UML for ABM

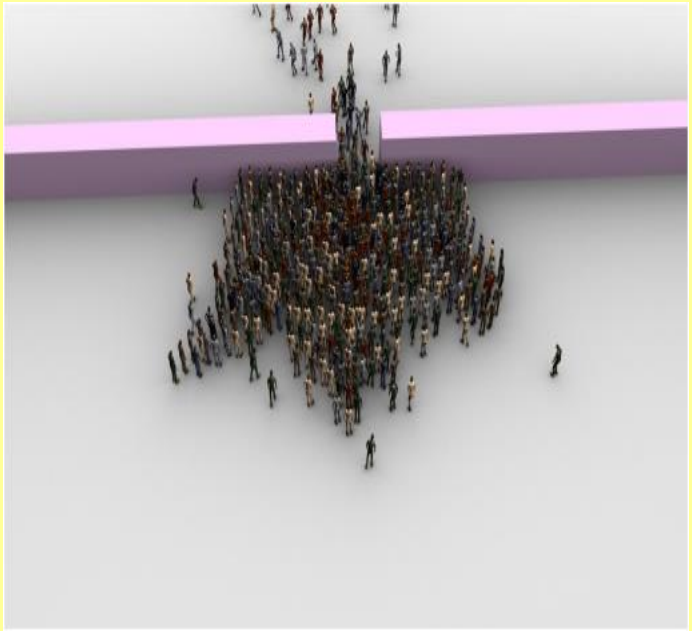
Hugues Bersini
IRIDIA/CODE

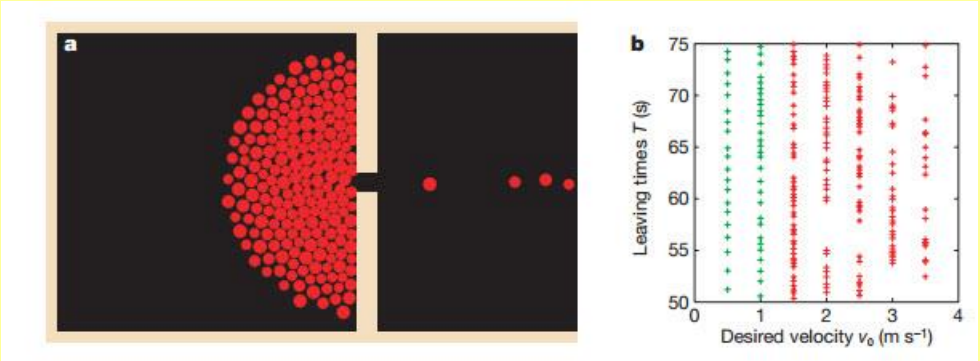
Plan

- Why modelling socio-economical complex systems ?
- How to model them ?

Social simulations – The crowd behaviour

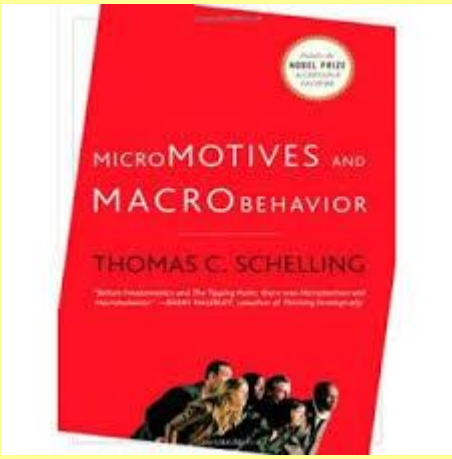
Dirk Helbing

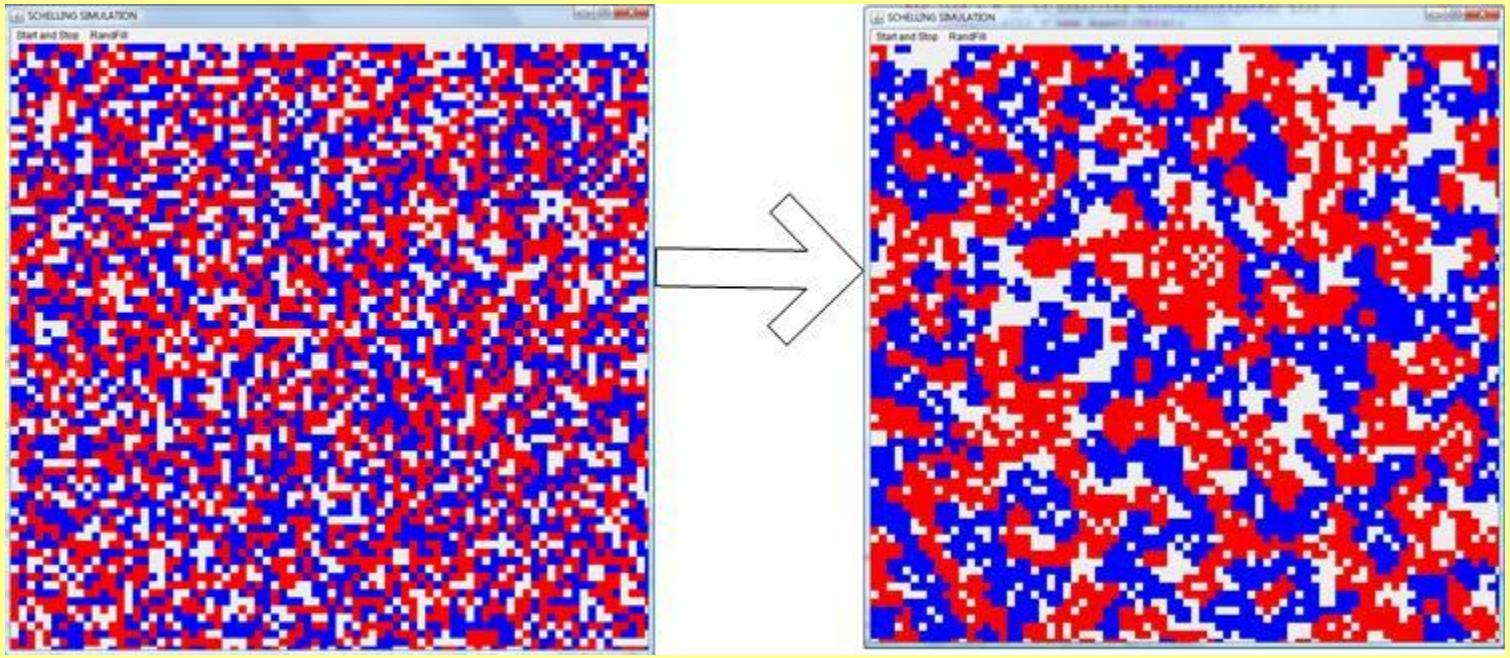


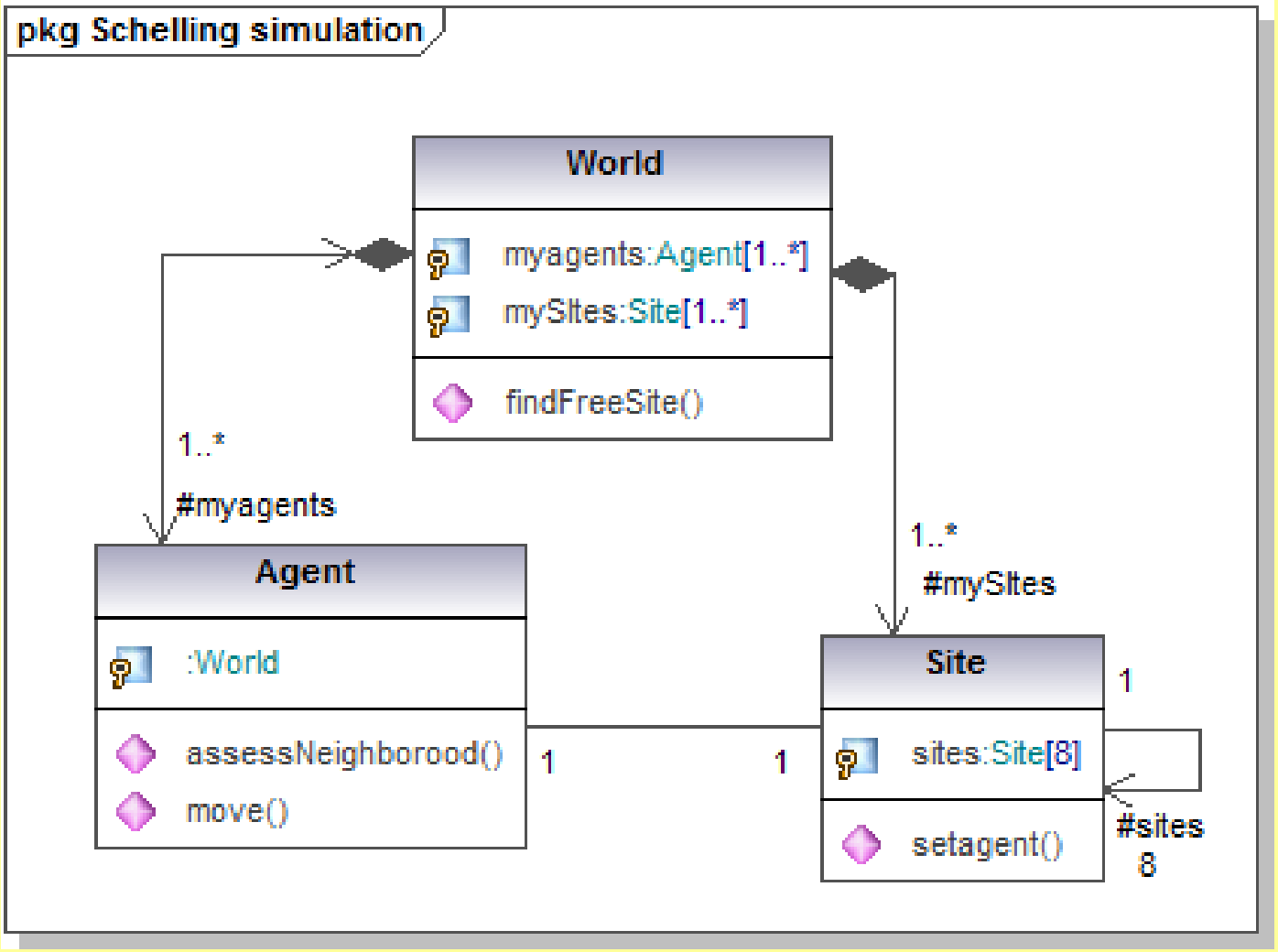

a
b
Leaving times T (s)
75
70
65
60
55
50
0
1
2
3
4
Desired velocity v_0 (m s^{-1})

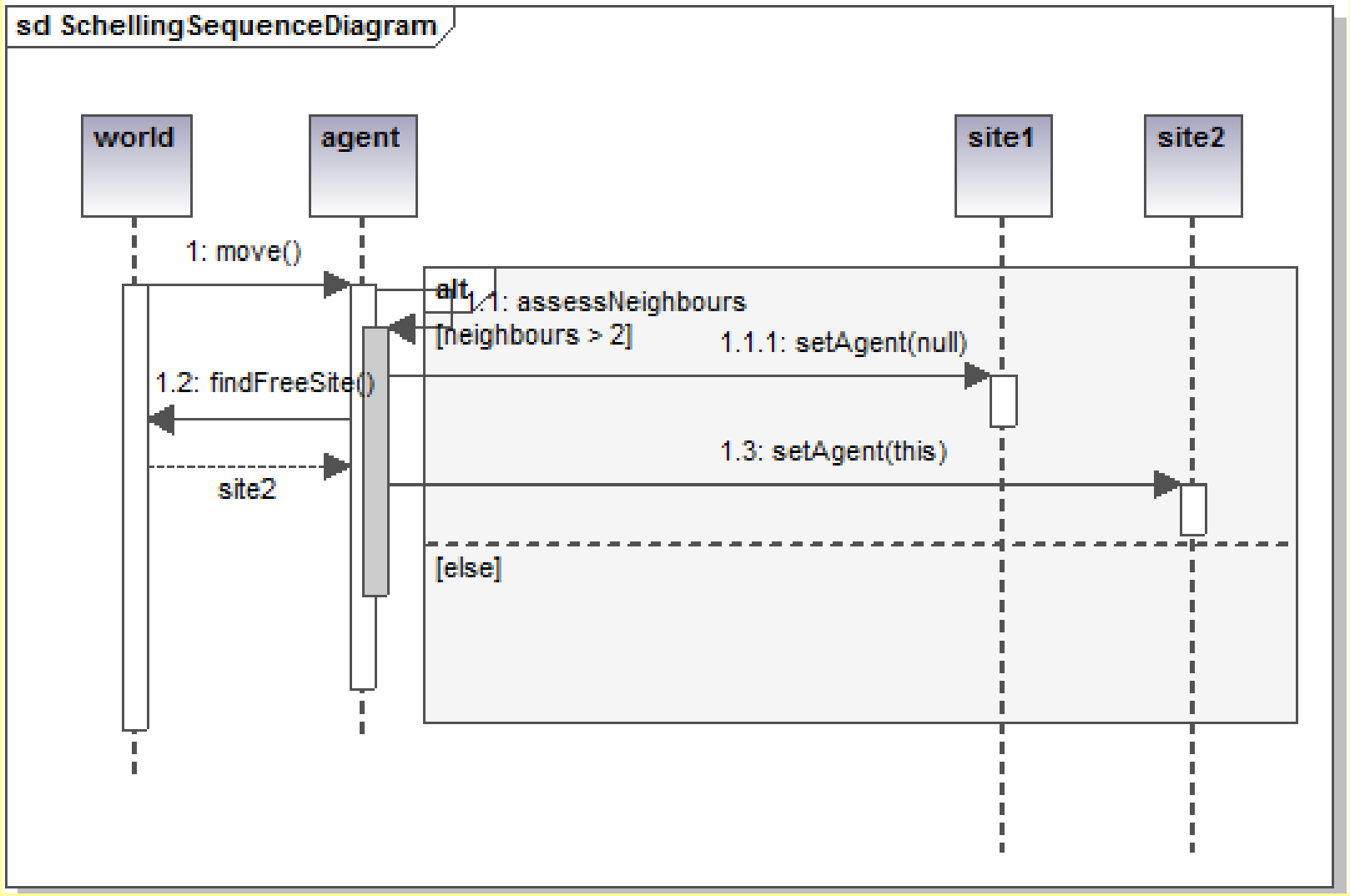
Sociological emergent effect

- Thomas Schelling experiment
- How weak racism creates strong seggregation

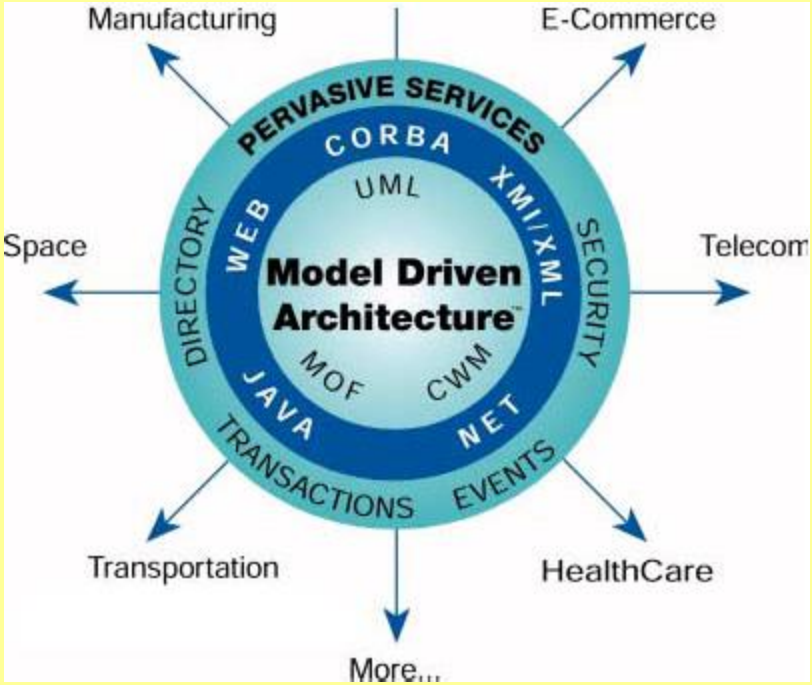



SCHELLING SIMULATION
Start and Stop
RandFill
SCHELLING SIMULATION
Start and Stop
RandFill


pkg Schelling simulation
World
myagents:Agent[1..*]
mySites:Site[1..*]
findFreeSite()
1..*
#myagents
1..*
#mySites
Agent
:World
assessNeighborood()
move()
1
1
Site
sites:Site[8]
setagent()
1
#sites
8


sd SchellingSequenceDiagram
world
agent
site1
site2
1: move()
alt
1: assessNeighbours
[neighbours > 2]
1.1.1: setAgent(null)
1.2: findFreeSite()
1.3: setAgent(this)
site2
[else]

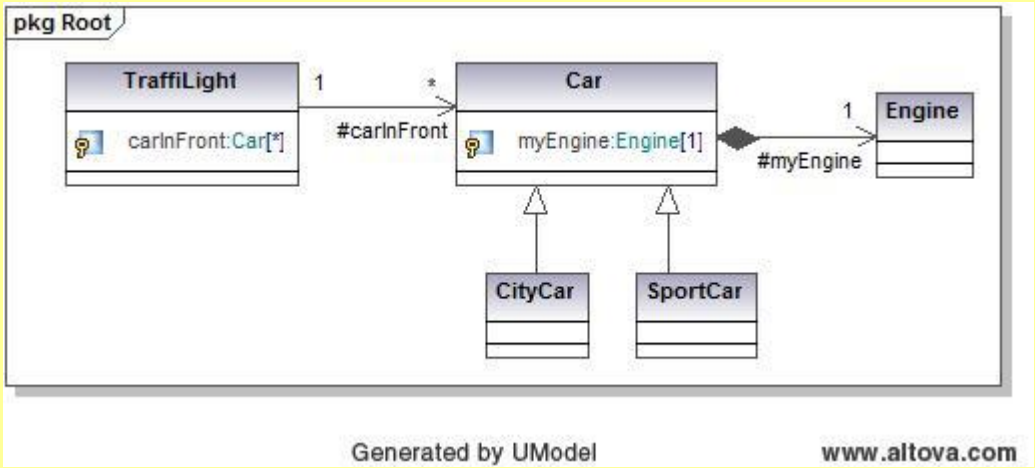
UML -> MDA



→ Third generation programming language

- UML could be the language adopted by natural scientists to express their knowledge
- Could help to homogenize and cross-fertilize existing models.
- We need to stop with “write once run only once !!!”
- Could improve their own understanding of their own field.
- Could disambiguate some of their knowledge
- Just three diagrams need to be understood

Introduction to class, sequence and state diagrams



1) ***class*** *TrafficLight {* ***private*** *ArrayList<Car> carsInFront;* ***public*** *TrafficLight () { carsInFront =* ***new*** *ArrayList<Car>();}* ***public void*** *addCar(Car c) {carsInFront.add(c);}} –*
2) ***class*** *Car{* ***private*** *Engine myEngine;* ***public*** *Car() {myEngine =* ***new*** *Engine();}} –*
3) ***class*** *Engine{}*
4) ***class*** *CityCar* ***extends*** *Car{}*
5) ***class*** *SportCar* ***extends*** *Car{} .*

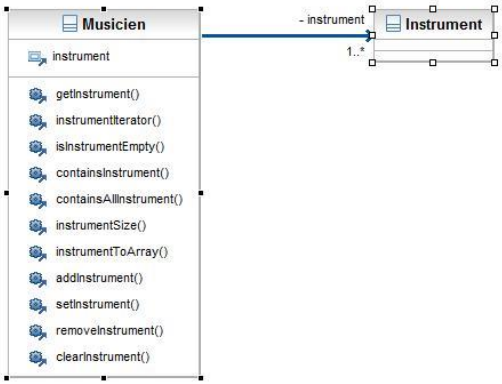
Java - Diagramme de Classes UML Musiciens/src/Musicien-aid.ucd - Eclipse Platform

File Edit View Navigate Search Project Run Window Help

Package Exp | Hierarchy

- Banque
- BioMed
- BioMed1
- BioMedSimple
- Capac
- CoursPolytech
- CoursUML
- EntreeSortie
- EssaiUML
- Examen-Cours
- FeuRouge
- HelloWorld
- ImmunoChaoForrest
- ImmunoFromChao
- MonHello
- Musiciens
  - src
    - (default package)
      - Instrument.java
      - Musicien.java
    - Musicien-aid.ucd
  - JRE System Library [jre1.6.0_02]
  - Musiciens.uml
  - reverse_settings.xml
- NeuralNetwork
- Oberveur
- Orchestre
- PolytechTestUML
- TechnoUML
- Tennis
- TestCompteBanque
- TestCoursUML
- TestGenericite
- TestGraphique
- TestIO
- TestJava
- TestPolytech
- TestTableau
- TestThread
- TestUML3
- TestUML4
- ThreadPolytech
- TrafficLight
- UMLEssai
- UMLPolytech
- Veronique
- Welcome

Test1-i.ucd | *Musicien-aid.ucd



ClassDiagram Musicien-aid

Musicien.java | Instrument.java

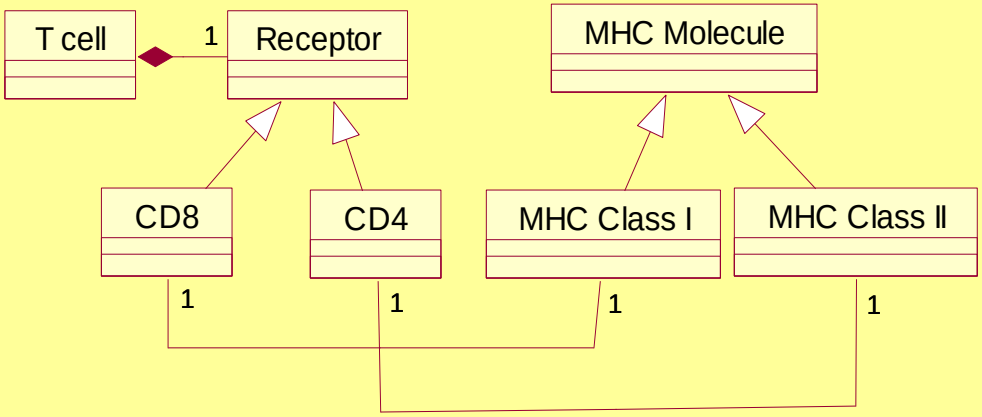
```
public Collection<Instrument> getInstrument()
{
    return instrument;
}

/**
 * Returns an iterator over the elements in this collection.
 *
 * @return an <tt>Iterator</tt> over the elements in this collection
 * @see java.util.Collection#iterator()
 *
 */
```

Problems | Javadoc | Declaration

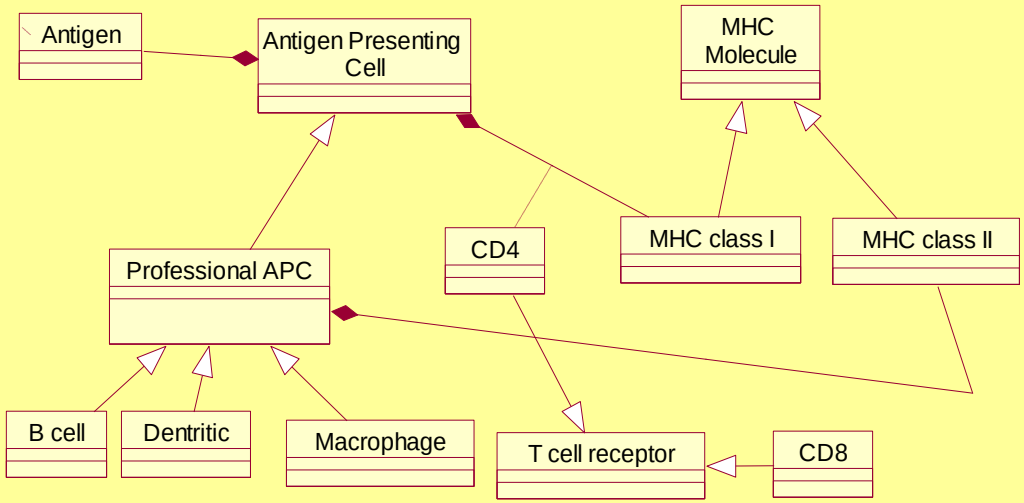
Task List | Outline | Font and Colors

UML Class diagram examples: taken from the Janeway et al's immune bible

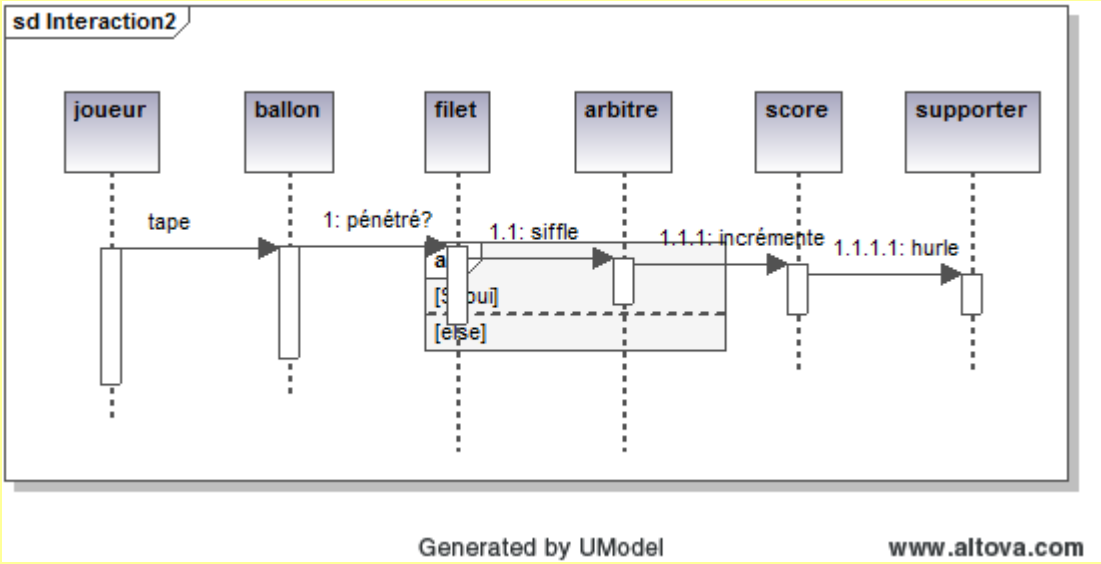
The antigen-specific activation of these effector T cells is aided by co-receptors on the T-cell surface that distinguish between the two classes of MHC molecule; cytotoxic cells express the CD8 co-receptor, which binds MHC class I molecules, whereas MHC Class II molecules specific T cells express the CD4 co-receptor, which has specificity for MHC Class II molecules.

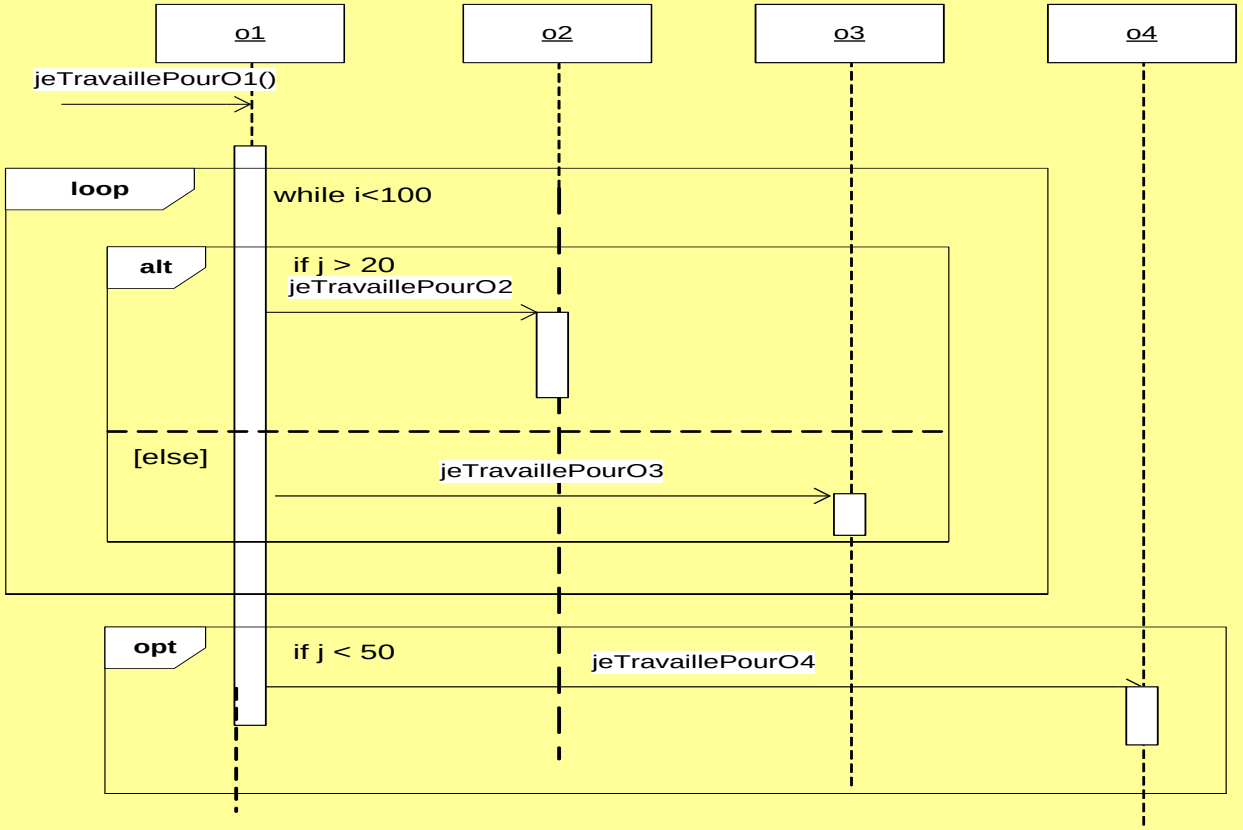


T cells are activated to produce armed effector T cells when their encounter their specific antigen in the form of a peptide: MHC complex on the surface of an activated antigen-presenting cell (APC) … The most important APC are the highly specialized dentritic cells … Macrophages can also be activated to express co-stimulatory and MHC class II molecules … B cells can also serve as APC in some circumstances...
Dentritic cells, macrophages and B cells are often known as professional antigen presenting cells



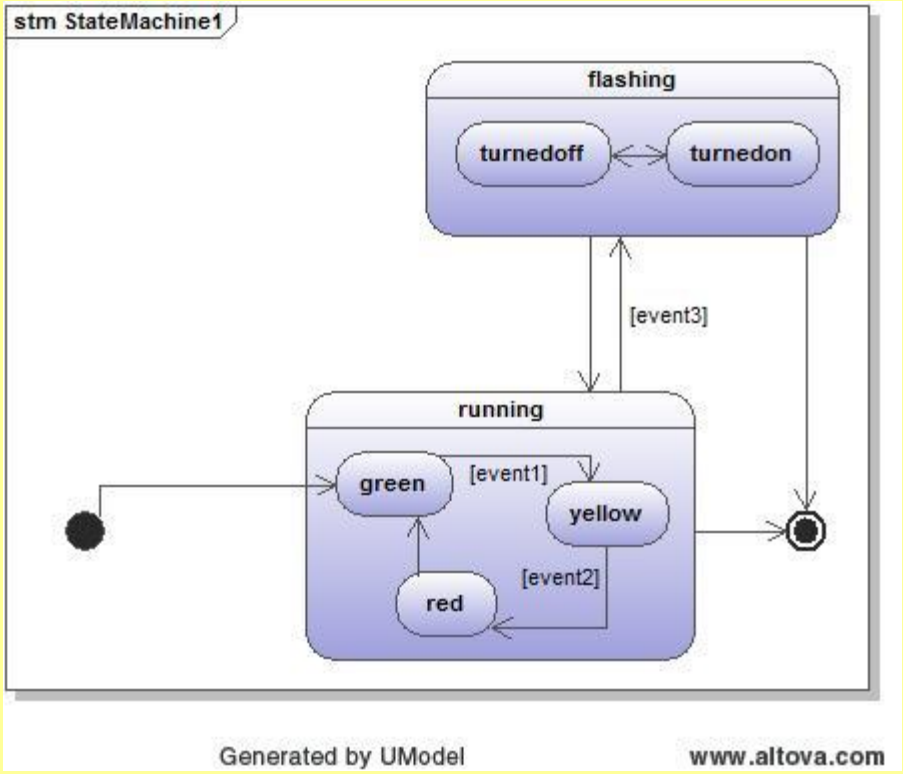
Sequence Diagrams

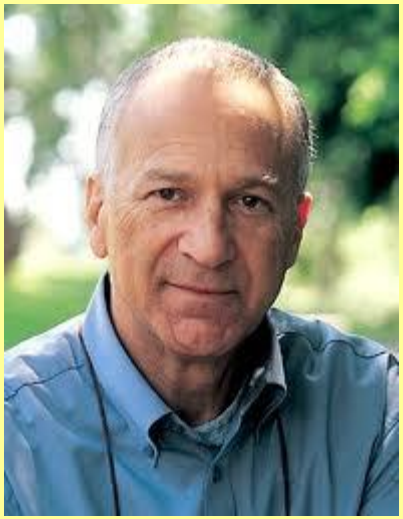





```
public class O1 {
  private int attribute1;
  private O2 lienO2;
  private O3 lienO3;
  public void jeTravaillePourO1(int a) {
  while (i<100) {
    if (j > 20){
      lienO2.jeTravaillePourO2();
    }
    else{
      lienO3.jeTravaillePourO3(a);
    }
  }
  if (j < 50) {
      lienO4.jeTravaillePourO4();
  }
}

class O2 {
  private O3 lienO3;
  public void jeTravaillePourO2() {
    lienO3.jeTravaillePourO3(6);
  }
}
```

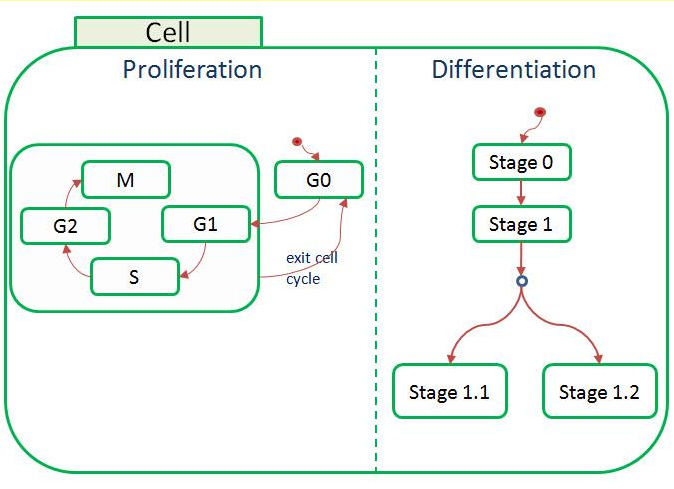
State-transition diagram

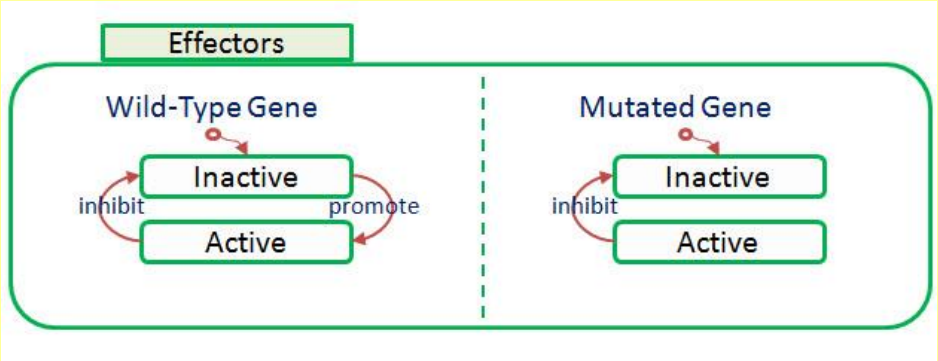


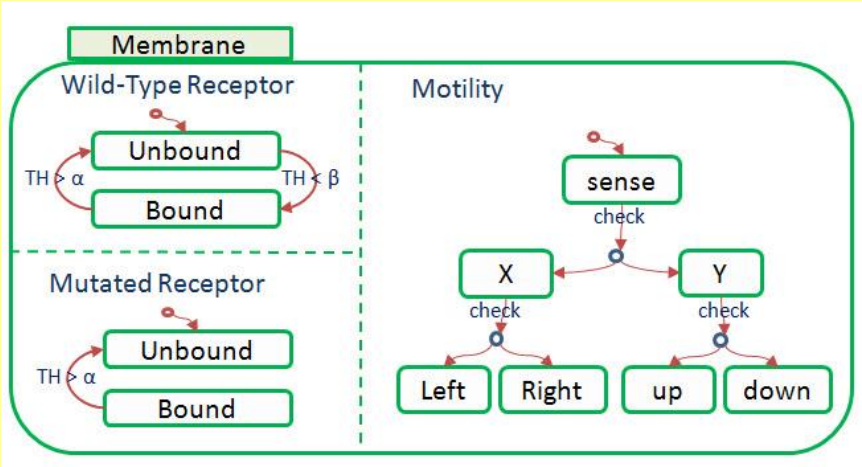


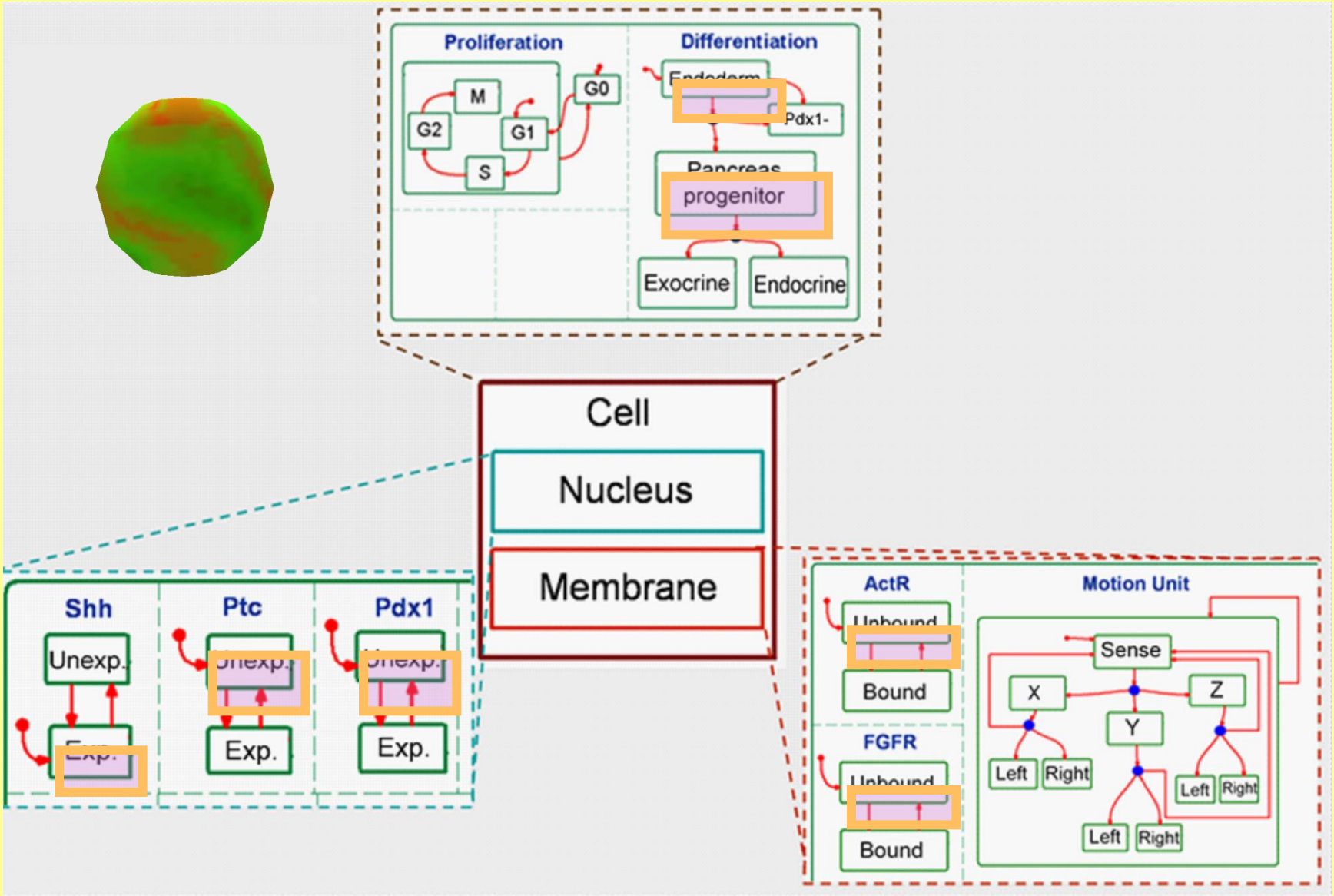
David Harel

Creator of State Diagrams and IBM Rhapsody

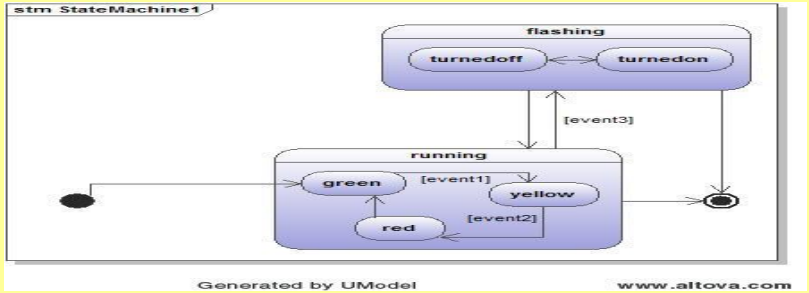

Cell
Proliferation
Differentiation
M
G0
G2
G1
S
exit cell cycle
Stage 0
Stage 1
Stage 1.1
Stage 1.2


Effectors
Wild-Type Gene
Inactive
inhibit
promote
Active
Mutated Gene
Inactive
inhibit
Active


Membrane
Wild-Type Receptor
Unbound
TH > α
TH < β
Bound
Mutated Receptor
Unbound
TH > α
Bound
Motility
sense
check
X
check
Y
check
Left
Right
up
down

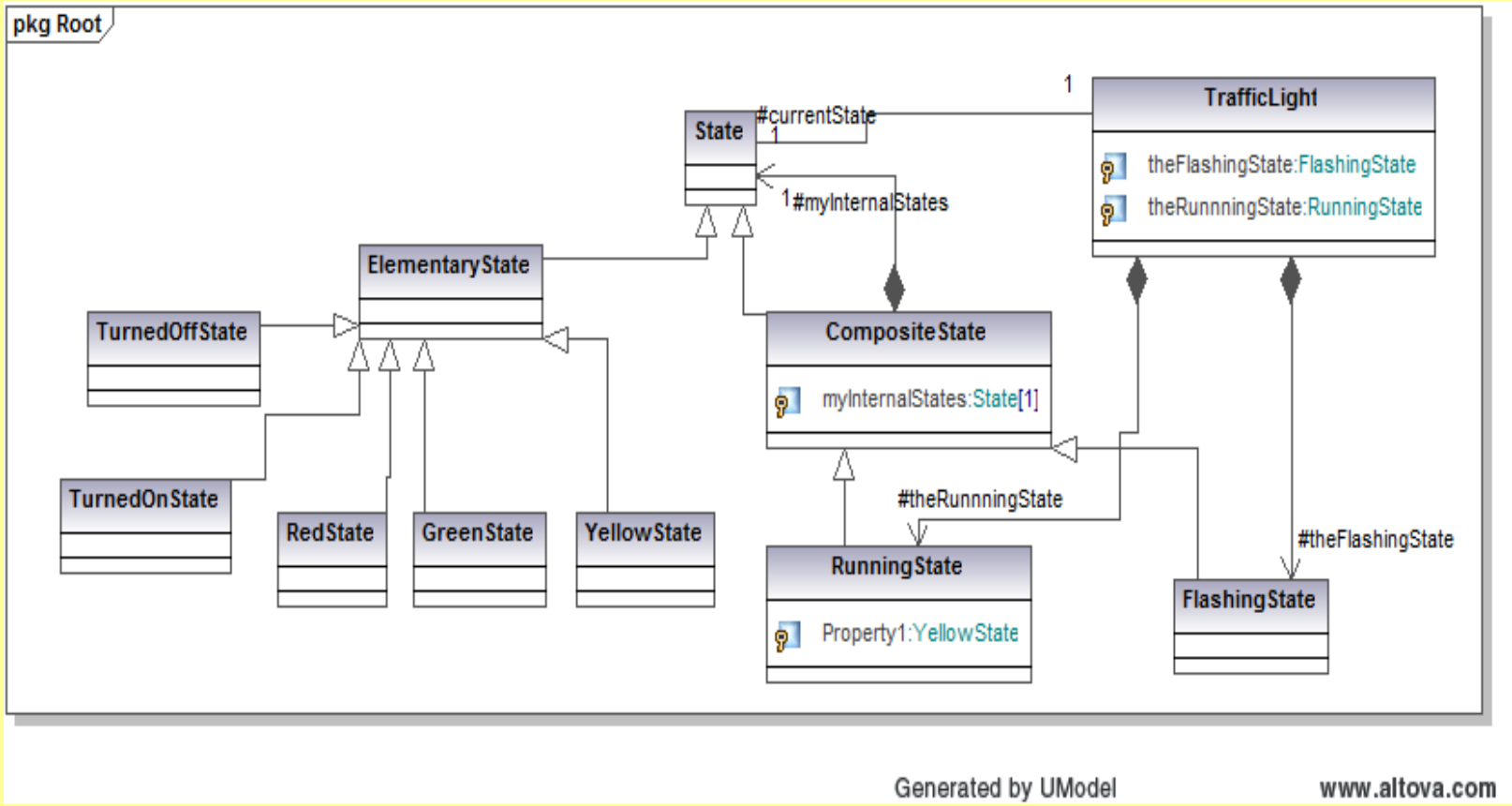

Proliferation
M
G0
G2
G1
S
Differentiation
Pdx1-
progenitor
Exocrine
Endocrine
Cell
Nucleus
Membrane
Shh
Unexp.
Ptc
Exp.
Pdx1
Exp.
ActR
Bound
FGFR
Bound
Motion Unit
Sense
X
Z
Y
Left
Right
Left
Right
Left
Right

From the diagram to XMI




```
<subvertex xmi:type="uml:State" xmi:id="U2d38031e-88c0-4487-a358-10033319382f"
xmi:uuid="2d38031e-88c0-4487-a358-10033319382f" name="running">
        <region xmi:type="uml:Region" xmi:id="U9975393a-c217-402a-8c28-b4ad044f4921"
                xmi:uuid="9975393a-c217-402a-8c28-b4ad044f4921" name="Region1">
        <subvertex xmi:type="uml:State" xmi:id="U0d02c7cb-1cc5-4cdb-9951-f98db987fa6b"
                xmi:uuid="0d02c7cb-1cc5-4cdb-9951-f98db987fa6b" name="yellow">
        <outgoing xmi:idref="U255848fd-71c6-4c37-9320-49f74f21bf91"/>
        <incoming xmi:idref="U89943d6b-b6dd-4e18-aa2d-1998dd625af9"/>
        </subvertex>
<subvertex xmi:type="uml:State" xmi:id="U95ab8769-8478-41a6-b00f-51caa61d343b"
xmi:uuid="95ab8769-8478-41a6-b00f-51caa61d343b" name="green">
```

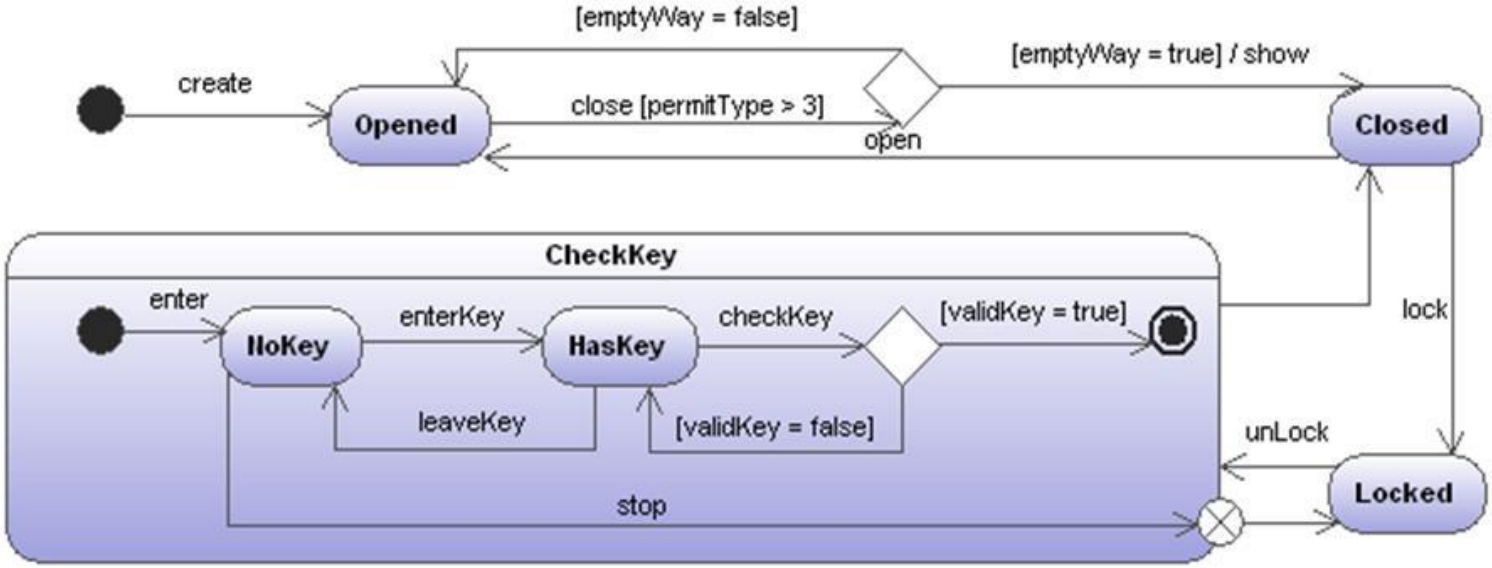
Automatic code generation: the state pattern

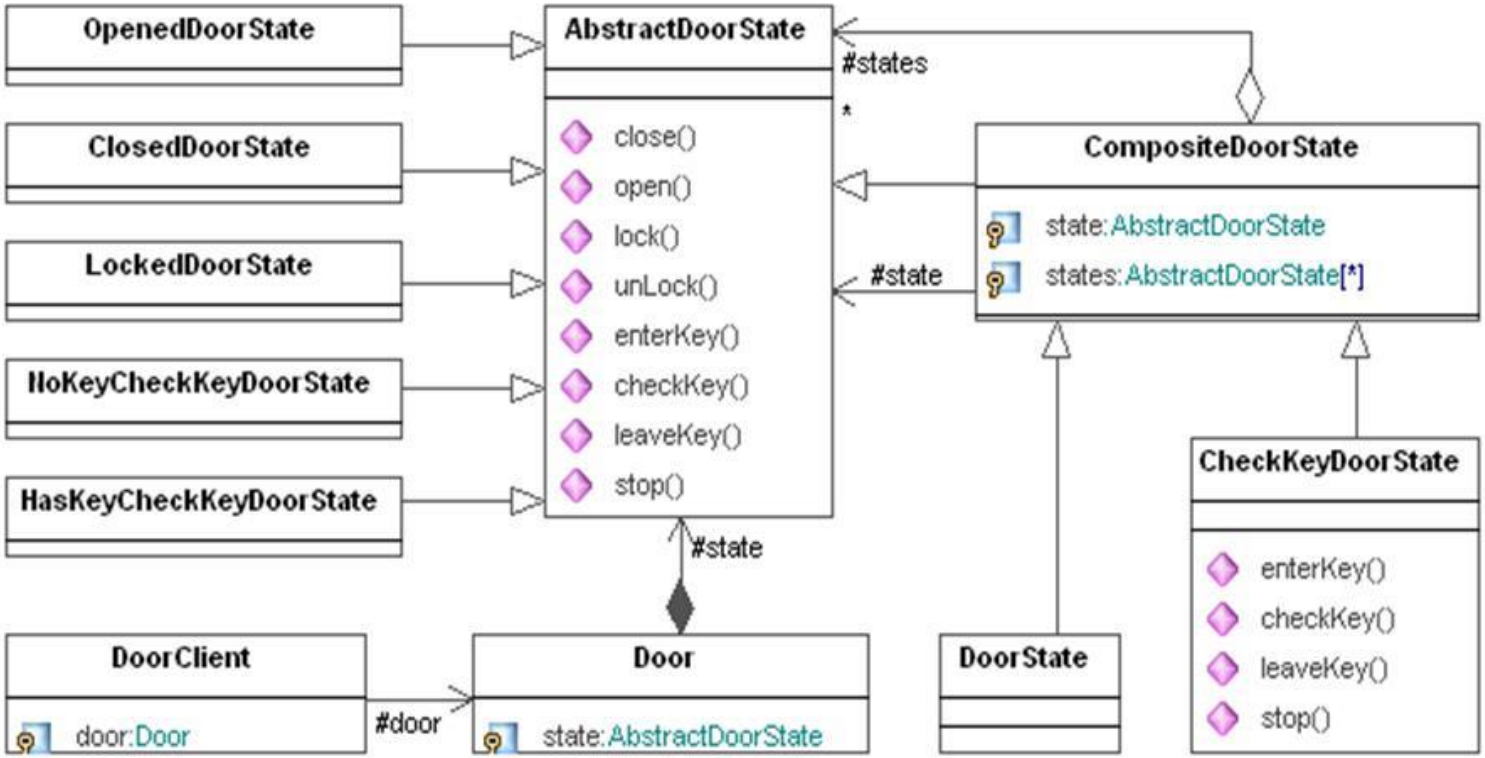


Automatic Code Generation

```
class TrafficLight {
        private ElementaryState currentState;
        private CompositeState currentCompositeState;
        private RunningState runningState;
        private FlashingState flashingState;
        public TrafficLight (ElementaryState currentState){
                carsInFront = new ArrayList<Car>();
                this.currentState = currentState;
                runningState = new RunningState(this);
                flashingState = new FlashingState(this);
        }
        public ElementaryState getYellowState() {
                return runningState.getYellowState();
        }
        public void changeState(ElementaryState newState){
                currentState.exitState();
                currentState = newState;
                currentState.enterState();
}
public void simulate() {
        while (true) {
                currentState.leaveState();
        }
}
}
```

```
abstract class State {
        private TrafficLight theTrafficLight;
        public State (TrafficLight theTrafficLight){
                this.theTrafficLight = theTrafficLight;
        }
        public TrafficLight getTheTrafficLight(){
                return theTrafficLight;
        }
        public abstract void enterState();
        public abstract void exitState();
        public abstract void leaveState();
}

class GreenState extends ElementaryState {
        public GreenState(TrafficLight theTrafficLight){
                super(theTrafficLight);
        }
        public void enterState() {}
        public void exitState() {}
        public void leaveState() {
                if (event1){
                        greenToYellow();
                }
        }
        public void greenToYellow (){
        getTheTrafficLight().changeState(getTheTrafficLight()..getYellowState());
    }
}
```

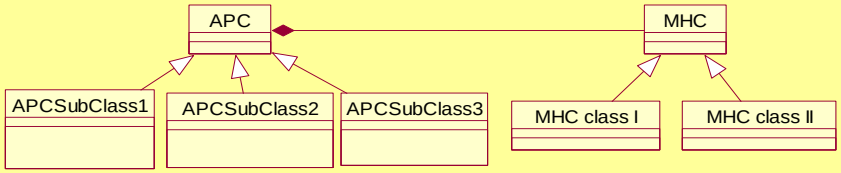

[emptyWay = false]
create
Opened
close [permitType > 3]
[emptyWay = true] / show
open
Closed
CheckKey
enter
NoKey
enterKey
HasKey
checkKey
[validKey = true]
leaveKey
[validKey = false]
stop
lock
unLock
Locked


OpenedDoorState
ClosedDoorState
LockedDoorState
NoKeyCheckKeyDoorState
HasKeyCheckKeyDoorState
AbstractDoorState
close()
open()
lock()
unLock()
enterKey()
checkKey()
leaveKey()
stop()
#states
*
#state
CompositeDoorState
state:AbstractDoorState
states:AbstractDoorState[*]
#state
CheckKeyDoorState
enterKey()
checkKey()
leaveKey()
stop()
DoorClient
door:Door
#door
Door
state:AbstractDoorState
DoorState

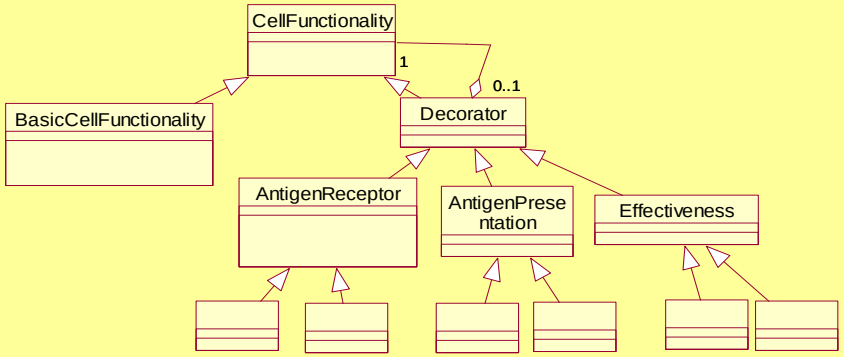
Design Patterns : « precise » your reality

- The “prototype” DP : how a new object is created by cloning an existing one. Shallow copying (a T cell would be cloned without equally cloning the antigen receptor it is composed of) is different from deep copying (where the cloning of the container implies the cloning of the content).
- The “flyweight” DP: looking for common parts in the description of many objects. Forces the programmer and the immunologist to have a clear idea of what is unique to each clone and what is common to all of them (for instance, if the genetic sequence of their receptor is unique, it can be stored only once in the original lymphocyte and make all clones refer to it).
- The “singleton” DP: Class possessing a single object

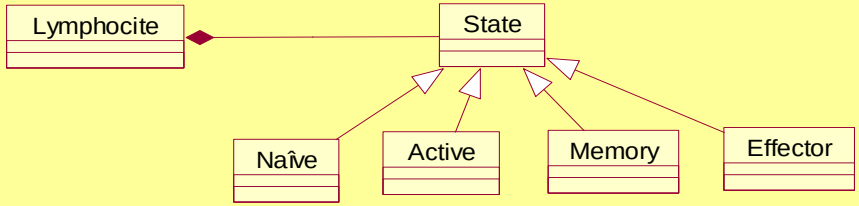
Design Patterns (2)



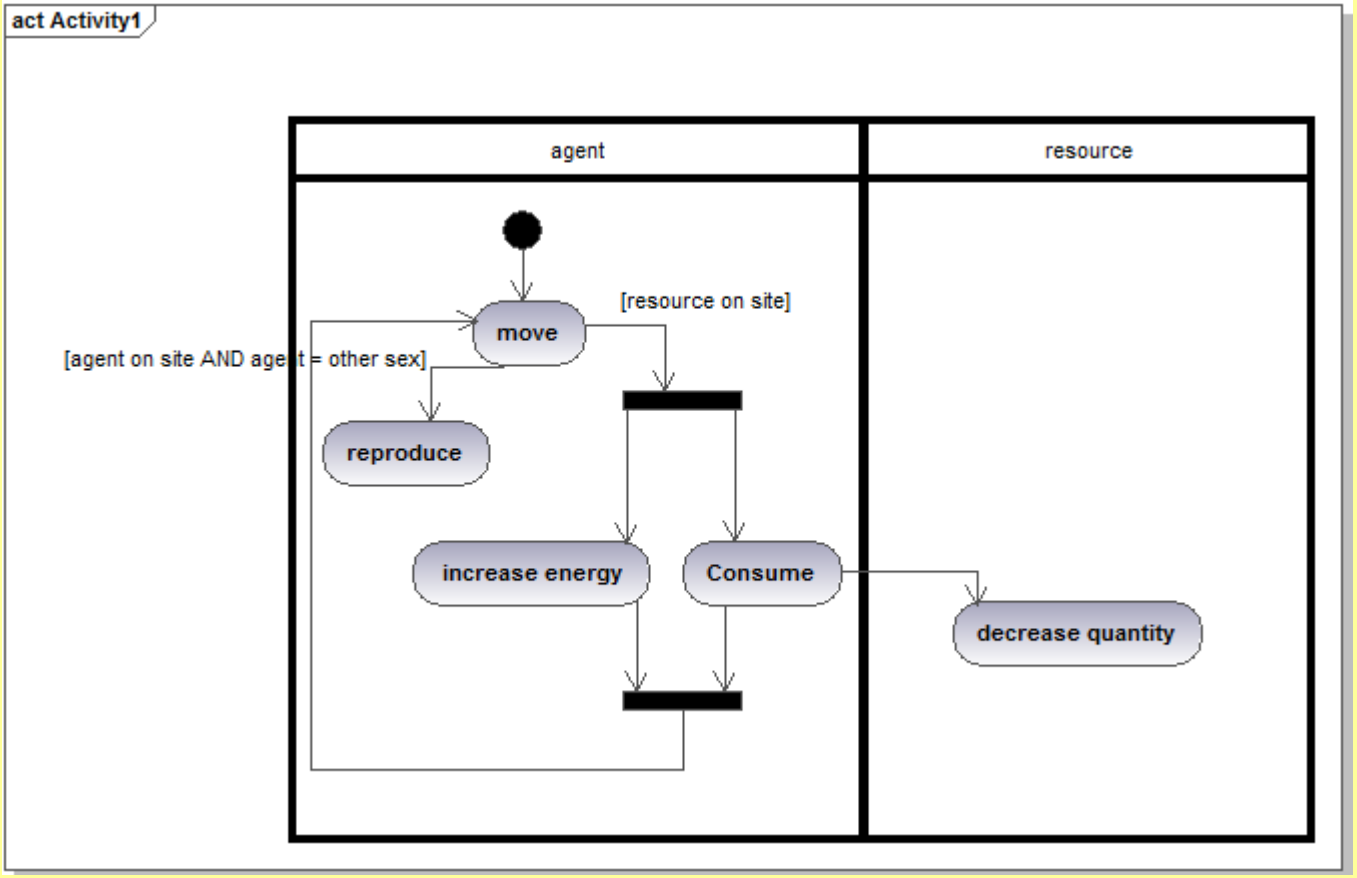
The “bridge DP”

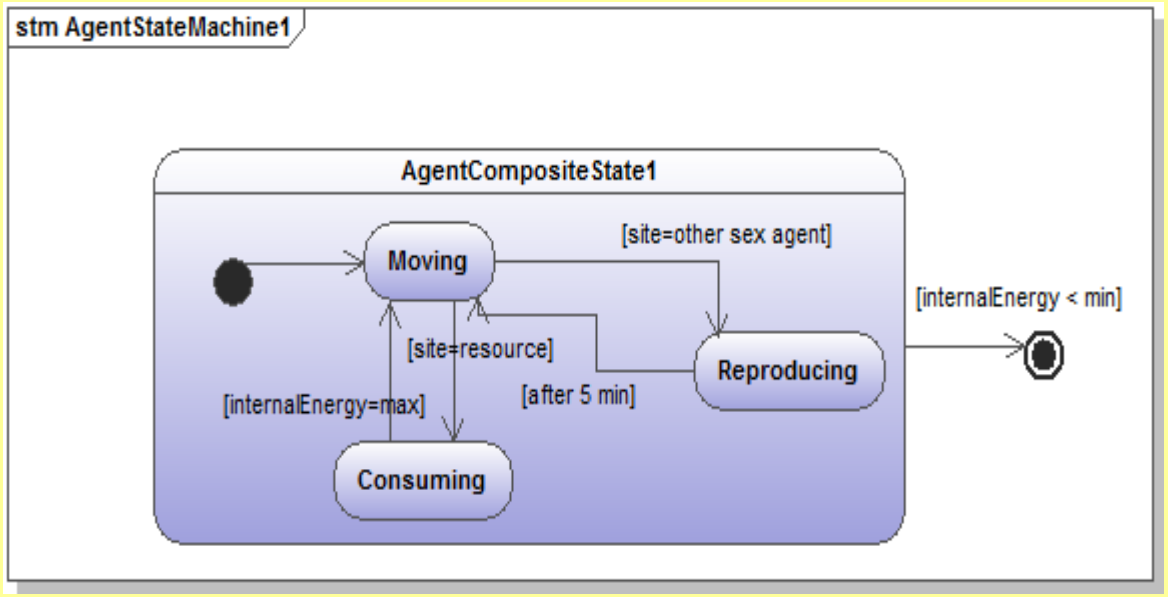


The “Decorator DP”



The “State DP”


act Activity1
agent
resource
move
[resource on site]
[agent on site AND agent = other sex]
reproduce
increase energy
Consume
decrease quantity


stm AgentStateMachine1
AgentCompositeState1
Moving
[site=other sex agent]
[internalEnergy < min]
[site=resource]
Reproducing
[after 5 min]
[internalEnergy=max]
Consuming

Evolutionary Game Theory

The prisoner's dilemma

P1/P2	Cooperate	Compete
Cooperate	(1,1)	(-2,3)
Compete	(3,-2)	(-1,-1)

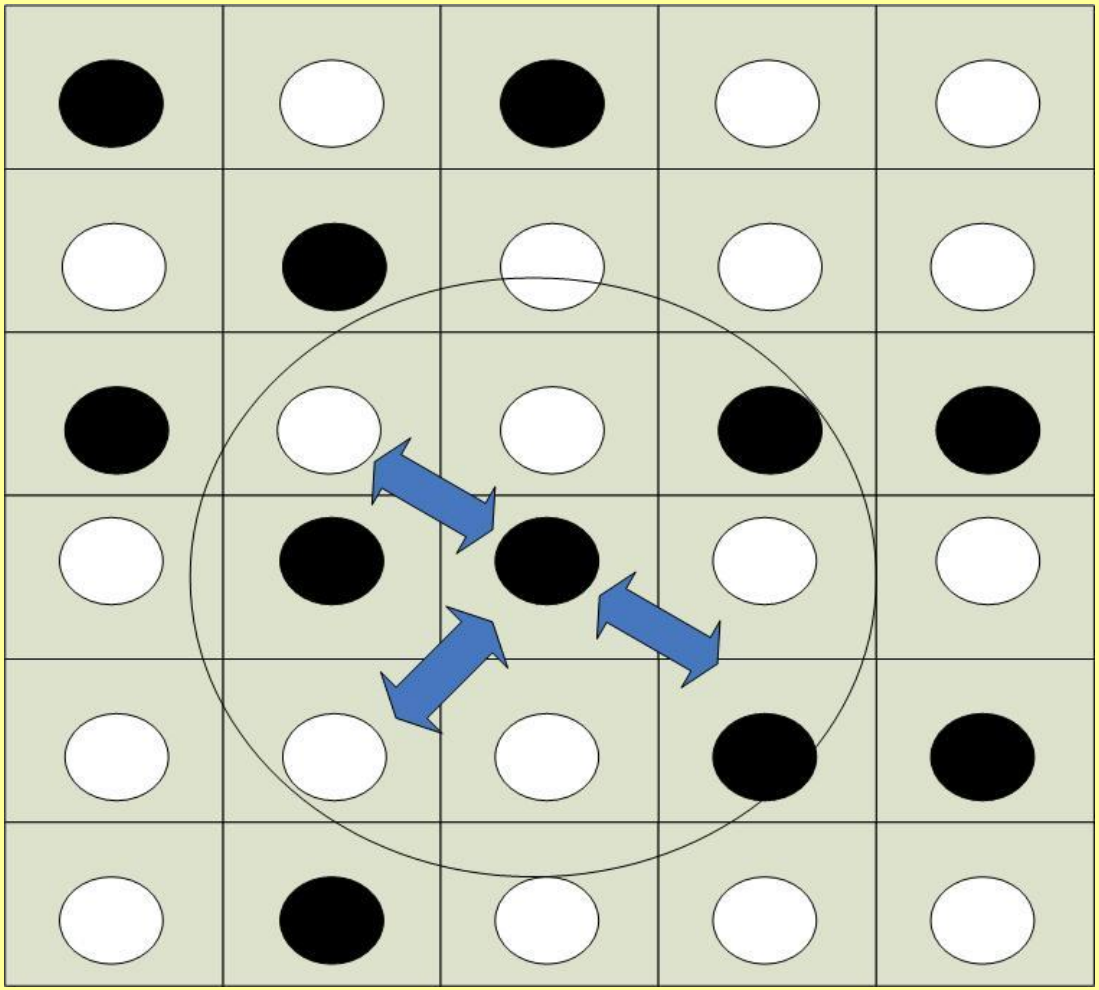
The winning strategy for both players is to compete. But doing so, they miss the cooperating one which is collectively better. The common good is subverted by individual rationality and self-interest.

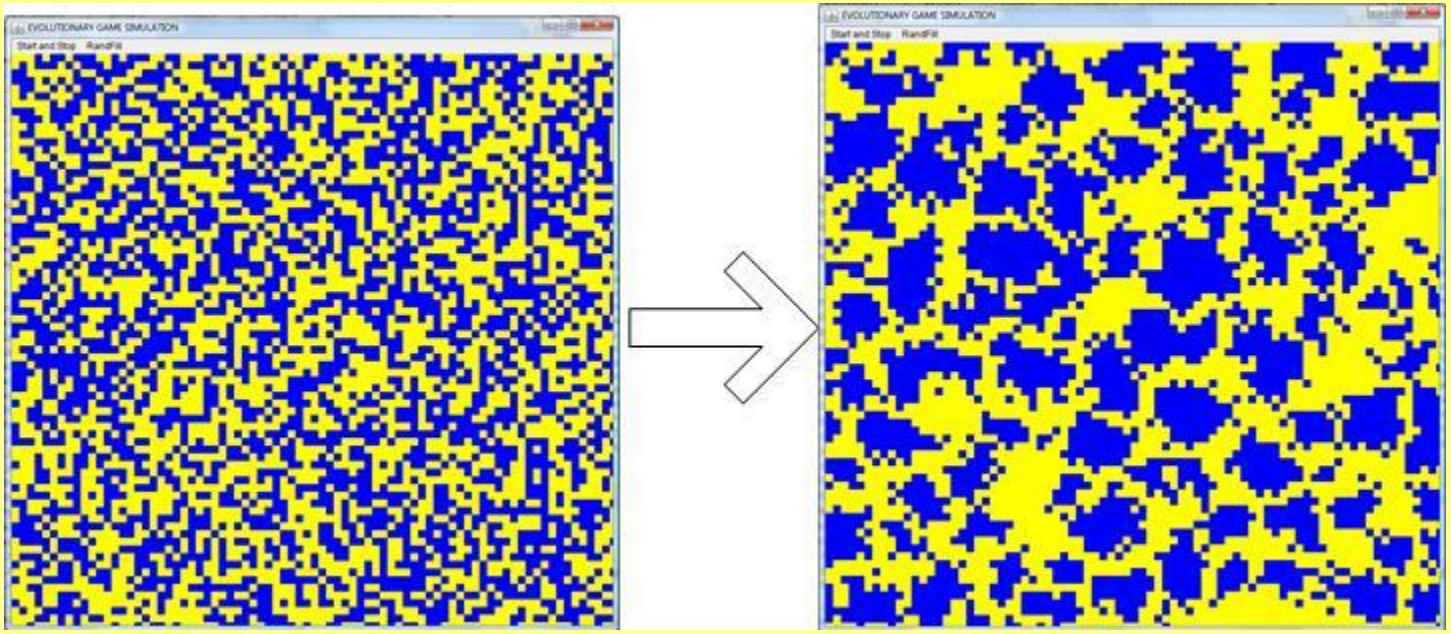
But is competitive behaviour and collective distress avoidable ?

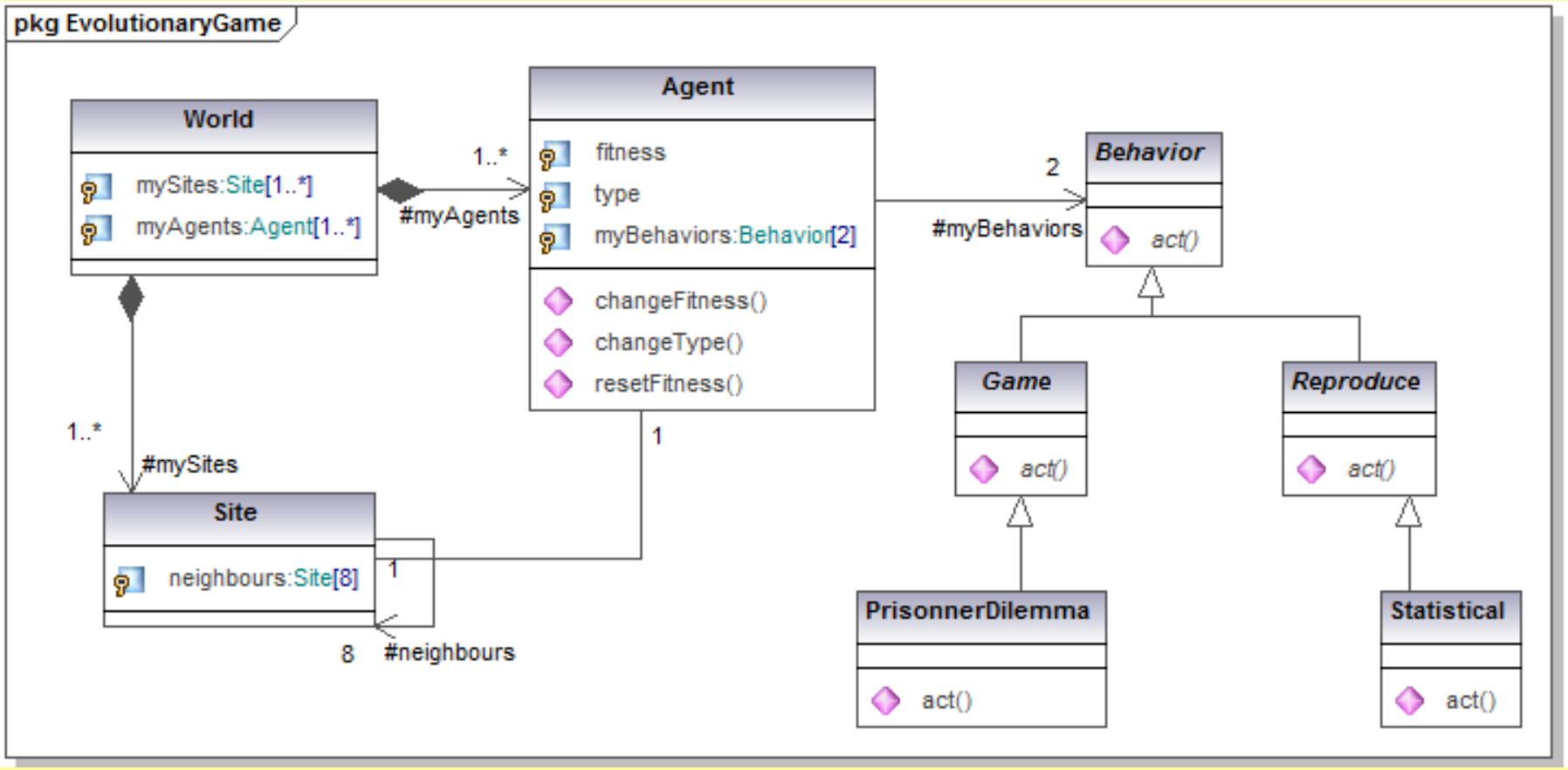
- So far the prisoner's dilemma is lacking some crucial quality that real world situations have.
- 1) Iterated version: play several moves and cumulate your reward over these moves.
  - Cooperation can set in → Tit for Tat
- 2) Distribute spatially the players (CA): each cell just cooperates with its immediate neighbours and adapts the local best strategy. Cluster of nice individuals emerge and can prosper in hostile environments -> EVOLUTIONARY GAME THEORY

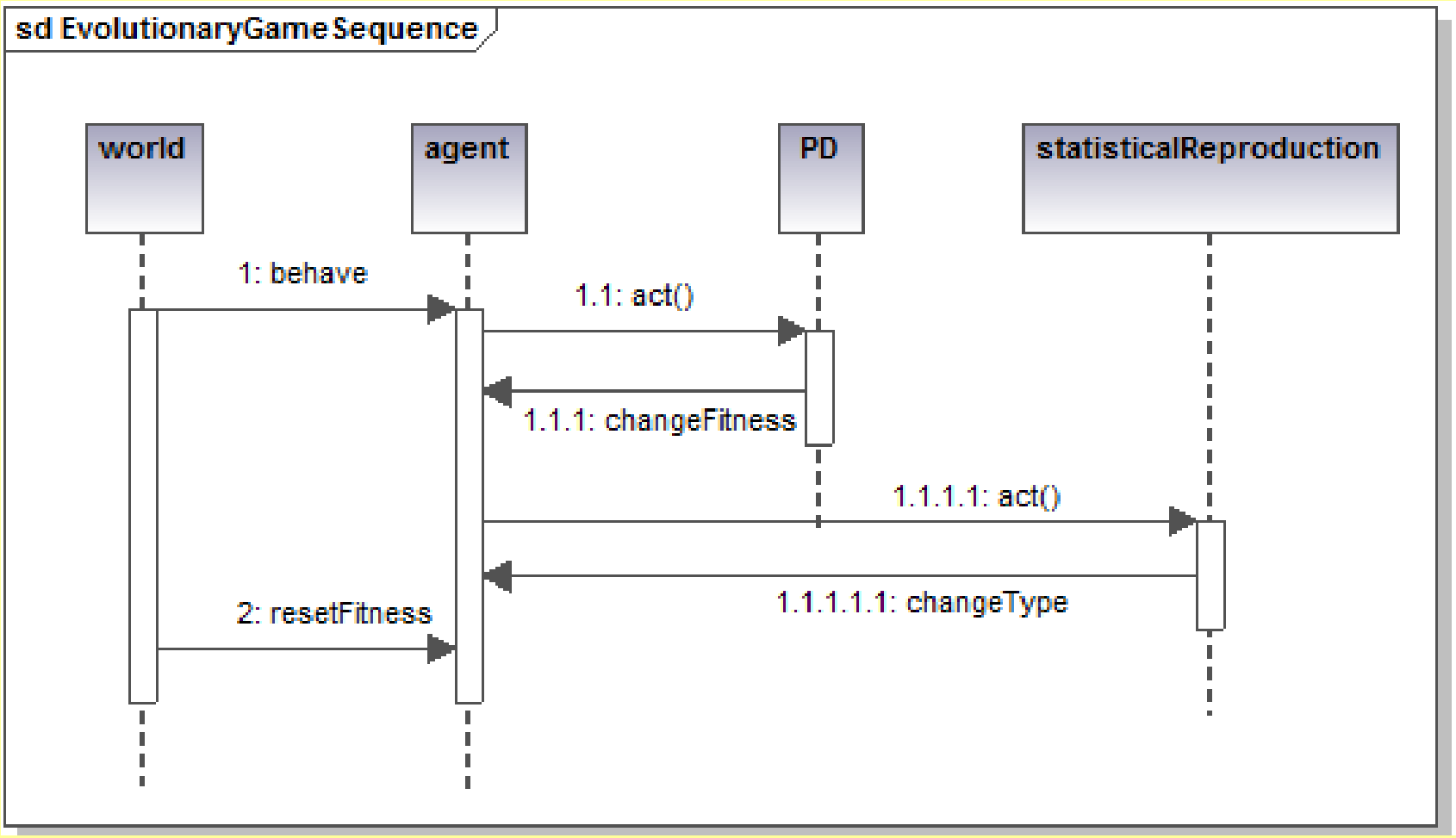
The spatial cellular automata simulation

- Largely inspired by Nowak's work on spatial prisoner dilemma
- A cellular automata in which every cell contains one agent (specialist or generalist)
- In all cells, asynchronously, an agent will subsequently:
  - interact with its neighbors (Moore neighborhood) to "consume" them.
    - Sum the payoff according to the payoff matrix
  - replicate
    - Adopt the identity of the fittest neighbor
- For a given number of iteration steps

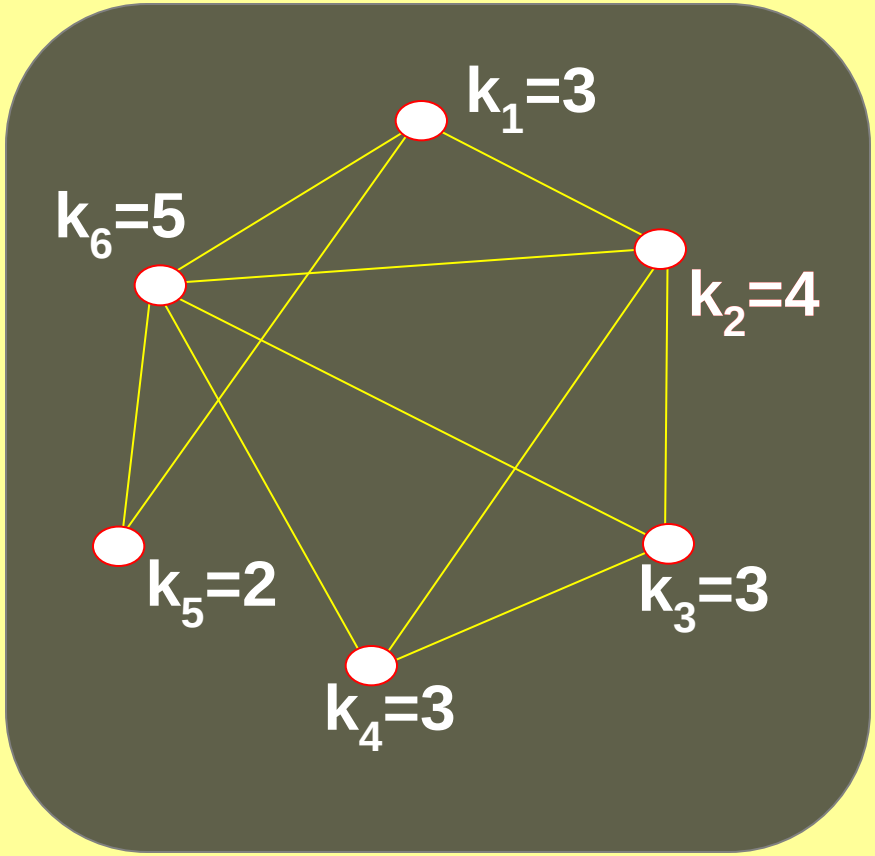



EVOLUTIONARY GAME SIMULATION
Start and Stop
RandFill
EVOLUTIONARY GAME SIMULATION
Start and Stop
RandFill


pkg EvolutionaryGame
World
mySites:Site[1..*]
myAgents:Agent[1..*]
1..*
#myAgents
Agent
fitness
type
myBehaviors:Behavior[2]
changeFitness()
changeType()
resetFitness()
2
#myBehaviors
Behavior
act()
1..*
#mySites
1
Site
neighbours:Site[8]
1
8
#neighbours
Game
act()
Reproduce
act()
PrisonnerDilemma
act()
Statistical
act()


sd EvolutionaryGameSequence
world
agent
PD
statisticalReproduction
1: behave
1.1: act()
1.1.1: changeFitness
1.1.1.1: act()
1.1.1.1.1: changeType
2: resetFitness

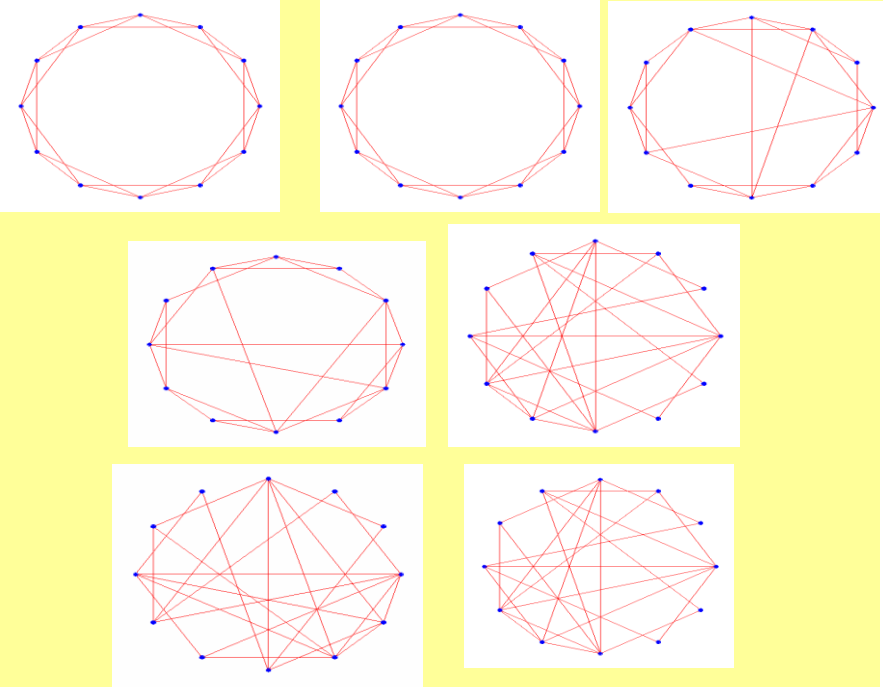
Evolutionary strategy on networks

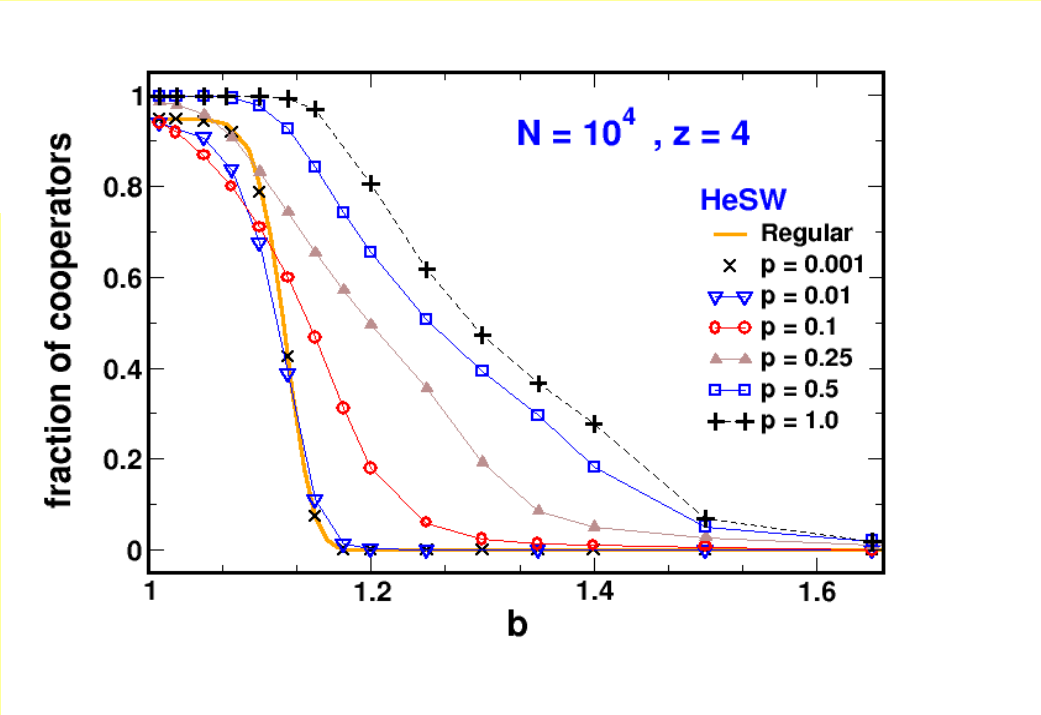


- Stochastic replicator dynamics:
  - Vertex x plays k_x times per generation and accumulates payoff f_x.
  - Choose a random neighbor y with payoff f_y.
  - Replace strategy m_x by m_y with probability:

$$p = \max\left[0, \frac{f_x - f_y}{k_>(T-S)}\right]$$

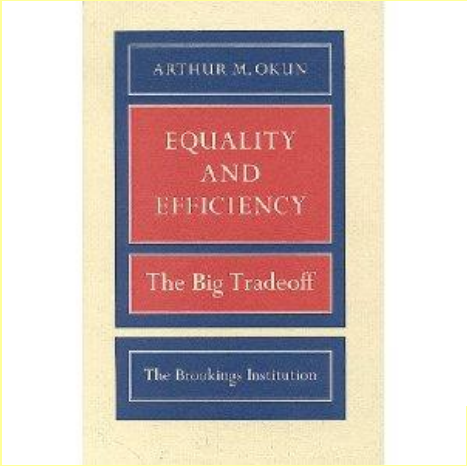
Games on graphs

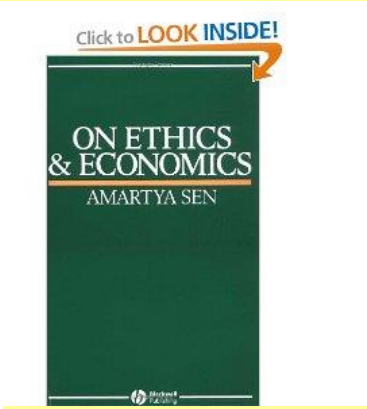




- Conclusions:
  - The more heterogeneous, the more cooperative.
  - Cs benefit most from heterogeneity.

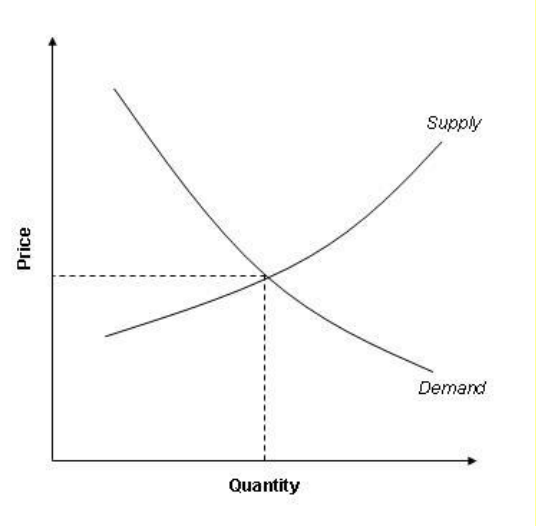
The efficiency/equality trade-off in economy





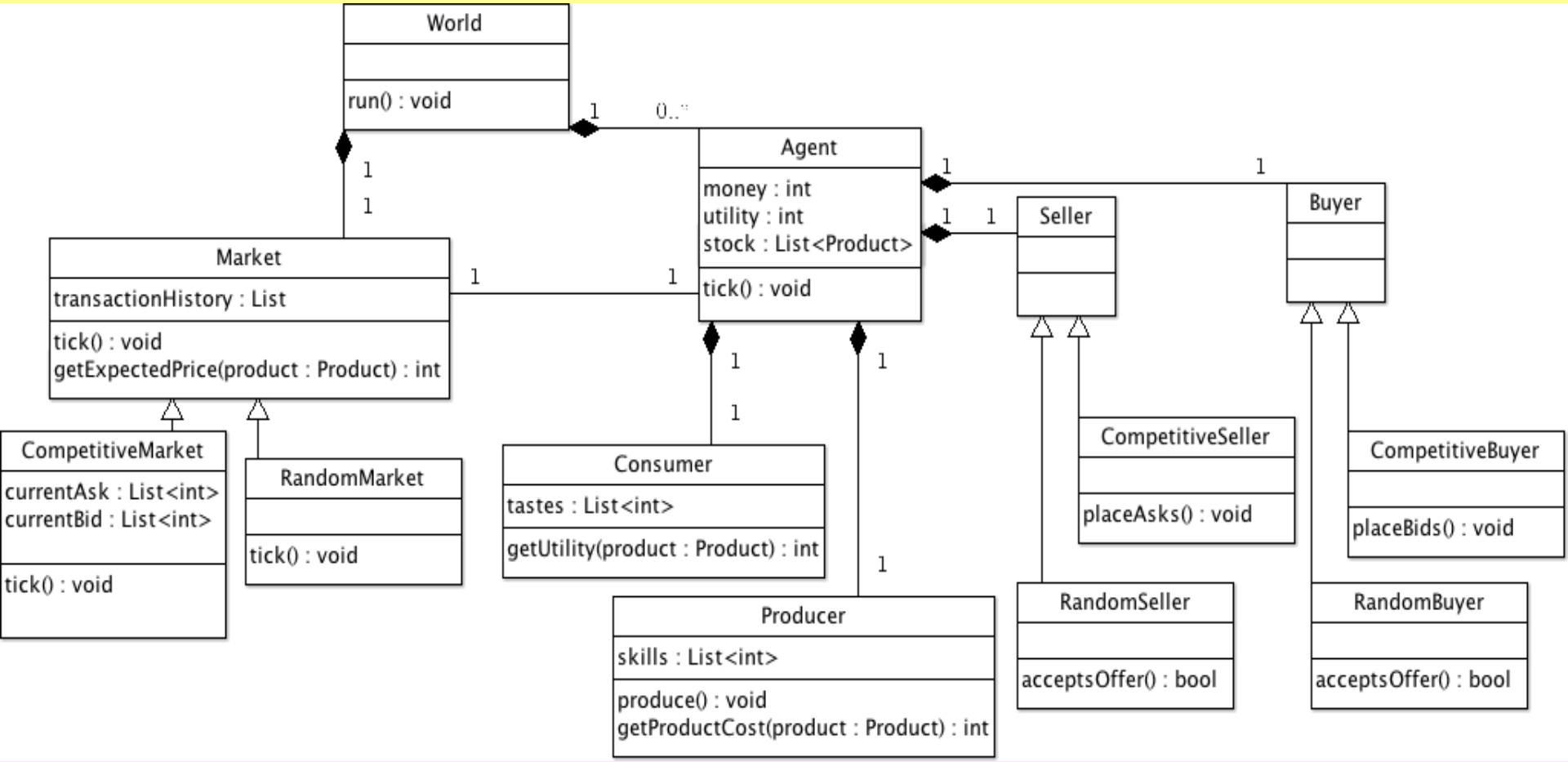
If both equality and efficiency are valued,
and neither takes absolute priority over the other,
then, in places where they conflict,
compromises ought to be struck.
In such cases, some equality will be sacrificed
for the sake of efficiency and some efficiency for the sake of equality.
But any sacrifice of either has to be justified
as a necessary means of obtaining more of the other.

Free market : the perfect concurrence

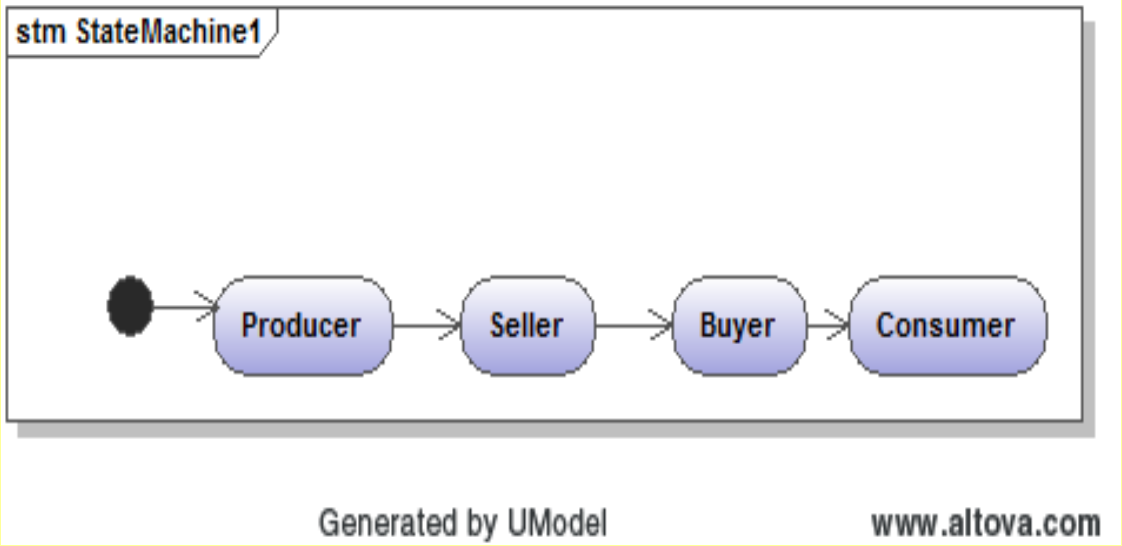


Buyers compete by raising up the buying price and sellers compete by decreasing the selling offer. Buyers are happy of the sellers competition and vice versa. But is that moral ????

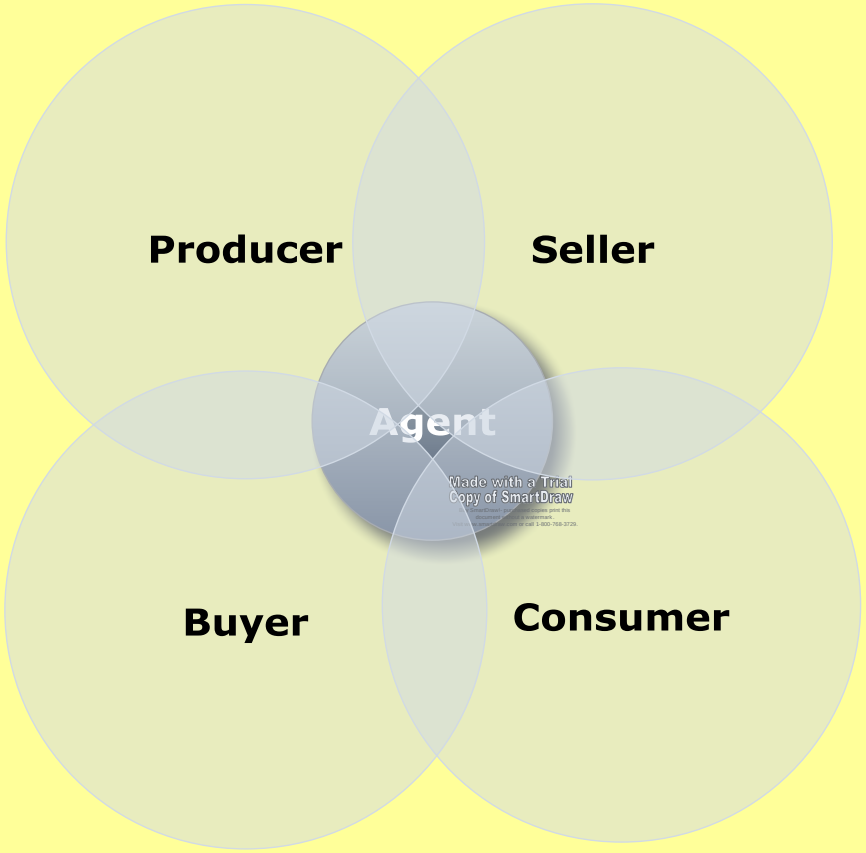
Stylised simulation



Agent’s behaviour



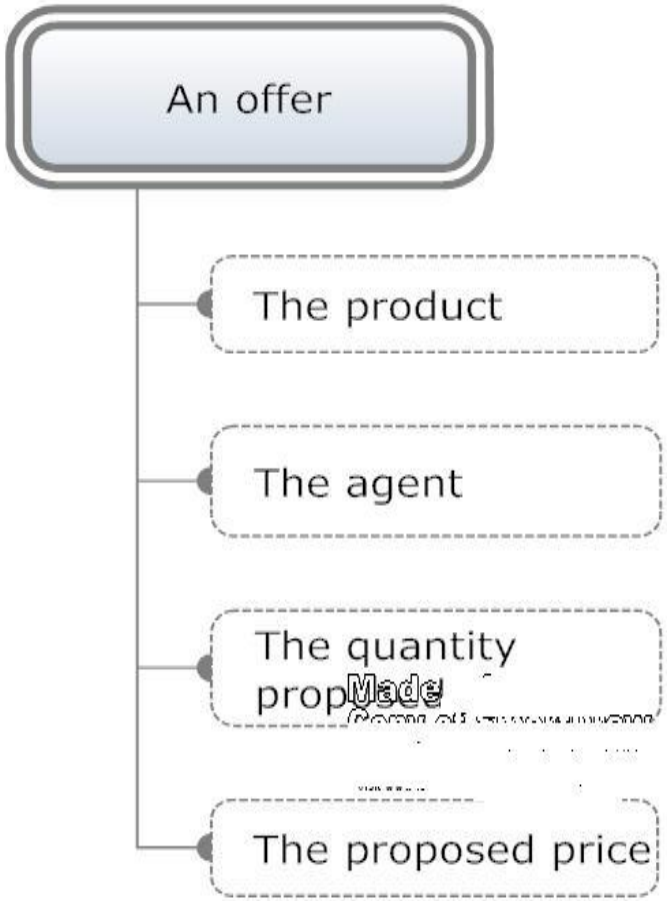
2) Agent = welfare + money + tastes + skills



Tastes and skills are random vectors of products

- The consumer consumes what he has bought. His welfare increases.
- The producer produces either randomly or as a function of the current market. Price feedback effect on the production. The resources decrease. His money decreases.
- The seller makes a selling offer composed of the products just produced: price = fabrication price + benefit (either random in a distributive economy or invertely related to his richness in a competitive one)
- The buyer makes a buying offer. He selects a product out of the blackboard (closest to his tastes). His price is the selling price + a bid (either random in a distributive economy or related to his richness in a competitive one).

3) Offer / either selling or buying



Distributive vs competitive

Distributive:

Selling Offer	Agent	Product	Quantity	Price	UnitPrice
1	A	P1	100	200	2
2	B	P2	50	100	2
3	C	P3	50	50	1
4	A	P1	25	100	4
5	B	P2	50	100	2

Buying Offer	Agent	Product	Quantity	Price	UnitPrice
1	A	P1	50	150	3
2	D	P2	100	600	6
3	B	P1	25	100	4
4	C	P3	25	75	3
5	B	P3	100	300	3

Random Transaction

Fig. 2. The two lists of offers which compose the blackboard, above the selling list below the buying list.

Random choice of the transaction:

Selling offer 3 and Buying offer 4

Competitive:

Selling Offer	Agent	Product	Quantity	Price	UnitPrice
1	A	P1	100	200	2
2	B	P2	50	100	2
3	C	P3	50	50	1
4	A	P1	25	100	4
5	B	P2	50	100	2

Buying Offer	Agent	Product	Quantity	Price	UnitPrice
1	A	P1	50	150	3
2	D	P2	100	600	6
3	B	P1	25	100	4
4	C	P3	25	75	3
5	B	P3	100	300	3

Fig. 2. The two lists of offers which compose the blackboard, above the selling list below the buying list.

Competitive economy:
Max(buyingPrice - sellingPrice)

choice of the MAX transaction:
SellingOffer 2 and BuyingOffer2

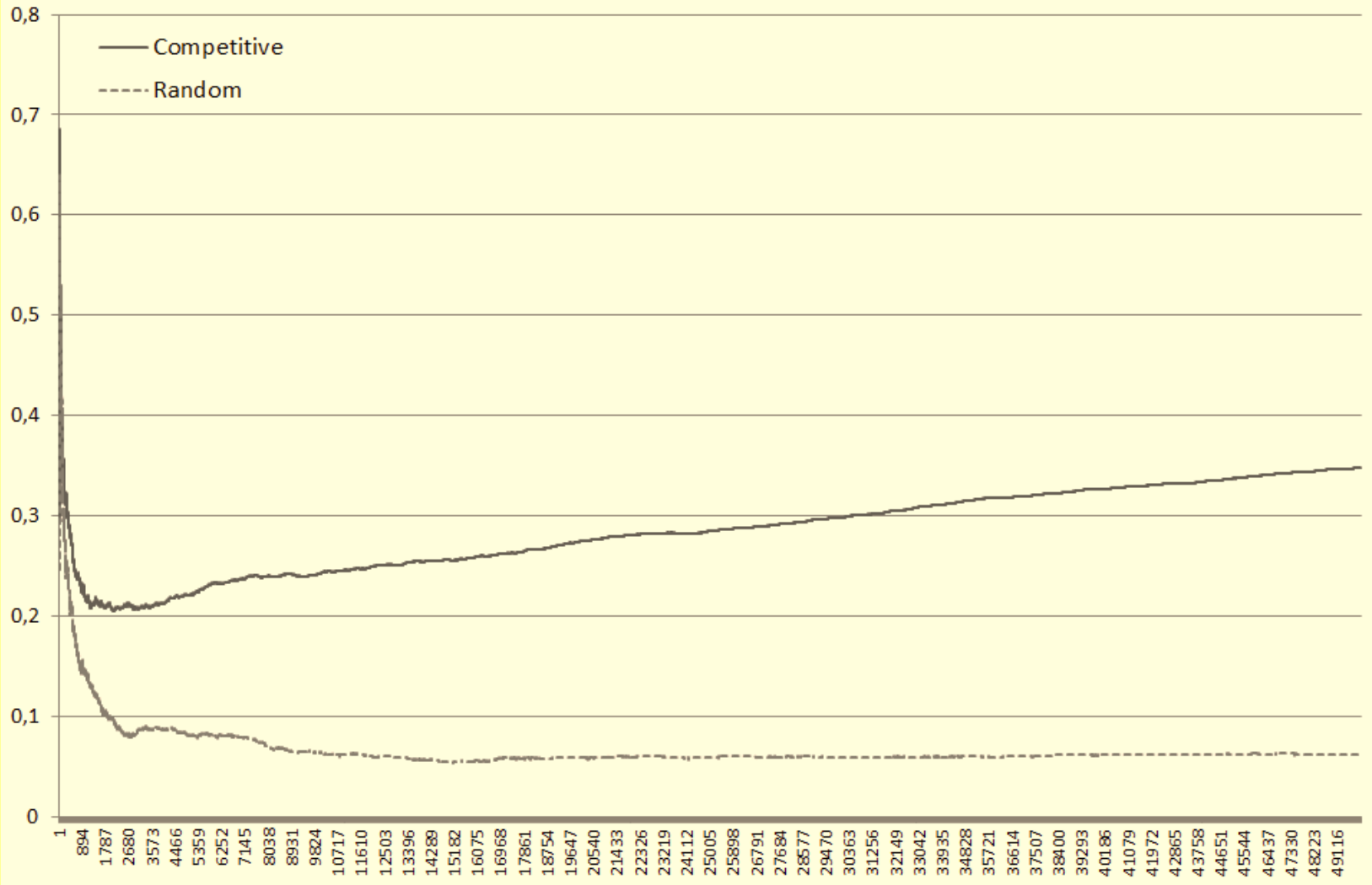
Preliminary results:

100 agents (the same initial money and welfare)

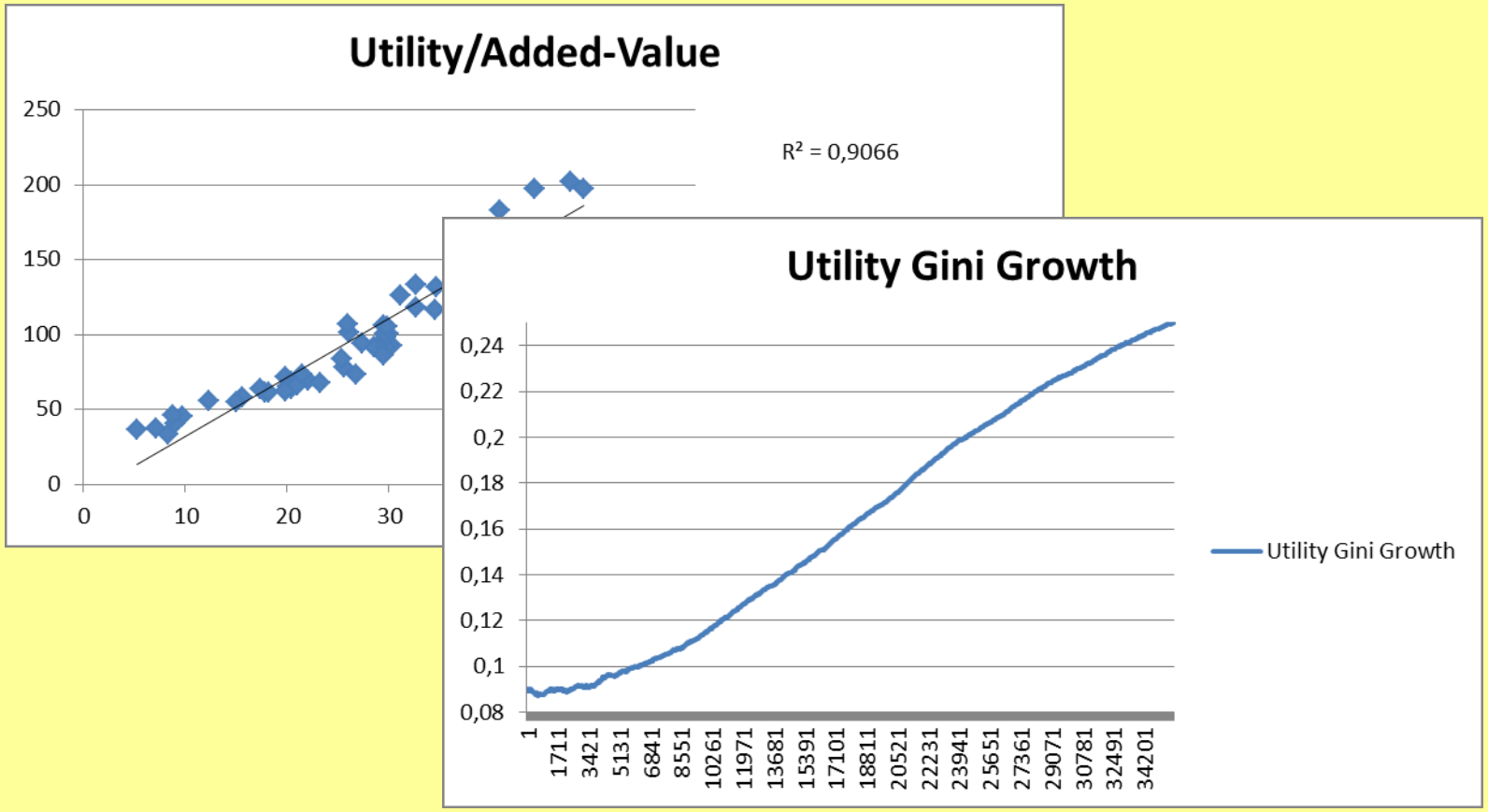
Random skills and tastes

Money	100	80	60	50	25	20	15	10
Utility	5465	5340	5464	5452	5433	5401	5330	5246
Money	4182	3275	2093	1735	494	284	83	17
Ad. Val.	3680	3720	3264	3380	2675	2400	1817	1424
G(Util)	0.05	0.061	0.077	0.070	0.088	0.12	0.16	0.25
G(Mon)	0.006	0.007	0.013	0.010	0.016	0.017	0.014	0.08
G(AV)	0.08	0.092	0.11	0.10	0.13	0.15	0.17	0.20
MF	0	0	0	0	0	26	270	914

Gini index


Competitive
Random
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1
894
1787
2680
3573
4466
5359
6252
7145
8038
8931
9824
10717
11610
12503
13396
14289
15182
16075
16968
17861
18754
19647
20540
21433
22326
23219
24112
25005
25898
26791
27684
28577
29470
30363
31256
32149
33042
33935
34828
35721
36614
37507
38400
39293
40186
41079
41972
42865
43758
44651
45544
46437
47330
48223
49116

Indice de Gini



Comparison lottery/concurrence

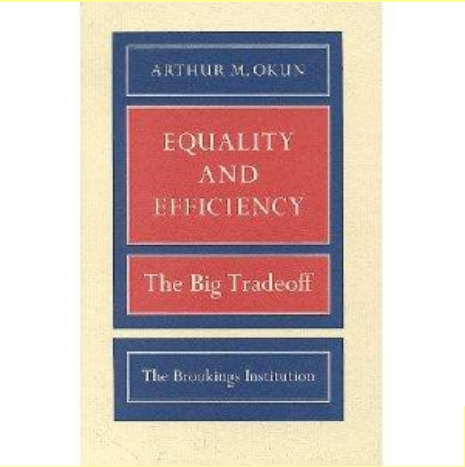
Lottery

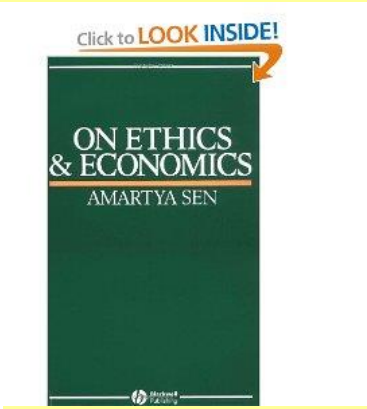
- Low Gini → More equality
- Low aggregate utility → Less efficient

Concurrence

- High Gini → Less equality
- High aggregate utility → More efficient

Fondamental economy dilemma





Solutions: Redistributions (But Okun), regulation, random economy