The Art of Iterating: Update-Strategies in Agent-Based Simulation

Jonathan Thaler
jonathan.thaler@nottingham.ac.uk
School of Computer Science, University of Nottingham

Peer-Olaf Siebers
peer-olaf.siebers@nottingham.ac.uk
School of Computer Science, University of Nottingham

Abstract

When developing a model for an Agent-Based Simulation (ABS) it is very important to select the update-strategy which reflects the semantics of the model as simulation results can vary vastly across different update-strategies. This awareness, we claim, is still underdeveloped in the majority of the field of ABS. In this paper we propose a new terminology to classify update strategies and then identify different strategies using this terminology. This will allow implementers and researchers in this field to use a general terminology, removing ambiguities when discussing ABS and their models. We will give results of simulating a discrete and a continuous game using our update-strategies and show that in the case of the discrete game only one specific strategy seems to be able to produce its emergent patterns whereas the pattern of the continuous game seems to be robust under varying update-strategies.

Keywords: Agent-Based Simulation, Parallelism, Concurrency, Emergence

1. Introduction

Agent-based simulation (ABS) is a method for simulating the emergent behaviour of a system by modelling and simulating the interactions of its sub-parts, called agents. Examples for an ABS is simulating the spread of an epidemic throughout a population or simulating the dynamics of segregation within a city. Central to ABS is the concept of an agent who needs to be updated in regular intervals during the simulation so it can interact with other agents and its environment. In this paper we are looking at two different kind of simulations to show the differences update-strategies can make in ABS and support our main message that *when developing a model for an ABS it is of most importance to select the right update-strategy which reflects and supports the corresponding semantics of the model*. As we will show due to conflicting ideas about update-strategies this awareness is yet still under-represented in the field of ABS and is lacking a systematic treatment. As a remedy we undertake such a systematic treatment in proposing a new terminology by identifying properties of ABS and deriving all possible update-strategies. The outcome is a terminology to communicate in a unified way about this very important matter, so researchers and implementers can talk about it in a common way, enabling better discussions, same understanding and better reproducibility and continuity in research. The two simulations we use are discrete and continuous games where in the former one the agents act synchronized at discrete time-steps whereas in the later one they act continuously in continuous time. We show that in the case of simulating the discrete game the update-strategies have a huge impact on the final result whereas our continuous game seems to be stable under different update-strategies. The contribution of this paper is: Identifying general properties of ABS, deriving update-strategies from these properties and establishing a general terminology for talking about these update-strategies.

2. Background

In this section we define our understanding of *agent* and ABS and how we understand and use it in this paper. Then we will give a description of the two kind of games which were the motivators for our research and case-studies. Finally we will present related work.

2.1. Agent-Based Simulation

We understand ABS as a method of modelling and simulating a system where the global behaviour may be unknown but the behaviour and interactions of the parts making up the system is known. Those parts, called agents, are modelled and simulated out of which then the aggregate global behaviour of the whole system emerges. So the central aspect of ABS is the concept of an agent which can be understood as a metaphor for a pro-active unit, situated in an environment, able to spawn new agents and interacting with other agents in a network of neighbours by exchange of messages (Wooldridge, 2009). It is important to note that we focus our understanding of ABS on a very specific kind of agents where the focus is on communicating entities with individual, localized behaviour from out of which the global behaviour of the system emerges. We informally assume the following about our agents:

- They are uniquely addressable entities with some internal state.
- They can initiate actions on their own e.g. change their internal state, send messages, create new agents, kill themselves.
- They can react to messages they receive with actions as above.
- They can interact with an environment they are situated in.

An implementation of an ABS must solve two fundamental problems:

1. **Source of pro-activity** How can an agent initiate actions without the external stimuli of messages?
2. **Semantics of Messaging** When is a message m, sent by agent A to agent B, visible and processed by B?

In computer systems, pro-activity, the ability to initiate actions on its own without external stimuli, is only possible when there is some internal stimulus, most naturally represented by a continuous increasing time-flow. Due to the discrete nature of computer-system, this time-flow must be discretized in steps as well and each step must be made available to the agent, acting as the internal stimulus. This allows the agent then to perceive time and become pro-active depending on time. So we can understand an ABS as a discrete time-simulation where time is broken down into continuous, real-valued or discrete natural-valued time-steps. Independent of the representation of the time-flow we have the two fundamental choices whether the time-flow is local to the agent or whether it is a system-global time-flow. Time-flows in computer-systems can only be created through threads of execution where there are two ways of feeding time-flow into an agent. Either it has its own thread-of-execution or the system creates the illusion of its own thread-of-execution by sharing the global thread sequentially among the agents where an agent has to yield the execution back after it has executed its step. Note the similarity to an operating system with cooperative multitasking in the latter case and real multi-processing in the former.

The semantics of messaging define when sent messages are visible to the receivers and when the receivers process them. Message-processing could happen either immediately or delayed, depending on how message-delivery works. There are two ways of message-delivery: immediate or queued. In the case of immediate message-deliver the message is sent directly to the agent without any queuing in between e.g. a direct method-call. This would allow an agent to immediately react to this message as this call of the method transfers the thread-of-execution to the agent. This is not the case in the queued message-delivery where messages are posted to the message-box of an agent and the agent pro-actively processes the message-box at regular points in time.

2.2. A discrete game: Prisoners Dilemma

As an example of a discrete game we use the *Prisoners Dilemma* as presented in (Nowak & May, 1992). In the prisoners dilemma one assumes that two persons are locked up in a prison and can choose to cooperate with each other or to defect by betraying the other one. Looking at a game-theoretic approach there are two options for each player which makes four possible outcomes. Each outcome is associated with a different payoff in the prisoner-dilemma. If both players cooperate both receive payoff R; if one player defects and the other cooperates the defector receives payoff T and the cooperator payoff S; if both defect both receive payoff P where $T > R > P > S$. The dilemma is that the safest strategy for an individual is to defect but the best payoff is only achieved when both cooperate. In the version of (Nowak & May, 1992) NxN agents are arranged on a 2D-grid where every agent has 8 neighbours except at the edges. Agents don't have a memory of the past and have one of two roles: either cooperator or defector. In every step an agent plays the game with all its neighbours, including itself and sums up the payoff. After the payoff sum is calculated the agent changes its role to the role of the agent with the highest payoff within its

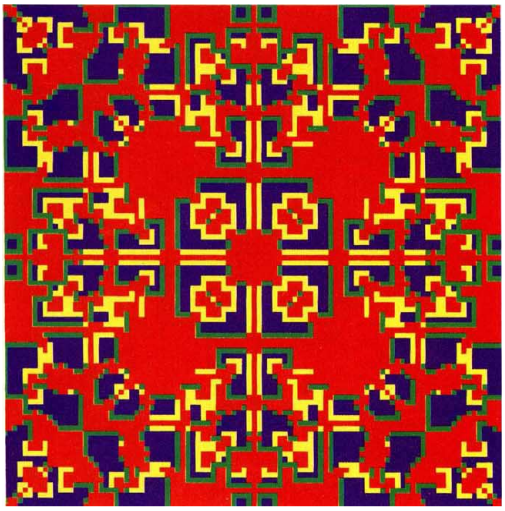


Figure 1: Patterns formed by playing the *Prisoners Dilemma* game on a 99x99 grid with $1.8 < b < 2$ after 217 steps with all agents being cooperators except one defector at the center. Blue are cooperators, red are defectors, yellow are cooperators which were defectors in the previous step, green are defectors which were cooperators in the previous step. Picture taken from (Nowak & May, 1992).

neighbourhood (including itself). The authors attribute the following payoffs: S=P=0, R=1, T>b, where b>1. They showed that when having a grid of only cooperators with a single defector at the center, the simulation will form beautiful structural patterns as shown in Figure 1.

In (Huberman & Glance, 1993) the authors show that the results of simulating the *Prisoners Dilemma* as above depends on a very specific strategy of iterating the simulation and show that the beautiful patterns seen in Figure 1 will not form when selecting a different update-strategy. They introduced the terms of synchronous and asynchronous updates and define synchronous to be as agents being updated in unison and asynchronous where one agent is updated and the others are held constant. Only the synchronous updates are able to reproduce the results. The authors differentiated between the two strategies but their description still lacks precision and detail, something we will provide in this paper. Although they published their work in the area on general computing, it has implications for ABS as well which can be generalized in the main message of our paper as emphasised in the introduction. We will show that there are more than two update-strategies and will give results of simulating this discrete game using all of them. As will be shown later, the patterns emerge indeed only when selecting a specific update-strategy.

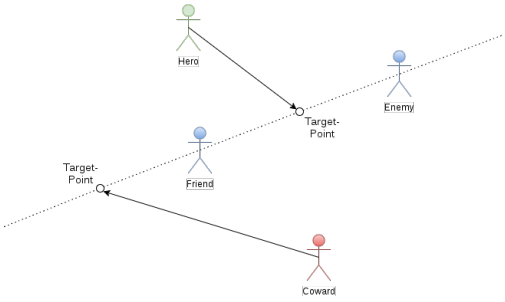


Figure 2: A conceptual diagram of the *Heroes & Cowards* game. Hero (green) and coward (red) have the same agents as friend and enemy but act different: the hero tries to move in between the friend and enemy whereas the coward tries to hide behind its friend.

2.3. A continuous game: Heroes & Cowards

As an example for a continuous game we use the *Heroes & Cowards* game introduced by (Wilensky & Rand, 2015). In this game one starts with a crowd of agents where each agent is positioned randomly in a continuous 2D-space which is bounded by borders on all sides. Each of the agents then selects randomly one friend and one enemy (except itself) and decides with a given probability whether the agent acts in the role of a hero or a coward - friend, enemy and role don't change after the initial set-up. In each step the agent will move a small distance towards a target point as seen in Figure 2. If the agent is in the role of a hero this target point will be the half-way distance between the agents friend and enemy - the agent tries to protect the friend from the enemy. If the agent is acting like a coward it will try to hide behind the friend also the half-way distance between the agents friend and enemy, just in the opposite direction. Note that this simulation is determined by the random starting positions, random friend and enemy selection, random role selection and number of agents. Note also that during the simulation-stepping no randomness is incurred and given the initial random set-up, the simulation-model is completely deterministic. As will be shown later the results of simulating this model are invariant under different update-strategies.

2.4. Related Research

Besides (Nowak & May, 1992) and (Huberman & Glance, 1993) which both discuss asynchronous and synchronous updates, multiple other works mention these kind of updates but the meaning is different in each. Asynchronous updates in the context of cellular automata was defined by (Bersini & Detours, 1994) as picking a cell at random and updating it and syn-

chronous as all cells updating at the same time and report different dynamics when switching between the two. The authors also raise the question which of both is correct and most faithful to reality as in an ideal solution both should deliver the similar spatio-temporal dynamics. They conclude that developers of simulations should pay attention to the fact that the dynamics and results are sensible to the updating procedures. Asynchronous vs. synchronous updates are mentioned in the book of (Wilensky & Rand, 2015) where they define asynchronous updates as having the property that changes made by an agent are seen immediately by the others whereas in synchronous updating the changes are only visible in the next tick. They also look into the notion of sequential vs. parallel actions and identify as sequential when only one agent acts at a time and parallel when agents act truly parallel, independent from each other. The same argumentation is followed by (Railsback & Grimm, 2011) where they discuss the importance of order of execution of the agents and describe asynchronous and synchronous updating. Yet another definition of synchronous and asynchronous updates is given in (Page, 1997). They define asynchronous updating as updating agents sequentially one after another and synchronous updating as updating all agents at virtually the same time. They go further and discuss also random updates where the order of the agent-sequence is shuffled before updating all agents. They also introduce incentive based asynchronous updating where the agent which gains the most from the update is updated first, thus introducing an ordering on the sequence which is sorted by the benefit each agent gains from its update. They also compare the differences synchronous, random-asynchronous and incentive-asynchronous updating has on dynamics and come to the conclusion that the order of updating the agents has an impact on the dynamics and should be considered with great care when implementing a simulation. Asynchronous and synchronous time-models are mentioned in (Dawson, Siebers, & Vu, 2014) where the authors describe basic inner workings of ABS environments and compare their implementation in C++ to the existing ABS environment AnyLogic which is programmed in Java. They interpret asynchronous time-models to be the ones in which an agent acts at random time intervals and synchronous time-models where agents are updated all in same time intervals. A different interpretation of synchronous and asynchronous time-models is given in (Yuxuan, 2016). He identifies the asynchronous time-model to be one in which updates are triggered by the exchange of messages and the synchronous ones which trigger changes immediately without the indirection of messages. A different approach was taken in (Botta, Mandel, & Ionescu, 2010) where they sketch a minimal ABS implementation in Haskell. Their research applies primarily to economic simulations and instead of iterating a simulation with a global time, their focus is on how to synchronize agents which have internal, local transition times. A very different approach to updating and iterating agents in ABS than to mechanisms used in existing software like AnyLogic or NetLogo was given in (Lysenko, D'souza, & Rahmani, 2008) where the authors mapped ABS on GPUs. They discuss execution order at length, highlight the problem of inducing a specific execution-order in a model which is problematic for parallel execution and give solutions how to circumvent these shortcomings. Although we haven't mapped our ideas to GPUs we explicitly include an approach for data-parallelism which can be utilized to roughly map their approach onto our terminology.

3. A new terminology

When looking at related work, we observe that there seems to be a variety of meanings attributed to the terminology of asynchronous and synchronous updates but the very semantic and technical details are unclear and not described very precisely. To develop a standard terminology, we propose to abandon the notion of synchronous and asynchronous updates and, based on the discussion above we propose six properties characterizing the dimensions and details of the internals of an ABS. Having these properties identified we then derive all meaningful and reasonable update-strategies which are possible in a general form in ABS. These update-strategies together with the properties will form the new terminology we propose for speaking about update-strategies in ABS in general. We will discuss all details programming-language agnostic and for each strategy we give a short description, the list of all properties and discuss their semantics, variations and implications selecting update-strategies for a model. A summary of all update-strategies and their properties is given in Table 1.

3.1. ABS Properties

We identified the following properties of agent-based simulations which are necessary to derive and define the differences between the update-strategies.

Iteration-Order Is the collection of agents updated *sequential* with one agent updated after the other or are all agents updated in *parallel*, at virtually the same time?

Global Synchronization Is a full iteration over the collection of agents happening in lock-step at global points in time or not (*yes/no*)?

Thread of Execution Does each agent has a *separate* thread of execution or does it *share* it with all others? Note that it seems to add a constraint on the Iteration-Order, namely that *parallel* execution forces separate threads of execution for all agents. We will show that this is not the case, when looking at the *parallel strategy* in the next section.

Message-Handling Are messages handled *immediately* by an agent when sent to them or are they *queued* and processed later? Here we have the constraint, that an immediate reaction to messages is only possible when the agents share a common thread of execution. Note that we must enforce this constraint as otherwise agents could end up having more than one thread of execution which could result in them acting concurrently by making simultaneous actions. This is something we explicitly forbid as it is against our definition of agents which allows them to have only one thread of execution at a time.

Visibility of Changes Are the changes made (messages sent, environment modified) by an agent which is updated during an Iteration-Order visible (during) *In-Iteration* or only *Post-Iteration* at the next Iteration-Order? More formally: do agents $a_{n>i}$ which are updated after agent a_i see the changes by agent a_i or not? If yes, we refer to *In-Iteration* visibility, to *Post-Iteration* otherwise.

Repeatability Does the ABS has an external source of non-determinism which it cannot influence? If this is the case then we regard an update-strategy as *non-deterministic*, otherwise *deterministic*. It is important to distinguish between *external* and *internal* sources of non-determinism. The former are race-conditions due to concurrency, creating non-deterministic orderings of events which has the consequence that repeated runs may lead to different results with the same configuration, rendering an ABS non-deterministic. The latter, coming from random-number generators, can be controlled using the same starting-seed leading to repeatability and deemed deterministic in this context.

3.2. ABS Update-Strategies

3.2.1. Sequential Strategy. This strategy has a globally synchronized time-flow and in each time-step iterates through all the agents and updates one agent after another. Messages sent and changes to the environment made by agents are visible immediately.

Iteration-Order: Sequential
Global Synchronization: Yes
Thread of Execution: Shared
Message-Handling: Immediate (or Queued)
Visibility of Changes: In-Iteration
Repeatability: Deterministic

Semantics: There is no source of randomness and non-determinism, rendering this strategy to be completely deterministic in each step. Messages can be processed either immediately or queued depending on the semantics of the model. If the model requires to process the messages immediately the model must be free of potential infinite-loops.

Variation: If the sequential iteration from agent [1..n] imposes an advantage over the agents further ahead or behind in the queue (e.g. if it is of benefit when making choices earlier than others in auctions or later when more information is available) then one could use random-walk iteration where in each time-step the agents are shuffled before iterated. Note that although this would introduce randomness in the model the source is a random-number generator implying it is still deterministic. If one wants to have a very specific ordering, e.g. 'better performing' agents first, then this can be easily implemented too by exposing some sorting-criterion and sorting the collection of agents after each iteration.

3.2.2. Parallel Strategy. This strategy has a globally synchronized time-flow and in each time-step iterates through all the agents and updates them in parallel. Messages sent and changes to the environment made by agents are visible in the next global step. We can think about this strategy in a way that all agents make their moves at the same time.

Iteration-Order: Parallel
Global Synchronization: Yes
Thread of Execution: Separate (or Shared)
Message-Handling: Queued
Visibility of Changes: Post-Iteration
Repeatability: Deterministic

Semantics: If one wants to change the environment in a way that it would be visible to other agents this is regarded as a systematic error in this strategy. First it is not logical because all actions are meant to happen

at the same time and also it would implicitly induce an ordering, violating the *happens at the same time* idea. To solve this, we require different semantics for accessing the environment in this strategy. We introduce a *global* environment which is made up of the set of *local* environments. Each local environment is owned by an agent so there are as many local environments as there are agents. The semantics are then as follows: in each step all agents can *read* the global environment and *read/write* their local environment. The changes to a local environment are only visible *after* the local step and can be fed back into the global environment after the parallel processing of the agents. It does not make a difference if the agents are really computed in parallel or just sequentially - due to the isolation of information, this has the same effect. Also it will make no difference if we iterate over the agents sequentially or randomly, the outcome *has to be* the same: the strategy is event-ordering invariant as all events and updates happen *virtually at the same time*. If one needs to have the semantics of writes on the whole (global) environment in ones model, then this strategy is not the right one and one should resort to one of the other strategies. A workaround would be to implement the global environment as an agent with which the non-environment agents can communicate via messages introducing an ordering but which is then sorted in a controlled way by an agent, something which is not possible in the case of a passive, non-agent environment. It is important to note that in this strategy a reply to a message will not be delivered in the current but in the next global time-step. This is in contrast to the immediate message-delivery of the *sequential* strategy where within a global time-step agents can have in fact an arbitrary number of messages exchanged.

3.2.3. Concurrent Strategy. This strategy has a globally synchronized time-flow and in each time-step iterates through all the agents and updates all agents in parallel but all messages sent and changes to the environment are immediately visible. So this strategy can be understood as a more general form of the *parallel strategy*: all agents run at the same time but act concurrently.

Iteration-Order: Parallel
Global Synchronization: Yes
Thread of Execution: Separate
Message-Handling: Queued
Visibility of Changes: In-Iteration
Repeatability: Non-Deterministic

Semantics: It is important to realize that, when running agents in parallel which are able to see actions by others immediately, this is the very definition of concurrency: parallel execution with mutual read/write access to shared data. Of course this shared data-access needs to be synchronized which in turn will introduce event-orderings in the execution of the agents. At this point we have a source of inherent non-determinism: although when one ignores any hardware-model of concurrency, at some point we need arbitration to decide which agent gets access first to a shared resource arriving at non-deterministic solutions. This has the very important consequence that repeated runs with the same configuration of the agents and the model may lead to different results.

3.2.4. Actor Strategy. This strategy has no globally synchronized time-flow but all the agents run concurrently in parallel, with their own local time-flow. The messages and changes to the environment are visible as soon as the data arrive at the local agents - this can be immediately when running locally on a multi-processor or with a significant delay when running in a cluster over a network. Obviously this is also a non-deterministic strategy and repeated runs with the same agent- and model-configuration may (and will) lead to different results.

Iteration-Order: Parallel
Global Synchronization: No
Thread of Execution: Separate
Message-Handling: Queued
Visibility of Changes: In-Iteration
Repeatability: Non-Deterministic

Semantics: It is of most importance to note that information and also time in this strategy is always local to an agent as each agent progresses in its own speed through the simulation. In this case one needs to explicitly *observe* an agent when one wants to e.g. visualize it. This observation is then only valid for this current point in time, local to the observer but not to the agent itself, which may have changed immediately after the observation. This implies that we need to sample our agents with observations when wanting to visualize them, which would inherently lead to well known sampling issues. A solution would be to invert the problem and create an observer-agent which is known to all agents where each agent sends a *'I have changed'* message with the necessary information to the observer if it has changed its internal state. This also does not guarantee that the observations will really reflect the actual state the agent is in but is a remedy against the notorious

sampling. Problems can occur though if the observer-agent can't process the update-messages fast enough, resulting in a congestion of its message-queue. The concept of Actors was proposed by (Hewitt, Bishop, & Steiger, 1973) for which (Greif, 1975) and (Clinger, 1981) developed semantics of different kinds. These works were very influential in the development of the concepts of agents and and can be regarded as foundational basics for ABS.

Variation: This is the most general one of all the strategies as it can emulate all the others by introducing the necessary synchronization mechanisms.

3.3. ABS Toolskits

There exist a lot of tools for modelling and running ABS. We investigated the abilities of two of them to capture our update-strategies and give an overview of our findings in this section.

3.3.1. NetLogo. NetLogo is probably the most popular ABS toolkit around as it comes with a modelling language which is very close to natural language and very easy to learn for non-computer scientists. It follows a strictly single-threaded computing approach when running a single model, so we can rule out both the *concurrent* and *actor strategy* as both require separate threads of execution. The tool has no built-in concept of messages and it is built on global synchronization which is happening through advancing the global time by the 'tick' command. It falls into the responsibility of the model-implementer to iterate over all agents and let them perform actions on themselves and on others. This allows for very flexible updating of agents which also allows to implement the *parallel strategy*. A NetLogo model which implements the prisoners dilemma game synchronous and asynchronous to reproduce the findings of (Huberman & Glance, 1993) can be found in chapter 5.4 of (Jansen, 2012).

3.3.2. AnyLogic. AnyLogic follows a rather different approach than NetLogo and is regarded as a multi-method simulation tool as it allows to do system dynamics, discrete event simulation and agent-based simulation at the same time where all three methods can interact with each other. For ABS it provides the modeller with a high-level view on agents and does not provide the ability to iterate over all agents - this is done by AnyLogic itself and the modeller can customize the behaviour of an agent either by modelling diagrams or programming in Java. As NetLogo, AnyLogic runs a model using a single thread thus the *concurrent* and *actor strategy* are not feasible in AnyLogic. A feature this toolkit provides is communication between agents using messages and it supports both queued and immediate messaging. AnyLogic does not provide a mechanism to directly implement the *parallel strategy* because all changes are seen immediately by the other agents but using queued messaging we think that the *parallel strategy* can be emulated nevertheless.

3.3.3. Summary. To conclude, the most natural and common update-strategy in these toolkits is the *sequential strategy* which is not very surprising. The primary target are mostly agent-based modellers which are non-computer scientists so the toolkits also try to be as simple as possible and multi-threading and concurrency would introduce lots of additional complications for modellers to worry about. So the general consensus is to refrain from multi-threading and concurrency as it is obviously harder to develop, debug and introduces non-repeatability in the case of concurrency and to stick with the *sequential strategy*. The *parallel strategy* is not supported *directly* by any of them but can be implemented using various mechanisms like queued message passing and custom iteration over the agents.

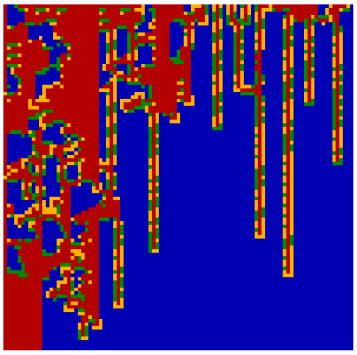
4. Case-Studies

In this section we present two case-studies in simulating the *Prisoners Dilemma* and *Heroes & Cowards* games for discussing the effect of using different update-strategies. As already emphasised, both are of different nature. The first one is a discrete game, played at discrete time-steps. The second one is a continuous game where each agent is continuously playing. This has profound implications on the simulation results shown below. We implemented the simulations for all strategies except the *actor strategy* in Java and the simulations for the *actor strategy* in Haskell and in Scala with the Actor-Library.

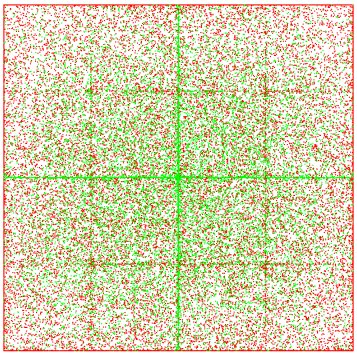
4.1. Prisoners Dilemma

The agent-based model Our agent-based model of this game works as follows: at the start of the simulation each agent sends its state to all its neighbours which allows to incrementally calculate the local payoff. If all neighbours' states have been received then the agent will send its local payoff to all neighbours which allows to compare all payoffs in its neighbourhood and calculate the best. When all neighbours' local payoffs have been received the agent will adopt the role of the highest payoff and sends its new state to all its neighbours, creating a circle.

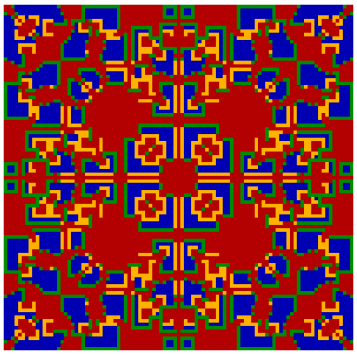
Prisoners Dilemma | **Heroes & Cowards**



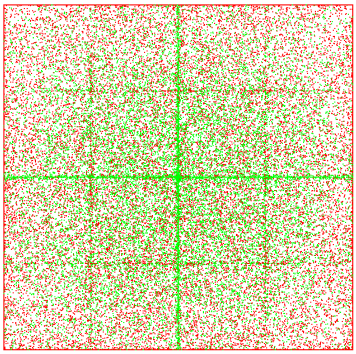
(a) sequential queued



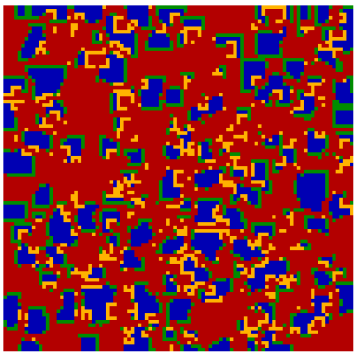
(b) sequential



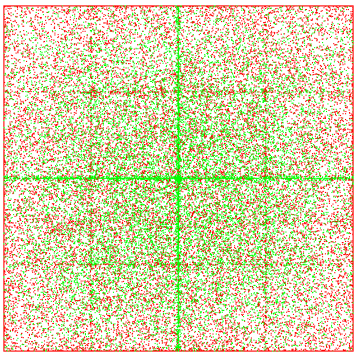
(c) parallel



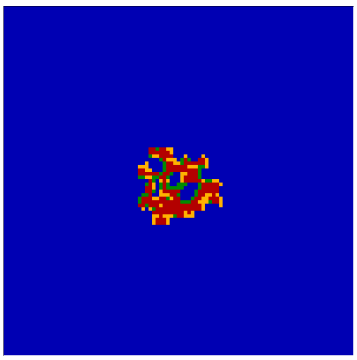
(d) parallel



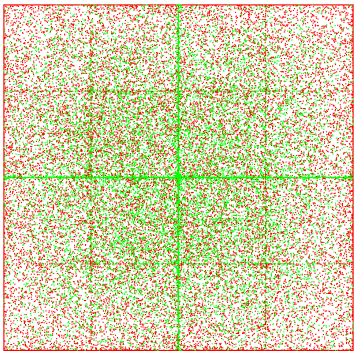
(e) concurrent



(f) concurrent



(g) actor



(h) actor

Figure 3: Effect on results simulating the Prisoners Dilemma and Heroes & Cowards with all four update-strategies.

Table 1: Update-Strategies in ABS

	Sequential	**Parallel**	**Concurrent**	**Actor**
Iteration-Order	Sequential	Parallel	Parallel	Parallel
Global-Sync	Yes	Yes	Yes	No
Thread	Shared	Separate	Separate	Separate
Messaging	Immediate	Queued	Queued	Queued
Visibility	In	Post	In	In
Repeatability	Yes	Yes	No	No

Care must be taken to ensure that the update-strategies are comparable because when implementing a model in an update-strategy it is necessary to both map the model to the strategy and try to stick to the same specification - if the implementation of the model differs fundamentally across the update-strategies it is not possible to compare the solutions. So we put great emphasis and care keeping all four implementations of the model the same just with a different update-strategy running behind the scenes which guarantees comparability.

Results The results as seen in the left column of Figure 3 were created with the same configuration as reported in (Nowak & May, 1992). When comparing the pictures with the one from the reference seen in Figure 1 the only update-strategy which is able to reproduce the matching result is the *parallel strategy* - all the others clearly fail to reproduce the pattern. From this we can tell that only the *parallel strategy* is suitable to simulate this model.

To reproduce the pattern of Figure 1 the simulation needs to be split into two global steps which must happen after each other: first calculating the sum of all payoffs for every agent and then selecting the role of the highest payoff within the neighbourhood. This two-step approach results in the need for twice as many steps to arrive at the matching pattern when using *queued* messaging as is the case in the *parallel, concurrent* and *actor strategy*.

For the *sequential strategy* one must further differentiate between *immediate* and *queued* messaging. We presented the results using the *queued* version, which has the same implementation of the model as the others. When one is accepting to change the implementation slightly, then the *immediate* version is able to arrive at the pattern after 217 steps with a slightly different model-implementation: because immediate messaging transfers the thread of control to the receiving agent that agent can reply within this same step. This implies that we can calculate a full game-round (both steps) within one global time-step by a slight change in the model-implementation: an agent sends its current state in every time-step to all its neighbours.

The reason why the other strategies fail to reproduce the pattern is due to the non-parallel and unsynchronized way that information spreads through the grid. In the *sequential strategy* the agents further ahead in the queue play the game earlier and influence the neighbourhood so agents which play the game later find already messages from earlier agents in their queue thus acting differently based upon these informations. Also agents will send messages to itself which will be processed in the same time-step. In the *concurrent* and *actor strategy* the agents run in parallel but changes are visible immediately and concurrently, leading to the same non-structural patterns as in the *sequential* one. Although agents don't change unless all their neighbours have answered, this does not guarantee a synchronized update of all agents because every agent has a different neighbourhood which is reflexive but not transitive. If agent a is agent's b neighbour and agent c is agent's b neighbour this does not imply that agent c is agent's a neighbour as well. This allows the spreading of changes throughout the neighbourhood, resulting in a breaking down of the pattern. This is not the case in the *parallel strategy* where all agents play the game at the same time based on the frozen state of the previous step, leading to a synchronized update as required by the model. Note that the *concurrent* and *actor strategy* produce different results on every run due to the inherent non-deterministic event-ordering introduced by concurrency.

4.2. Heroes & Cowards

The agent-based model Our agent-based model of this game works as follows: in each time-step an agent asks its friend and enemy for their positions which will answer with a corresponding message containing their current positions. The agent will have its own local information about the position of its friend and enemy and will calculate its move in every step based on this local information.

Results The results as seen in the right column of Figure 3 were created with 100.000 agents where 25% of them are heroes running for 500 steps. Although the individual agent-positions of runs with the same configuration differ between update-strategies the cross-patterns are forming in all four update-strategies. For the patterns to emerge it is important to have significant more cowards than heroes and to have agents in the tens of thousands - we went for 100.000 because then the patterns are really prominent. The patterns form because the heroes try to stay halfway between their friend and enemy: with this high number of cowards it is very likely that heroes end up with two cowards - the cowards will push towards the border as they try to escape, leaving the hero in between. We can conclude that the *Heroes & Cowards* model seems to be robust to the selection of its update-strategy and that its emergent property - the formation of the cross - is stable under differing strategies.

5. Conclusion and future research

In this paper we have presented general properties of ABS, derived four general update-strategies and discussed their implications. By doing this we proposed a unified terminology which allows to speak about update-strategies in a common and unified way, something that the ABS community is currently lacking. We hope our classification and terminology will help the community to better understand the details necessary to consider implementing an agent-based simulations. Again we cannot stress enough that selecting the right update-strategy is of most importance and must match the semantics of the model one wants to simulate. We showed that the *Prisoners Dilemma* game on a 2D-grid can only be simulated correctly when using the *parallel strategy* and that the other strategies lead to a breakdown of the emergent pattern reported in the original paper. On the other hand using the *Heroes & Cowards* game we showed that there exist models whose emergent patterns exhibit a stability under varying update-strategies. Intuitively we can say that this is due to the nature of the model specification which does not require specific orderings of actions but it would be interesting to put such intuitions on firm theoretical grounds in future research.

References

Bersini, H., & Detours, V. (1994). Asynchrony induces stability in cellular automata based models. In *In Proceedings of Artificial Life IV* (pp. 382–387). MIT Press.

Botta, N., Mandel, A., & Ionescu, C. (2010). *Time in discrete agent-based models of socio-economic systems* (Documents de travail du Centre d'Economie de la Sorbonne No. 10076). Université Panthéon-Sorbonne (Paris 1), Centre d'Economie de la Sorbonne.

Clinger, W. D. (1981). *Foundations of Actor Semantics* (Tech. Rep.). Cambridge, MA, USA: Massachusetts Institute of Technology.

Dawson, D., Siebers, P. O., & Vu, T. M. (2014, September). Opening pandora's box: Some insight into the inner workings of an Agent-Based Simulation environment. In *2014 Federated Conference on Computer Science and Information Systems* (pp. 1453–1460). doi: 10.15439/2014F335

Greif, I. (1975). *Semantics of communicating parallel processes* (Tech. Rep.). Cambridge, MA, USA: Massachusetts Institute of Technology.

Hewitt, C., Bishop, P., & Steiger, R. (1973). A Universal Modular ACTOR Formalism for Artificial Intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence* (pp. 235–245). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Huberman, B. A., & Glance, N. S. (1993, August). Evolutionary games and computer simulations. *Proceedings of the National Academy of Sciences*, *90*(16), 7716–7718.

Jansen, M. (2012). *Introduction to Agent-Based Modeling.* Retrieved from `https://www.openabm.org/book/introduction-agent-based-modeling`

Lysenko, M., D'souza, R., & Rahmani, K. (2008). *A Framework for Megascale Agent Based Model Simulations on the GPU*.

Nowak, M. A., & May, R. M. (1992, October). Evolutionary games and spatial chaos. *Nature*, *359*(6398), 826–829. doi: 10.1038/359826a0

Page, S. E. (1997, February). On Incentives and Updating in Agent Based Models. *Comput. Econ.*, *10*(1), 67–87. doi: 10.1023/A:1008625524072

Railsback, S., & Grimm, V. (2011). *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press.

Wilensky, U., & Rand, W. (2015). *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NETLogo*. MIT Press.

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems* (2nd ed.). Wiley Publishing.

Yuxuan, J. (2016). *The Agent-based Simulation Environment in Java*. Unpublished doctoral dissertation, University Of Nottingham, School Of Com-

puter Science.