Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Modeling and Simulating Social Systems with MATLAB

Lecture 7 – Game Theory / Agent-Based Modeling

Stefano Balietti, Olivia Woolley, Lloyd Sanders, Dirk Helbing

Computational Social Science

Repetition: Cellular Automata

- Cellular automata are abstract representations of dynamics in discrete space and time
- Reduction of complexity to simple microscopic interaction rules based on neighborhoods in a grid
- Their simplicity is also a limitation of cellular automata: system dynamics cannot always be reduced to neighbor-based interaction rules in a grid
- Can be test beds for complex systems: for example comparison of results of cellular automata and dynamical systems

Microscopic modeling

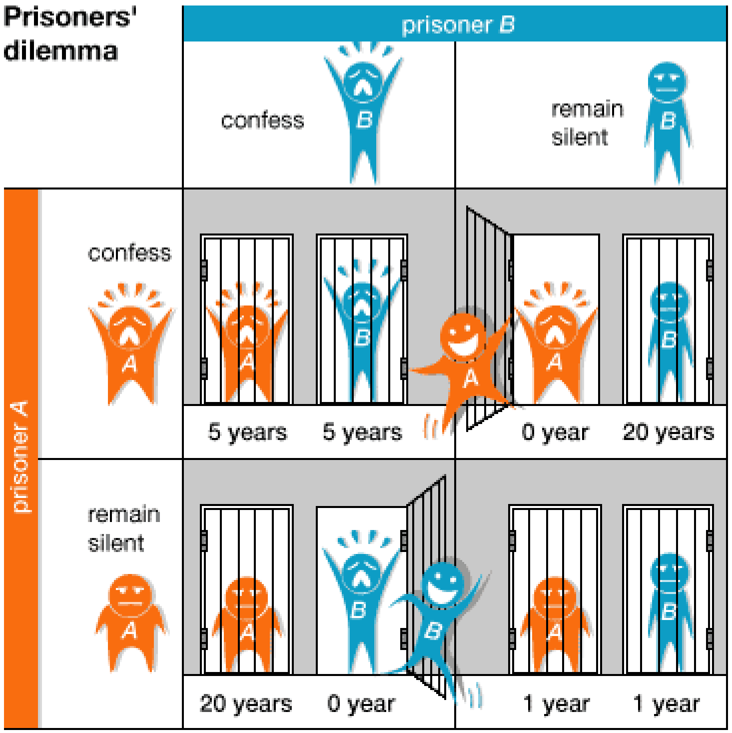
- Some real world processes can be reduced to simple local interaction rules (e.g. swarm intelligence)
- Microscopic modeling requires that dynamic components of a system are 'atomizable' in the given context (e.g. pedestrians behave like particles)
- Represents a reductionist (or mechanistic) point of view in contrast to a systemic (or holistic) approach
- In line with methods that have fueled the success of natural science research in the past 200 years
- Game theory provides a powerful framework to formalize and reduce complex (strategic) interactions

Game Theory

- Mathematical framework for strategic interactions of individuals
- Formalizes the notion of finding a ‘best strategy’ (Nash equilibrium) when facing a well-defined decision situation
- Definition and study of ‘games’
- Underlying assumption is that individuals optimize their ‘payoffs’ (or more precisely: ‘utility’) when faced with strategic decisions
- Repeated interactions are interesting for simulations (results can be completely different from one-shot games)

Game Theory - Human Cooperation

- Prisoner’s dilemma



15,15	20,0
0,20	19,19

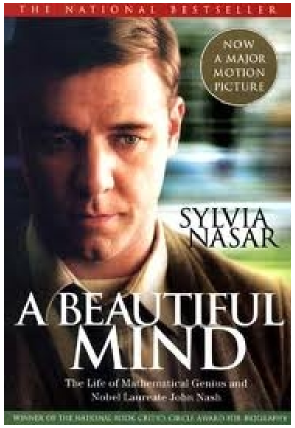
Game Theory: Assumptions

- **Rationality:** alignment between preferences, belief, and actions
- Players want to maximize their payoff
- A player's belief is something like:

  "I am rational, you are rational. I know that you are rational, and you know that I know that you rational. Everybody knows that I know that you are rational, and you know …"
- Players act accordingly: best response

Nash Equilibrium

- Is the strategy that players always play with no regrets: best response
- No player has an incentive to deviate from a Nash equilibrium
- In many circumstances, there is more than one Nash equilibrium



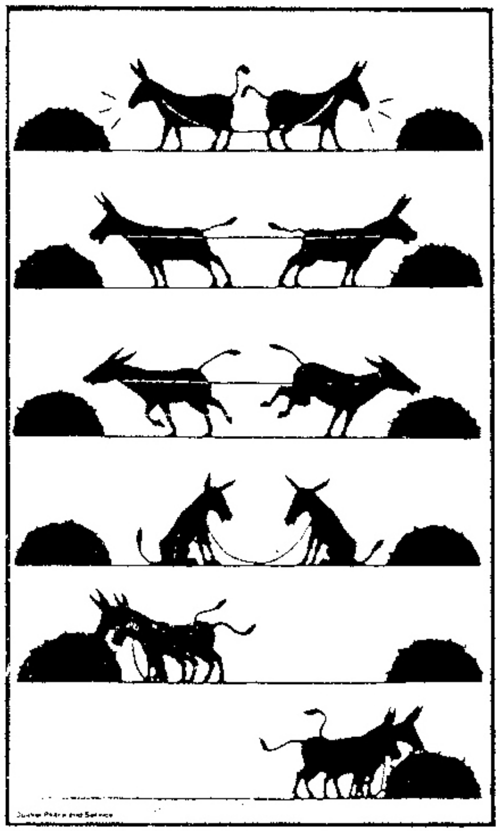
Some questions

- Is Nash an optimal strategy?
- What is the difference between a Pareto-efficient equilibrium and a Nash Equilibrium?
- Why do players play Nash? Do they?

The Evolution of Cooperation (Axelrod, 1984)

- Often a very good strategy: Tit for tat
  - be nice: cooperate first
  - then do whatever your opponent did in the last round (punish defection; reward cooperation)
- Other possible strategies:
  - Always cooperate / always defect
  - Tit for tat, but defect on first round
  - Win–Stay, Lose–Shift: repeat behavior if successful
- Shadow of the future
  - probability that there will be a next round

Game Theory - Coordination Games



1,1	0,0
0,0	1,1

Game Examples:

- Stag-hunt / assurance game
- Chicken / hawk-dove game

Evolutionary Game Theory

- Classical game theory has its limitations too: assumption of hyper-rational players; strategy selection problem; static theory
- Evolutionary game theory introduces evolutionary dynamics for strategy selection, i.e. the evolutionary most stable strategy dominates the system dynamics
- Very suited for agent-based modeling of social phenomena as it allows agents to learn and adapt

Evolutionary Learning

- The main assumption underlying evolutionary thinking is that the entities which are more successful at a particular time will have the best chance of being present in the future

- Selection
- Replication
- Mutation

Evolutionary Learning

- **Selection** is a discriminating force that favors successful agents
- **Replication** produces new agents based on the properties of existing ones (“inheritance”)
- Selection and replication work closely together, and in general tend to reduce diversity
- The generation of new diversity is the job of the **mutation** mechanism

How agents can learn

- Imitation: Agents copy the behavior of others, especially behavior that is popular or appears to yield high payoffs
- Reinforcement: Agents tend to adopt actions that yielded a high payoff in the past, and to avoid actions that yielded a low payoff
- Best reply: Agents adopt actions that optimize their expected payoff given what they expect others to do (choosing best replies to the empirical frequency distribution of their opponents' previous actions: "fictitious play")

From Factors to Actors (Macy, 2002)

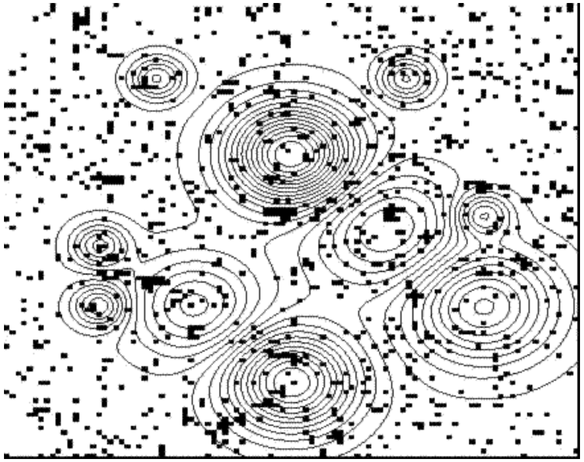
- Simple and predictable local interactions can generate complex and sometimes surprising global patterns
- Examples: diffusion of information, emergence of norms, coordination of conventions, or participation in collective action
- Simulation of individual **actors** instead of modeling based on aggregated **factors** (e.g. dynamical systems)

From Factors to Actors (Macy, 2002)

- Agents are autonomous
- Agents are interdependent
- Agents follow simple rules
- Agents are adaptive and backward-looking

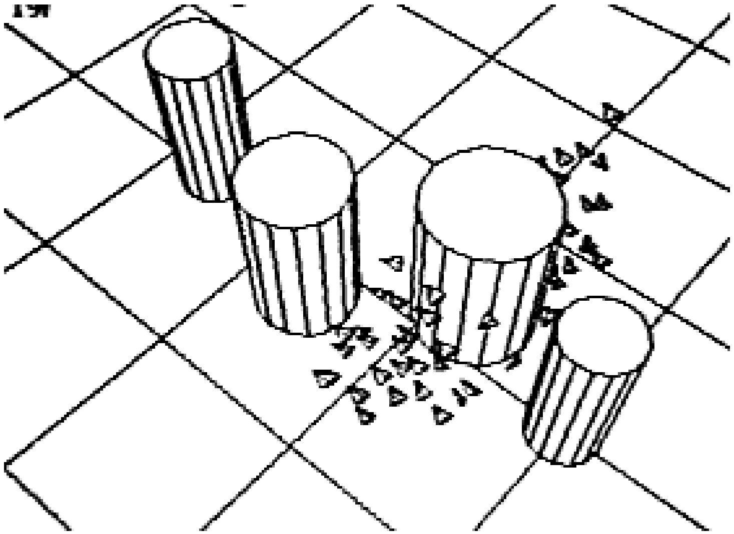
Swarm Intelligence

- Ant colonies, bird flocking, animal herding, bacterial growth, fish schooling
- Key Concepts:
  - decentralized control
  - interaction and learning
  - self-organization

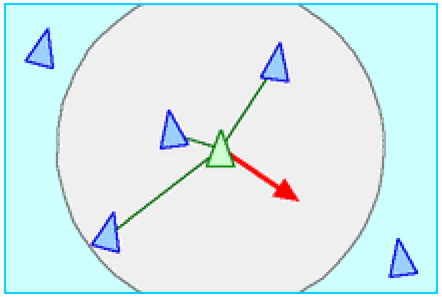


Boids (Reynolds, 1986)

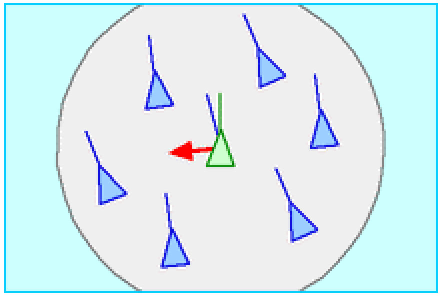
- Boids (bird-like objects) as simulated individual objects moving in a 3-dimensional virtual space



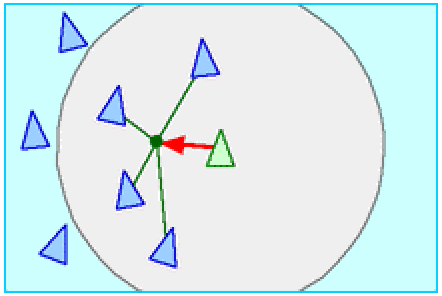
More information: http://www.red3d.com/cwr/boids/



Separation: steer to avoid crowding local flockmates

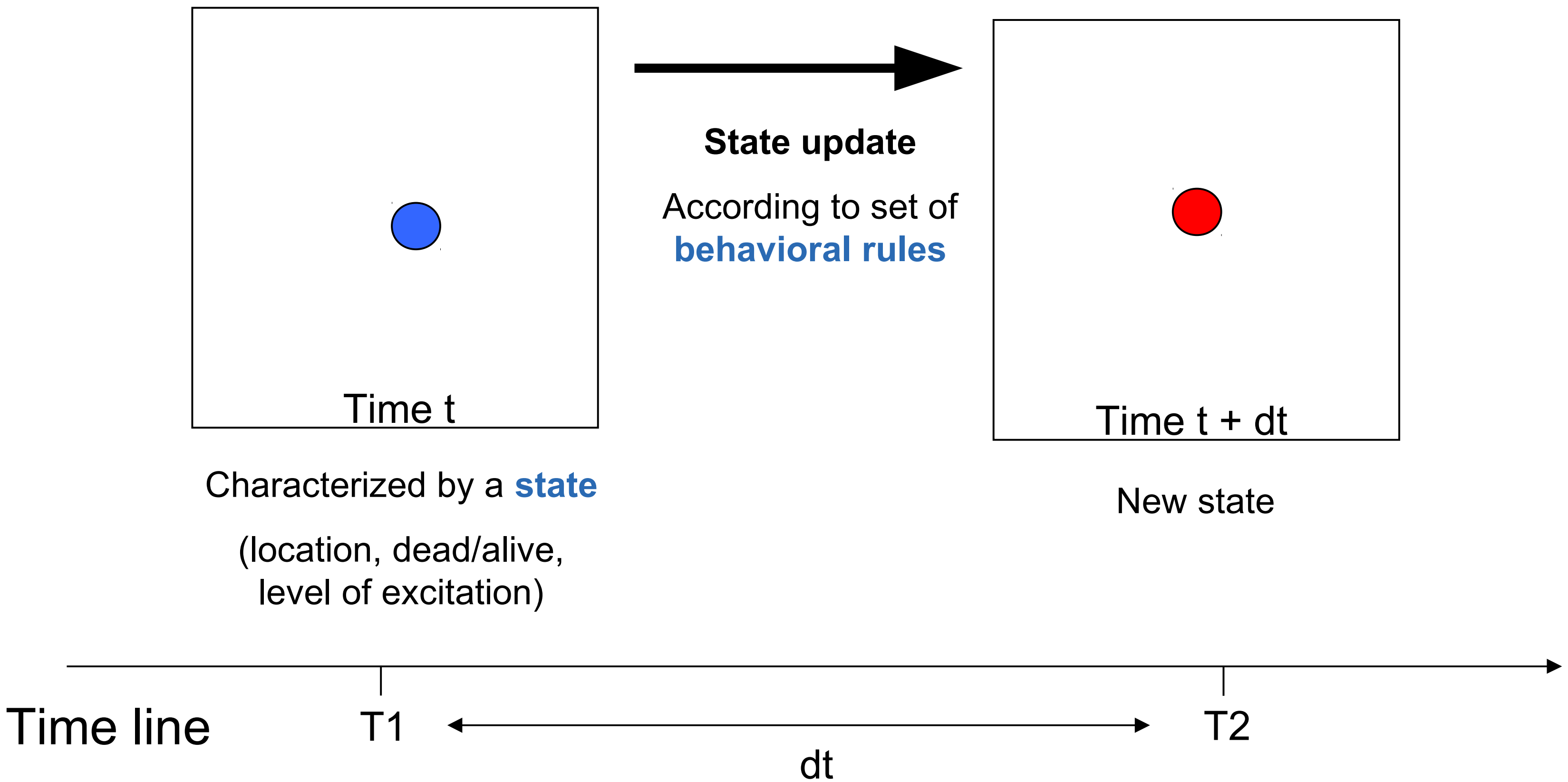


Alignment: steer towards the average heading of local flockmates

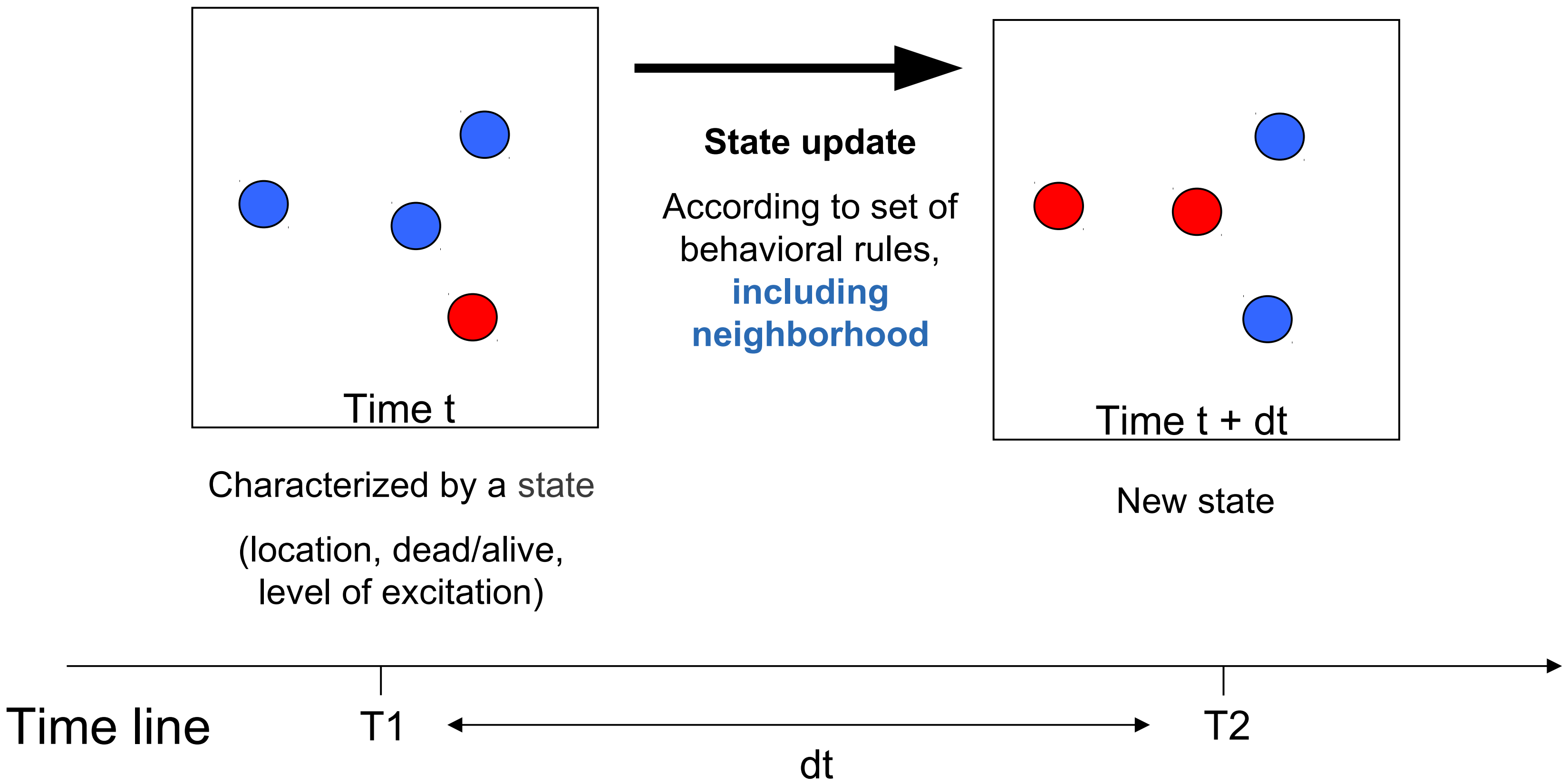


Cohesion: steer to move toward the average position of local flockmates

Programming Agent-Based Simulations

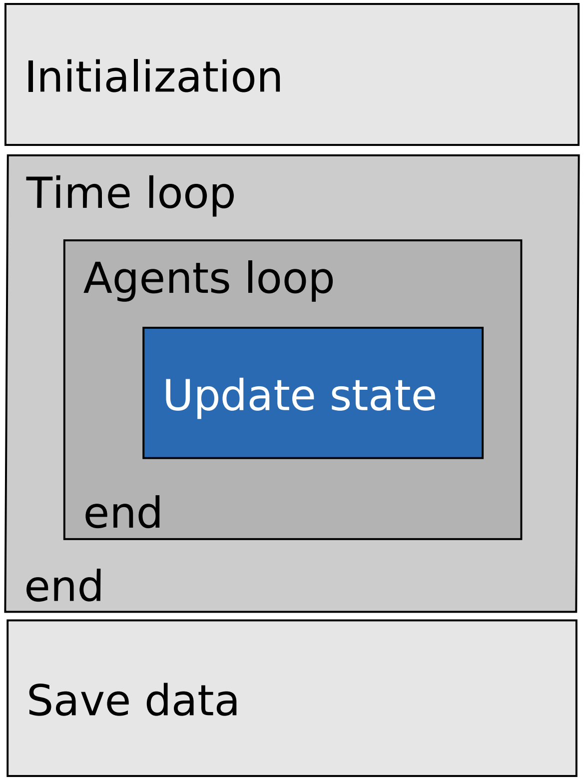


Programming Agent-Based Simulations



Programming a simulator

- Programming an agent-based simulator often goes in five steps
  - Initialization
    - Initial state; parameters; environment
  - Time loop
    - Processing each time steps
  - Agents loop
    - Processing each agents
  - Update
    - Updating agent *i* at time *t*
  - Save data
    - For further analysis



Programming a simulator

Efficient Structure

- Define and initialize all variables in the very beginning of your program
- Execute the actual simulation as a function with the most important variables as inputs and the most important result values as outputs
- Automated analysis, visualization, etc. of the simulation results after retrieving the output of the main function

Programming a simulator

- Step 1: Defining the initial state & parameters

```
% Initial state

t0 = 0;        % begining of the time line
dt = 1;        % time step
T = 100;       % number of time steps

% Initial state of the agents

State1 = [0 0 0 0];
State2 = [1 1 1 1];

% etc…
```

Programming a simulator

- Step 1: Defining the initial state & parameters

```
% Initial state

t0 = 0;      % begining of the time line
dt = 1;      % time step
T = 100;     % number of time steps

% Initial state of the agents

State1 = zeros(1,50);
State2 = rand(1,50);
```

Programming a simulator

- Step 2: Covering each time step

```
% Time loop

for t=t0:dt:T

     % What happens at each time step?
     % - Update environment
     % - Update agents

end
```

Programming a simulator

- Step 3: Covering each agent

```
% Agent loop

for i=1:length(States)

     % What happens for each agent?

end
```

Programming a simulator

- Step 4: Updating agent i at time t

```
% Update
%
% Example: Each agent has 60% chance to
% switch to state 1

randomValue = rand();

if (randomValue < 0.6)
    State(i)=1;
else
    State(i)=0;
end
```

Programming a simulator

- Step 4: Updating agent i at time t

```
% Update
%
% Example: Each agent has chance 'probsw'
% to switch to state 1

randomValue = rand();

if (randomValue < probsw)
    State(i)=1;
else
    State(i)=0;
end
```

define in first part

Programming a simulator

- Step 4: Updating agent i at time t

```
% Initial state

t0 = 0;         % begining of the time line
dt = 1;         % time step
T = 100;        % number of time steps
probsw = 0.6;   % probability to switch state
```

Programming a simulator

- Step 5: Final processing

```
% Outputs and final processing

propAlive = sum(states)/length(states);

% propAlive is the main output of the
% simulation
```

Programming a simulator

- Encapsulating main part in a function:

```
% simulation function
% input: probsw
% output: propAlive

function [propAlive] = simulation(probsw)

  % ...

  propAlive = sum(states)/length(states);

end
```

Programming a simulator

- Running the simulation:

```
>> p=simulation(0.6)
p =
0.54
>> p=simulation(0.6)
p =
0.72
```

Programming a simulator

- Automatically run a large number of simulations:

```
% Running N simulations

N = 1000;      % number of simulations
probsw = 0.6; % probability to switch state

for n=1:N
    p(n) = simulation(probsw);
end
```

Programming a simulator

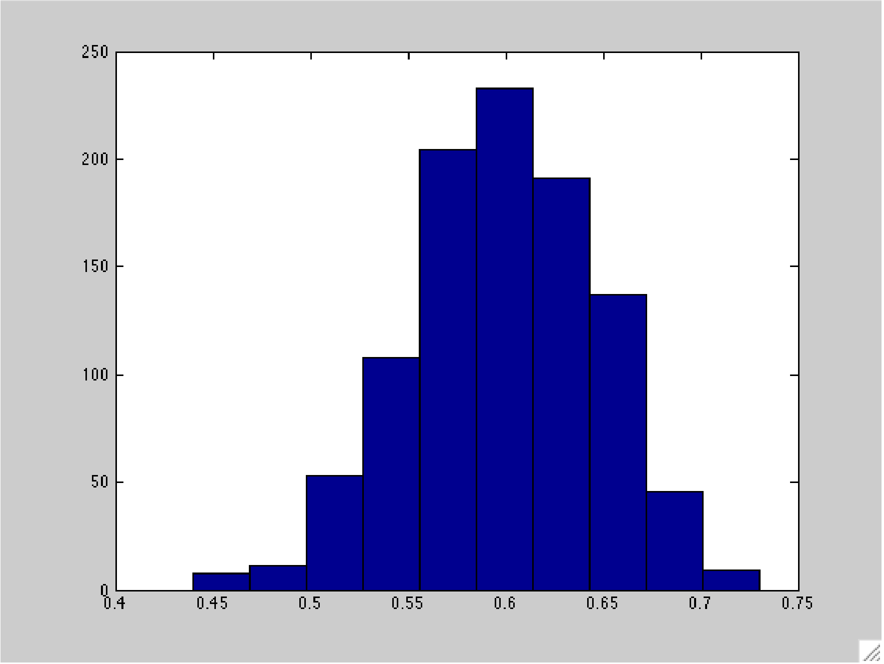
- Global variables:

```
global alpha beta  % mark as global
alpha = 0.1;
beta = 0.9;

for n=1:N
    p(n) = simulation(probsw);
end


function [propAlive] = simulation(probsw)
  % mark as global with the function too:
  global alpha beta
  % ...
end
```

Programming a simulator

- Alternatively:

```
>> hist(p)
```



Object-oriented Programming in MATLAB

- MATLAB has support for object-oriented programming
- However, we do not encourage to use this feature, as it is not an efficient way of programming in MATLAB
- Use of vectors and matrices instead of objects

If you still want to have a look at how object-oriented programming works in MATLAB:
http://www.mathworks.com/company/newsletters/articles/introduction-to-object-oriented-programming-in-matlab.html

Exercise 1

- There is some additional material on the iterated prisoner’s dilemma available in the directory 'prisoner' of https://github.com/msssm/lecture_files
- Experiment with the code and try to find out what it does and how to interpret the results

Exercise 2

- Start thinking about how to adapt the general simulation engine shown today for your project
- Problems:
  - Workspace organization (files, functions, data)
  - Parameters initialization
  - Number of agents
  - Organization of loops
  - Which data to save, how frequently, in which format
  - What kind of interactions are relevant
  - …

References

- Fudenberg & Tirole, *Game Theory*, MIT Press (2000)
- Meyerson, *Game Theory*, Harvard University Press (1997)
- Herbert Gintis, *Game Theory Evolving*, Princeton Univ. Press, 2nd Edition (2009)
- Uhrmacher, Weyns (Editors), *Multi-Agent Systems – Simulation and Applications*, CRC Press (2009)
- A step by step guide to compute the Nash equilibrium: http://uwacadweb.uwyo.edu/Shogren/jaysho/NashNotes.pdf
- https://github.com/Sandermatt/ETHAxelrodNoise