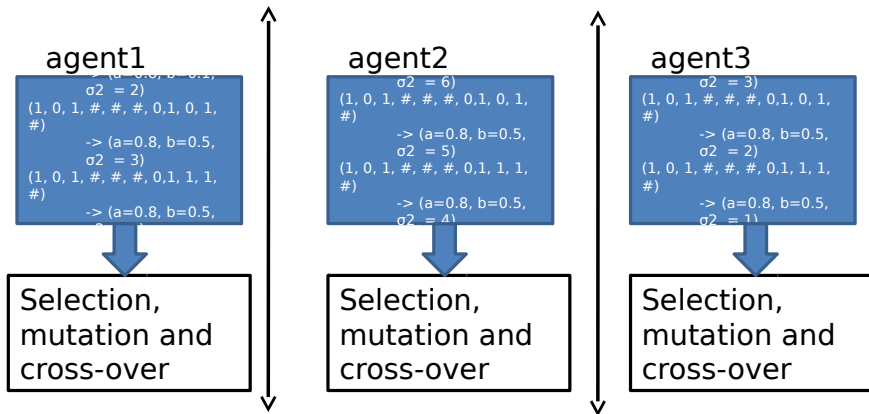
Multi-agent reinforcement-learning

Desiderata for reinforcement algorithms in multi-agent contexts [Shoham and Leyton-brown, 2010, ch. 7]:

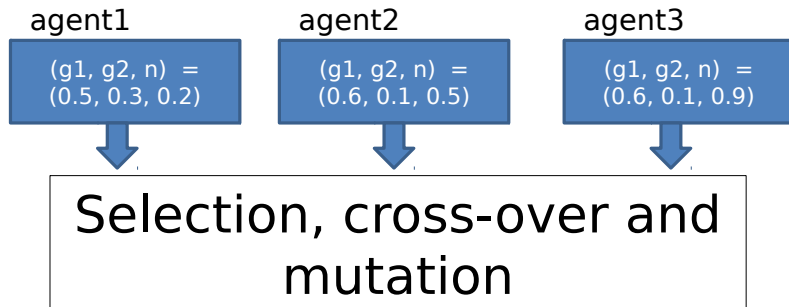
- A learning algorithm is *safe* if it is able to guarantee the security value of the game (the minimum payoff that can be obtained regardless of the strategies chosen by other agents).
- A learning algorithm is *rational* if it converges to a best-response when its opponents(s) adopt static strategies.
- *no-regret* specifies that the learning-algorithm is able to gain a superior expected payoff compared with what could have obtained by following any one of its pure strategies.

The development of multi-agent learning algorithms that satisfy one or more of these criteria is an active field of research within the agents community [Bowling, 2005, Busoniu et al., 2008, Kaisers and Tuyls, 2010]

Using a GA to model individual-based learning



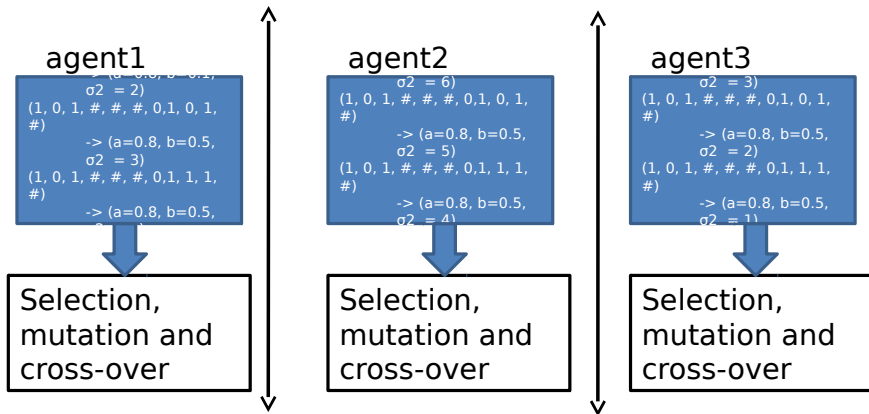
Using a GA to model social learning



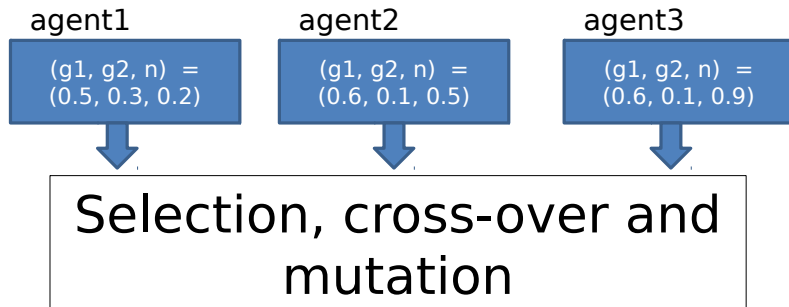
Social learning and co-evolution

- Co-evolutionary algorithms can be interpreted as models of *social* learning [Boyd and Richerson, 1988, Vriend, 2000, Skyrms, 2005, Richerson and Boyd, 2006]
- interpretation: agents have a probability of switching to another agent's strategy if it is observed to obtain a higher payoff than their current strategy.
- learning as evolution: behaviour is learnt through observation resulting in *copying* (reproduction) with *error* (mutation).
- The effect of social learning has been explored in several recent agent-based models of trading behaviour in financial markets [Eguíluz and Zimmermann, 2000, LeBaron and Yamamoto, 2007, Rayner et al., 2012].
- Issues [Ficici and Pollack, 1998, Ficici and Pollack, 2000, Ficici et al., 2005]

Using a GA to model individual-based learning



Using a GA to model social learning



Empirical Game Theory

- Empirical Game-Theory uses a combination of simulation *and* game-theoretic analysis
[Walsh et al., 2002, Wellman, 2006, Phelps et al., 2010].
- Makes use of a *heuristic* payoff matrix
- “Empirical” because we use empirical methods to estimate payoffs
- Simplifying assumptions: game is symmetric

Heuristic Payoff Matrix

- For a game with j strategies, we represent entries in the payoff matrix as vectors of the form $\mathbf{p} = (p_1, \dots, p_j)$ where p_i specifies the number of agents who are playing the i^{th} strategy.
- Each entry $\mathbf{p} \in P$ is mapped onto an outcome vector $\mathbf{q} \in Q$ of the form $\mathbf{q} = (q_1, \dots, q_j)$ where q_i specifies the expected payoff to the i^{th} strategy.
- For a game with n agents, the number of entries in the payoff matrix is given by $s = \frac{(n+j-1)!}{n!(j-1)!}$.

Example Heuristic Payoff Matrix

$n(T4T)$	$n(S)$	$n(D)$	T4T	S	D
0	0	3			0.0
0	1	2		-0.03	0.015
1	0	2	-0.006		0.03
0	2	1		0.15	0.03
1	1	1	0.26	0.28	0.04
2	0	1	0.45		0.06
0	3	0		0.9	
1	2	0	0.9	0.9	
2	1	0	0.9	0.9	
3	0	0	0.9		

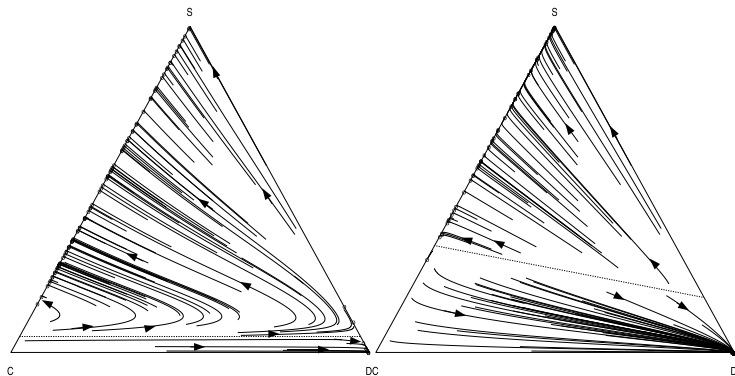
Heuristic payoff matrix for 3 agents and 3 strategies

Evolutionary game-theory

In an evolutionary game-theoretic model, pairs of agents are chosen at random from a very large population. By assuming an infinite population and no mutation we can specify the dynamics of the system using a simple ordinary differential equation. The standard approach is to use the replicator dynamics equation [Weibull, 1997] to model how the frequency of each strategy in the larger population changes over time:

$$\dot{m}_i = [u(e_i, \mathbf{m}) - u(\mathbf{m}, \mathbf{m})] m_i \quad (1)$$

Dynamics of learning



Direction field for $n = 10$ agents and $N = 100$ pairwise interactions per generation (left) compared with $N = 13$ (right). C denotes unconditional altruists, D unconditional defectors and S discriminators who cooperate in the first round. Each line represents a trajectory whose termination is represented by an open circle. The arrows show the direction of change.

Agent interaction

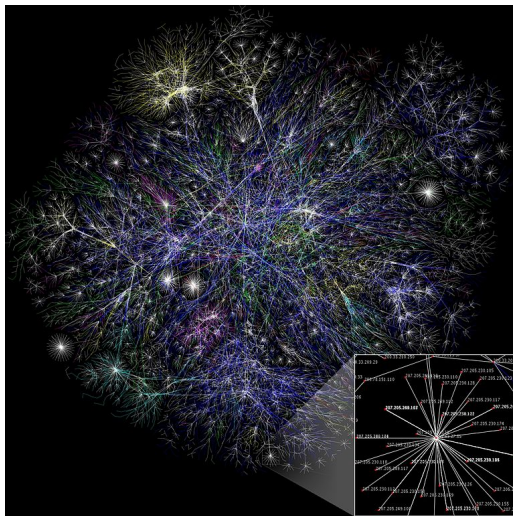
- Random pair-wise interaction (mean-field)
- Networks [Newman, 2010]
 - ▶ Exogenous
 - ▶ Endogenous
 - ▶ Adaptive

Networked models

We know from every-day experience that interactions between agents have more *structure* than this. For example:

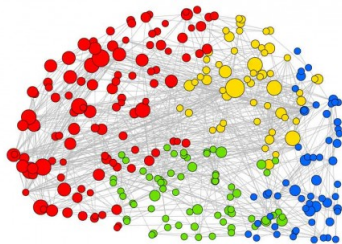
- More likely to interact with friends than strangers
- More likely to purchase from known suppliers
- Banks lend only to certain other banks
- Not everybody has the same level of access to a financial exchange
- Some nodes on the internet have better connections (latency or bandwidth) to other nodes

Example - Structure of the internet



<http://www.opte.org/maps/>

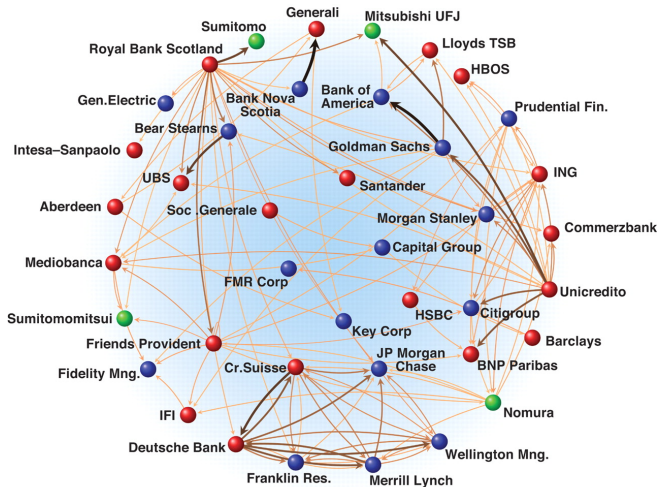
Example - The human brain



<http://www.cam.ac.uk/research/news/wiring-the-brain/>

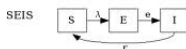
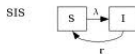
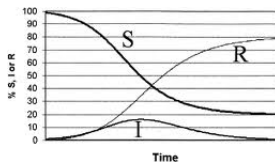
http://www.youtube.com/watch?v=f3P15X_62xQ

Example - financial interdependency network



[Schweitzer et al., 2009]

Spatial contagion

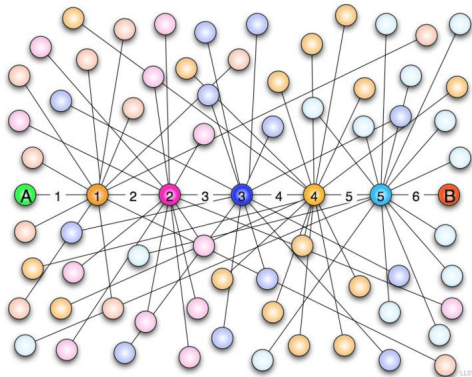


<http://www.youtube.com/watch?v=tvgrpHP6voBc>

Networked contagion

- Biological epidemic:
<http://www.youtube.com/watch?v=rzhKyD19ZEY>
- Financial contagion [Cont and Moussa, 2009]

Small worlds



- Empirically validated [Travers and Milgram, 1969]
- See animation

Clustering

- What role does the network play? Can we explain this using a network-less model?
- Suppose everyone has on average 100 friends
- For n degrees of separation I can reach 100^n people. For $n = 5$ I can reach approx 9 billion.
- Why is this unrealistic?

Definition of a network

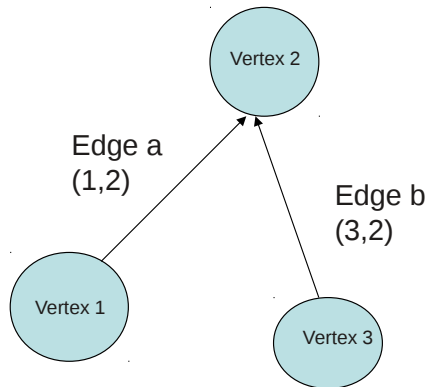
- Networks are also called *graphs*.
- In computer-science see graph theory.
- A graph consists of a set of *vertices* (also called *nodes*) and a set of *edges* (also called links).
- Edges connect pairs of vertices (however, not every pair of vertices is connected).
- Formally a graph is a pair (V, E) where V is a set of vertices and E is a set of edges $E \subseteq \{(u, v) | u, v \in V\}$

Simple example

vertices: $V = \{1, 2, 3\}$ edges: $E = \{a, b\}$

$a = (1, 2)$

$b = (3, 2)$



- Networks are present in many different kinds of system in economics, computer science and biology
- Are there common properties across these diverse systems?
- How can we analyse these properties?
- *Network science* attempts to answer these questions

Network metrics

- Network size: the total number of vertices N
- Maximum possible number of edges: $\frac{N(N-1)}{2} \approx \frac{N^2}{2}$
- Distance between vertices u and v : either number of edges on the shortest path from u to v , or ∞ if there is no path.
- Diameter of the network
 - ▶ worst-case diameter: largest distance between any pair of nodes
 - ▶ average-case diameter: average distance of all pairs of nodes
- If the distance between all pairs is finite, we say the network is *connected*; otherwise it has multiple *components*.
- The *degree* of vertex v is the number of edges connected to v .
- The *degree distribution* is the statistical distribution of the degrees of all vertices in the network.

More network metrics: clustering coefficient

- The clustering coefficient measures the proportion of neighbours of a node that are connected with *each other*
- Define the neighbourhood of node i as

$$N_i = \{v_j : (i, j) \in E \wedge (j, i) \in E\}$$

- The local clustering coefficient of node i is

$$C_i = \frac{|\{(j, k) : v_j, v_k \in N_i, (j, k) \in E\}|}{k_i(k_i - 1)}$$

- Global clustering coefficient of the entire network is

$$\bar{C} = \frac{1}{n} \sum_{i=1}^n C_i \quad (2)$$

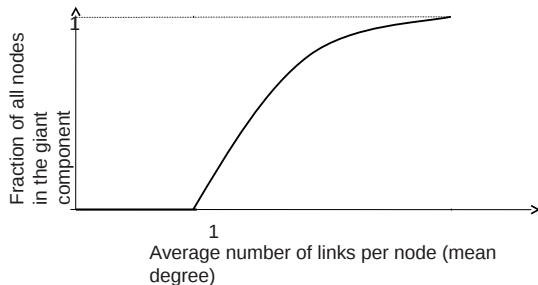
Natural networks

Empirical networks typically have:

- Few connected components (often only 1 or small number given the network size).
- Small diameter
 - ▶ often a constant independent of network size
 - ▶ or perhaps growing only very slowly with network size
 - ▶ typically look at average; exclude infinite distances
- A high degree of edge clustering
 - ▶ when compared with a random network
 - ▶ despite having small diameter
- A heavy-tailed degree distribution
 - ▶ small number of high-degree vertices (hubs)
 - ▶ often follows a *power law*

Non-linearity: phase transitions

<http://ccl.northwestern.edu/netlogo/models/GiantComponent>



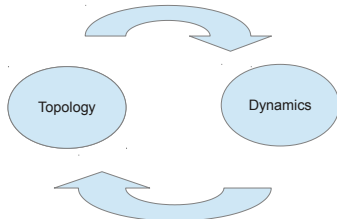
Network Topology (*Statics*)

- Looks at *structural properties* of the network
 - ▶ network metrics such as: size, diameter, connectivity, degree distribution
 - ▶ these metrics define the *topology* of the network
- Structure can reveal:
 - ▶ community
 - ▶ the important nodes
 - ▶ robustness
 - ▶ constraints on dynamics
- Does not analyse what actually occurs on the network

- What happens on networks, eg:
 - ▶ spread of disease over social network
 - ▶ mapping spread of a fad
 - ▶ computation in the brain
 - ▶ traffic flows on the Internet
 - ▶ asset or debt flowing in an economic network
- Statics and dynamics are often coupled in a feedback loop:
 - ▶ distribution of wealth depends on network topology
 - ▶ in an epidemic, the rate of infection (dynamics) depends on the network connectivity (statics)

Dynamic Networks

In practice the network topology is rarely static over time
[Kossinets and Watts, 2006] [Zimmermann et al., 2000]
[Gross and Sayama, 2009].



Models of network formation

- Real-world networks are the result of complex underlying process.
- Are there simple statistical models that capture commonalities across lots of different networks?
- Analogous to random number generation, is there a useful abstraction to generate random networks?

Random graphs

- Repeatedly connect pairs of nodes randomly chosen at uniform
- Analysed mathematically by [Erdos and Rényi, 1960]
- Random graphs are often used as a null model.
- Structural properties:
 - ▶ few components and small diameter ✓
 - ▶ heavy-tailed degree distributions ×
 - ▶ high clustering coefficient ×

- [Watts and Strogatz, 1998]
- http://en.wikipedia.org/wiki/Watts_and_Strogatz_model
- Structural properties:
 - ▶ few components and small diameter ✓
 - ▶ heavy-tailed degree distributions ×
 - ▶ high clustering coefficient ✓

Preferential attachment

- [Albert and Barabasi, 2002]
- Probability of connecting a new node to existing node i :
$$p_i = \frac{k_i}{\sum_j k_j}$$
- Structural properties:
 - ▶ few components and small diameter ✓
 - ▶ heavy-tailed degree distributions ✓
 - ▶ high clustering coefficient ✕

Dependency injection

This section describes how several design problems that arise in implementing agent-based models in the form of simulation software are naturally solved by a software engineering design pattern [Gamma et al., 1995] called “dependency injection” [Fowler, 2004, Prasanna, 2009] which promotes “separation of concerns” between the configuration of an object model and the implementation thereof.

Dependency-injection is particularly attractive for experiments involving Monte-Carlo simulation because in a typical simulation experiment we execute the model very many times, which presents some subtle design issues. On the one hand, each run of the simulation requires that we initialise the agents in our model afresh, drawing new values of random variables independently from previous runs. On the other hand, however, it is important that some components of our simulation retain state across different runs: for example, we may want to collect summary statistics on certain dependent variables in our model in order to estimate their expected value, and the Pseudo-Random Number Generator (PRNG) itself must have persistent state across runs in order to prevent spurious correlation in random variates.

Example model - The El Farol Bar Problem



- Introduced by [Arthur, 1994] to motivate *adaptive* expectations [Lo, 2005] over rational expectations.
- $N = 100$ people decide whether or not to go the El Farol Bar each week.
- No fun if overcrowded; ie number attending > 60

Rational Expectations

- Optimal decision is a function of expectations
- What is the “rational” expectation?
- Paradox; what is the outcome if
 - all believe that few will go?
 - all believe that many will go?
- Expectations are self-denying
- Can we design a model based on evolution or learning?

Induction

- Deductive reasoning
 - Use logical argument to derive new knowledge from existing facts.
- Inductive reasoning
 - forming generalisations from a finite set of empirical observations.
- Both types of reasoning are used in Artificial Intelligence.
- Induction is not “rational”, eg:
 - Observation:
 - All of the swans we have seen are white. TRUE
 - Conclusion:
 - All swans are white. FALSE

Inductive Expectations

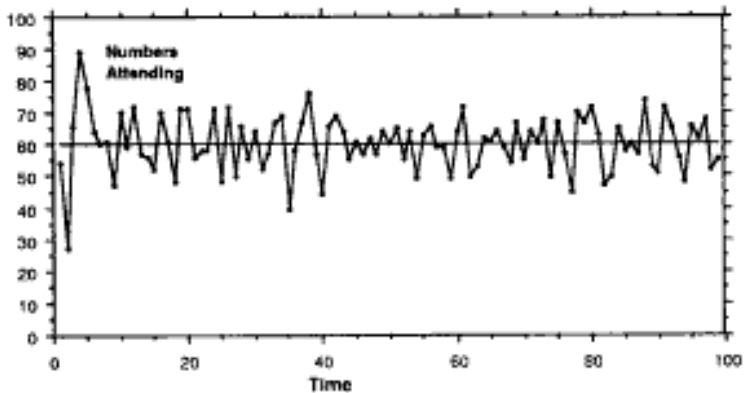
- Use heuristic rules of thumb to form expectations based on historical time series data, eg suppose history of attendance on a weekly basis is:
 - 44, 78, 56, 15, 23, 67, 84
 - 34, 45, 76, 40, 56, 22, 35
- Then choose amongst the following heuristics..

Example heuristics for predicting attendance

1. The same as last week's
 2. A mirror image around 50 of last week's
 3. An average of the last four weeks
 4. The trend in last 8 weeks, bounded by 0,100
 5. The same as 2 weeks ago
 6. The same as 5 weeks ago
- Etc..

Heuristics as species

- We can think of these heuristics as strategies or actions
- Apply reinforcement learning (RL) or evolutionary models
- Eg each predictor is an “arm” in an n-armed bandit problem
- Alternatively think of each predictor as being coded for by a genome in a GA population
- Use forecast error (eg mean-squared error) as reward or fitness function
- However, important difference from RL is that we can assess the fitness of all rules even if they were not actually used
 - No need to explore – always play the greedy action (the rule with the smallest error)



Dynamic Equilibrium

- “Where cycle-detector predictors are present, cycles are quickly ‘arbitraged’ away so there are no persistent cycles.”
- “Mean attendance converges always to 60”
- “The predictors self-organize into an equilibrium pattern or ‘ecology’ in which, on average 40 percent are forecasting above 60, 60 percent below 60”.
- This is a *dynamic equilibrium*.

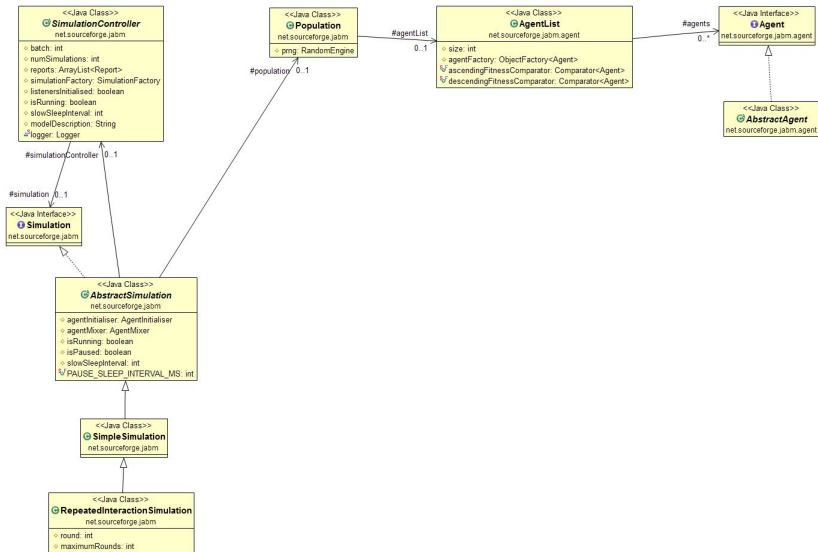
The Adaptive Markets Hypothesis

- (A1) Individuals act in their own self-interest.
- (A2) Individuals make mistakes.
- (A3) Individuals learn and adapt.
- (A4) Competition drives adaptation and innovation.
- (A5) Natural selection shapes market ecology.
- (A6) Evolution determines market dynamics.

Lo, A. W., 2005. Reconciling Efficient Markets with Behavioural Finance: The Adaptive Markets Hypothesis. *Journal of Investment Consulting* 7 (2), 21-44. [\[online\]](#)

The Minority Game

- The Minority Game is a simplified version of the El Farol Bar problem:
 - Binary outcomes (no threshold parameter)
 - Symmetric actions
- Used in (econo-)physics
- Analysed mathematically
- Also touted as one of the simplest agent-based models of financial markets:
 - C. H. Yeung and Y.-C. Zhang, "Minority games," Nov 2008. [\[Online\]](#)
 - D. Challet, M. Marsili, and Y.-C. Zhang, "Modeling market mechanism with minority game," Tech. Rep., 1999. [\[Online\]](#)
 - D. Challet, A. Chessa, M. Marsili, and Y. C. Zhang, "From minority games to real markets," *Quantitative Finance*, vol. 1, no. 1, pp. 168-176, January 2001. [\[Online\]](#)

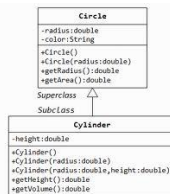
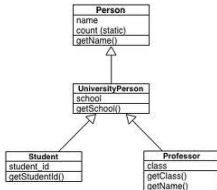


UML diagram showing the key components of JABM

Monte Carlo and OO

- How to execute OO programs as MC simulations?
- Ideally: program without having to differentiate between random variables and standard variables
- Ideally we want to implement a model and then run it under different conditions, eg:
 - Certain parameters drawn from random distributions
 - Control experiment in which test a range of parameter values
 - Constant parameter settings
- A good toolkit will allow us to do this ***without changing any code in our model***

Main reason: agent-based models are naturally conceptualised as Object-Oriented programs...



Agent-based model	Object-oriented program
Population of agents	Collection of object <i>instances</i>
Heterogeneous population	Different types or <i>classes</i> of object
Agent properties	Class attributes

Principle: agents should be first-class citizens in our program

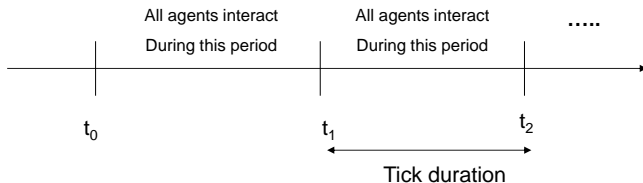
Object-oriented agent-based modelling

- Object-oriented programming languages provide a natural way of expressing all of these different conceptual frameworks for modelling agents, their interactions and how they learn or evolve.
- Agents can be represented as instances of objects.
- Different types of agents can be represented as different classes of object.
- Properties of agents correspond to class attributes.
- Temporal aspects can be represented within a discrete-event simulation framework [Banks and Carson, 1984].
- Events can be modelled as objects; different types of events as different classes.
- Agents and events automatically become *first-class citizens* [Burstall, 2000] in whatever OO language we use

POJOs – Plain Old Java Objects

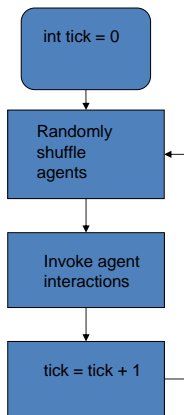
```
public class BiddingAgent {  
  
    /**  
     * The agent's private valuation for the item  
     * it is currently bidding on.  
     */  
    double valuation;  
  
    double stdevValuation = 1;  
  
    double meanValuation = 100;  
  
    public BiddingAgent(Random prng) {  
        this.valuation = prng.nextGaussian()  
            * stdevValuation + meanValuation;  
    }  
  
}
```

A simple discrete model of time



- Divide time into equally-spaced discrete chunks – sometimes called *ticks*.
- A tick is not necessarily a small unit of time – could be hours, days or years
- Typically a tick is the smallest unit of time in which all agents (eg buyers, sellers, auctioneer) have the opportunity to interact
- Suppose eg average arrival rate of:
 - buyers is 10 per hour
 - sellers is 1 per hour
- Then we would set:
 - tick duration = 1 hour
 - Number of sellers = 1
 - Number of buyers = 10
- The execution of each tick of the simulation is sometimes called the simulation *cycle*

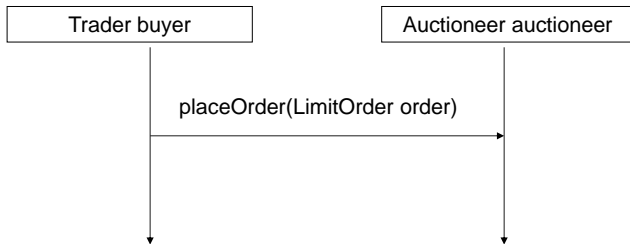
The simulation cycle



```
public void run() {  
    for (tick=0; tick<MAX_TICKS; tick++) {  
        Collections.shuffle(Arrays.asList(agents));  
        invokeAgentInteractions();  
    }  
}
```

Interaction between agents

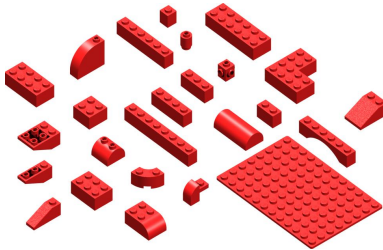
- We can think of methods a bit like “messages”. In the previous example we can think of the buyer object “sending” a `placeOrder` “message” to the auctioneer



Problems with this approach

- Intuitively we are trying to model an event, or occurrence, in the auction, but modelling events using methods has several problems.
- What if other types of object need to know when a bid has been placed? Eg:
 - An object which records the progress of the auction to a log file
 - If we are modelling an “open outcry” auction then other agents may need to take action when an order (bid) has been placed.
 - We need to record how long the event lasted
 - via eg a Clock class.

Programming with components



- Ideally we can plug in different types of component for different scenarios without having to write any code
- Programming becomes a kind of configuration or “wiring-up” exercise – to put together **new** solutions we string together **existing** components in different ways

Design Patterns

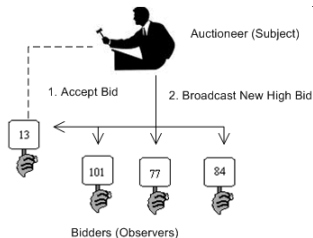
- This kind of problem has occurred time and time again in many other software engineering contexts
- There is a widely-known solution
- Commonly solved problems are catalogued into Design Patterns:

– *“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”*

» C. Alexander et al. *A Pattern Language*. OUP, New York, 1977, p.

x

The Observer design pattern



- The Observer design pattern establishes a publish-subscribe relationship between classes.
- Classes monitoring events are called Observers.
- Classes originating events (eg order placed event) are Observable. They maintain a list of Observers. Observable objects are also called Subjects.

In Java

- Java defines two classes to help implement the Observer pattern:
 - `java.util.Observer`
 - `java.util.Observable`
- Observerables (subjects) can notify Observers by calling `notifyObservers()`
- Must call `setChanged()` beforehand
- Observers are notified via the `update()` method
- Observers can subscribe for updates by invoking `addObserver()` on the subject.

java.util.Observable

```
package org.ccfea.auction;

import java.util.Observable;

import java.util.Random;

public class Trader extends Observable {

    float fundamentalValue;
    Random prng;
    String name;
    LimitOrder currentBid;

    public void requestBid(Auctioneer auctioneer) {
        float shade = prng.nextFloat() * MAX_PRICE;
        float bidPrice = Math.max(0.0f, fundamentalValue - shade);
        LimitOrder bid = new LimitOrder(this, 1, bidPrice);
        auctioneer.placeOrder(bid);
        // Call setChanged() to indicate that our state has changed
        setChanged();
        notifyObservers(bid);
    }
}
```

Defined already (inherited from Observable)

java.util.Observer

```
package org.ccfea.auction;

import java.util.Observable;
import java.util.Observer;

public class Auctioneer implements Observer {


    public void placeOrder(LimitOrder bid) {
        System.out.println("Received bid " + bid);
        if (bid.getPrice() > highestBid.getPrice()) {
            highestBid = bid;
        }
    }

    public void update(Observable o, Object arg) {
        if (o instanceof Trader) {
            Trader trader = (Trader) o;
            LimitOrder order = (LimitOrder) arg;
            placeOrder(order);
        }
    }
}
```

The `update()` method is executed whenever the `Observable` (subject) calls `notify()` - in this case a `Trader`. This is handled by code in the superclass `java.util.Observable`.

Subscribing to an Observable

```
public Simulation() {  
    traders = new Trader[NUM_TRADERS];  
    prng = new Random();  
    auctioneer = new Auctioneer();  
    for(int i=0; i<NUM_TRADERS; i++) {  
        traders[i] = new Trader("Trader#" + i, prng);  
        traders[i].addObserver(auctioneer);  
    }  
}
```



The `addObserver()` method is defined in `Observable` which is available in `Trader` via inheritance. The auctioneer will now receive updates from the trader.

Events

- We have “decoupled” Trader from Auctioneer:
 - We can add additional `Observers` without interfering with `Trader`’s code
- Limitations: the `notifyObserver()` method notifies the observers that something has happened but not what has happened.
- Solution: Model events just as any other entity in an object-oriented model: ie using classes, eg:
 - `OrderPlacedEvent`
 - `AuctionFinishedEvent`
 - Etc..

Event notification

```
package org.ccfea.auction;

import java.util.Observable;

import java.util.Random;

public class Trader extends Observable {

    float fundamentalValue;
    Random prng;
    String name;
    LimitOrder currentBid;

    public void requestBid(Auctioneer auctioneer) {
        float shade = prng.nextFloat() * (MAX_PRICE / 10.0f);
        float bidPrice = Math.max(0.0f, fundamentalValue - shade);
        LimitOrder bid = new LimitOrder(this, 1, bidPrice);
        auctioneer.placeOrder(bid);
        setChanged();
        notifyObservers(new OrderPlacedEvent(bid));
    }
}
```

Explicitly create an event object

Event handling

```
public void update(Observable o, Object arg) {  
    if (arg instanceof TradeOccurredEvent) {  
        Trade trade =  
            ((TradeOccurredEvent) arg).getTrade();  
        if (trade.getBuyer() == this) {  
            // Process the trade event  
            notify(trade);  
        }  
    }  
    if (arg instanceof AuctionFinishedEvent) {  
        System.out.println(this +  
            ": profits = " + payoff);  
    }  
}
```

Model View Controller (MVC) pattern

- A similar pattern in user-interface design called Model View Controller:
 - <http://msdn.microsoft.com/en-us/library/ms978748.aspx>
 - A Model is simply an object that can be observed and generates (“fires”) events
 - A View is typically an object representing a graphical component such as a graph. Typically redraws itself when an event happens
 - A Controller is an object that dispatches event in response to real-world events (eg mouse clicks)
- Many variants

MVC Advantages

- Loose coupling between visualisation components and the actual model
- Makes it easy to run simulations “headless” when running on a cluster
- Makes it easy to reuse different visualisation components across different models

“Wiring up” an MVC app

Q: How do we configure the subscription dependencies between our components?

A: One way of doing this is by introducing configuration code into our model, eg:

```
public class BiddingAgent extends AbstractModel {  
  
    double valuation;  
  
    public BiddingAgent(Auctioneer, Random prng, double mean) {  
        this.valuation = prng.nextGaussian() * mean;  
        auctioneer.addListener(this);  
    }  
  
    //...  
  
    public void generateBid() {  
        fireEvent(new BidPlacedEvent(20.5));  
    }  
}
```

Configuration using dependency injection

- Delegate configuration to a separate “container” component
- Express configuration and dependency relationships declaratively – eg in XML
- Container “injects” dependencies into our POJOs
- POJOs are passive in the “wiring up” process
- JABM uses the industry-standard [Spring framework](#)
- The container in Spring is called the *bean factory*
- Every object or agent in our simulation is described by a *bean definition*

```
public class BiddingAgent {  
  
    /**  
     * The agent's private valuation for the item  
     * it is currently bidding on.  
     */  
    double valuation;  
  
    public double getValuation() {  
        return valuation;  
    }  
  
    public void setValuation(double valuation) {  
        this.valuation = valuation;  
    }  
  
}
```

```
<bean id="myAgent" class="BiddingAgent">  
    <property name="valuation" value="10.0"/>  
</bean>
```



```

<bean id="agent1" class="BiddingAgent" scope="prototype">
    <property name="valuation" ref="valuationRandomVariate" />
</bean>

<bean id="agent2" class="BiddingAgent" scope="prototype">
    <property name="valuation" ref="valuationRandomVariate" />
</bean>

<!-- This bean takes different values each time it is referenced -->
<bean id="valuationRandomVariate"
    class="net.sourceforge.jabm.spring.RandomDoubleFactoryBean"
    scope="prototype">
    <property name="distribution" ref="commonValuationDistribution"/>
</bean>

<!-- The common probability distribution used to draw agents
    valuations -->
<bean id="commonValuationDistribution" scope="singleton"
    class="cern.jet.random.Normal">
    <constructor-arg value="100.0" />
    <constructor-arg value="1.0" />
    <constructor-arg ref="prng" />
</bean>

<!-- The Pseudo-Random Number Generator (PRNG) used to
    generate all random values in the simulation -->
<bean id="prng" scope="singleton"
    class="cern.jet.random.engine.MersenneTwister64">
</bean>

```

Listing 5 Configuring random variables using a properties file in JABM

```
# Configure the valuation attribute of an agent as a random variable
# drawn from a Uniform distribution.

agent1.valuation = U(50,100)
```

Listing 6 Configuring random variables and constant parameters in the same properties file

```
# Run the simulation over 1000 independent runs
simulationController.numSimulations = 1000

# Our population consists of 100 agents
population.numAgents = 100

# Configure the valuation attribute of an agent as a random variable
# drawn from a Uniform distribution.
agent.valuation = U(50,100)
```

Listing 7 Performing a parameter sweep

```
qLearner.learningRate = 0.1:0.1:0.9
qLearner.discountRate = 0.1:0.1:0.9
```

Advantages of dependency injection

- Promotes loose coupling
- Avoids hard-wiring dependencies
 - In particular we do not need to use Singletons, because [Singletons Are Evil](#)
- Configuration is expressed declaratively
- Supported by many IDEs
- Random variates can be injected into the OO model
- Bean factory ensures we obtain a fresh configuration on each execution – independent samples

Summary

- JABM is a toolkit that allows developers to build agent-based models using well-established Object-Oriented programming principles.
- Available under Open-source licence from <http://jabm.sourceforge.net/>
- Allows us to execute object-oriented programs in a Monte-Carlo setting.



Albert, R. and Barabasi, A. (2002).

Statistical mechanics of complex networks.

Reviews of modern physics, 74:47–97.



Arthur, W. B. (1994).

Inductive Reasoning and Bounded Rationality.

The American Economic Review, 84(2):406–411.



Axelrod, R. and Cohen, M. D. (2001).

Harnessing Complexity: Organizational Implications of a Scientific Frontier.

Basic Books.



Banks, J. and Carson, J. S. (1984).

Discrete-Event System Simulation.

Prentice Hall.



Bowling, M. (2005).

Convergence and No-Regret in Multiagent Learning.

In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 209–216. MIT Press, Cambridge, MA.



Boyd, R. and Richerson, P. J. (1988).

Culture and the Evolutionary Process.

University Of Chicago Press.



Burstall, R. (2000).

Christopher Strachey: Understanding Programming Languages.

Higher-Order and Symbolic Computation, 13:51–55.



Busoni, L., Babuska, R., and De Schutt, B. (2008).

A Comprehensive Survey of Multiagent Reinforcement Learning.

IEEE Transactions on Systems Man and Cybernetics - Part C: Applications and Reviews, 38(2):156–172.



Cai, K., Gerding, E., McBurney, P., Niu, J., Parsons, S., and Phelps, S. (2009).

Overview of CAT: A Market Design Competition.

Technical report, University of Liverpool.



Chmieliauskas, A., Chappin, E. J. L., and Dijkema, G. P. J. (2012).

Modeling Socio-technical Systems with AgentSpring.

In *CESUN Third International Proceedings of the Third International Engineering Systems Symposium: Design and Governance in Engineering Systems - Roots Trunk Blossoms*, Delft, Netherlands.



Cont, R. and Moussa, A. (2009).

Too interconnected to fail: contagion and systemic risk in financial networks.



DeLong, J. B. (1998).

Estimating World GDP, One Million B.C. to Present.



Dubitzky, W., Kurowski, K., and Schott, B., editors (2011).

Repast SC++: A Platform for Large-scale Agent-based Modeling.

Wiley.

in press.



Eguíluz, V. M. and Zimmermann, M. G. (2000).

Transmission of Information and Herd Behavior: An Application to Financial Markets.

Physical Review Letters, 85(26):5659–5662.



Eiben, A. E. and Smith, J. E. (2007).

Introduction to Evolutionary Computing.

Natural Computing. Springer, second edition.



Erdos, P. and Rényi, A. (1960).

On the evolution of random graphs.

Publications of the Mathematical Institute of the Hungarian Academy of Sciences, 5:17–61.



Ficici, S. G., Melnik, O., and Pollack, J. B. (2005).

A game-theoretic and dynamical-systems analysis of selection methods in coevolution.



Ficici, S. G. and Pollack, J. B. (1998).

Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states.

In *Proceedings of of ALIFE-6*, pages 238–247.



Ficici, S. G. and Pollack, J. B. (2000).

A game-theoretic approach to the simple coevolutionary algorithm.

In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, H.-P. S., editor, *Parallel Problem Solving from Nature — PPSN VI 6th International Conference*, pages 16–20, Paris, France. Springer Verlag.



Fowler, M. (2004).

Inversion of Control Containers and the Dependency Injection pattern.



Franklin, S. and Graesser, A. (1996).

Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents.

In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35. Springer Verlag.



Gamma, E., Helm, R., Johnson, R., and John Vlissides (1995).

Design Patterns: Elements of Reusable Object-Oriented Software.

Addison-Wesley.



Gross, T. and Sayama, H. (2009).

Adaptive networks: Theory, models and applications.

Springer-Verlag, Heidelberg, Germany.



Kaisers, M. and Tuyls, K. (2010).

Frequency adjusted multi-agent Q-learning.

In Van Der Hoek, Kamina, Lespérance, Luck, and Sen, editors, *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 309–316. International Foundation for Autonomous Agents and Multiagent Systems.



Kazlev, M. A. (2002).

The Cambrian Period of the Paleozoic Era: 542 to 488 Million Years Ago.



Kossinets, G. and Watts, D. J. (2006).

Empirical Analysis of an Evolving Social Network.

Science, 311(5757):88–90.



LeBaron, B. and Yamamoto, R. (2007).

Long-memory in an order-driven market.



Lo, A. (2005).

Reconciling Efficient Markets with Behavioural Finance: The Adaptive Markets Hypothesis.
Journal of Investment Consulting, 7(2):21–44.



Luke, S. (2005).

MASON: A Multiagent Simulation Environment.
Simulation: Transactions of the society for Modeling and Simulation International, 82(7):517–527.



Minar, N., Burkhart, R., Langton, C., and Askenazi, M. (1996).

The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations.
Technical Report 96-06-042, Santa Fe Institute, Santa Fe.



Moreira, J. E., Midkiff, S. P., Gupta, M., Artigas, P. V., Snir, M., and Lawrence, R. D. (2000).

Java programming for high-performance numerical computing.
IBM Systems Journal, 39(1):21–56.



Newman, M. (2010).

Networks: An Introduction.
Oxford University Press.



North, M. J. and Macal, C. M. (2005).

Escaping the Accidents of History: An Overview of Artificial Life Modeling with Repast.
In *Artificial Life Models in Software*, chapter 6, pages 115–141. Springer.



Parsons, R., MacKenzie, J., and Fowler, M. (2000).

Plain old Java Object.
online; accessed 27/11/2011.



Phelps, S. (2012).

Emergence of social networks via direct and indirect reciprocity.
Journal of Autonomous Agents and Multi-Agent Systems (forthcoming).



Phelps, S., McBurney, P., and Parsons, S. (2010).

A Novel Method for Strategy Acquisition and its application to a double-action market game.
IEEE Transactions on Systems, Man, and Cybernetics: Part B, 40(3):668–674.



Prasanna, D. R. (2009).

Dependency Injection: With Examples in Java, Ruby, and C#.
Manning Publications, 1st edition.



Rayner, N., Phelps, S., and Constantinou, N. (2012).

Learning is Neither Sufficient Nor Necessary: An Agent-Based Model of Long Memory in Financial Markets.
AI Communications, (forthcoming).



Reynolds, C. W. (1987).

Flocks, Herds, and Schools: A Distributed Behavioral Model.
Computer Graphics, 21(4):25–34.



Richerson, P. J. and Boyd, R. (2006).

Not by Genes Alone: How Culture Transformed Human Evolution.
University Of Chicago Press.



Schweitzer, F., Fagiolo, G., Sornette, D., Vega-Redondo, F., Vespignani, A., and White, D. R. (2009).

Economic Networks: The New Challenges.
Science, 325(5939):422–425.



Shoham, Y. and Leyton-brown, K. (2010).

Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations.
version 1. edition.



Skyrms, B. (2005).

Dynamics of Conformist Bias.
The Monist, 88(2):260–269.



Sutton, R. S. and Barto, A. G. (1998).

Reinforcement Learning: An Introduction.
MIT Press.



Tesfatsion, L. (2002).

Agent-based computational economics: growing economies from the bottom up.
Artificial Life, 8(1):55–82.



Travers, J. and Milgram, S. (1969).

An experimental study of the small world problem.
Sociometry, 32(4):425–443.



Vriend, N. J. (2000).

An illustration of the essential difference between individual and social learning, and its consequences for computational analyses.
Journal of Economic Dynamics and Control, 24:1–19.



Walsh, W. E., Das, R., Tesauro, G., and Kephart, J. O. (2002).

Analyzing complex strategic interactions in multi-agent games.
In *AAAI-02 Workshop on Game Theoretic and Decision Theoretic Agents*.



Watts, D. J. and Strogatz, S. H. (1998).

Collective dynamics of ‘small-world’ networks.
Nature, 393(6684):440–442.



Weibull, J. W. (1997).

Evolutionary Game Theory.
MIT Press, First MIT edition.



Wellman, M. P. (2006).

Methods for empirical game-theoretic analysis.
In *Proceedings of the Twenty First National Conference on Artificial Intelligence (AAAI-06)*, pages 1152–1155.



Wilensky, U. and Rand, W. (2011).

An introduction to agent-based modeling: Modeling natural, social and engineered complex systems with NetLogo.
MIT Press, Cambridge, MA.
in press.



Zimmermann, M. G., Eguíluz, V. M., Miguel, M. S., and Spadaro, A. (2000).

Cooperation in an Adaptive Network.
Advances in Complex Systems, 3(1-4):283–297.