

# MASON Retirement Age

Ernesto Carrella

RA

May 2 2011

# Outline

- 1 Introduction
- 2 Replication
- 3 Object Oriented Programming
- 4 The Joy of Garbage Collection

# The two souls of the paper

- An experiment in replication

# The two souls of the paper

- An experiment in replication
- An experiment in MASON

- If we choose a very simple model, can we achieve “numerical identity”

# Replication

- If we choose a very simple model, can we achieve “numerical identity”
- No

# Replication

- If we choose a very simple model, can we achieve “numerical identity”
- No
- Numerical identity is really hard

- *Models are completely independent from visualization, which can be added, removed, or changed at any time*



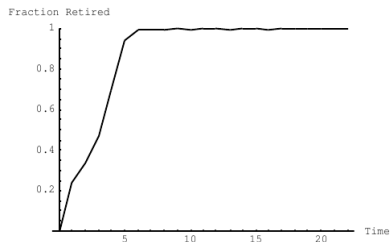
- *Models are completely independent from visualization, which can be added, removed, or changed at any time*
- Can we believe advertisement?

- *Models are completely independent from visualization, which can be added, removed, or changed at any time*
- Can we believe advertisement?
- No, not really.

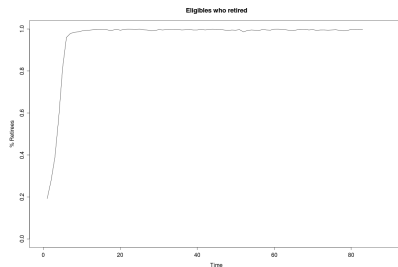
# Outline

- 1 Introduction
- 2 Replication**
- 3 Object Oriented Programming
- 4 The Joy of Garbage Collection

# The Good



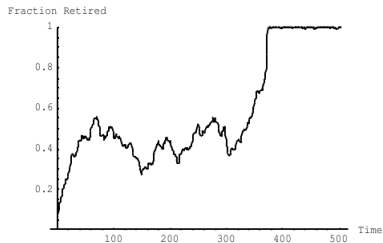
(a) Original Paper



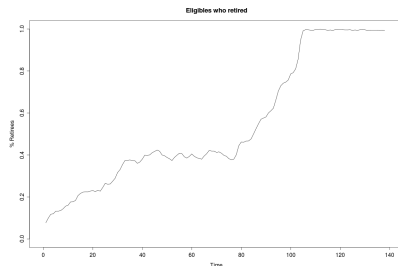
(b) MASON replication

Figure: 20% rational agents case comparison

# The Good



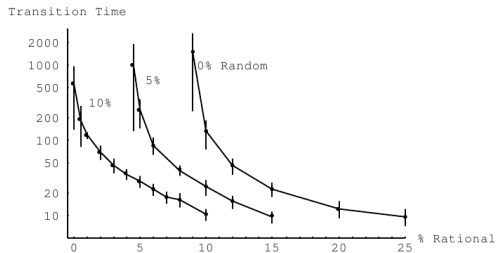
(a) Original Paper



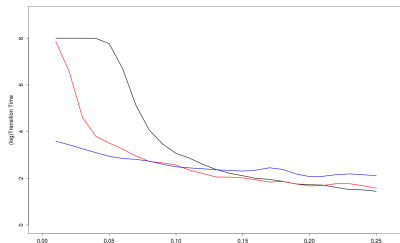
(b) MASON replication

Figure: 5% rational agents case comparison

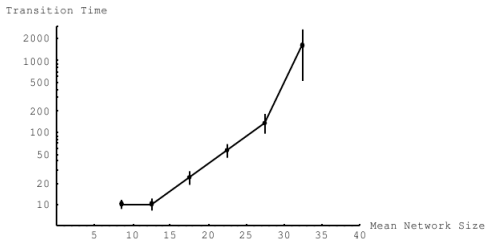
# The good



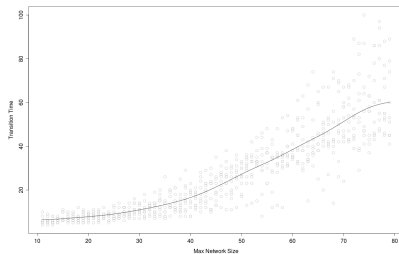
(a) Original Paper



# The bad



(a) Original Paper



# The really bad

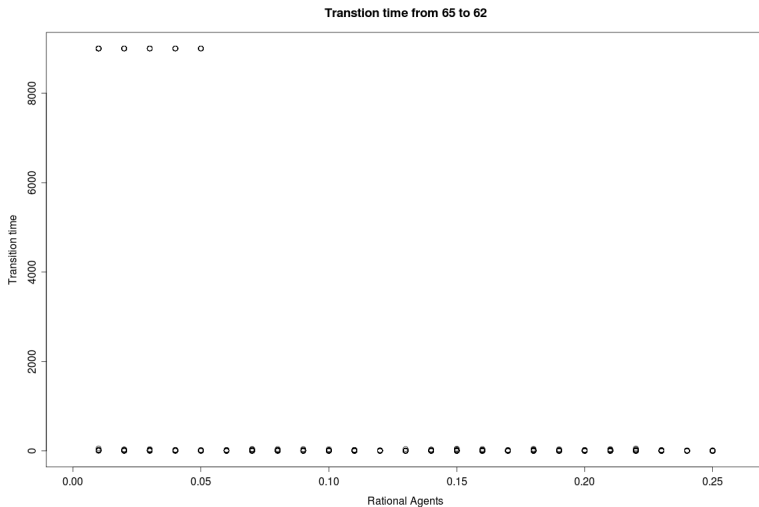
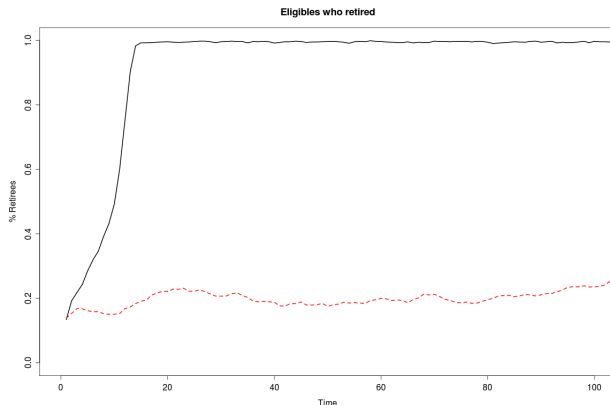


Figure: Weird dynamics



# The weird



**Figure:** The black line is the simulation with strict threshold, the red line is without

# Outline

- 1 Introduction
- 2 Replication
- 3 Object Oriented Programming**
- 4 The Joy of Garbage Collection

# Amateur Perspective

- Never write twice

# Amateur Perspective

- Never write twice
- Small meaningful functions

# Amateur Perspective

- Never write twice
- Small meaningful functions
- Implementation Hiding

# Amateur Perspective

- Never write twice
- Small meaningful functions
- Implementation Hiding ... sort of

- Three types of agents

# Agent Type

- Three types of agents
  - ▶ Rational



# Agent Type

- Three types of agents
  - ▶ Rational
  - ▶ Random

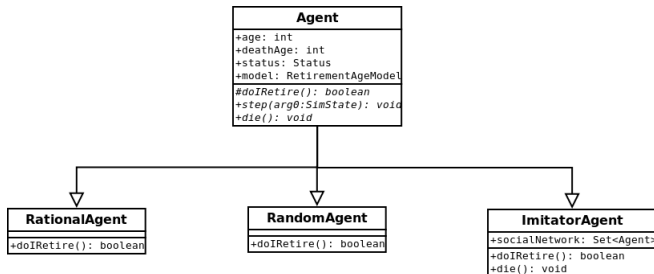
- Three types of agents
  - ▶ Rational
  - ▶ Random
  - ▶ Imitator

- Three types of agents
  - ▶ Rational
  - ▶ Random
  - ▶ Imitator
- Keep them as a class

# Agent Type

- Three types of agents
  - ▶ Rational
  - ▶ Random
  - ▶ Imitator
- Keep them as a class
- Instantiate them in proportion

# It's UML Time!



- Interface vs Abstract Class

- Interface vs Abstract Class

```
public void step(SimState arg0) {  
  
    age++;  
  
    if (age >= deathAge)  
        this.die();  
  
    else if (status == Status.WORKING)  
        status = doRetire();  
}
```

# Overriding

- The difference in agents is only in the method *doRetire()*



# Overriding

- The difference in agents is only in the method *doRetire()*
- ImitatorAgent extends *die()*

# Overriding

- The difference in agents is only in the method *doRetire()*
- ImitatorAgent extends *die()*
- All we need to do is schedule them.

# Outline

- 1 Introduction
- 2 Replication
- 3 Object Oriented Programming
- 4 The Joy of Garbage Collection

# The Garbage Collector

- It “automatically” destroys unused objects

# The Garbage Collector

- It “automatically” destroys unused objects
- Automatically: unlinked objects

# The Garbage Collector

- It “automatically” destroys unused objects
- Automatically: unlinked objects
- Cannot be done manually

# Delete agents

- Do we really need to destroy them?

# Delete agents

- Do we really need to destroy them?
- Yes



# Delete agents

- Do we really need to destroy them?
- Yes
- What links to them?

# Delete agents

- Do we really need to destroy them?
- Yes
- What links to them?
  - ▶ Simstate's table containing them

# Delete agents

- Do we really need to destroy them?
- Yes
- What links to them?
  - ▶ Simstate's table containing them
  - ▶ and that's it?

# Schedule and Scheduling

- We schedule each agent separately

# Schedule and Scheduling

- We schedule each agent separately
- *Schedule.scheduleRepeating()*

# Schedule and Scheduling

- We schedule each agent separately
- *Schedule.scheduleRepeating()*
- How does schedule works?

# Schedule and Scheduling

- We schedule each agent separately
- *Schedule.scheduleRepeating()*
- How does schedule works?
- Heap

# How to remove a repeating steppable from the heap

- Make the array steppable instead



# How to remove a repeating steppable from the heap

- Make the array steppable instead
- Use Stoppable

# How to remove a repeating steppable from the heap

- Make the array steppable instead
- Use Stoppable
- **public** *Stoppable scheduleRepeating(Steppable agent)*

# Delete agents 2

- Do we really need to destroy them?
- Yes
- What links to them?
  - ▶ Simstate's table containing them
  - ▶ Schedule

# Delete agents 2

- Do we really need to destroy them?
- Yes
- What links to them?
  - ▶ Simstate's table containing them
  - ▶ Schedule
  - ▶ and that's it?

- A set of Agents

# Social Network

- A set of Agents
- These are references to your friends

# Social Network

- A set of Agents
- These are references to your friends
- As long as one of your friend is alive (or a friend of that friend...) you will not be recycled

# Social Network

- A set of Agents
- These are references to your friends
- As long as one of your friend is alive (or a friend of that friend...) you will not be recycled
- This created enormous slowdowns, even after fixing for the other two



# Social Network

- A set of Agents
- These are references to your friends
- As long as one of your friend is alive (or a friend of that friend...) you will not be recycled
- This created enormous slowdowns, even after fixing for the other two
- Morale: Garbage collector thinks you are smarter than you are