CELLULAR AUTOMATON

Pau Fonseca i Casas, pau@fib.upc.edu

Key message

- IF you know how to formalize a model using SDL you know:
  - How to model ODEs
  - How to model MAS
  - How to model CA
  - How to model distributed simulation
  - How to integrate IoT
- We still follow SDL syntax in this introduction to CA.

Other alternatives?

- Yes
  - Wolfram rules
  - CELL-DEVS
  - Mathematical representation

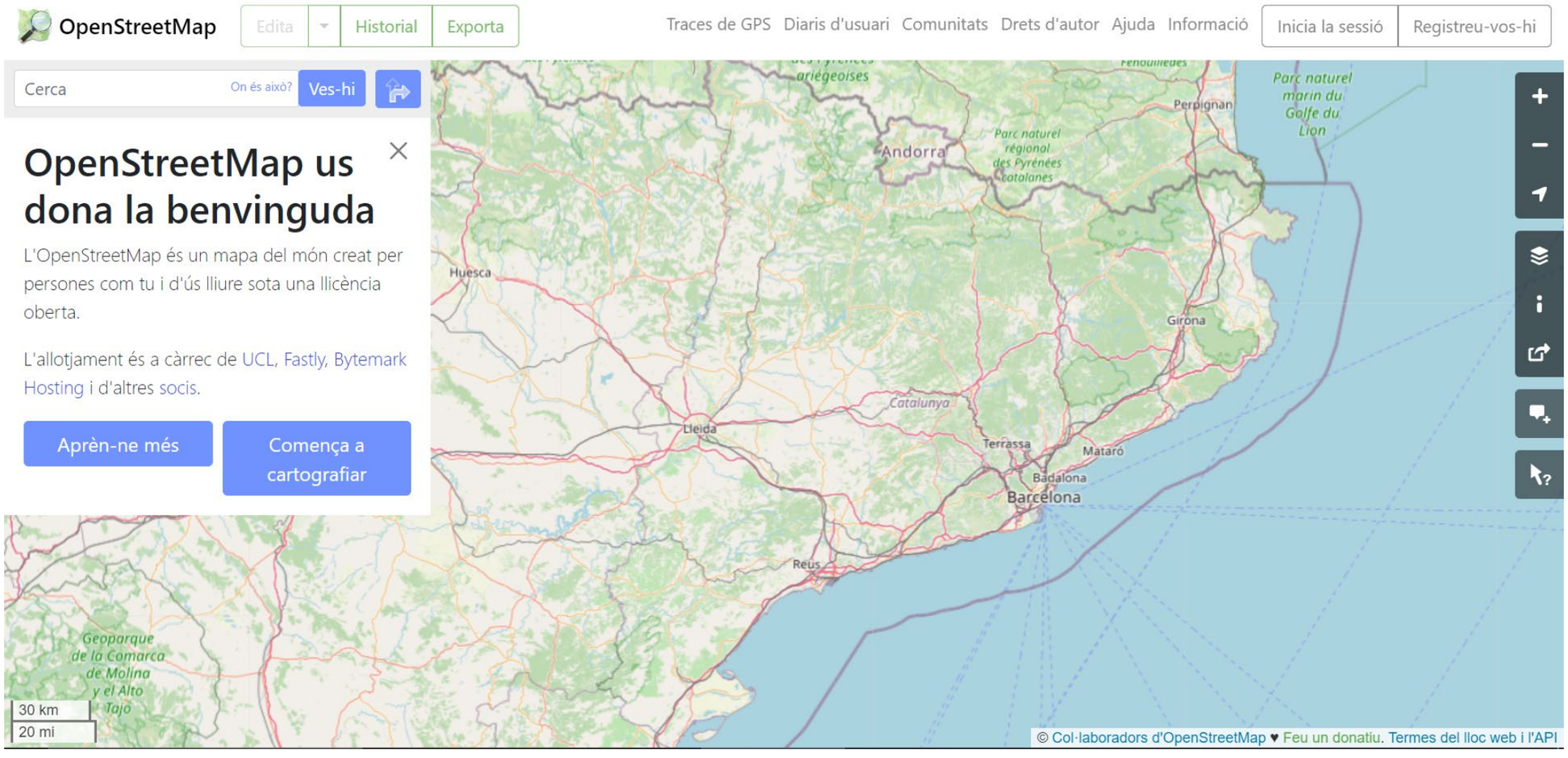
GIS data

GIS data structure and classification.

GIS data


Mapes
Cerca
Avinguda Diagonal
Carrer de Pau Gargallo
Carrer de Trias i Giró
Passeig dels Tilers
UPC Campus Nord
Jardins del Palau de Pedralbes
Metro-Palau Reial
Auditori Antoni Caparros
Medichem
DiR Campus
Bowling Pedralbes
Cementiri les Corts
Camp Nou
Jardins de la Maternitat
Metro-Maria Cristina
Aqua Diagonal Wellness Centre
Sony
CORREOS
Carrer de Manila
Passeig Manuel Girona
Carrer de Menéndez y Pelayo
Avinguda de Joan XXIII
Carrer d'Aristides Maillol
bing
© 2016 Microsoft Corporation, © 2016 HERE

Open Street map



Layers in a GIS

- Vector Layers: Represent geographic features using points, lines, and polygons.
- Types:
  - Point Layer: Represents discrete locations (e.g., cities, wells, or landmarks).
  - Line Layer: Represents linear features (e.g., roads, rivers, or pipelines).
  - Polygon Layer: Represents areas (e.g., land parcels, lakes, or administrative boundaries).
  - Use Cases: Urban planning, transportation networks, land use analysis.

Layers in a GIS

- Raster Layers:: Represent geographic features as a grid of cells or pixels, where each cell has a value (e.g., elevation, temperature, or land cover).
- Types:
  - Satellite Imagery: High-resolution images of the Earth's surface.
  - Digital Elevation Models (DEMs): Represent terrain elevation.
  - Land Cover Maps: Show vegetation, water, and urban areas.
  - Use Cases: Environmental modeling, terrain analysis, remote sensing.

Layers in a GIS

- Attribute Layers: Contain non-spatial data (attributes) linked to spatial features in vector or raster layers.
- Examples:
  - Population data linked to administrative boundaries.
  - Soil properties linked to land parcels.
  - Use Cases: Data analysis, decision-making, and reporting.

Layers in a GIS

- Thematic Layers: Represent specific themes or subjects (e.g., population density, rainfall distribution, or crime rates).
- Types:
  - Choropleth Maps: Show data using color gradients.
  - Heatmaps: Visualize intensity or density of phenomena.
  - Use Cases: Socioeconomic analysis, environmental monitoring.

Layers in a GIS

- Base Layers: Provide a background or reference for other layers (e.g., maps, satellite imagery, or topographic maps).
- Examples:
  - OpenStreetMap.
  - Google Maps.
  - Use Cases: Contextual reference for spatial analysis.

Layers in a GIS

- Network Layers: Represent interconnected linear features (e.g., road networks, utility networks, or river systems).
- Use Cases: Routing, logistics, and network analysis.

Layers in a GIS

- Temporal Layers: Represent data that changes over time (e.g., historical land use, weather patterns, or population growth).
- Use Cases: Time-series analysis, trend monitoring.

Layers in a GIS

- 3D Layers: Represent geographic features in three dimensions (e.g., buildings, terrain, or subsurface structures).
- Use Cases: Urban planning, disaster management, and visualization.

Layers in a GIS

- Annotation Layers: Contain text or labels for features (e.g., city names, road labels, or contour values).
- Use Cases: Map labeling and cartographic design.

Layers in a GIS

- Surface Layers: Represent continuous surfaces (e.g., elevation, temperature, or pollution levels).
- Types:
  - Contour Lines: Show elevation or other continuous values.
  - TIN (Triangulated Irregular Network): Represent terrain surfaces.
- Use Cases: Terrain analysis, hydrological modeling.

Layers in a GIS

- Geodatabase Layers: Layers stored in a geodatabase, which is a structured collection of geographic data.
- Types:
  - Feature Classes: Store vector data.
  - Raster Datasets: Store raster data.
- Use Cases: Data management and organization.

Layers in a GIS

- WMS/WFS Layers (Web Services): Layers served over the web using standards like Web Map Service (WMS) or Web Feature Service (WFS).
- Use Cases: Sharing and accessing spatial data online.

Layers in a GIS

- CAD Layers: Imported from Computer-Aided Design (CAD) systems, representing engineering or architectural designs.
- Use Cases: Urban planning, infrastructure design.

Layers in a GIS

- LiDAR Layers: Represent 3D point cloud data collected using LiDAR (Light Detection and Ranging) technology.
- Use Cases: Flood modeling, forestry, and urban planning.

Layers in a GIS

- Statistical Layers: Represent statistical data (e.g., census data, crime rates, or disease incidence) spatially.
- Use Cases: Demographic analysis, public health.

Layers in a GIS

Layer Type	Description	Examples
Vector	Points, lines, polygons	Roads, cities, land parcels
Raster	Grid of cells/pixels	Satellite imagery, DEMs
Attribute	Non-spatial data linked to spatial features	Population data, soil properties
Thematic	Specific themes (e.g., population density)	Choropleth maps, heatmaps
Base	Background/reference layers	OpenStreetMap, Google Maps
Network	Interconnected linear features	Road networks, utility networks
Temporal	Data that changes over time	Historical land use, weather patterns
3D	Three-dimensional features	Buildings, terrain
Annotation	Text or labels for features	City names, road labels
Surface	Continuous surfaces (e.g., elevation)	Contour lines, TIN
Geodatabase	Structured collection of geographic data	Feature classes, raster datasets
WMS/WFS	Web-based layers	Online maps, shared datasets
CAD	Imported from CAD systems	Engineering designs
LiDAR	3D point cloud data	Flood modeling, forestry
Statistical	Statistical data represented spatially	Census data, disease incidence

Layers in a GIS

Layer	Description
DEM (raster layers)	Raster, thematic, base, temporal, geodatabase, WMS/WFS
2DLayers (vector layers)	Vector, annotation, surface, WMS/WFS, Statistical
3DLayers (3D features)	3D, WMS/WFS
Routes (GPS, IoT)	Track points, WMS/WFS
2DObjects (attribute layers)	Attribute, WMS/WFS
3DObjects (attribute layers)	WMS/WFS, CAD, LiDAR

2DLayers (Vectorial layers)

- Point, polylines, texts or lines.
- Usually represent information for the observer but not represent information for the simulation model (don't have any specific behavior).

3DLayers (Rasters Layers)

- To represent a fixed population of elements
  - Forest or a city (each tree have his own position over the DEM).
- All the elements have the same virtual representation and the same behaviour.

Objects

- To represent element with an individual or concrete behaviour (2DObjects or 3DObjects).
  - **2DObjects:** lines polylines, texts and points with an individual behaviour, (a point with a touch sensor that shows some information).
  - **3DObjects:** virtual objects that are not associated with any layer. These objects can have own behaviour represented by a script or program.

Routes

- The routes define 3DObjects movements through the virtual Landscape.
- These routes can be defined by the simulator.
- Enables the possibility of move different elements through to Landscape following the simulator logic.

- Connected with IoT (GPS, etc.)

Why use GIS data?

A classification of the models

Interactive/evolutionary models

	Static models	Dynamic models
Interactive	Static interactions. The only changes are in the composition. For instance, systems that are not modified over time. Through simulations, an approximate value can be obtained.	System interaction. Changes in the interactions between the different model components. For instance, an industrial plant.
Evolutionary	Evolutionary selection. Random acquisition of variations that change the composition of types.	Evolutionary system feedback that influences the supply of variation and the speed of evolution. Changes in type depend on the history of the system. For instance, the evolution of a society or wildfire with the interaction of an extinction model.

Source: Carl Henning Reschke, *Evolutionary Perspectives on Simulations of Social Systems*, J. Artif. Soc. Soc. Simul. **4**, (2001).

MAS

		Enviroment	
	Relation with the environment	Discrete	Continuous
Static	Strong	Automaton	Control system
	Weak	Semi-Isolated Evolution	
Dynamic	Weak	Complex Interactions	
	Strong	Ecosystem	

Source: C. A. Marín and N. Mehandjiev, *A Classification Framework of Adaptation in Multi-Agent Systems*, in *Cooperative Information Agents X*, edited by M. Klusch, M. Rovatsos, and T. R. Payne (Springer Berlin Heidelberg, Berlin, Heidelberg, 2006), pp. 198–212.

Using GIS in Simulation models

- Allows environment modeling.
- Dynamical use of the GIS data.
  - Dynamical modification of the GIS data.
  - Dynamical acquisition of the GIS data.

Cellular automata

Modeling the environment

Cellular Automaton (CA) definition

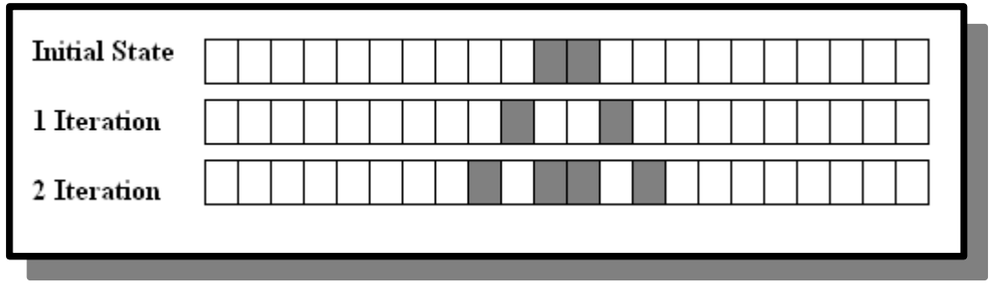
- Cellular Automaton (CA) consists of a regular grid of cells, each in one of a finite number of states, such as on and off (in contrast to a coupled map lattice). The grid can be in any finite number of dimensions.

Cellular Automaton (CA) definition

- For each cell, a set of cells called its neighborhood is defined relative to the specified cell. An initial state (time t = 0) is selected by assigning a state for each cell. A new generation is created (advancing t by 1), according to some fixed rule (generally, a mathematical function) that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood.
  - Source: https://en.wikipedia.org/wiki/Cellular_automaton

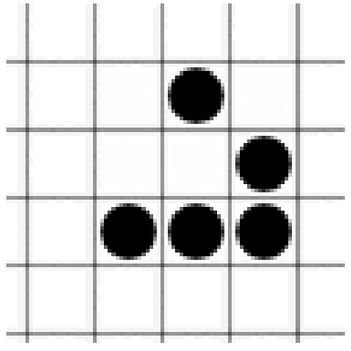
Cellular automaton

- Structure based in:
  - Set of rules.
  - Matrix of data.
- Modify the matrix following the set of rules.



Game of life

- The **Game of Life** is a cellular automaton devised by the British mathematician John Horton Conway in 1970. It is the best-known example of a cellular automaton.
- Glider gun and glider



Cellular automaton

- Simplifies the use of environment in a simulation model.
  - Torrens, Paul M. Benenson, Itzhak. 2004, Geosimulation, John Wiley & Sons Ltd., ISBN: 0-470-84349-7
  - Batty, Michael, 2005, Cities and complexity.The MIT Press.
  - Wolfram, Stephen 2002, A new Kind of Science, Wolfram Media, Inc.

Cellular automaton

- Considerations:
  - The space of states can be huge.
  - Not all the states can be reachable → maps of states space.
- Limitations:
  - Only one matrix of data.
  - Only one set of rules.
  - The space is discrete.

Basics of One-Dimensional Cellular Automata

- A one-dimensional cellular automaton consists of a line of cells, where each cell can be in one of two states: 0 (off/dead) or 1 (on/alive).
- The state of each cell in the next time step depends on its current state and that of its immediate neighbors (typically, the left and right neighbors).
- The evolution rule defines how the state of a cell changes based on the states of its neighborhood.

Neighborhood and Configurations

- In a one-dimensional cellular automaton with a neighborhood size of 1 (i.e., each cell has one neighbor to the left and one to the right), there are 8 possible configurations for the neighborhood of 3 cells (left, center, right):
  - 111, 110, 101, 100, 011, 010, 001, 000

- Each configuration is associated with a new state for the center cell in the next time step.

Wolfram Encoding Rule

- The Wolfram rule assigns an 8-bit binary number to each one-dimensional cellular automaton, where each bit represents the new state of the center cell for one of the 8 possible configurations.
- The binary number is converted to a decimal number between 0 and 255, which identifies the rule.

Example

- Suppose the rule assigns the following new states:
  - 111 → 0
  - 110 → 1
  - 101 → 1
  - 100 → 0
  - 011 → 1
  - 010 → 1
  - 001 → 1
  - 000 → 0
- The corresponding binary number is: 01101110. Converted to decimal: **110**.
- Therefore, this rule is called Rule **110**.

Notable Rules

- Rule 30: Generates chaotic patterns and is known for its complex behavior.
- Rule 90: Produces fractal patterns and follows the logic of the Sierpinski triangle.
- Rule 110: It is Turing-complete, meaning it can perform any computational calculation given the appropriate initial state.

- Rule 184: Used to model traffic flow.

Visualization of the Rules

Tools

- Wolfram rules can be visualized as transition tables or evolution diagrams:
  - Transition table: Shows the 8 neighborhood configurations and the corresponding new state.
  - Evolution diagram: Shows how the automaton evolves over time, row by row.

Example of a transition table for Rule 110

Neighborhood (left, center, right)	New State
1 1 1	0
1 1 0	1
1 0 1	1
1 0 0	0
0 1 1	1
0 1 0	1
0 0 1	1
0 0 0	0

Elementary Cellular Automata

$$A = (a(t), S_A, f_A)$$

lattice of cells — alphabet — update function

	$a_{n-1}(t)$	$a_n(t)$	$a_{n+1}(t)$	
		$a_n(t+1)$		

$S_A = \{0,1\}$

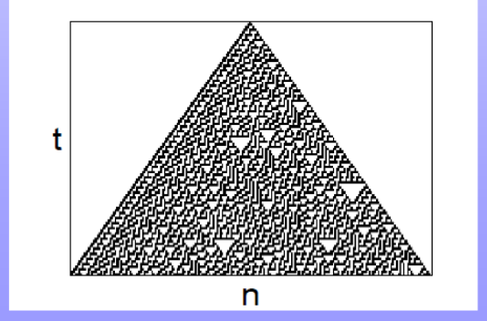
for elementary CA

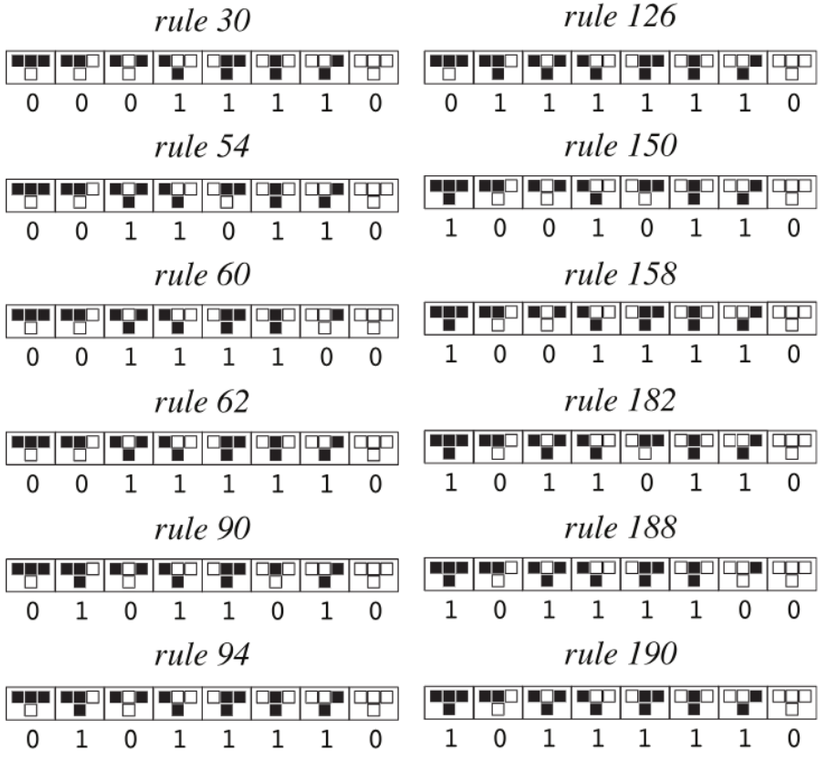
$$a_n(t+1) = f_A[a_{n-1}(t), a_n(t), a_{n+1}(t)]$$

(256 possible functions)

A dynamical system capable of rich behavior.

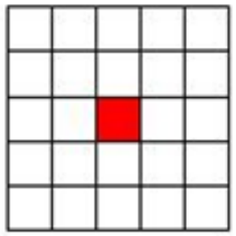
Wolfram's rule notation:
read 
as a binary number.



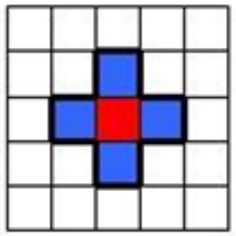


Neighbourhood

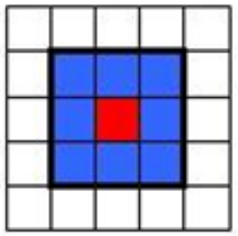
Vecindades



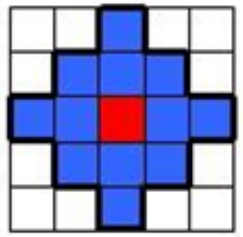
Vacía
$N = 0$



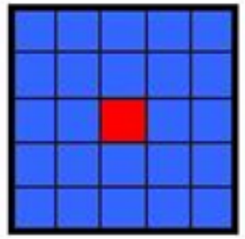
Von Neumann
$N = 4$



Moore
$N = 8$

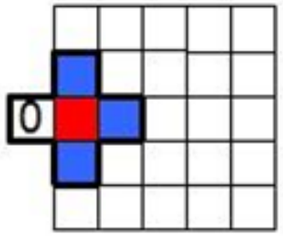


MvonN
$N = 12$

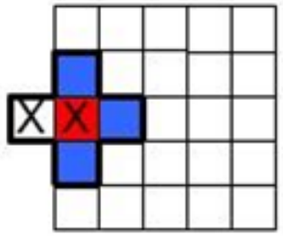


Moore expandida
$N = 24$

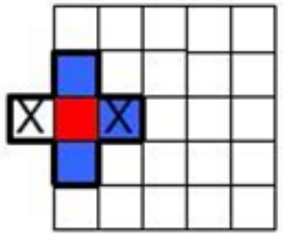
Fronteras



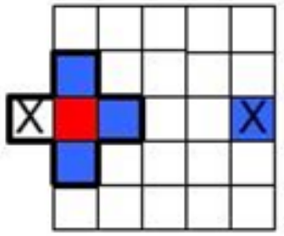
Fija



Adiabática



Reflexiva



Periodica

Source: https://centrobio.utec.edu.pe/automatas-celulares-una-tecnica-inteligente-para-modelar-sistemas-complejos/

Game of life

- The Environment
  - The Game of Life takes place on a two-dimensional grid of cells, typically square.
  - Each cell can be in one of two states: alive (1) or dead (0).
  - The state of each cell evolves in discrete time steps (called generations) based on its neighbors.
- Neighborhood: Moore

Game of life: Evolution Rules

- Birth:
  - A dead cell comes to life if it has exactly 3 live neighbors.
- Survival:
  - A live cell survives if it has 2 or 3 live neighbors.
  - If it has fewer than 2 live neighbors, it dies due to underpopulation.
  - If it has more than 3 live neighbors, it dies due to overpopulation.

CA classification

Differents ways to classify CA

Basic criteria for the classification

- Dimension
- Neighborhood structure
- Rule type
- Behaviour (Wolfram)

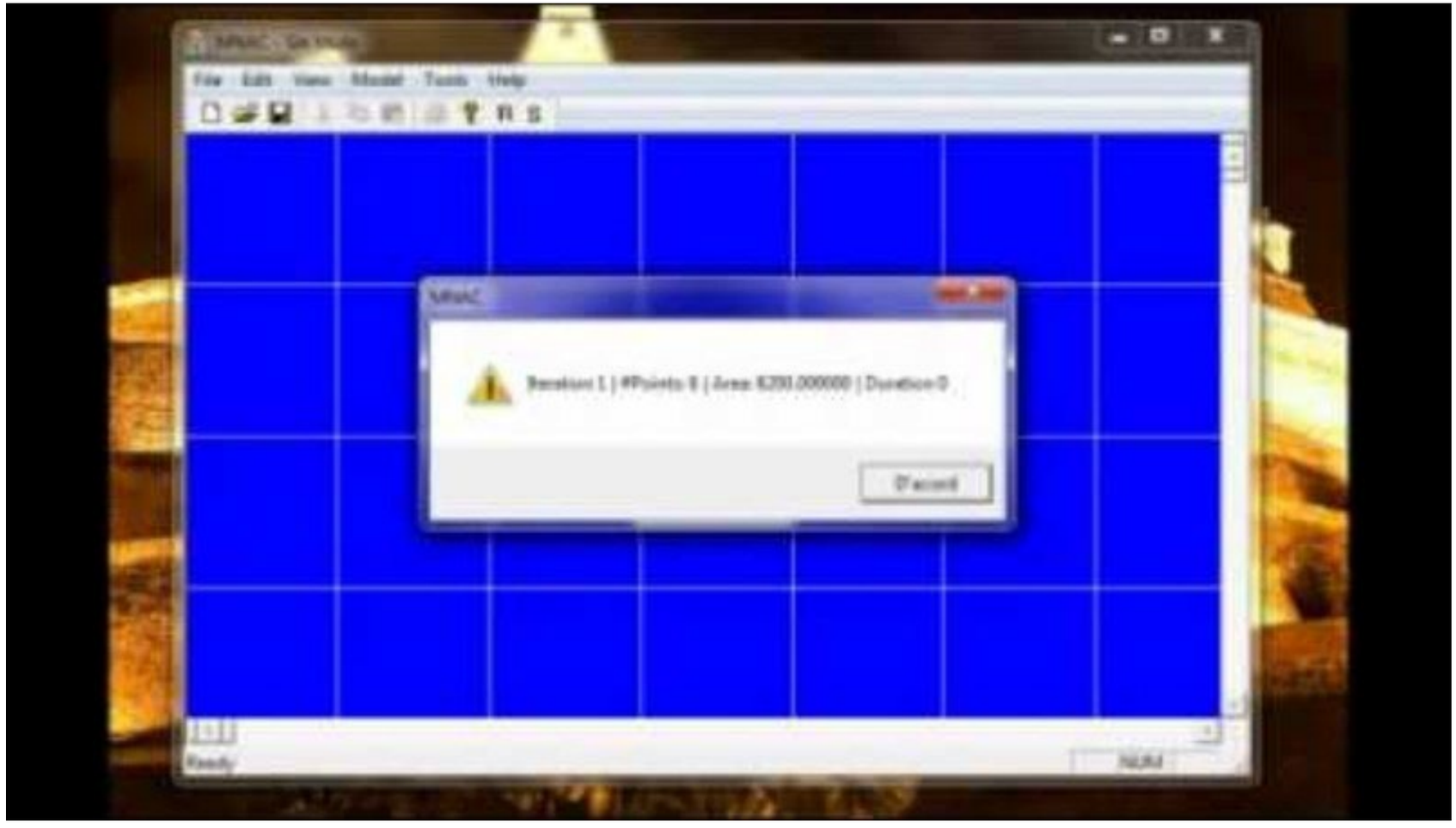
Classification by Dimensionality

- One-Dimensional (1D) Cellular Automata:
  - Cells are arranged in a linear array.
  - Example: Rule 30, Rule 110.
- Two-Dimensional (2D) Cellular Automata:
  - Cells are arranged in a grid (e.g., square, hexagonal, or triangular).
  - Example: Conway's Game of Life.
- Three-Dimensional (3D) Cellular Automata:
  - Cells are arranged in a 3D lattice.
  - Example: 3D extensions of Conway's Game of Life.

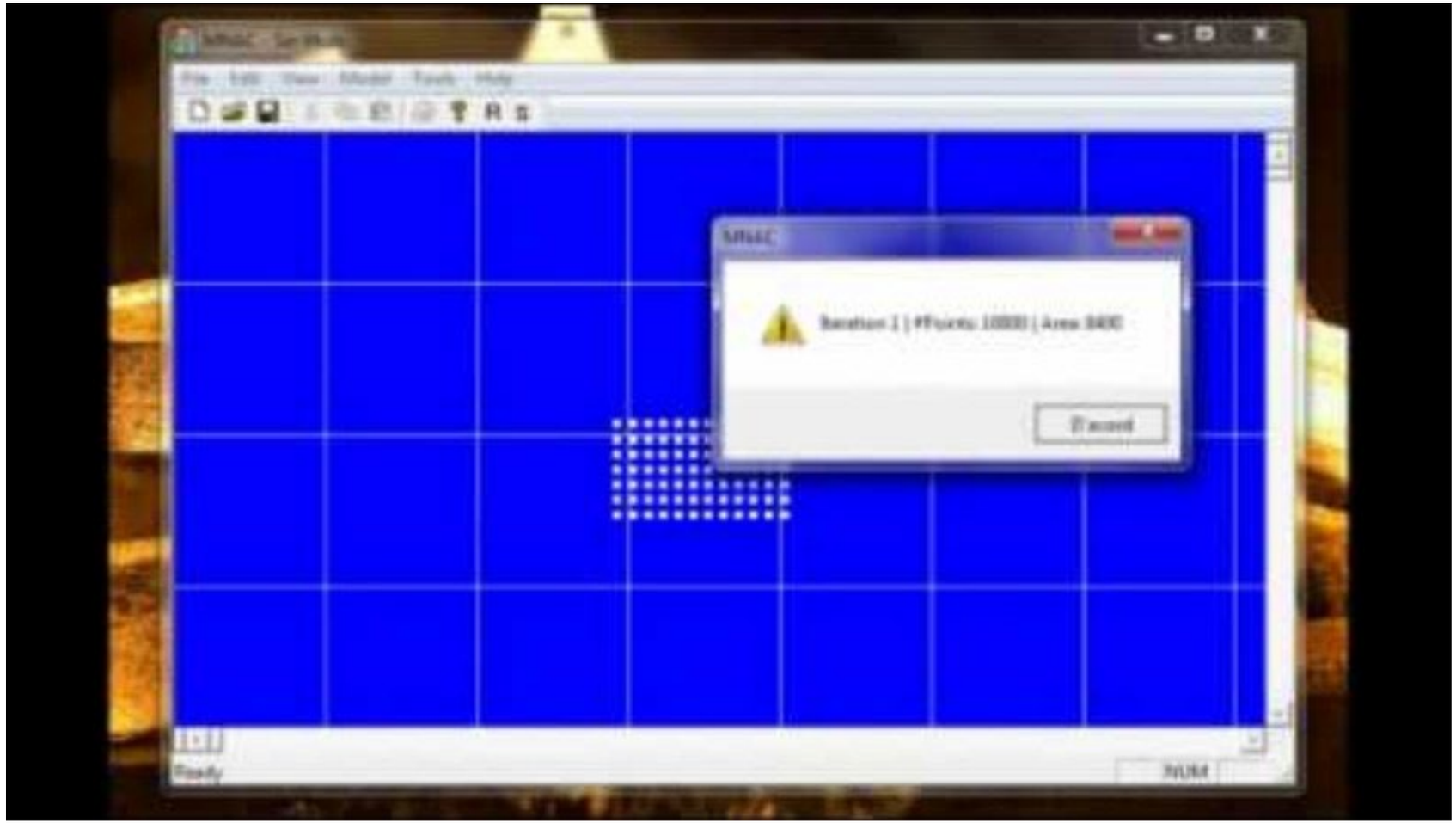
Classification by Neighborhood Structure

- Von Neumann Neighborhood:
  - Includes the four orthogonally adjacent cells (up, down, left, right).
  - Commonly used in 2D cellular automata.
- Moore Neighborhood:
  - Includes all eight surrounding cells (orthogonal and diagonal).
  - Commonly used in 2D cellular automata.
- Extended Neighborhoods:
  - Larger neighborhoods, such as radius-2 or radius-3 neighborhoods.
- Topological Neighborhood:
  - Neighborhoods defined using functions over a topological space, accepts changes, etc. like m:n-CA^{k}.

m:n-CA^k



https://youtu.be/PiXiq7j9A2s?si=hBVyCir3mZe09_ns



https://youtu.be/49OoqWE0REs?si=1ygpWmCYsk6MLcsc

Classification by Rule Type

- Totalistic Rules:
  - The new state of a cell depends on the sum of the states in its neighborhood.
  - Example: Brian's Brain (a 2D cellular automaton).
- Outer-Totalistic Rules:
  - The new state depends on the current state of the cell and the sum of the states in its neighborhood.
  - Example: Conway's Game of Life.
- Non-Totalistic Rules:
  - The new state depends on the specific configuration of the neighborhood, not just the sum.
  - Example: Rule 110 (1D cellular automaton).

Brian's Brain (2D Cellular Automaton)

- Brian's Brain is a three-state, two-dimensional cellular automaton created by Brian Silverman. It is known for its complex and visually interesting behavior, resembling the activity of neurons in a brain.
- States: (each cell can be in one of three states)
  - Off (0): The cell is inactive.
  - On (1): The cell is active.
  - Dying (2): The cell is in a refractory state (temporarily inactive).
  - Neighborhood: Uses the Moore neighborhood (all eight surrounding cells).
- Rules:
  - Activation:: A cell in the Off state becomes On if exactly two of its neighbors are On.
  - Refractory Period:: A cell in the On state transitions to the Dying state in the next step.
  - Recovery:: A cell in the Dying state transitions to the Off state in the next step.

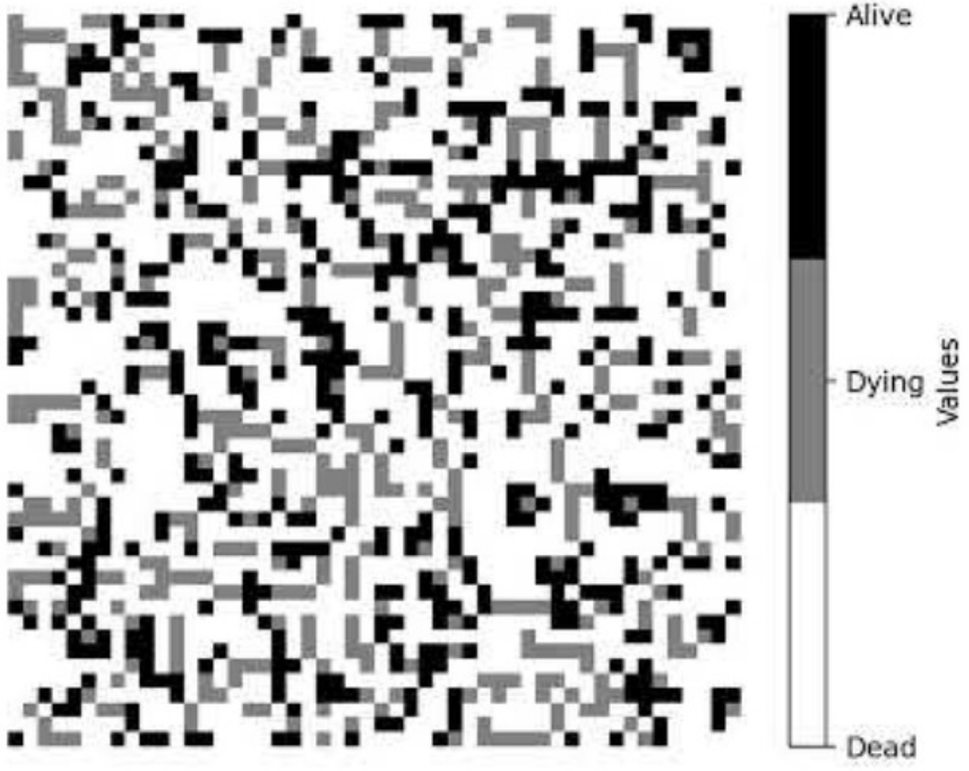
Brian's Brain (2D Cellular Automaton)

- Example Evolution:
  - Cells that were On at time t become Dying at time t+1.
  - Cells that were Off at time t may become On if exactly two neighbors were On.
- Behavior:
  - Gliders and Spaceships:
    - Brian's Brain exhibits moving patterns (gliders) that propagate across the grid.
  - Oscillators:
    - Some patterns oscillate between states, creating periodic behavior.
  - Chaotic Growth:
    - The automaton often produces chaotic, expanding patterns that resemble neural activity.
- Applications:
  - Neural Modeling:
  - Brian's Brain is used to model neural networks and brain activity.
  - Art and Visualization:

Time t	Tiime t+1
0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0	0 2 0 0 2 0 2 0 0 2 0 0 0 0 0 0

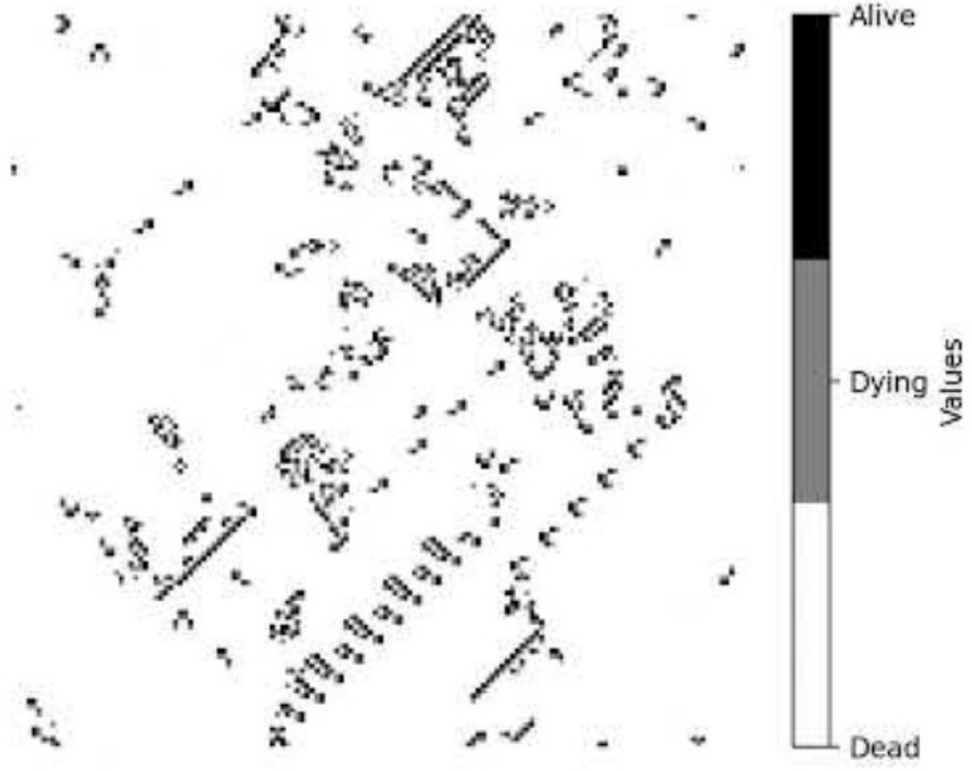
- Its visually complex behavior makes it popular for artistic and educational purposes.

Brian's Brain



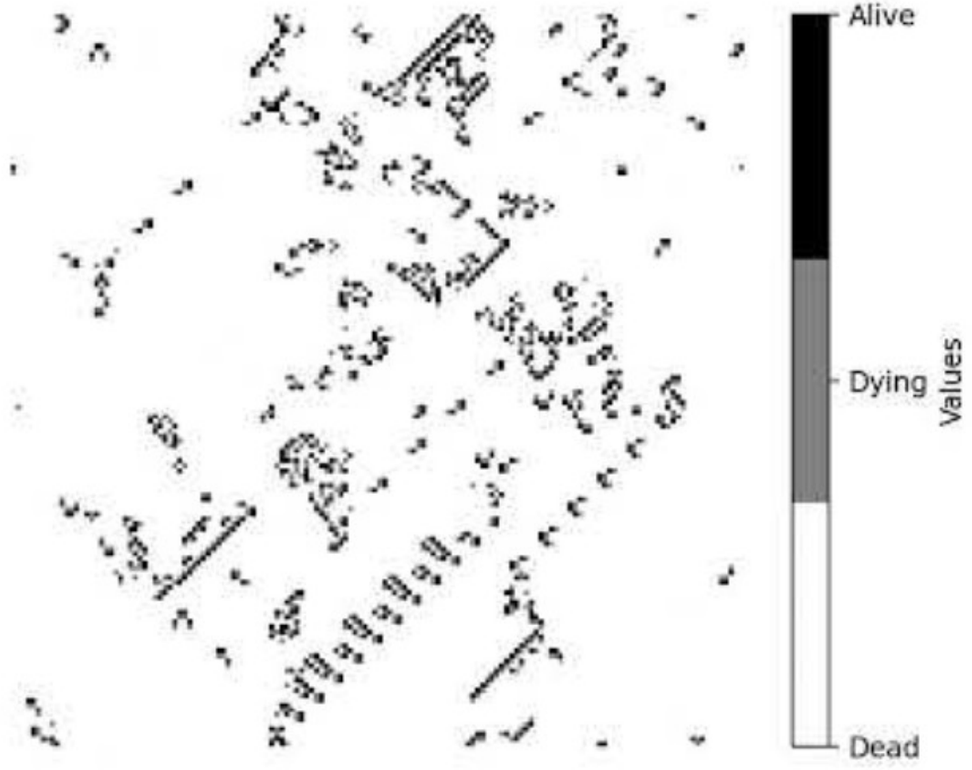
https://www.youtube.com/watch?v=2zXou3XQpb8

Brian's Brain



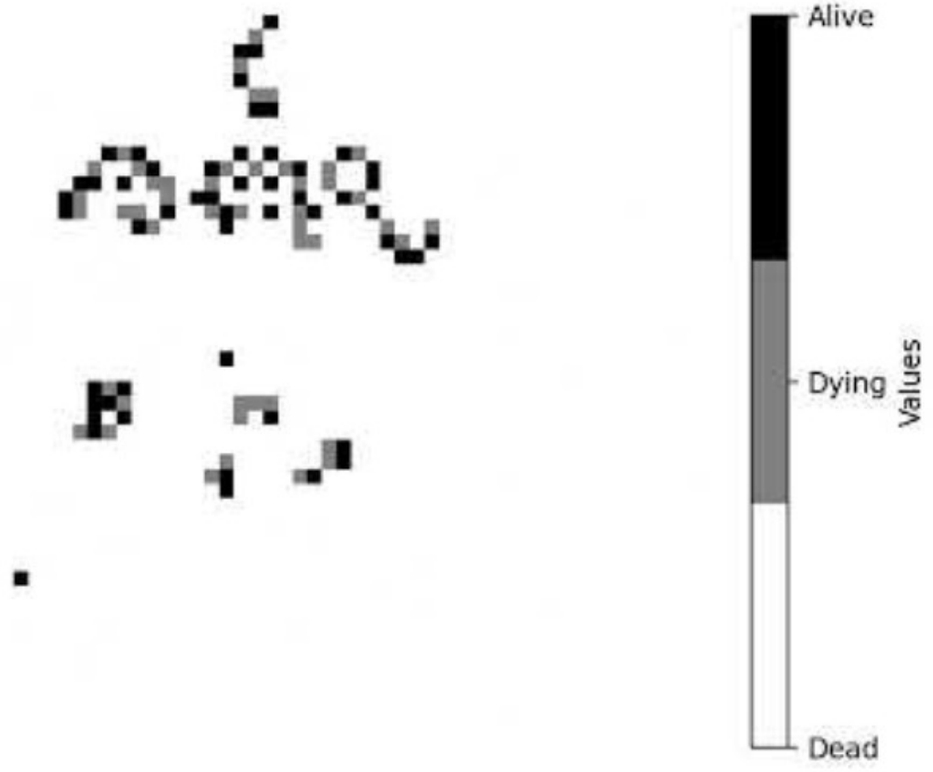
https://www.youtube.com/watch?v=LZew9LI1Qsk

Brian's Brain



https://www.youtube.com/watch?v=BDzmjGeg-Ho

Brian's Brain

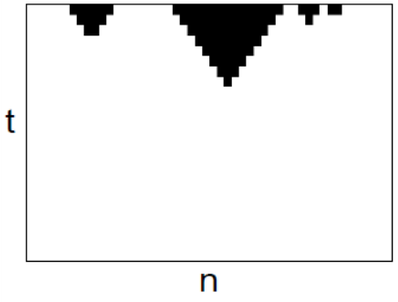
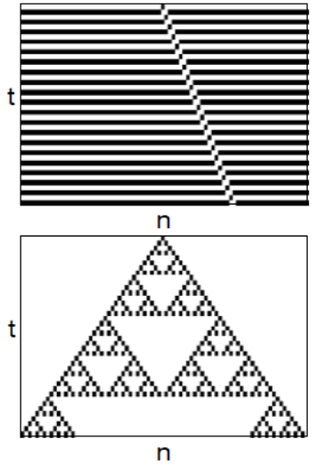
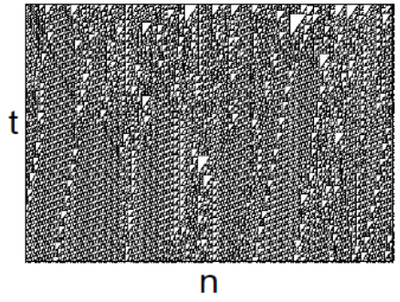
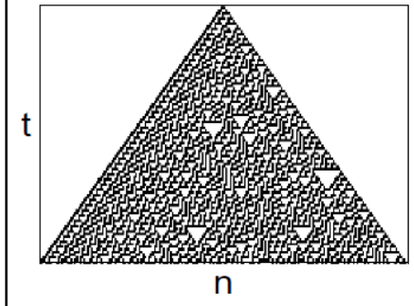


https://www.youtube.com/watch?v=QQxAe4jkiBQ

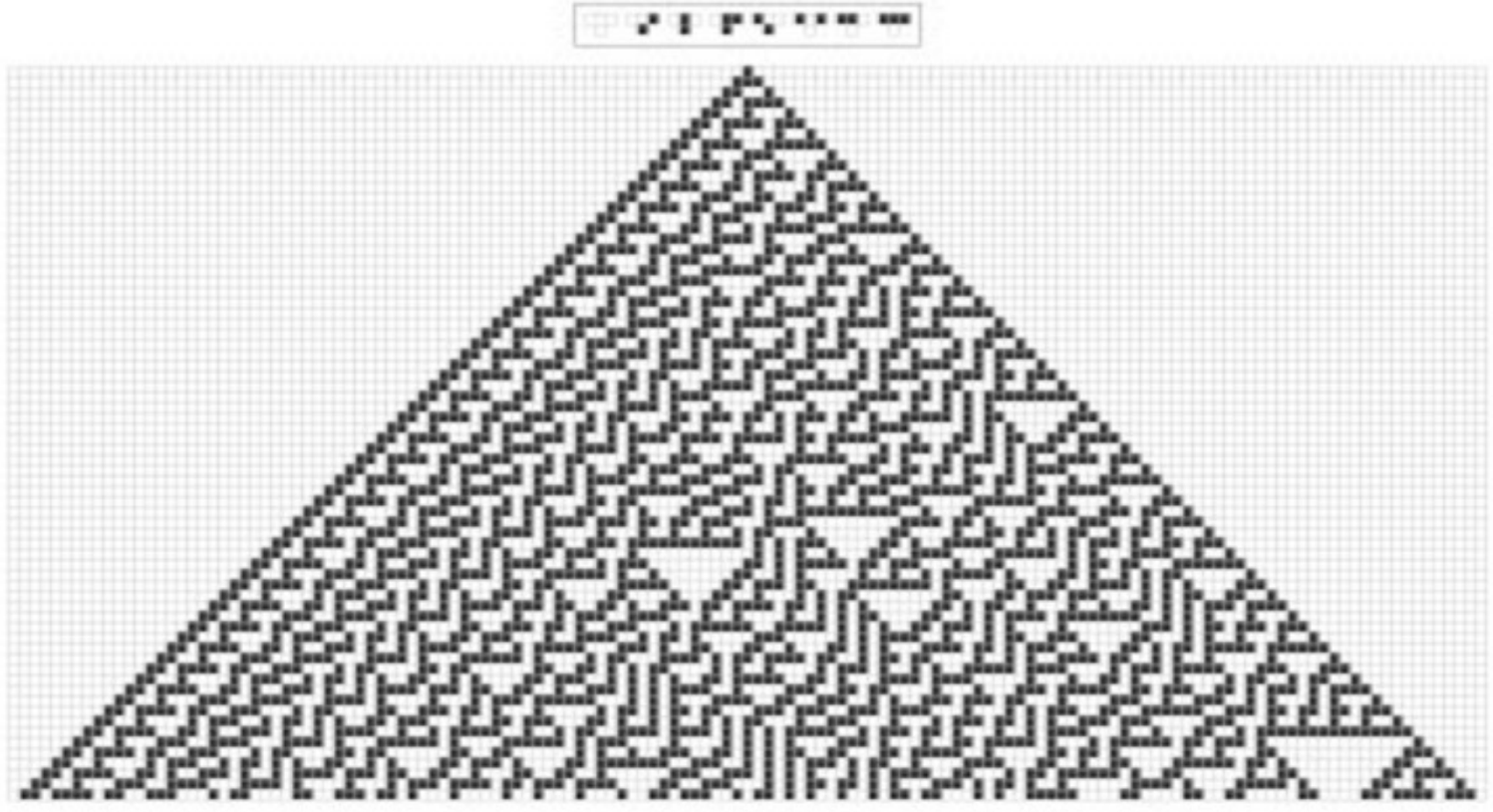
Classification by Behavior (Wolfram's Classes)

- Stephen Wolfram proposed a classification of cellular automata based on their long-term behavior:
- Class I: Homogeneity:
  - The system evolves to a uniform state (all cells have the same value).
  - Example: Rule 0 (all cells become 0).
- Class II: Periodicity:
  - The system evolves to a periodic or stable pattern.
  - Example: Rule 4 (repeats a simple pattern).
- Class III: Chaos:
  - The system evolves to a chaotic, aperiodic pattern.
  - Example: Rule 30 (produces complex, unpredictable behavior).
- Class IV: Complexity:
  - The system exhibits complex, localized structures that interact in

Wolfram's classification of CA

Class 1 Decaying structures	Class 2 Periodic or nested	Class 4 Interacting structures	Class 3 Chaotic behavior
			

Rule 30



https://www.youtube.com/watch?v=7izmHXpcc5A

Wolfram's classification of CA

- Classification is only qualitative, there is no classification algorithm.
- CA seem to cover the full range of complexity found in nature.
- Some CA are **computationally irreducible** (Wolfram hypothesized that most of class 3 and 4).
- Some CA are **universal Turing Machines** (Wolfram hypothesized that class 4 is the threshold).

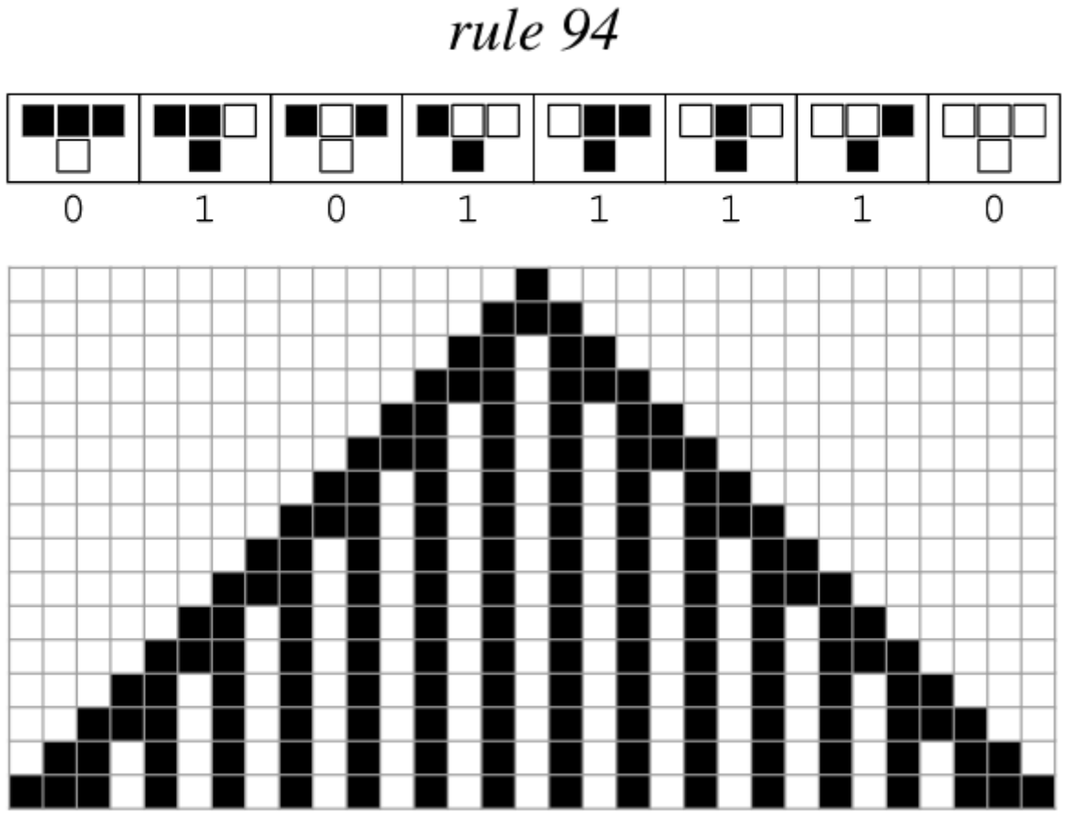
Computational Irreducibility and Predictability

- Computational irreducibility = the system behavior can be predicted only by running it. There is no computationally more efficient way.
- Some complex CA are computationally irreducible.
  - Wolfram hypothesized that most of class 3 and 4 CA are irreducible.
- In analogy to the real world:
  - Many physical processes that seem complex are probably computationally irreducible.
  - Most physical systems have too many degrees of freedom for direct simulation.
  - Are Complex Systems inherently unpredictable?

Working with them

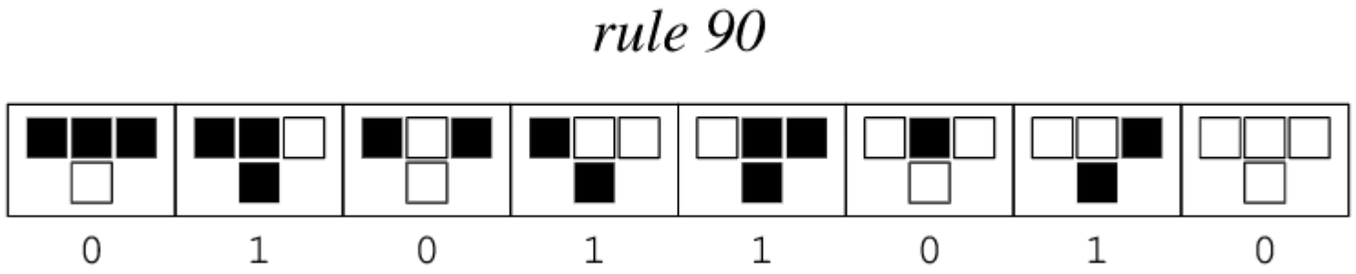
Some examples

Rule 94



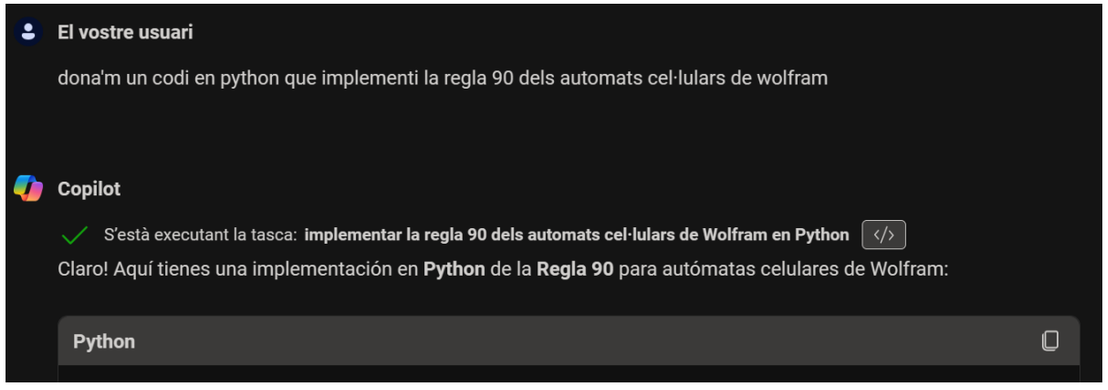
Source:: Weisstein, Eric W. "Rule 94." From *MathWorld*--A Wolfram Web Resource. https://mathworld.wolfram.com/Rule94.html

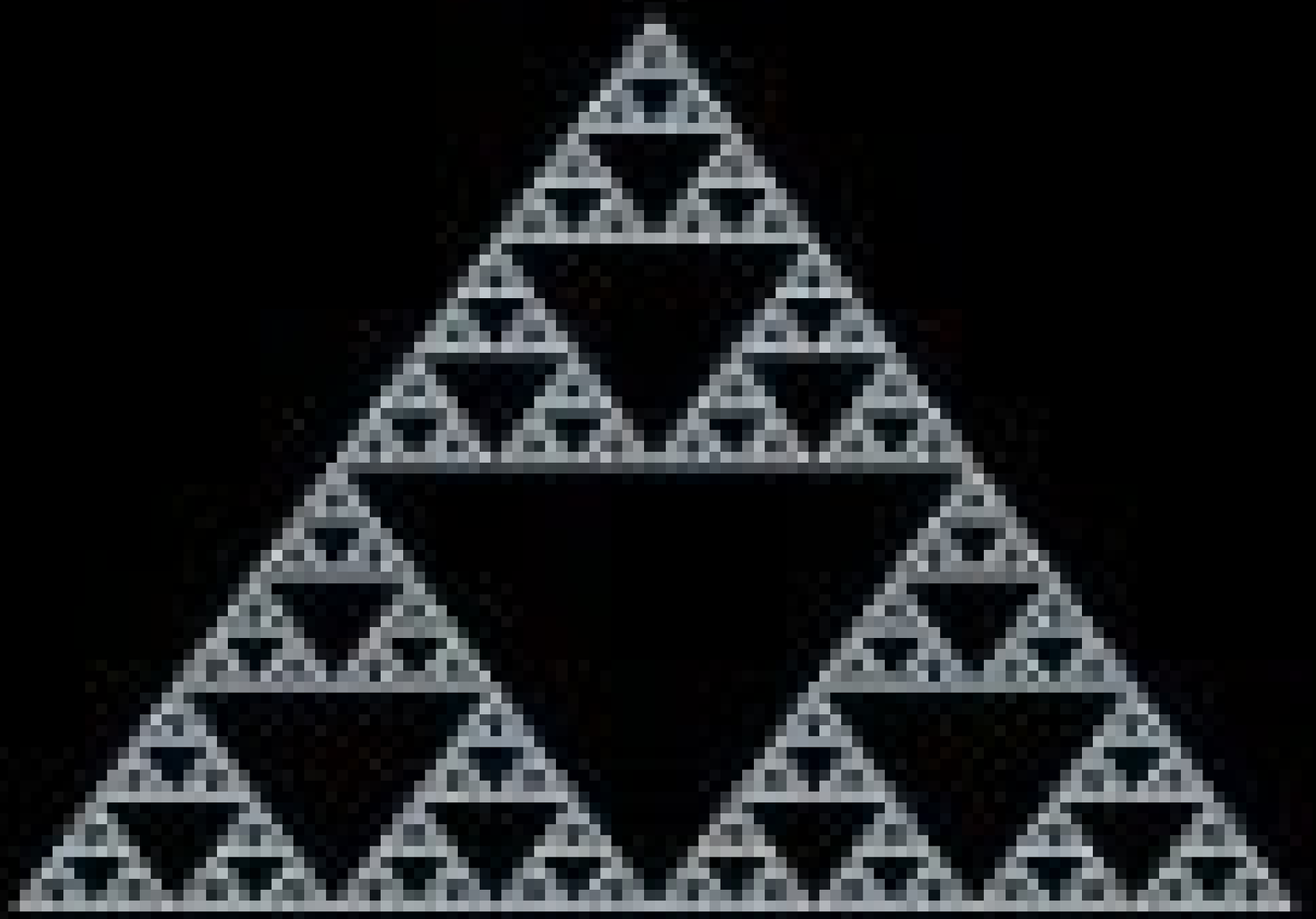
Rule 90



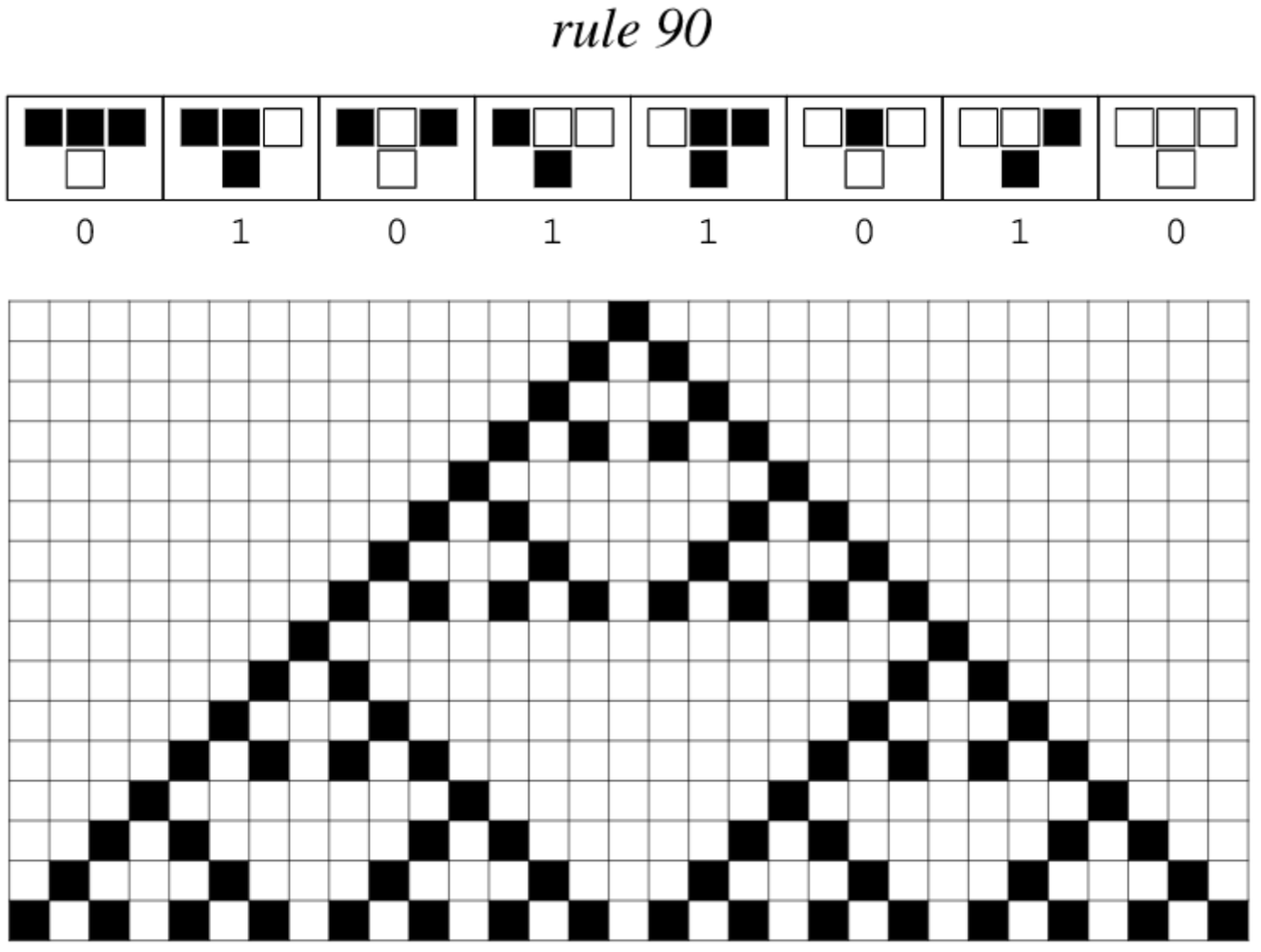
Source: Weisstein, Eric W. "Rule 90." From *MathWorld*--A Wolfram Web Resource. https://mathworld.wolfram.com/Rule90.html

```python
def regla_90(pasos):
    # Inicializamos la primera fila con una sola célula negra en el centro
    fila = [0] * pasos
    fila[pasos // 2] = 1

    # Inicializamos la lista de resultados con la primera fila
    resultado = [fila]

    # Generamos filas posteriores basadas en la regla 90
    for _ in range(pasos - 1):
        nueva_fila = []
        for i in range(pasos):
            izquierda = fila[i - 1] if i > 0 else 0
            centro = fila[i]
            derecha = fila[i + 1] if i < pasos - 1 else 0
            nueva_celula = (izquierda + centro + derecha) % 2
            nueva_fila.append(nueva_celula)
        fila = nueva_fila
        resultado.append(fila)

    return resultado

# Establecemos el número de pasos (filas) para el autómata
pasos = 20

# Generamos el autómata con la regla 90
automata = regla_90(pasos)

# Imprimimos el autómata
for fila in automata:
    print("".join(str(celula) for celula in fila))
```





Rule 90

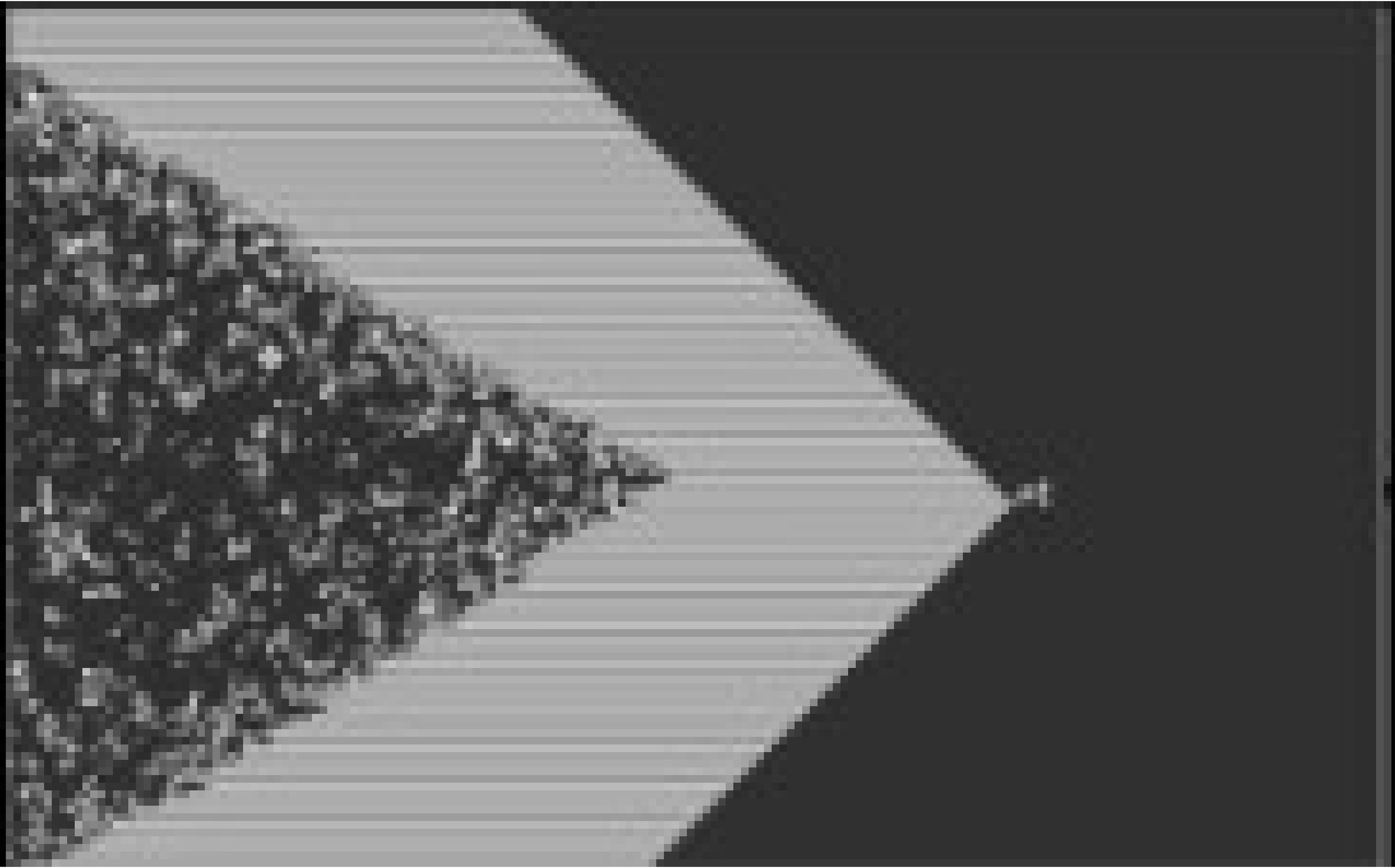


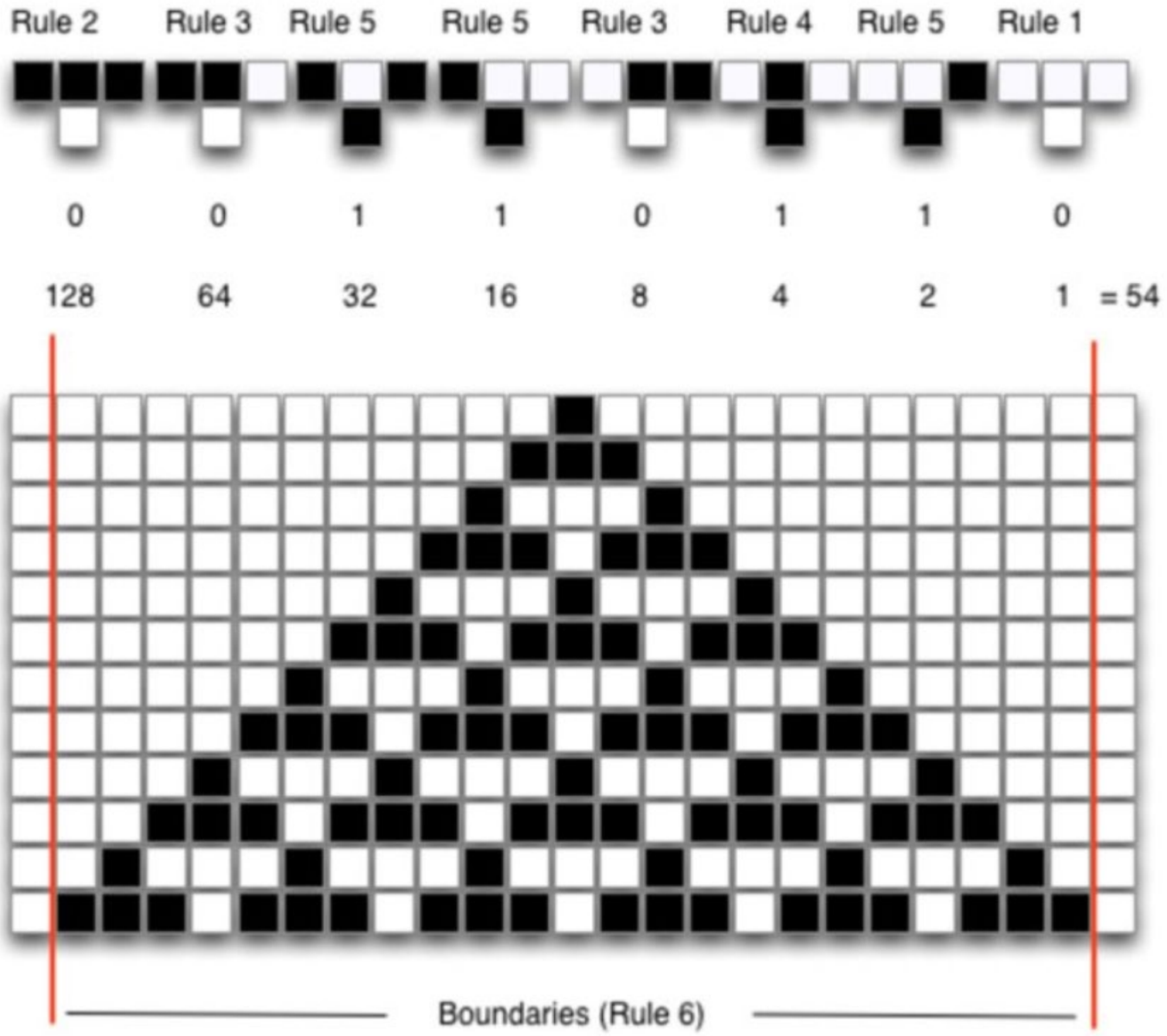
Source: Weisstein, Eric W. "Rule 90." From *MathWorld*--A Wolfram Web Resource. https://mathworld.wolfram.com/Rule90.html

Rule 54 Cellular Automaton

- Rule 54 Cellular Automaton

Source: https://www.researchgate.net/publication/317311801_Cellular_Automata_as_an_Example_for_Advanced_Beginners%27_Level_Coding_Exercises_in_a_MOOC_on_Test_Driven_Development/figures?lo=1




Rule 2
Rule 3
Rule 5
Rule 5
Rule 3
Rule 4
Rule 5
Rule 1
0
0
1
1
0
1
1
0
128
64
32
16
8
4
2
1
= 54
Boundaries (Rule 6)


Numberphile
,5,8,13,21

CA be coarse-grained?

Reduce CA space of states with a similar output

Predictable Coarse-Grained Behavior

- Usually, infinite precision is not required in Physics and coarse-grained or even statistical information is sufficient.
- Can CA be coarse-grained? (reduce its space of states with a similar output).
  - Coarse-grain-able CA are **compactable**. They have a coarse description with a smaller phase space.
  - Coarse-grain-able CA are **predictable** if they or their coarse-grained version are computationally reducible.
  - We found that at least some computationally irreducible CA has predictable coarse-grained behavior.

Coarse-grained procedure

- Coarse-graining is a procedure used to simplify the description of a system by grouping its components into larger units, thereby reducing the complexity of the system while preserving its essential features.
- In the context of cellular automata (CA), coarse-graining involves aggregating cells into blocks and defining new rules for the evolution of these blocks.
- This process can help reveal the large-scale behavior of the CA and identify emergent patterns or structures.

Coarse-grained procedure

1. Define the Original CA

- Start with a cellular automaton defined by:
  - A grid of cells, each in a finite state (e.g., 0 or 1).
  - A neighborhood (e.g., nearest neighbors).
  - A rule that determines the new state of a cell based on its current state and the states of its neighbors.

Coarse-grained procedure

2. Choose a Block Size

- Decide on the size of the blocks (also called "supercells") to which the cells will be aggregated. For example, you might group cells into blocks of size 2×2 or 3×3.
- Let k be the size of each block (e.g., k=2 for 2×2 blocks).

Coarse-grained procedure

3. Aggregate Cells into Blocks

- Divide the grid into non-overlapping blocks of size k×k.
- Each block will now represent a "supercell" in the coarse-grained system.

Coarse-grained procedure

4. Define the State of a Supercell

- Assign a new state to each supercell based on the states of the cells within the block. Common methods include:
  - Majority rule: The state of the supercell is the state that appears most frequently in the block.
  - Threshold rule: The supercell is assigned a state (e.g., 1) if the number of cells in the block with that state exceeds a certain threshold.
  - Averaging: If the states are numerical, the supercell's state could be the average of the states in the block.

Coarse-grained procedure

5. Define the Coarse-Grained Rule

- Determine how the supercells evolve over time. This involves defining a new rule for the coarse-grained CA based on the original rule and the block size.
- The new rule should capture the essential behavior of the original CA while operating on the larger scale of supercells.

Coarse-grained procedure

6. Simulate the Coarse-Grained CA

- Use the coarse-grained rule to simulate the evolution of the supercells over time.
- Compare the behavior of the coarse-grained CA with the original CA to ensure that the essential features are preserved.

Results of coarse-graining on elementary rules

- Based on wolfram's classification:
  - Coarse-grain 240 out of the 256 elementary CA by trying different choices of super-cell sizes and projection operators.
  - It might be possible to coarse-grain the other 16 rules (30, 45, 106, 154 and their symmetries) by trying larger super-cell sizes.
  - Many elementary CA can be coarse-grained by other elementary CA.
  - Coarse-grained-able CA include members of all four classes.
  - At least one computationally irreducible CA can be coarse-grained by a reducible CA and is therefore predictable on a coarse level

Results of coarse-graining elementary rules

- See, Israeli, N., & Goldenfeld, N. (2004). Computational Irreducibility and the Predictability of Complex Physical Systems. *Physical Review Letters*, *92*(7), 1–4. https://doi.org/10.1103/PhysRevLett.92.074105

Applications of Coarse-Graining in CA

- Simplifying Complex Systems:
  - Coarse-graining reduces the complexity of a CA, making it easier to analyze and simulate.
- Identifying Emergent Patterns:
  - By focusing on larger-scale structures, coarse-graining can reveal emergent patterns or behaviors that are not apparent at the level of individual cells.
- Studying Universality:
  - Coarse-graining can help determine whether a CA exhibits universal behavior (e.g., Turing completeness) at different scales.
- Connecting Microscopic and Macroscopic Behavior:
  - Coarse-graining bridges the gap between the microscopic rules of a CA and its macroscopic behavior, providing insights into how local interactions give rise to global phenomena.

Challenges in Coarse-Graining

- Loss of Detail:
  - Coarse-graining involves a loss of information, as the fine-scale details of the original CA are aggregated into larger units.
- Defining the Coarse-Grained Rule:
  - It can be challenging to define a coarse-grained rule that accurately captures the behavior of the original CA.
- Preserving Essential Features:
  - The coarse-grained CA must preserve the essential features of the original CA, such as emergent patterns or critical phenomena.

Coarse-graining example

Example: Coarse-Graining a 1D Cellular Automaton

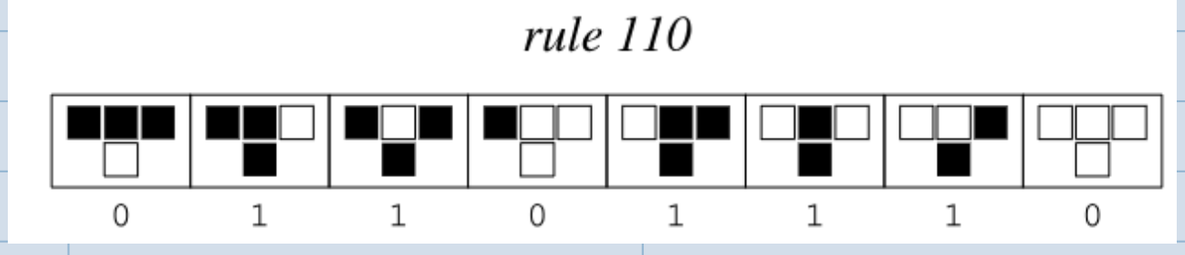
Example: Coarse-Graining a 1D Cellular Automaton

- Consider a 1D cellular automaton with the following properties:
- Each cell can be in state 0 or 1.
- The neighborhood consists of the cell itself and its two nearest neighbors (left and right).
- The rule is Rule 110 (a well-known Turing-complete rule).

Step 1: Define the Original CA

- The original CA is a 1D grid of cells evolving according to Rule 110.
  - REMEMBER: Rule 110 is a 1D cellular automaton where each cell can be in state 0 or 1. The new state of a cell depends on its current state and the states of its left and right neighbors. The rule table for Rule 110 is:

Left Neighbor	Current State	Right Neighbor	New State
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	1
0	0	0	0

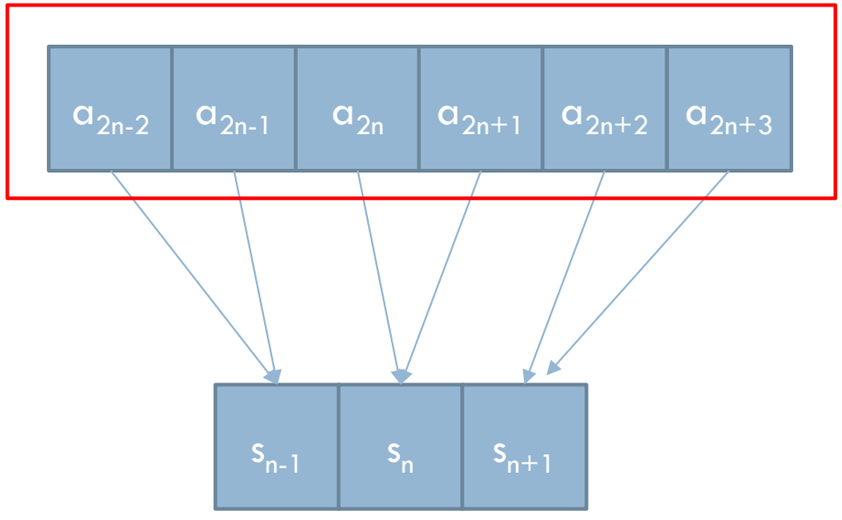


Step 2: Choose a Block Size

- Let k=2, meaning we will group cells into blocks of 2 adjacent cells.

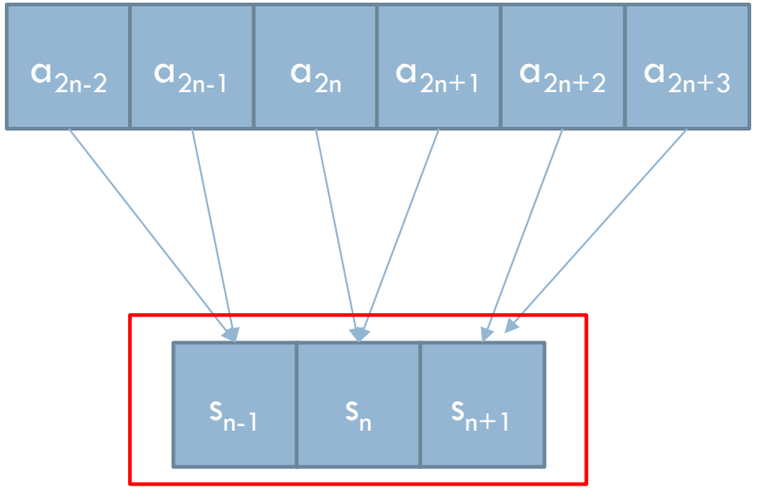
Step 3: Aggregate Cells into Blocks

- Divide the 1D grid into non-overlapping blocks of size 2. For example:
  - Original grid: …,0,1,1,0,1,0,0,1,…
  - Blocks: …,(0,1),(1,0),(1,0),(0,1),…



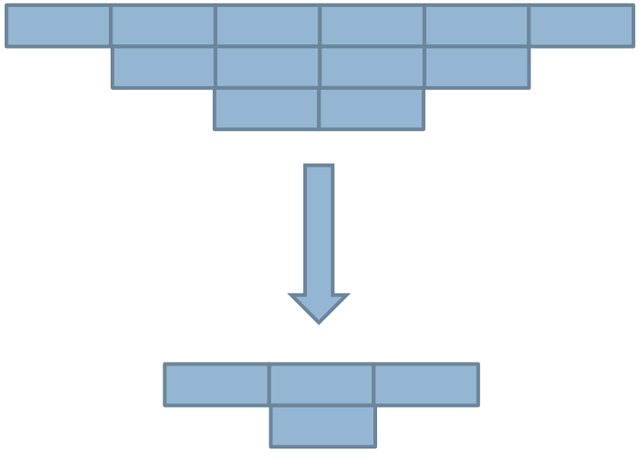
Step 4: Define the State of a Supercell

- Use the majority rule to define the state of each supercell:
  - If a block has more 1s than 0s, its supercell state is 1.
  - If a block has more 0s than 1s, its supercell state is 0.
  - If the block has an equal number of 1s and 0s, choose a default state (e.g., 0).



Step 5: Define the Coarse-Grained Rule

- The new rule for the coarse-grained CA should describe how the supercells evolve based on their current states and the states of their neighboring supercells.
- For example, you might define a rule table for the coarse-grained CA based on the behavior of the original Rule 110.



Step 5: Define the Coarse-Grained Rule

- The coarse-grained rule describes how the supercells evolve based on their current states and the states of their neighboring supercells. To define this rule, we need to:
  - Map the original Rule 110 to the supercell level.
  - Determine the new state of a supercell based on its current state and the states of its left and right neighboring supercells.

Step 5: Define the Coarse-Grained Rule

- Enumerate all possible states of a supercell and its neighbors:
  - Each supercell can be in one of 3 states: (0,0), (0,1), (1,0), or (1,1).
  - After applying the majority rule, the supercell states are:
    - (0,0)→0
    - (0,1)→0 (default)
    - (1,0)→0 (default)
    - (1,1)→1
- Simulate the evolution of supercells using the original Rule 110:
  - For each possible configuration of a supercell and its left and right neighbors, apply the original Rule 110 to the underlying cells and then apply the majority rule to determine the new supercell state.

Step 5: Define the Coarse-Grained Rule

- Construct the Coarse-Grained Rule Table
- We construct a rule table for the coarse-grained CA by considering all possible configurations of a supercell and its left and right neighbors. Here's how:
  - List all possible configurations:
    - Each supercell and its neighbors can be in one of 2 states (0 or 1), so there are $2^3 = 8$ possible configurations for the left neighbor, current supercell, and right neighbor.
  - For each configuration:
    - Expand the supercells into their underlying cells (blocks of 2 cells).
    - Apply the original Rule 110 to the underlying cells.
    - Apply the majority rule to the resulting block to determine the new supercell state.

Step 5: Define the Coarse-Grained Rule

- Consider the configuration where:
  - Left supercell = 1
  - Current supercell = 1
  - Right supercell = 0
- Expand the supercells into cells:
  - Left supercell (1): (1,1)
  - Current supercell (1): (1,1)
  - Right supercell (0): (0,0)

Step 5: Define the Coarse-Grained Rule

- Apply Rule 110 to the underlying cells:
  - The expanded neighborhood for each cell in the current supercell is:
    - For the first cell in the current supercell:
      - Left neighbor: Second cell of the left supercell (1)
      - Current state: First cell of the current supercell (1)
      - Right neighbor: Second cell of the current supercell (1)
      - Rule 110: (1,1,1)→0
    - For the second cell in the current supercell:
      - Left neighbor: First cell of the current supercell (1)
      - Current state: Second cell of the current supercell (1)
      - Right neighbor: First cell of the right supercell (0)
      - Rule 110: (1,1,0)→1

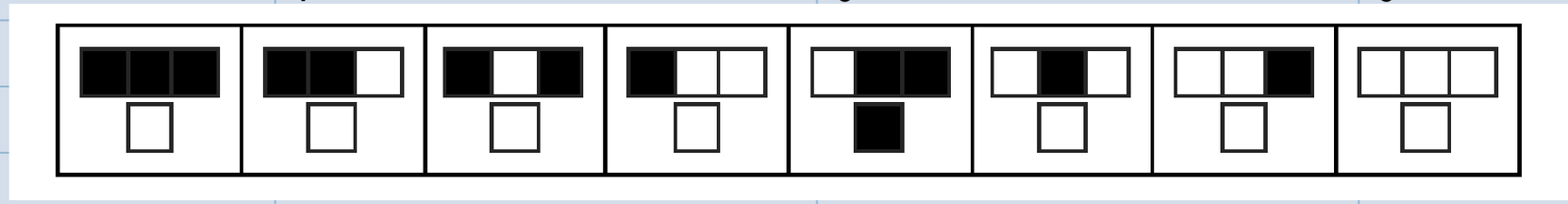
Step 5: Define the Coarse-Grained Rule

- Resulting block:
  - The new block is (0,1).
- Apply the majority rule:
  - The majority state in (0,1) is 0 (default).
- New supercell state:
  - The new supercell state is 0.

Step 5: Define the Coarse-Grained Rule

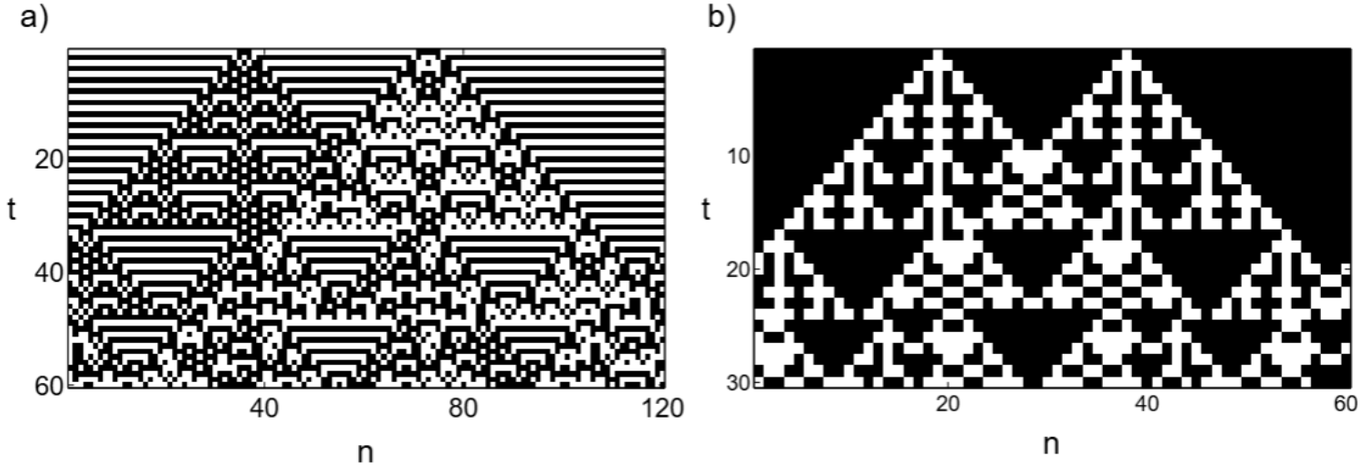
- Coarse-Grained Rule Table
- The coarse-grained rule table for k=2 using the majority rule is as follows:

Left Supercell	Current Supercell	Right Supercell	New Supercell State
1	1	1	0
1	1	0	0
1			
1			
0			
0	1	0	0
0	0	1	0
0	0	0	0



Step 6: Simulate the Coarse-Grained CA

- Use the coarse-grained rule table to simulate the evolution of the supercells.
- Compare the behavior of the coarse-grained CA with the original Rule 110 to ensure that the essential features are preserved.



https://www.researchgate.net/publication/7175386_Coarse-graining_of_cellular_automata_emergence_and_the_predictability_of_complex_systems

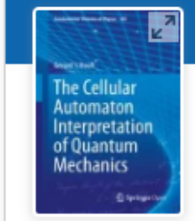
FIG. 2: Coarse-graining of rule 105 by rule 150. (a) shows results of running rule 105. The top line is the initial condition and time progress from top to bottom. (b) shows the results of running rule 150 with the coarse grained initial condition from (a).

Wolfram's classification of ECA

Wolfram's Class I									
0	8	32	40	128	136	160	168		
Wolfram's Class II									
1	2	3	4	5	6	7	9	10	11
12	13	14	15	19	23	24	25	26	27
28	29	33	34	35	36	37	38	41	42
43	44	46	50	51	56	57	58	62	72
73	74	76	77	78	94	104	108	130	132
134	138	140	142	152	154	156	162	164	170
172	178	184	200	204	232				
Wolfram's Class III									
18	**22**	**30**	**45**	**60**	**90**	**105**	**122**	**126**	**146**
150									
Wolfram's Class IV									
54	*106*	*110*							

m:n-CA^k

Extending the capabilities of the cellular automata



Book | Open Access | © 2016

The Cellular Automaton Interpretation of Quantum Mechanics

Home > Book

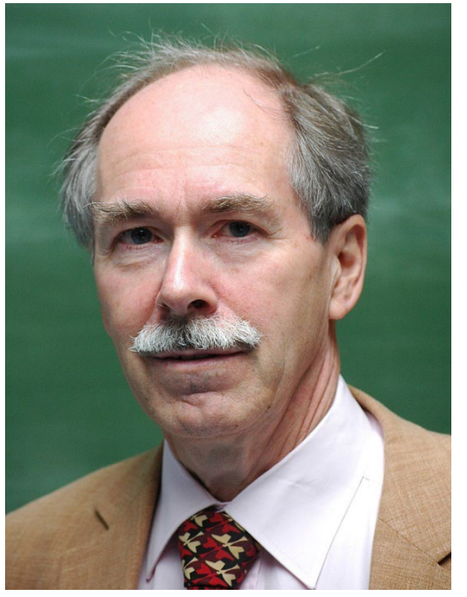
Authors: Gerard 't Hooft

A radically different approach to the challenges of quantum theory

Authored by one of today's most creative theoretical physicists

Pedagogically structured to first introduce concepts and main arguments, with technical details presented separately

Part of the book series: Fundamental Theories of Physics (FTPH, volume 185)



The Cellular Automaton Interpretation of Quantum Mechanics | SpringerLink

New!

What is Topology?

- Definition: Topology is a branch of mathematics that studies properties of space that are preserved under continuous deformations (stretching, bending, twisting, but not tearing or gluing).
- Key Idea: Focuses on the concept of "closeness" and "continuity" without relying on distance.
- Visual: Show a coffee cup transforming into a donut (classic example of topological equivalence).

Key Concepts in Topology

- Topological Space: A set of points with a collection of open sets satisfying certain axioms.
- Continuity: A function is continuous if the inverse image of an open set is open.
- Homeomorphism: A bijective continuous function with a continuous inverse (preserves topological properties).

Topological invariants

- A topological invariant is a property of a topological space that is preserved under homeomorphisms (continuous deformations). If two spaces have different invariants, they cannot be homeomorphic.

https://www.youtube.com/watch?v=ixduANVe0gg

Connectedness

- A topological space is connected if it cannot be divided into two disjoint non-empty open sets. Intuitively, it is "all in one piece."

Types of Connectedness:

- Connected:
  - A space is connected if it is not the union of two disjoint non-empty open sets.
  - Example: The interval [0,1] is connected.
- Path-Connected:
  - A space is path-connected if any two points can be connected by a continuous path.
  - Example: A circle is path-connected.
- Simply Connected:
  - A space is simply connected if it is path-connected and every loop can be continuously shrunk to a point.
  - Example: A sphere is simply connected, while a torus is not.
- Disconnected:
  - A space is disconnected if it can be divided into two disjoint non-empty open sets.
  - Example: The set [0,1)U(1,2] is disconnected.

Compactness

- A topological space is compact if every open cover has a finite subcover. Intuitively, it is "small" or "finite" in some sense.

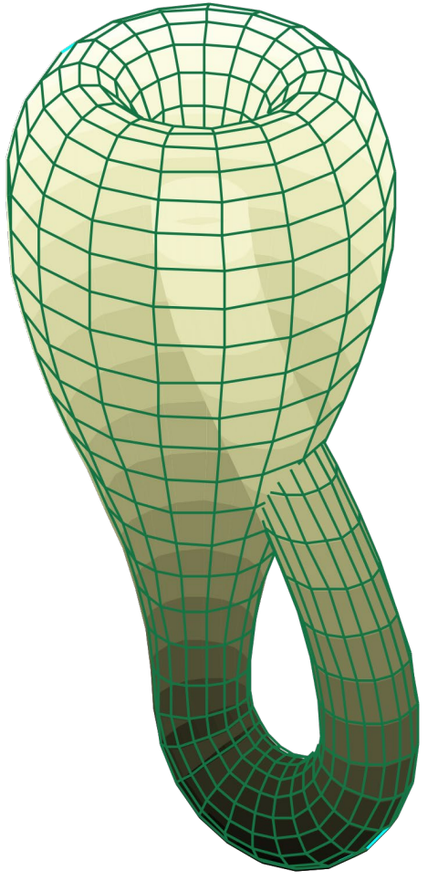
Examples of Topological Spaces

- Euclidean Space ($\mathbb{R}^n$): The standard space with the usual topology.
- Sphere (S^2): The surface of a ball.
- Torus (T^2): The surface of a donut.
- Möbius Strip: A surface with only one side and one edge.

Topological Equivalence

- Homeomorphism: Two spaces are topologically equivalent if there is a homeomorphism between them.
  - A coffee cup and a donut are homeomorphic.
  - A circle and a square are homeomorphic.
- Non-Examples:
  - A circle and a line are not homeomorphic.

Torus Eversion: Klein Bottles



By Tttrung - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=960446

Definition of an Open Set

- In topology, an open set is a set that satisfies certain axioms, which are designed to generalize the intuitive notion of "openness" in familiar spaces like $\mathbb{R}^n$.
- The empty set and the whole set are open:
  - $\emptyset \in \tau$
  - $X \in \tau$
- Arbitrary unions of open sets are open:
  - If $\{U_i\}_{i \in I}$ is a collection of sets in τ, then $\bigcup_{i \in I} U_i \in \tau$.
- Finite intersections of open sets are open:
  - If $U_1, U_2, \ldots, Un \in \tau$, then $U_1 \cap U_2 \cap \cdots \cap Un \in \tau$.
- The sets in τ are called open sets, and the pair (X,τ) is called a topological space.

Intuitive Understanding of Open Sets

- In $\mathbb{R}^n$, an open set is a set where every point has a "neighborhood" entirely contained within the set.
- For example:
  - An open interval (a,b) in $\mathbb{R}$ is an open set.
  - An open ball in $\mathbb{R}^n$ (a disk without its boundary) is an open set.
- In topology, the concept of an open set is abstracted to any set X, ven if X does not have a notion of distance.

Examples of Topologies Generated by Open Sets

- Standard Topology on $\mathbb{R}$:
  - The open sets are unions of open intervals (a,b)(a,b).
  - Example: (0,1)U(2,3) is an open set in $\mathbb{R}$.
- Discrete Topology:
  - Every subset of X is open.
  - Example: If X={a,b,c}X={a,b,c}, then τ={∅,{a},{b},{c},{a,b},{a,c},{b,c},X}.
- Indiscrete Topology:
  - Only ∅ and X are open.
  - Example: If X={a,b,c}, then τ={∅,X}.
- Finite Complement Topology:
  - A set is open if its complement is finite or the whole space.
  - Example: On $\mathbb{R}$, the set $\mathbb{R}$ \{1,2,3} is open.

Why Are Open Sets Important?

- An open set is a subset of a topological space that satisfies the axioms of a topology.
- Open sets are the building blocks of a topological space, defining its structure and enabling the study of continuity, convergence, and other topological properties.
- Different choices of open sets lead to different topologies, allowing for a wide range of mathematical exploration and application.
- Key properties
  - Flexibility: Open sets allow us to define a topology on any set, even if the set does not have a natural notion of distance.
  - Generalization: They generalize familiar concepts like continuity and convergence to abstract spaces.
  - Structure: They provide a way to describe the "shape" and "structure" of a space without relying on metrics.

Working with a topology

- Defining a topological space involves specifying a set X and a collection of subsets of X (called open sets) that satisfy certain axioms.
- These axioms generalize the intuitive notion of "openness" and provide a foundation for studying continuity, convergence, and other topological properties.

Multi n-dimensional CA

- A multi n dimensional cellular automaton is a cellular automaton generalization composed by m layers with n dimensions each one.
- Aim:
  - Allows multiple layers.
  - Allows vectorial layers (continuous space).
  - Allows multiple set of rules (evolution functions).

Multi n−dimensional CA

- The representation is: $m: n - AC^k$
- where
  - m: is the automaton number of layers.
  - n: is the different layers dimension.
  - k: is the number of main layers (1 by default).

For k=1, m=1 we have a common cellular automata.
For k=0 we have a dataset.

Multi n-dimensional CA

- Defined over the mathematical topology concept.
- $1{:}n - AC^{1}$ is the common cellular automata if the topology used is the discrete topology defined over $\mathbb{R}$ or $\mathbb{Z}$.
- The implementation, as is usual, can be a matrix.

State of the automata

- S_m^t state of the automaton, while E_m considering geo-references.
- Em[x1,..,xn], layer m state in x1,..,xn position
  - Em is a function describing cell state in position x1,..,xn of layer m.
- EG[x1,..,xn], automata status in x1,..,xn position.
  - EG returns automata global state in position referenced by coordinates x1,..,xn.

Evolution function Λm

- Evolution function allows global automaton state change through cells value modification.
- Defined for the layer m to modify its state through the state of others layers using combination function Ψ, and vicinity and nucleus functions.
- Is only defined in main layers.

Evolution function Λm

- Function defined for the layer m to modify its state through the state of others layers using combination function Ψ, and vicinity and nucleus functions.
- A m:n-AC automaton only presents one main layer, an m:n-ACk automaton presents k main layers.

Vicinity topology

- Topology defining the set of points (neighborhood) for layer m, to be considered for Λm calculus.
- Vicinity function vn(x1,..,xn):
  - Function returning minimum open set of vicinity topology containing point x1,..,xn, and including maximum points that accomplishes the restriction and minimum points not accomplishing the restriction.

Nucleous topology

- Topology defining the set of points (neighborhoods) for layer m, to be modified by Λm calculus
- Nucleus function nc(x1,..,xn)
  - Function returning minimum open set of nucleus topology containing point x1,..,xn, and including maximum points that accomplishes the restriction and minimum points not accomplishing the restriction.

Vicinity function example

- Over Z
  - vn(x1,..,xn) = {(x1-1,x2-1,xn-1), (x1-1,x2-1,Xn), (x1-1,x2-1,xn+1), (x1-1,x2,xn-1), (x1-1,x2,Xn), (x1-1,x2,xn+1), (x1-1,x2+1,xn-1), (x1-1,x2+1,Xn), (x1-1,x2+1,xn+1), (x1,x2-1,xn-1), (x1,x2-1,Xn), (x1,x2-1,xn+1), (x1,x2,xn-1), (x1,x2,Xn), (x1,x2,xn+1), (x1,x2+1,xn-1), (x1,x2+1,Xn), (x1,x2+1,xn+1), (x1+1,x2-1,xn-1), (x1+1,x2-1,Xn), (x1+1,x2-1,xn+1), (x1+1,x2,xn-1), (x1+1,x2,Xn), (x1+1,x2,xn+1), (x1+1,x2+1,xn-1), (x1+1,x2+1,xn), (x1+1,x2+1,xn+1)}
- Over R
  - vn(x1,..,xn)=returns the open set centered in the point x1,x2,x3 for the topology that defines the vicinity, $B(x,r) = \{y \in \Re^m / d(x,y) < r\}$

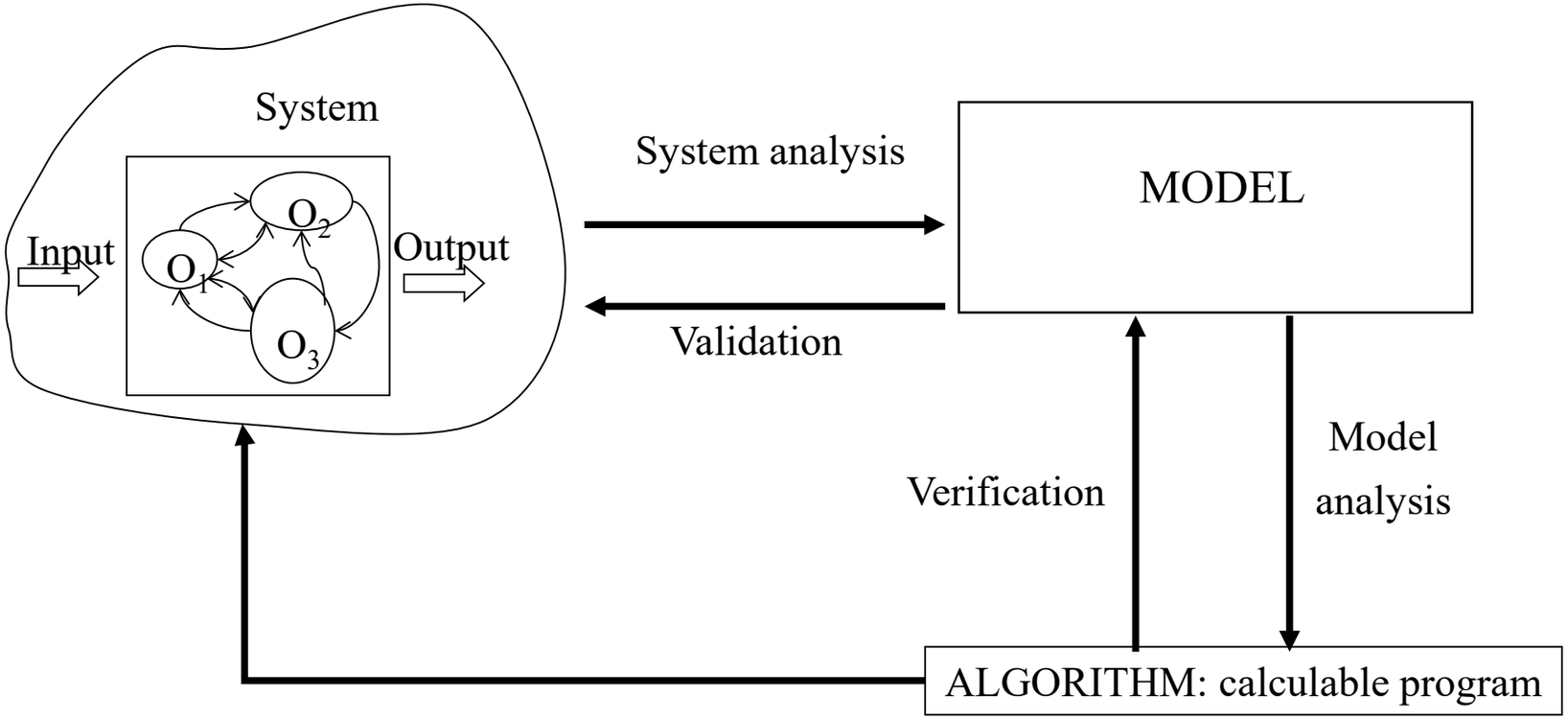
Nucleus function example

- Over Z
- nc(x1,..,xn) = {(x1,..,xn)}
- Over R
- nc(x1,..,xn)=returns the open set centered in the point x1,x2,x3 for the topology that defines the nucleus.

Using SDL

Defining CA models with SDL

Remember



Obtaining answers, data from the executions of the model.

Implementing the results?

Alternatives

- CELL-DEVS, SDL
- Both are equivalent. Also, we can use Petri Nets, see
  - I. Barragán, J. C. Seck-Tuoh, and J. Medina, *Relationship between Petri Nets and Cellular Automata for the Analysis of Flexible Manufacturing Systems*, in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 7630 LNAI (2013), pp. 338–349.

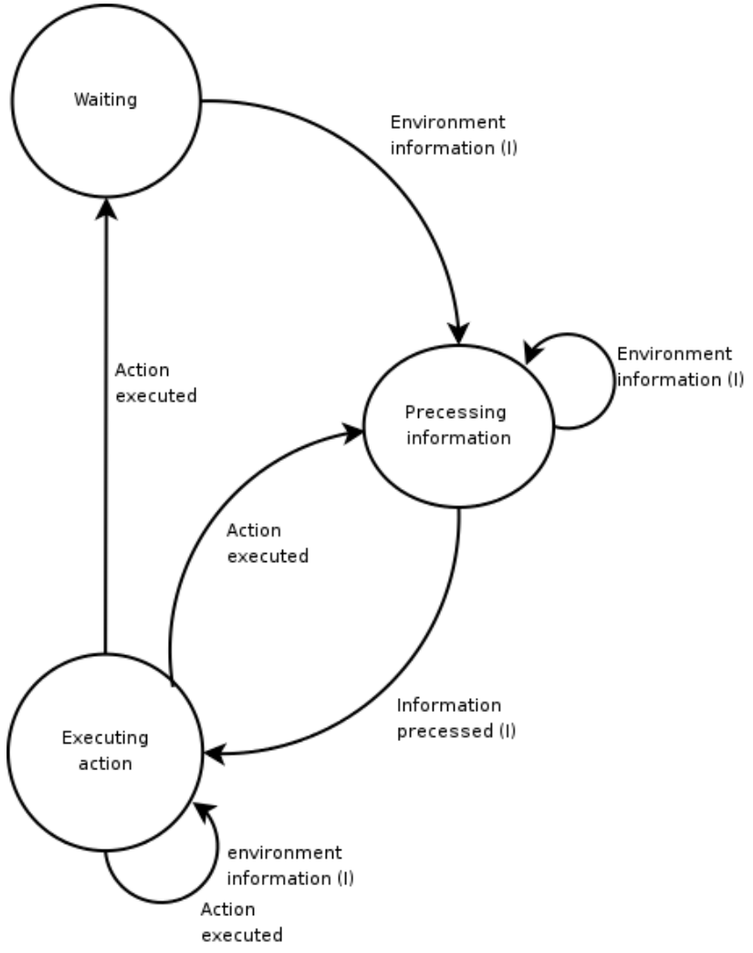
Alternatives

- We must consider
  - What is the tool we have?
  - What is the personnel involved in the project?
  - We have components to reuse?
  - Are this formalisms and languages ready to represent the needed structures for our models?

MAS

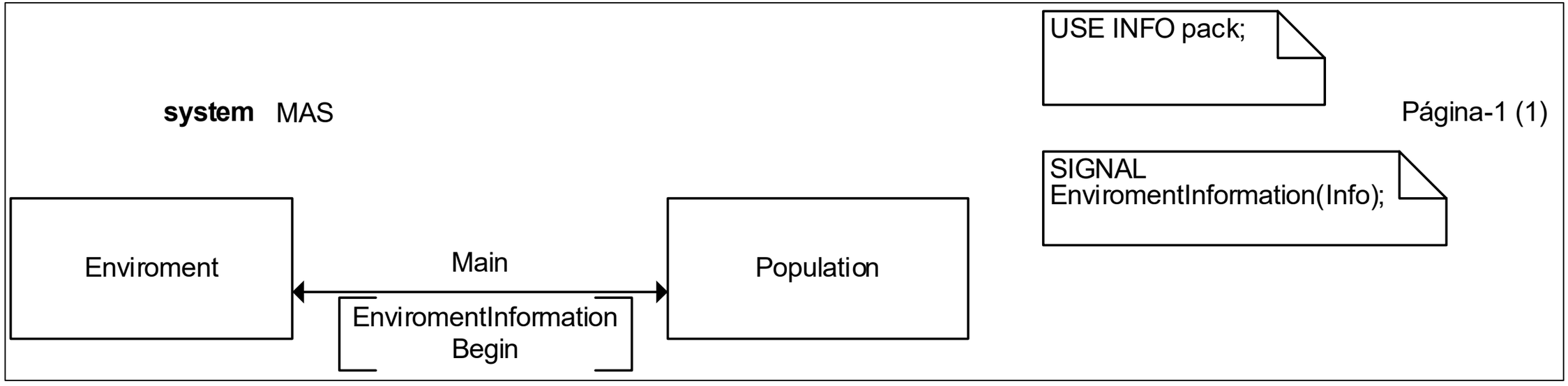
- But, let's start with a MAS (Multi Agent Simulation model).

Reflexive agent specification

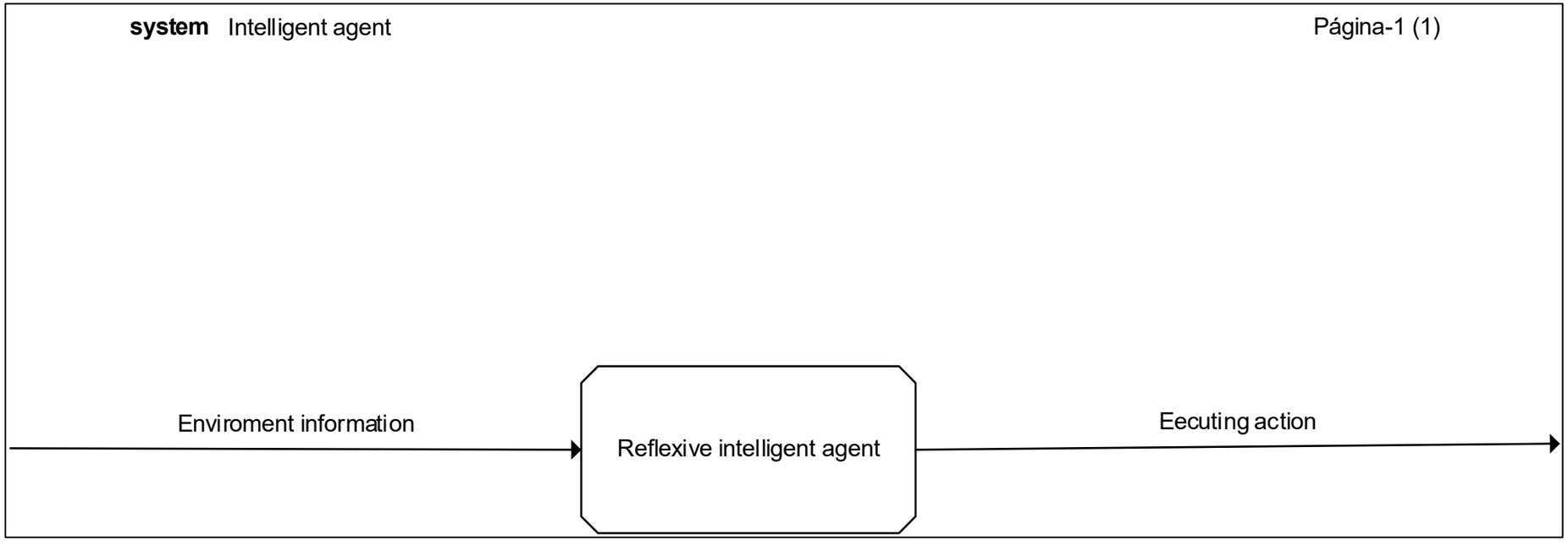


Reflexive agent specification

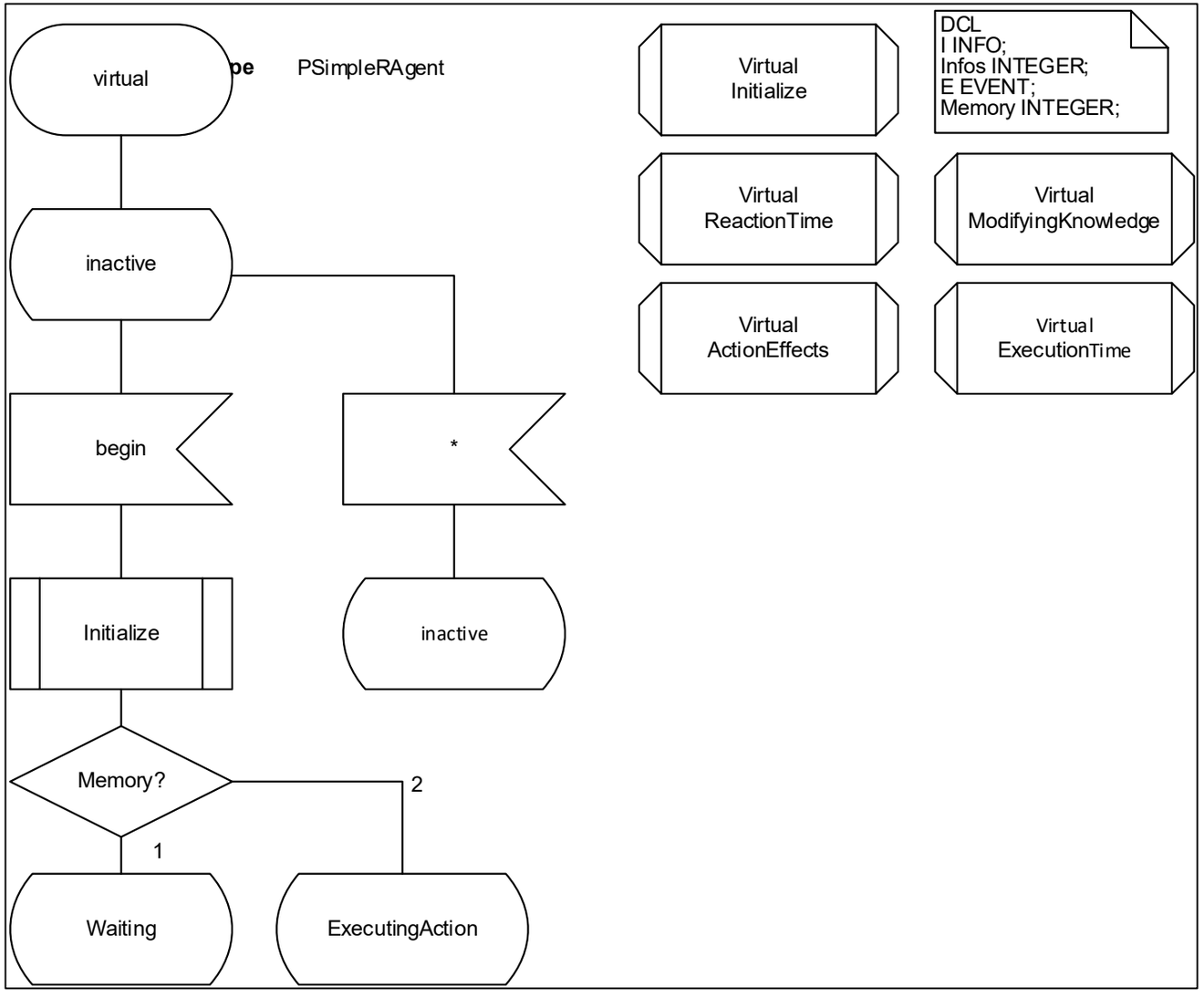
- Time to process information represent the delay due to the understanding of what happens in the world



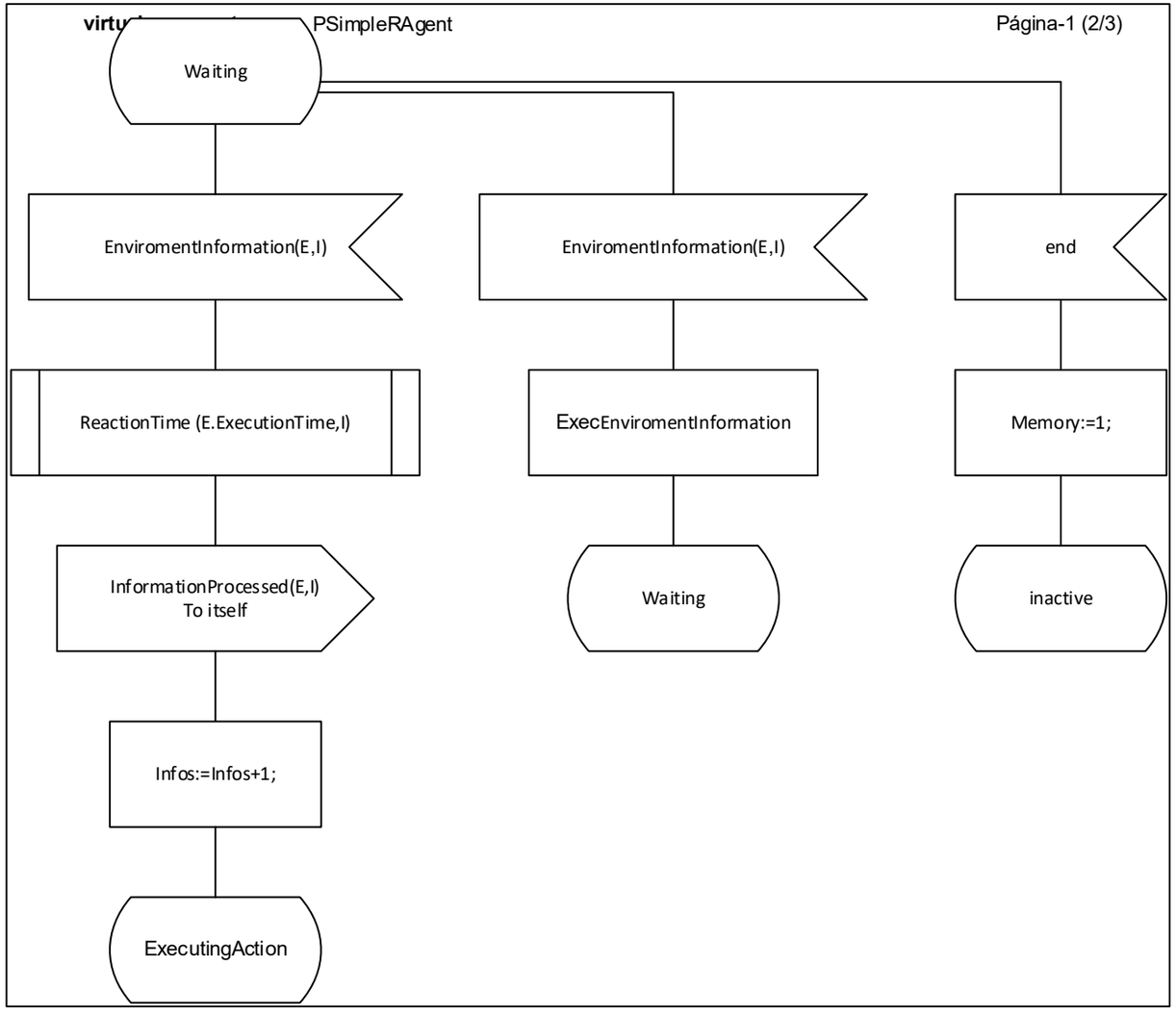
Reflexive agent specification

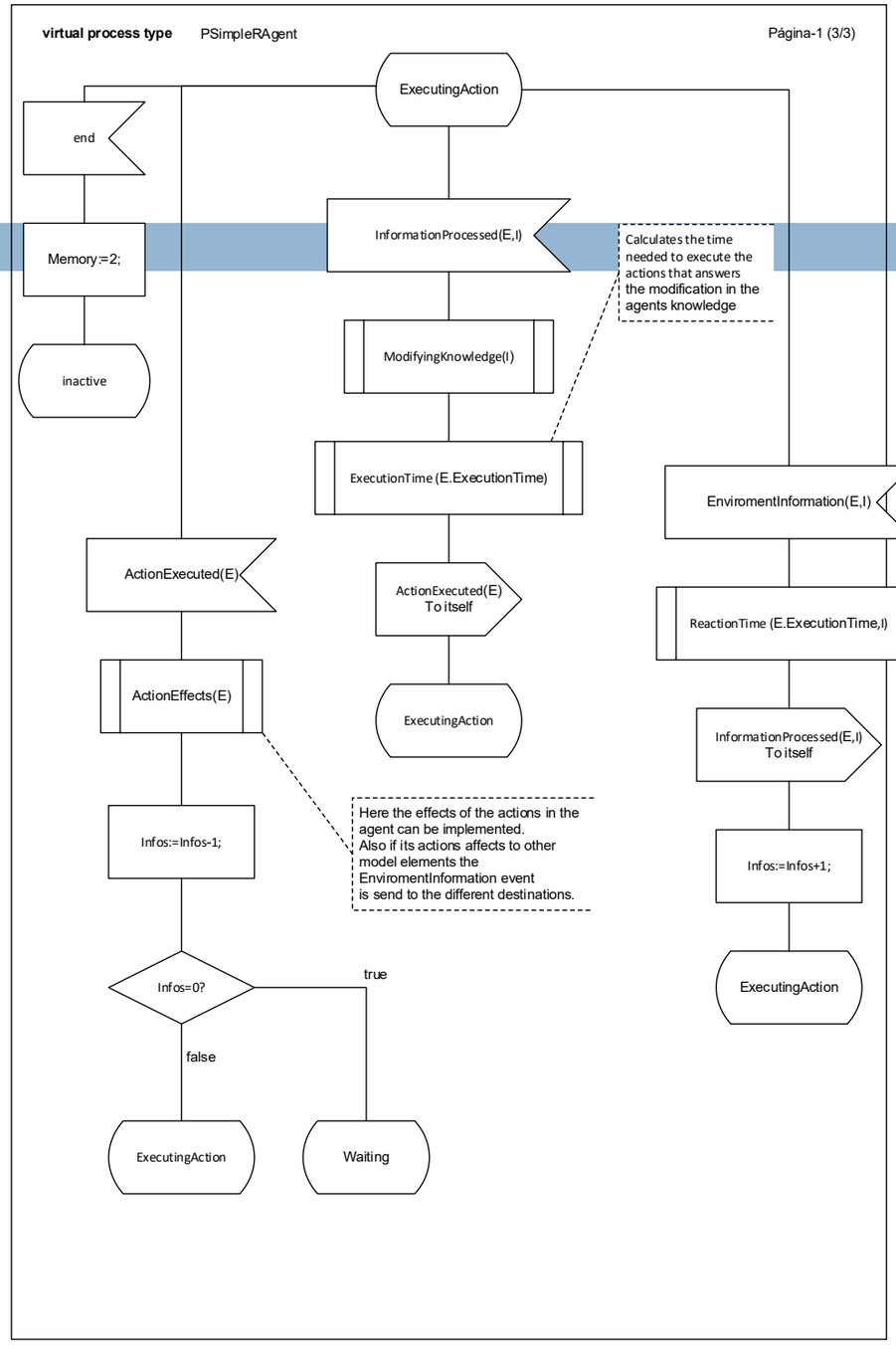


Reflexive agent specification



Reflexive agent specification



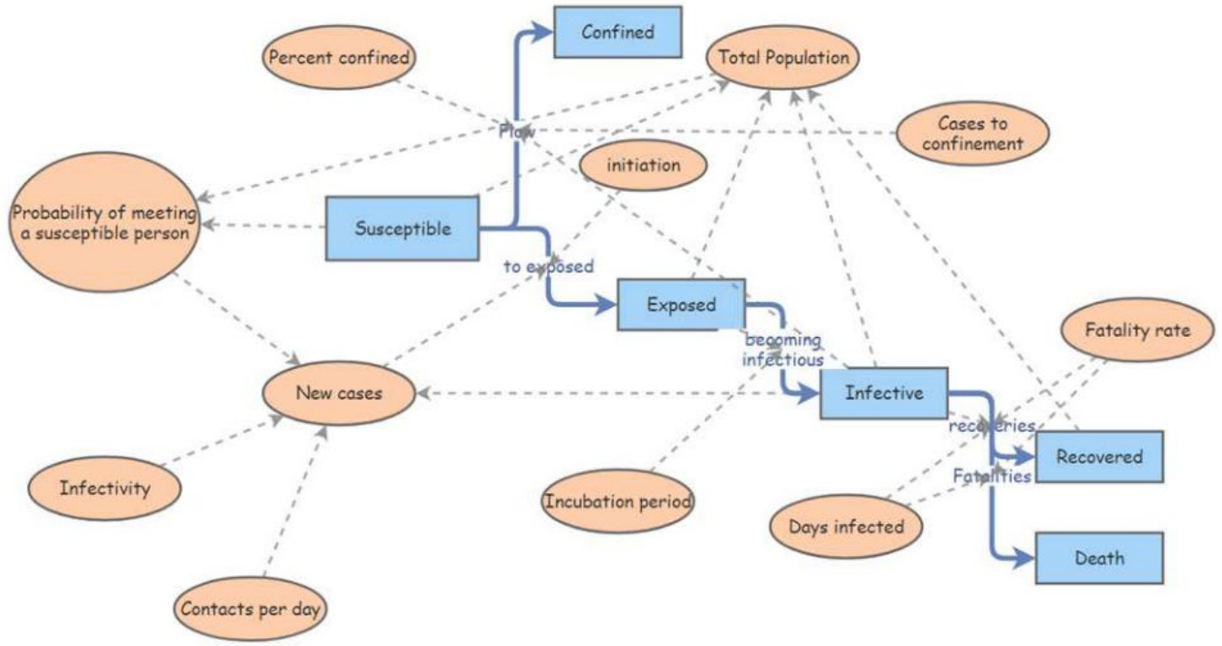


CA specification

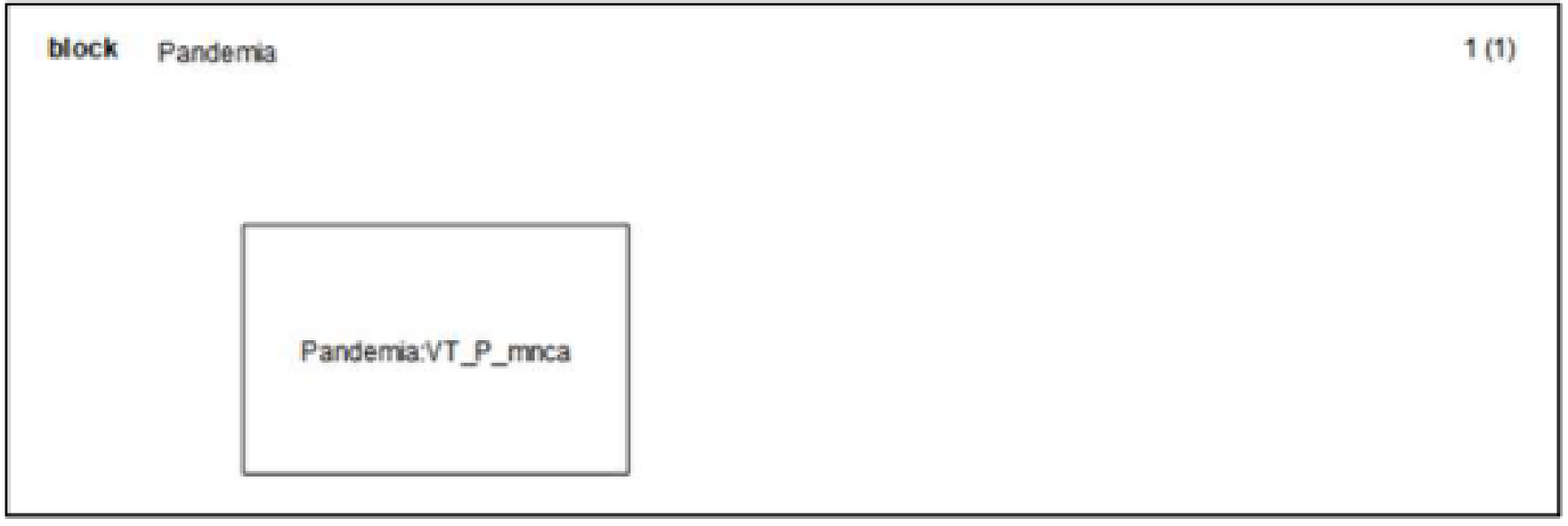
COVID-19 pandèmic model

Justification

- Need to simplify the validation process of the model.
- Need to replicate the behavior faster.



System

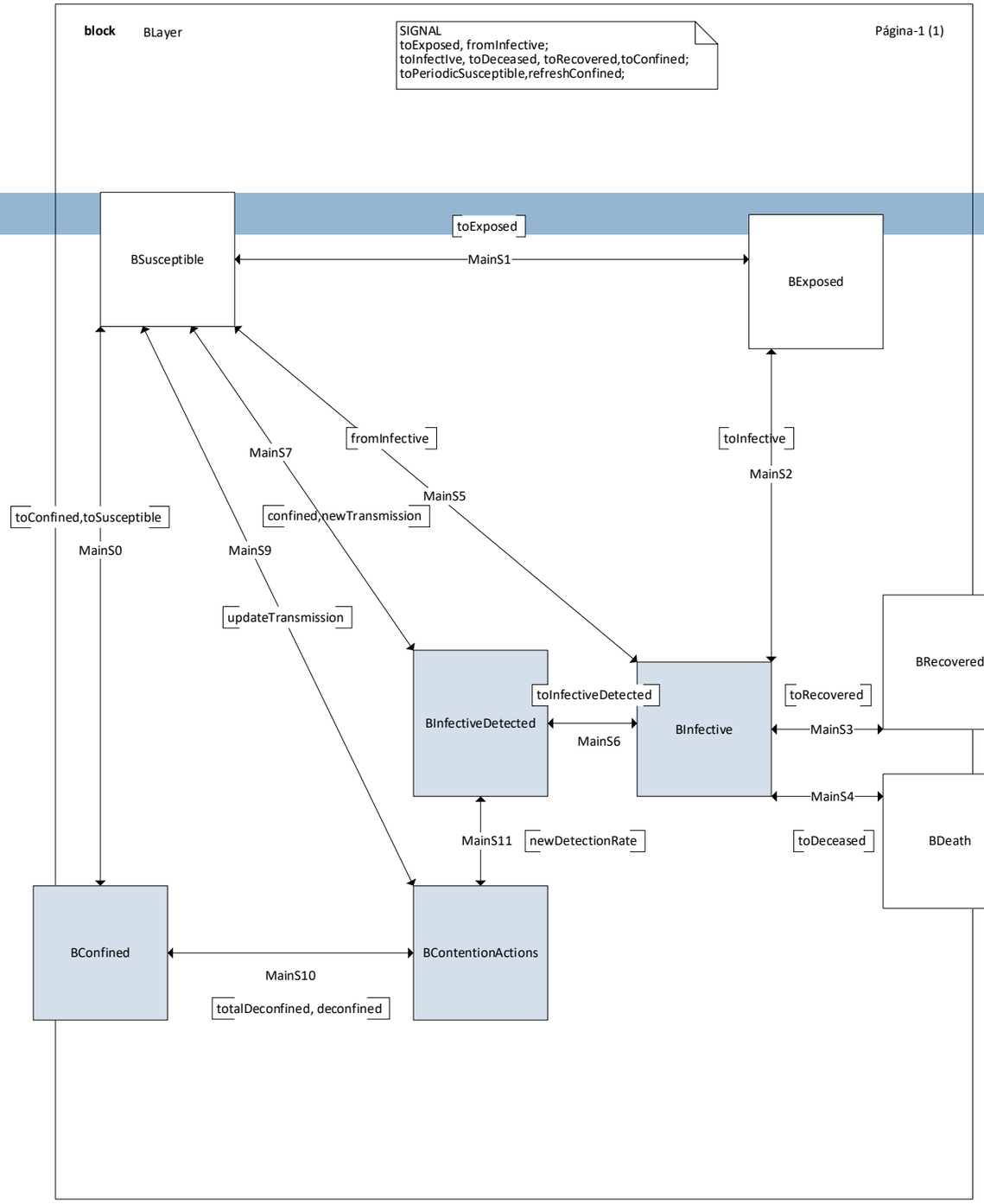


block MNCA_Pandemia inherits VT_P_mnca

BLayer

DCL

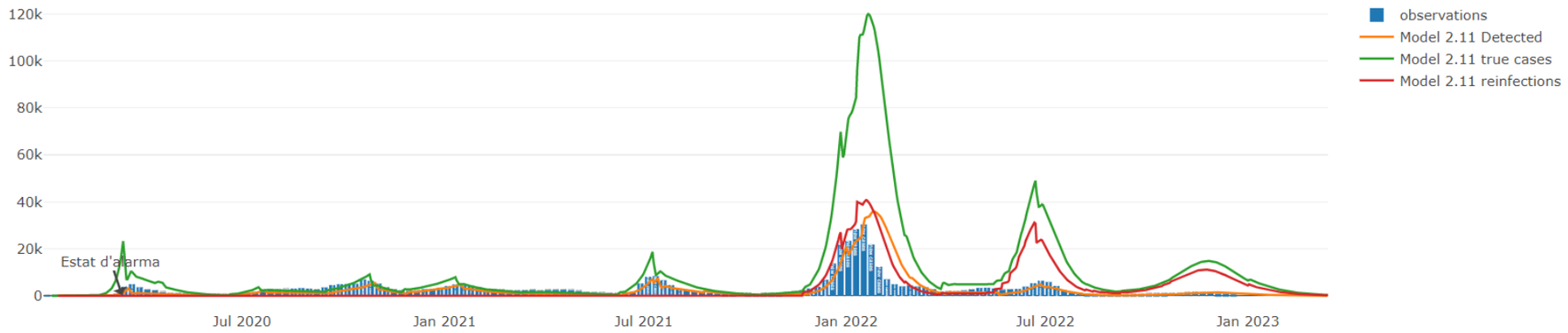
```
double layer_main='Initialize.img';
double layer_population='Population.img';
double layer_initial_infective='Initial_infective.img';
double layer_infective='Infective.img';
double layer_susceptible='Susceptible.img';
double layer_exposed='Exposed.img';
double layer_recovered='Recovered.img';
double layer_deceased='Deceased.img';
double layer_confined='Confined.img';
double layer_new_exposed='New_exposed.img';
double layer_new_infective='New_infective.img';
double layer_infective_detected='Infective_detected.img';
double layer_transmision='Transmision.img';
double layer_pdetected='pDetected.img';
double layer_confinement='Confinement.img';
double layer_new_confined_cases='New_confined_cases.img';
```


block BLayer
SIGNAL
toExposed, fromInfective;
toInfectIve, toDeceased, toRecovered,toConfined;
toPeriodicSusceptible,refreshConfined;
Página-1 (1)
BSusceptible
toExposed
MainS1
BExposed
fromInfective
MainS7
toInfective
MainS2
MainS5
toConfined,toSusceptible
MainS0
confined,newTransmission
MainS9
updateTransmission
BRecovered
BInfectiveDetected
toInfectiveDetected
MainS6
BInfective
toRecovered
MainS3
MainS4
MainS11
newDetectionRate
toDeceased
BDeath
BConfined
BContentionActions
MainS10
totalDeconfined, deconfined

SDL-PAND KPIs Model actual

New cases | R value (Digital Shadow) | Accumulated curve (Digital Shadow)

The simulation model allows to have a forecast of the evolution of the pandemic. It is corrected based on the actual data of the evolution of the pandemic. The current model is model 2.10. The R is calculated with the EpiEstim package of the statistical language "R".



KPI details	Effective growth. Digital Shadow	Incidence. Digital Shadow.	Effective growth. Digital Twin	Incidence. Digital Twin.
C3: 3 days moving average; C7: 7 days moving average; C14: 14 days moving average. C7 and C14 are more stable than C3, picking up trend variations in full week. Especially C3 has bias, an increase (very unstable) may represent a quick warning of possible worsening. Calculated for 100,000 inhabitants.	1.6 C3 (0–5) 1.9 C7 (0–5) 0.4 C14 (0–5) Detected cases.	4.6 I3 (0–100) 15.5 I7 (0–100) 13.3 I14 (0–100) Detected cases.	1 C3 (0–5) 0.9 C7 (0–5) 1 C14 (0–5) True cases.	531.3 I3 (0–100) 1.3K I7 (0–100) 2.6K I14 (0–100) True cases.

The wildfire model

Modeling the propagation of a wildfire.

Wildfire model.

- Motivations:
  - Dangerous environment.
  - Difficult to experiment.
  - Simulations involves naturals resources and personnel.

- To develop an experimental framework to simulate a wildfire
  - Propagation.
  - Extintion.

- Working with:
  - CREAF data.
  - Catalonia Fire Fighting Department

Wildfire propagation (over R)

- Implemented using SDLPS.
- BEHAVE model.
- Raster data describing the landscape.

GIS Data

- Input data files:
  - **Mapa**: file containing the DEM (Digital Elevation Model).
  - **Model**: file that represents the propagation model implemented for each cell.
  - **Slope, Aspect**: files that stores the slope and his direction. These files are calculated using the DEM. (Mapa files)
  - **M1, M10, M100, Mherb, Mwood**: files that contains the combustible description.
- The results files are two files:
  - **ignMap.dtm**: Stores ignition time.
  - **flMap.dtm**: Stores flames elevation.

GIS data: IDRISI32.

- 1987, Research program of Clark's University.

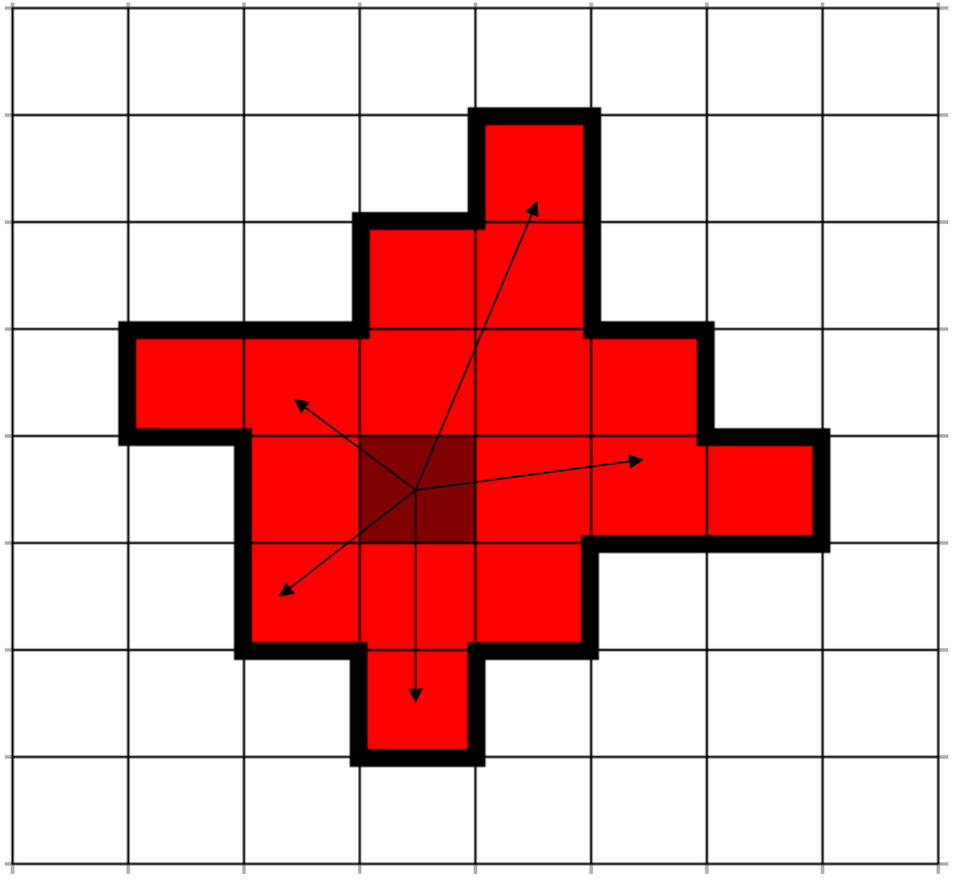
- We use the IDRISI32 file format.
  - One file for the data.
  - Other for the information related to the data.

20	30	10	90	178	12	89	12
34	23	56	12	342	56	34	38
12	12	34	15	76	87	12	32
...							

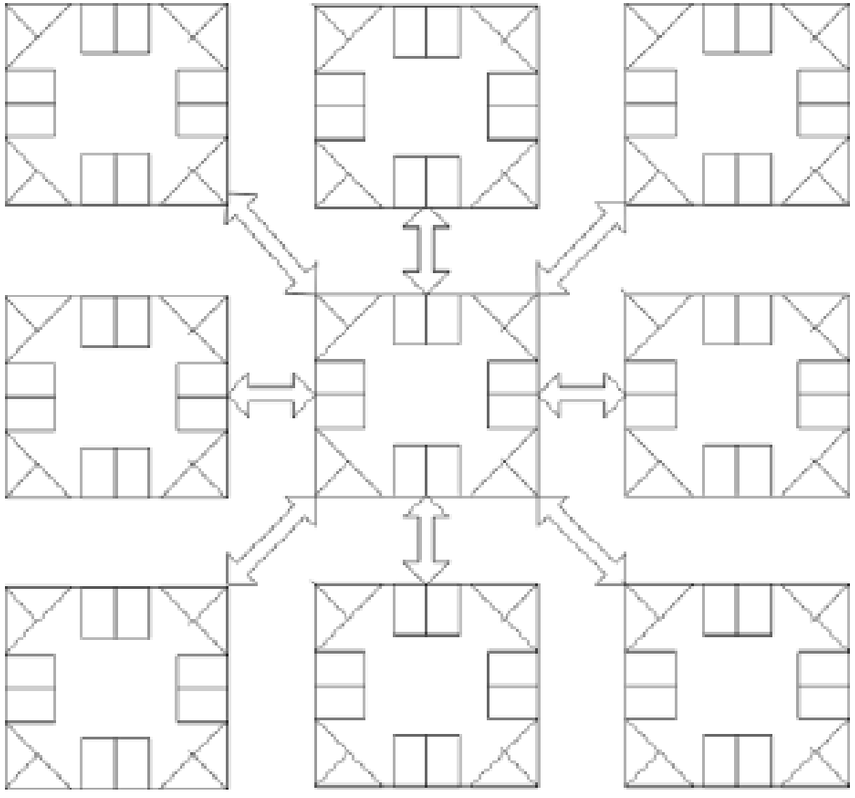
Propagation model events

- The events that lead propagation model are:
- **EBurn:** Associate to ignite fire into simulation cell.
- **EPropagation:** Programmed time for fire propagation to neighbor cell.
- **EExtinguish:** Programmed time to put out fire in a cell.
- **dataUpdate:** Event that represent a modification in the data used to calculate spread time. When this event is received is necessary to recalculate propagation model, (for instance a modification of the wind speed or direction).

BEHAVE model



Moore neighbourhood

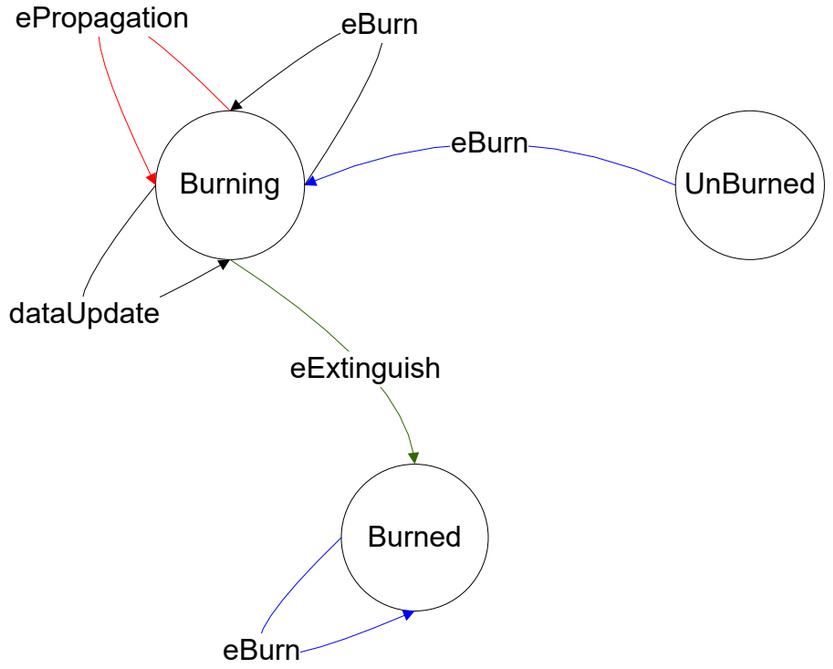


BEHAVE model

- The BEHAVE library, Andrews 1996.
- Based in a cellular automaton and a discrete simulation model.
- From a set of raster layers and an initial point the model calculates the ignition time and the elevation of the flames on each cell.
- In our model:
  - The fire starts in a know cell.
  - The results are calculated to the neighborhood cells.
  - Analyze what is the cell with the lowest ignition time.
  - Recalculate the results for this cell.
  - This loop is repeated while exist cells in the model.

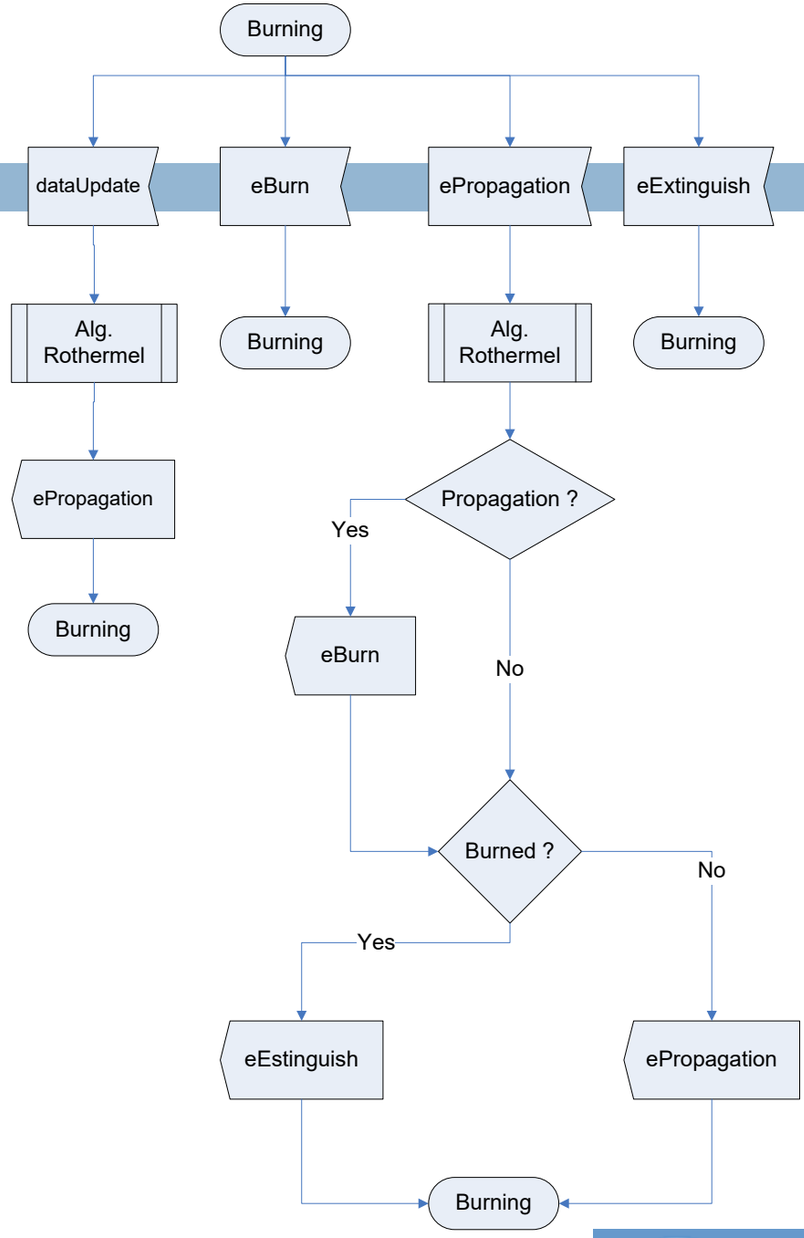
States diagram

- **EBurn**: Associate to ignite fire into simulation cell.
- **EPropagation**: Programmed time for fire propagation to neighbor cell.
- **EExtinguish**: Programmed time to put out fire in a cell.
- **dataUpdate**: Event that represent a modification in the data used to calculate spread time. When this event is received is necessary to recalculate propagation model, (for instance a modification of the wind speed or direction).

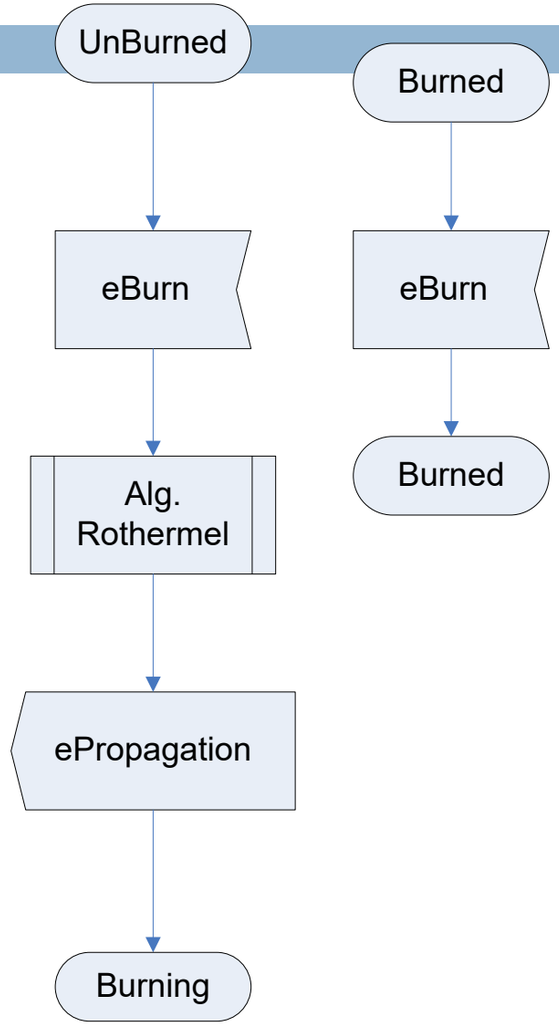


Process diagram

- The third level of the SDL formalism.

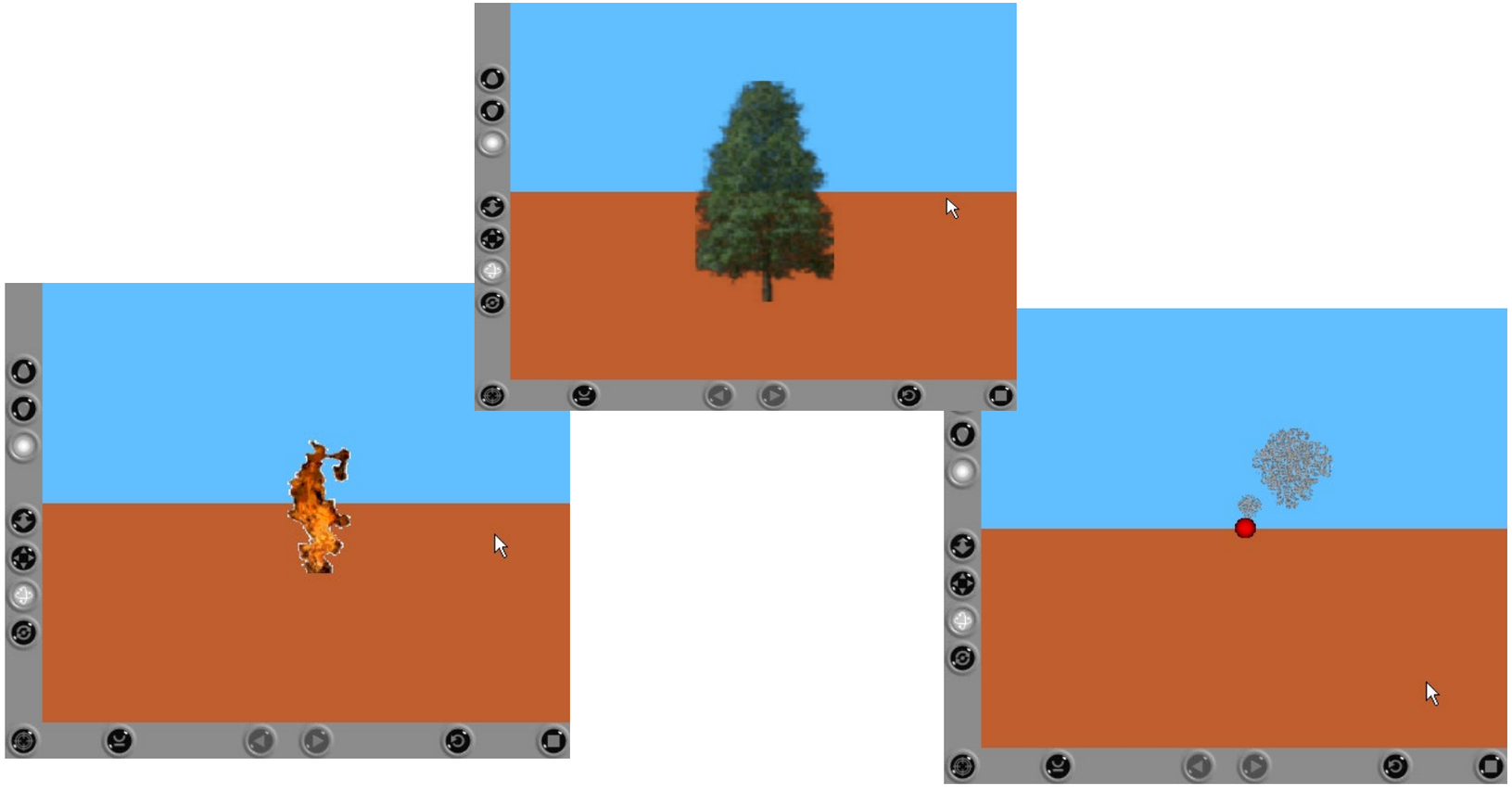


Process diagram



Simulation model

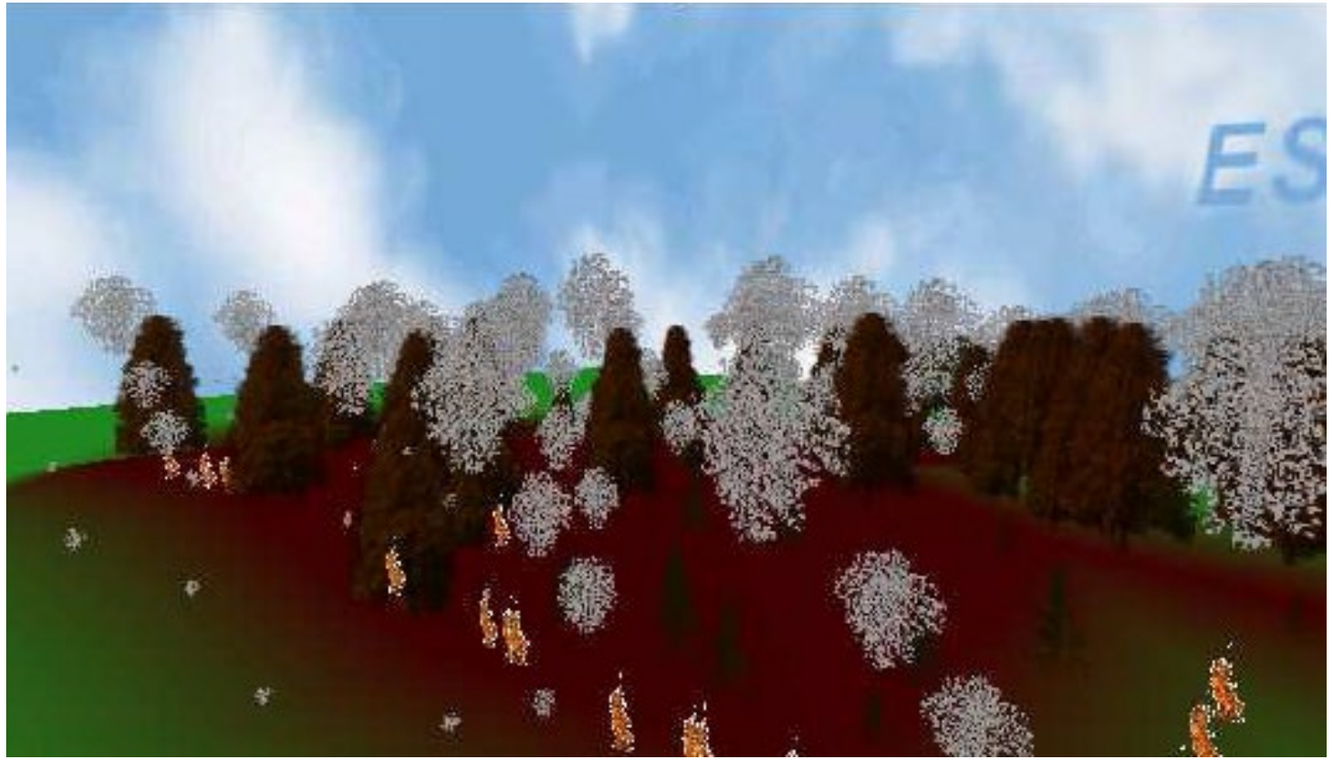
Visual efects: objects



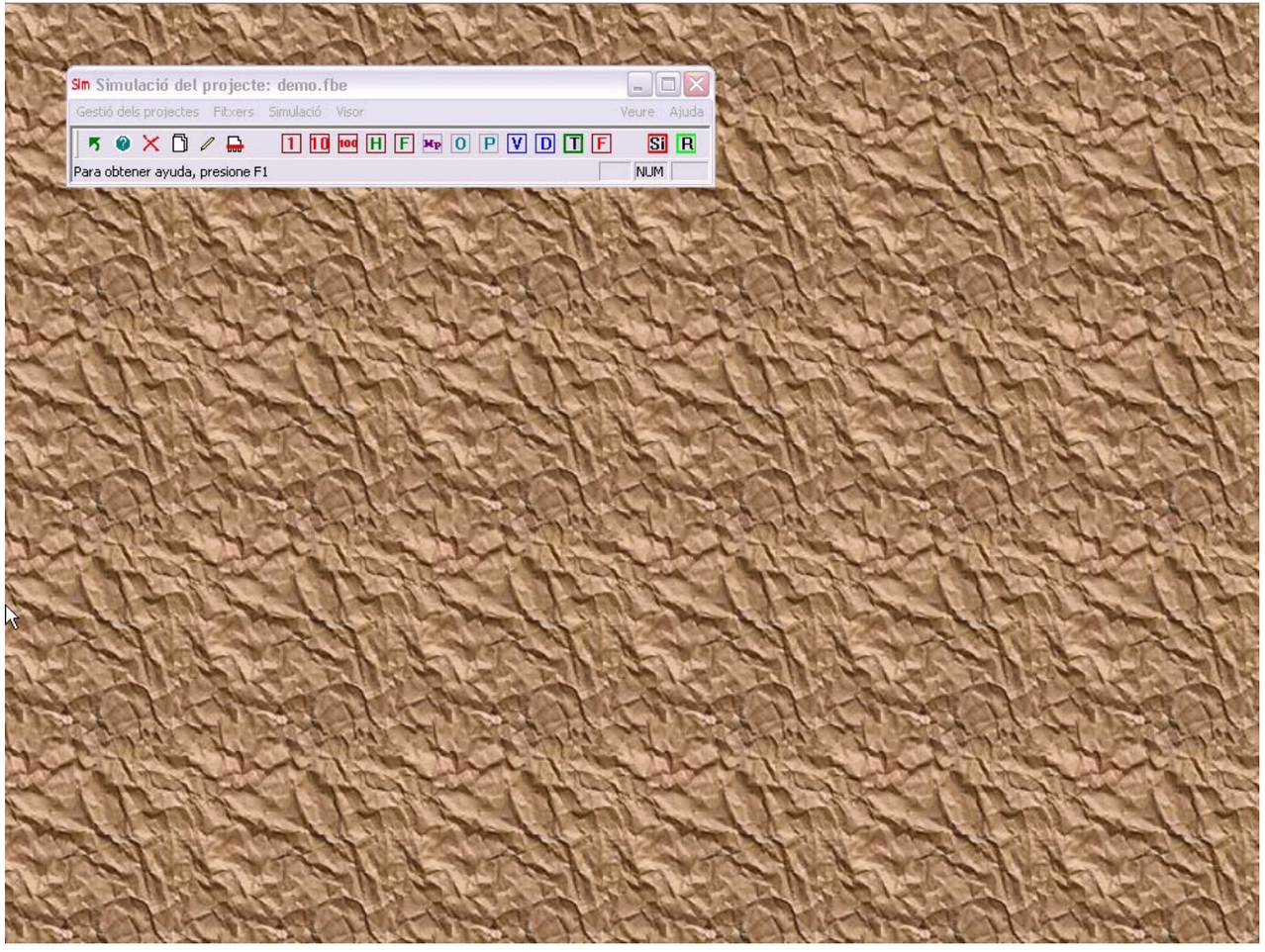
Visual efects: example

Visual efects: example

- Wildfire in action

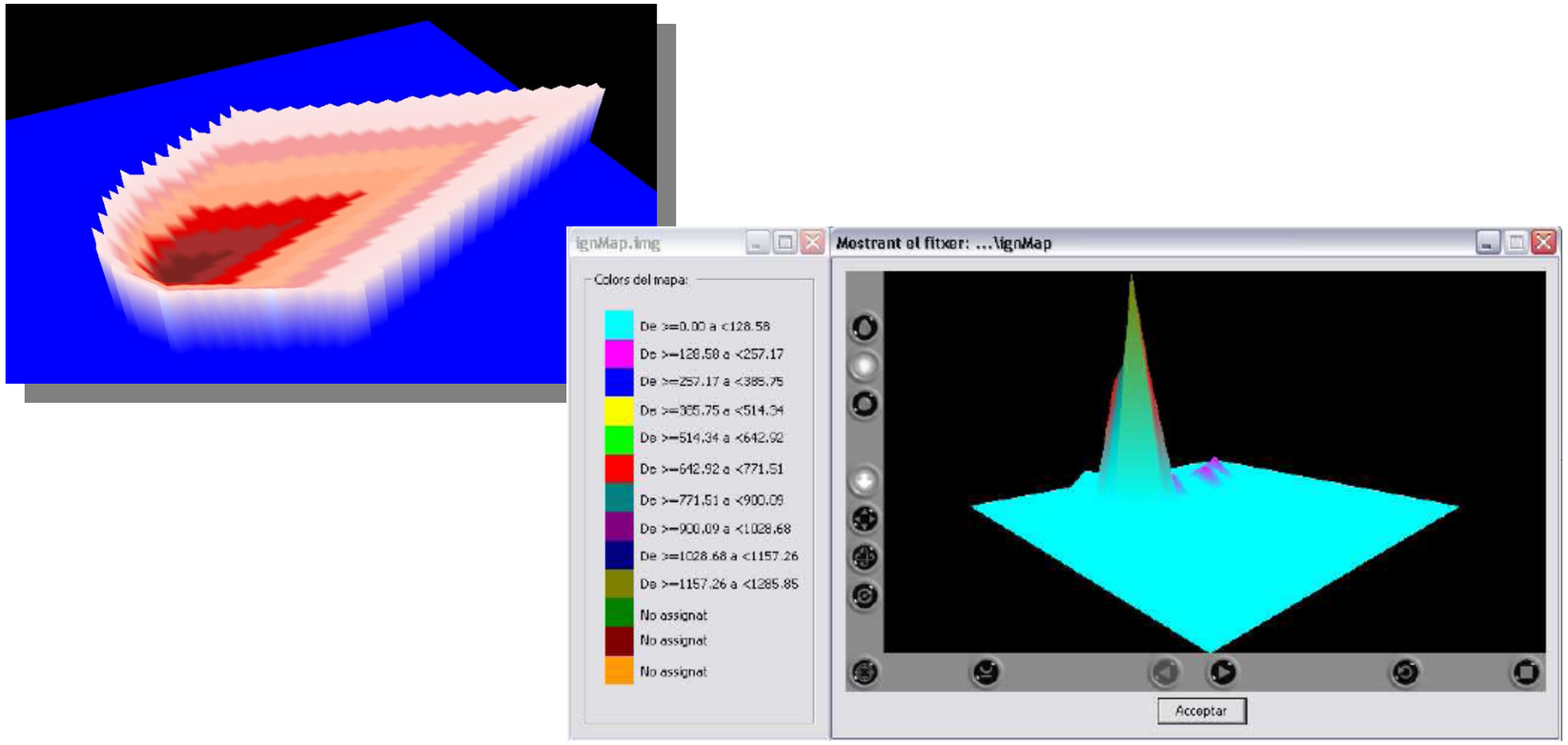


Wildfire simulation

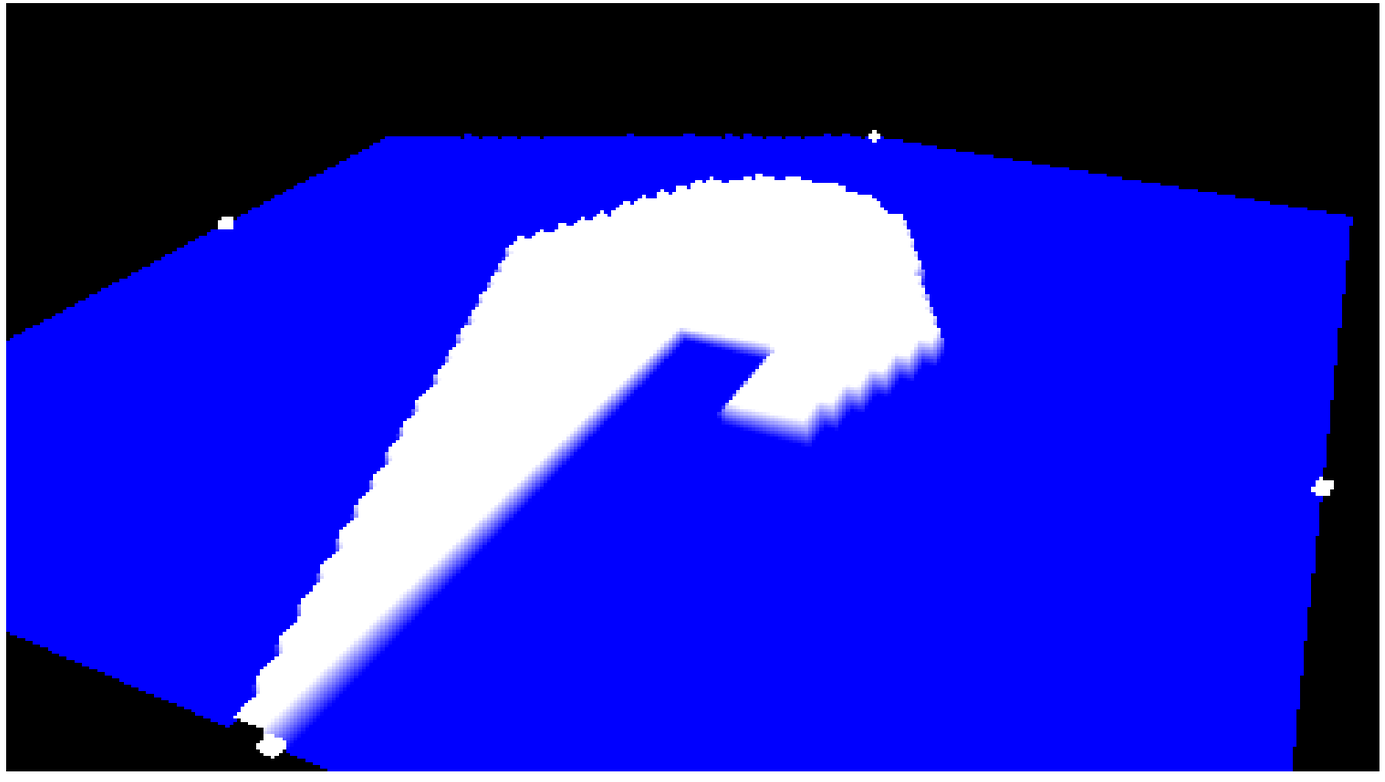


Results

- Height and temperature of the flames.



flMap.dtm (with IAgents)



Slap avalanche model

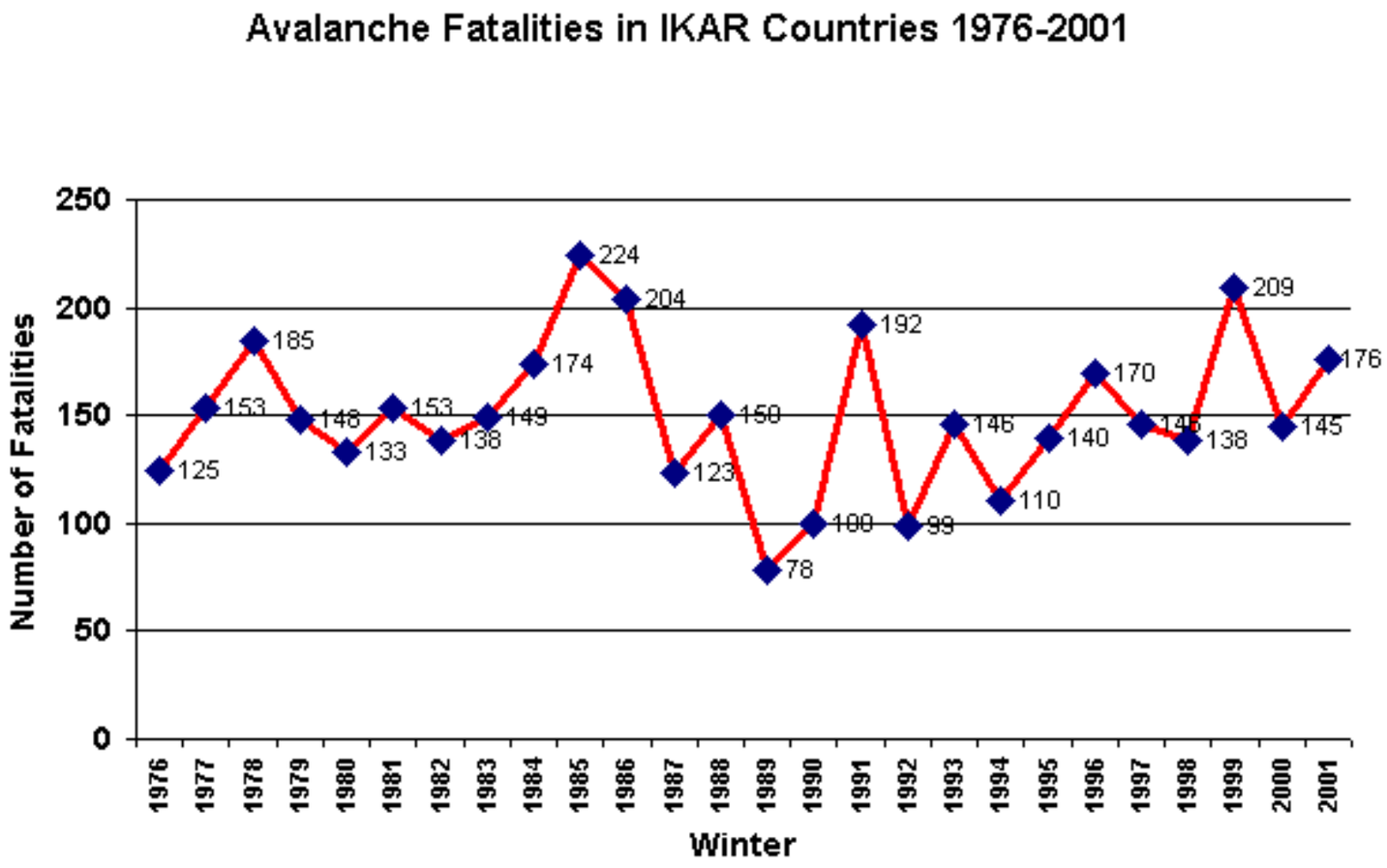
Slap avalanche model

Avalanche

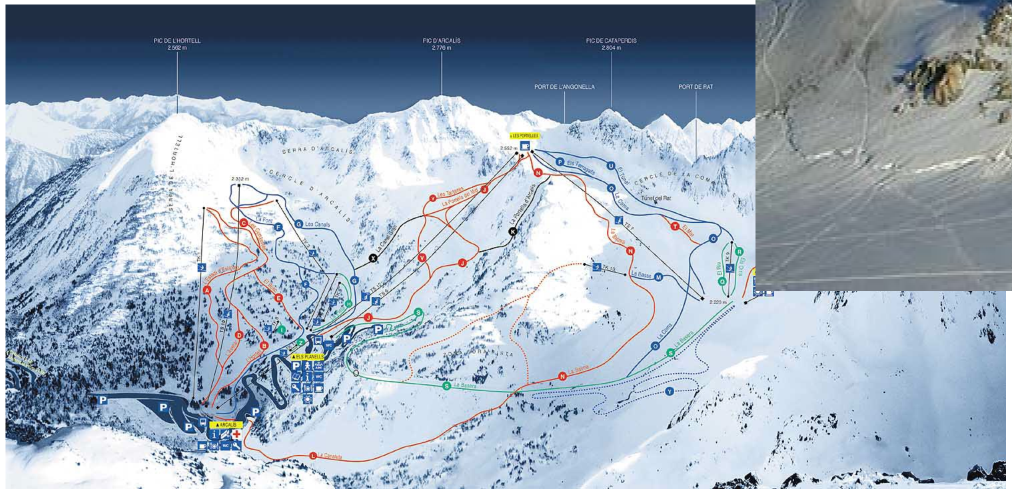
Two main types of **snow** avalanche:

- **Loose-snow** avalanche originates at a point and propagates downhill by successively dislodging increasing numbers of poorly cohering snow grains, typically gaining width as movement continues down slope.
- **Slab avalanche,** occurs when a distinct cohesive snow layer breaks away as a unit and slides because it is poorly anchored to the snow or ground below

Avalanche fatalities in IKAR Countries



Some photos



Avalanche Model data

Name	Type	Description	Qtt	Source	Modifiable
Height	Raster	Layer representing the height of the environment.	1	ICC	No
Thickness of the snow	Raster	Represents the thickness of the “slab snow”	1	Meteocat	Yes
Floor features	Raster	Represents the kind of surface (rocks, sand, snow, ice,..). Each surface has his own specific rough parameter.	1	Meteocat Creaf	No
Snow that causes the slab features	Raster	Density, compactness of the snow.	1	Meteocat	Yes*
Obstacles	Raster	Represents the obstacles that have the environment (small rocks, big rocks, houses, trees,…)	N	Creaf	Yes
Crack	Vectorial	Line representing the breakdown of the ice.	1	Input data	Yes, at beginning.
State of the snow	Raster	Shows the state of the terrain, **empty**, **static** and **dynamic**	1	Meteocat	Yes

Avalanche Model

- $6+N:2\text{-}AC^{4+N}$ on Z^2

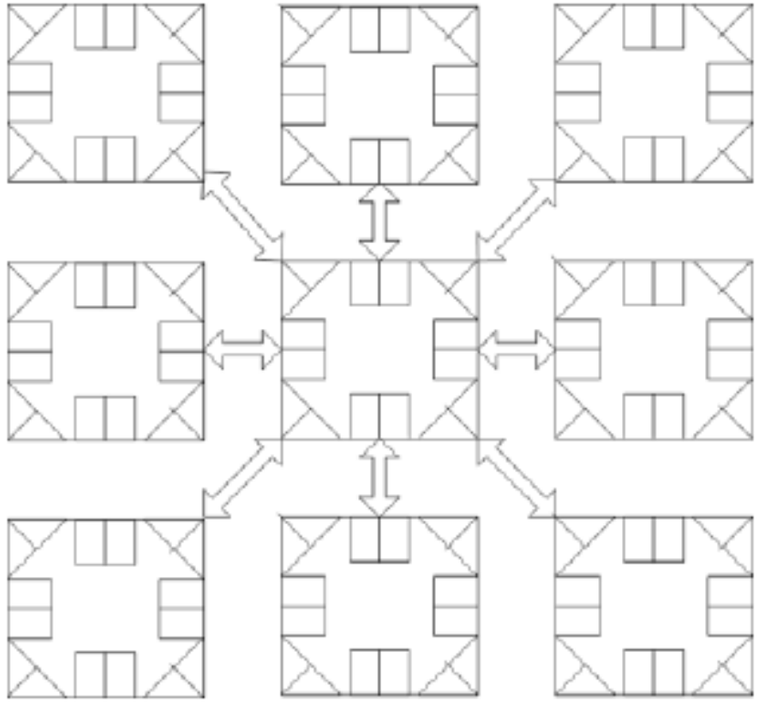
Vicinity and nucleus function

- Vicinity function: vn(x1,x1) = {(x1-1,x2-1), (x1-1,x2), (x1-1,x2+1), (x1,x2-1), (x1,x2), (x1,x2+1), (x1+1,x2-1), (x1+1,x2), (x1+1,x2+1)}
- Nucleus function: nc(x1,x1)= {(x1,x1)}

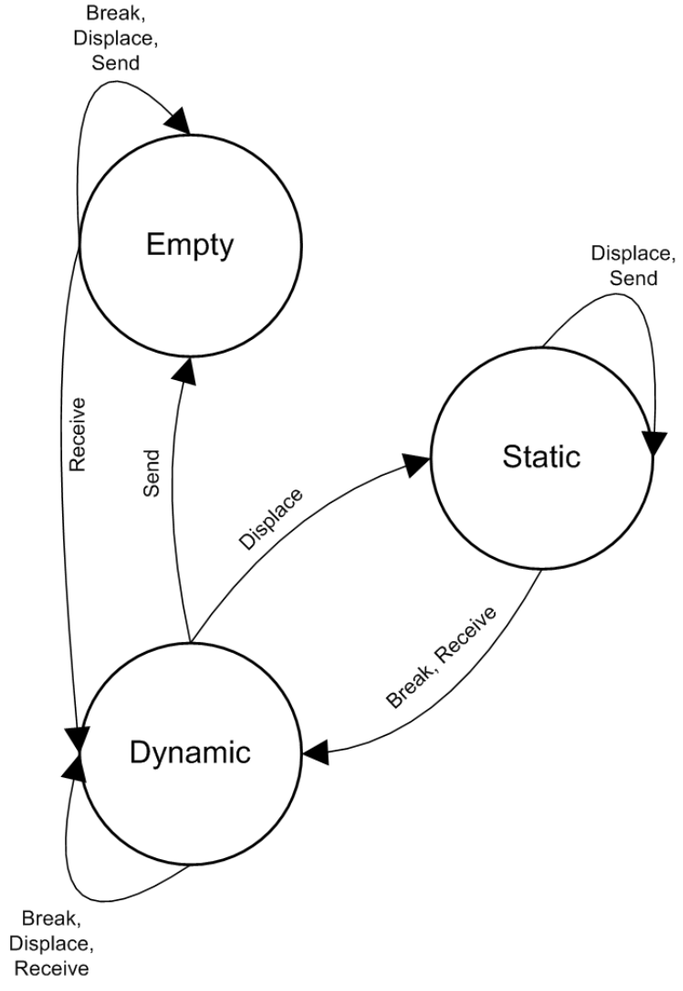
Evolution functions

- E_2[i]: Thickness of the snow. The function that rules this layer is “Modify information(p)”
- E_4[i]: Density, compactness of the snow, in our case is 0.5 (Mears 1976).
- E_6[i]: State of the snow. The function is defined in the next diagrams.
- E_N[i]: Obstacles. The function that defines the obstacles we use in the model.

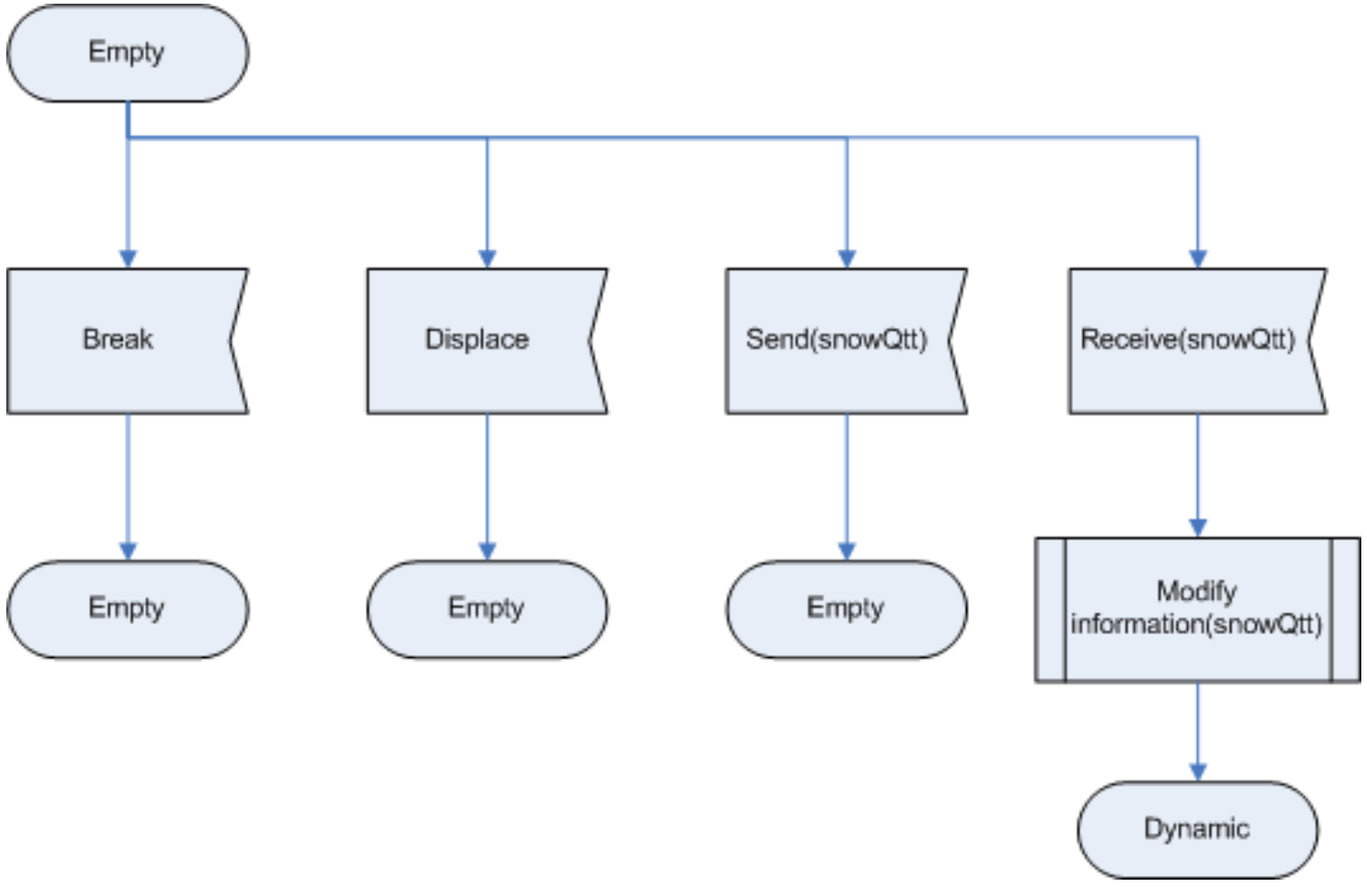
Moore neighbourhood



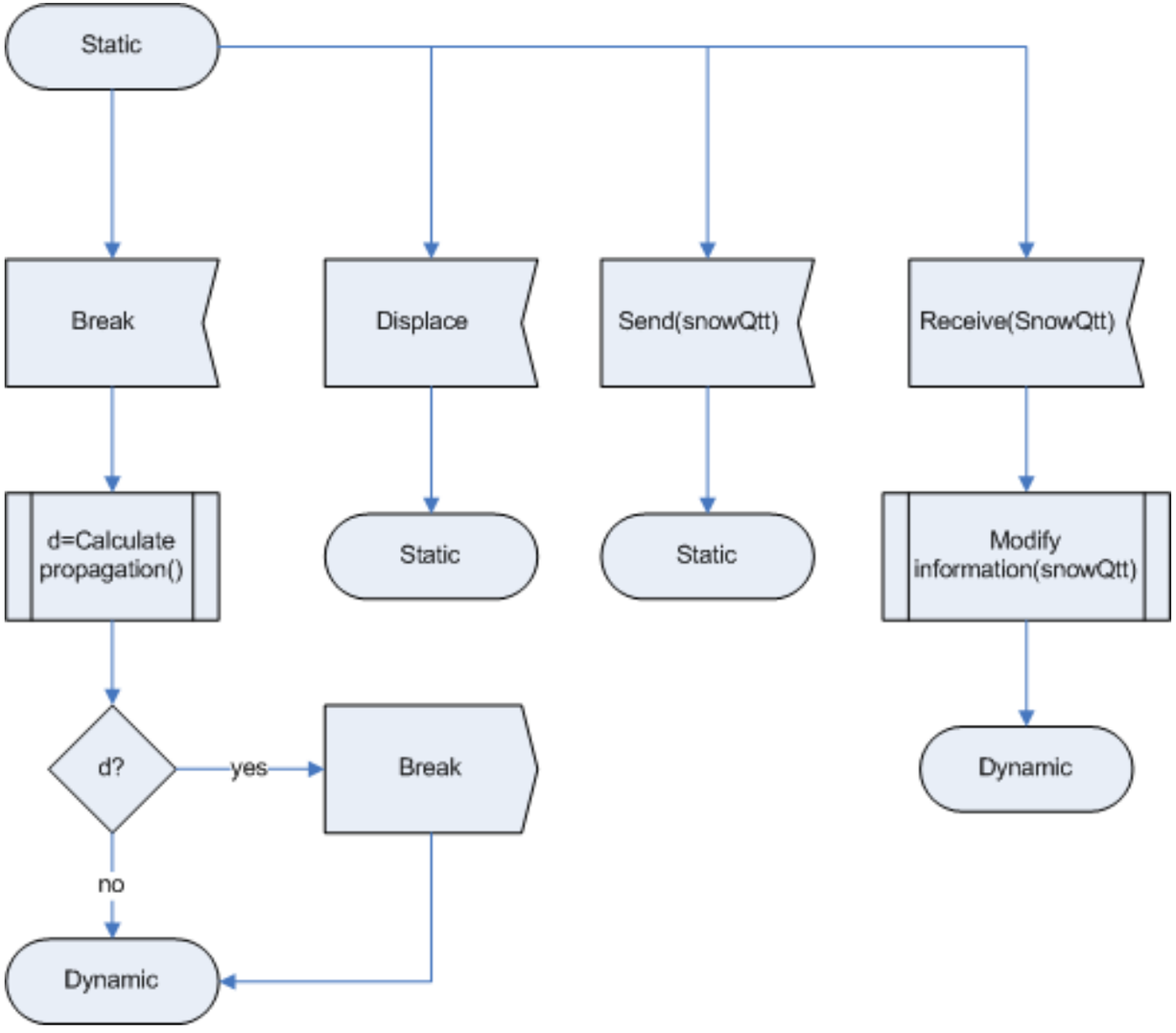
Λ:state of the snow



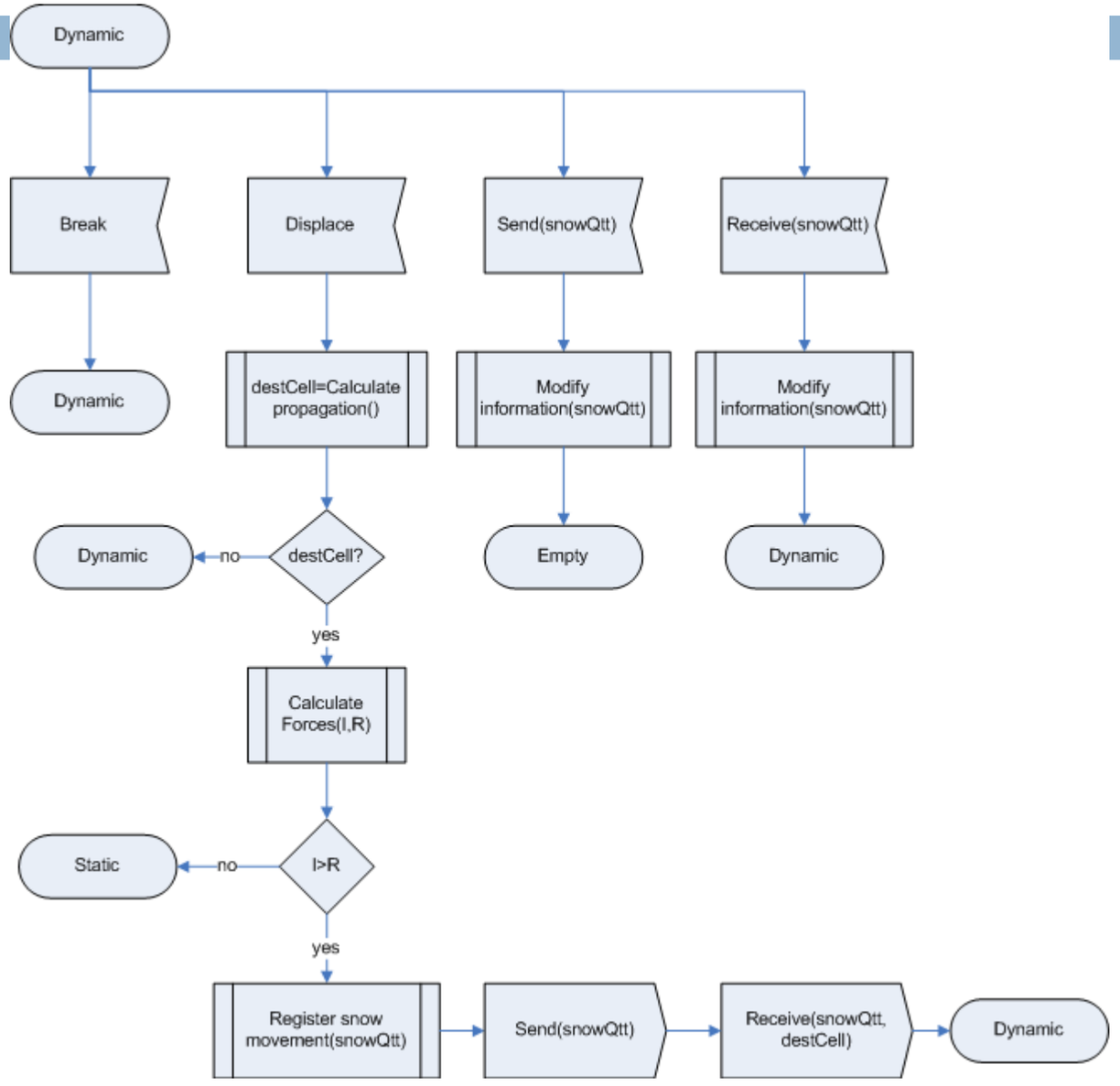
Empty process



Static process



Dynamic process



Evolution function

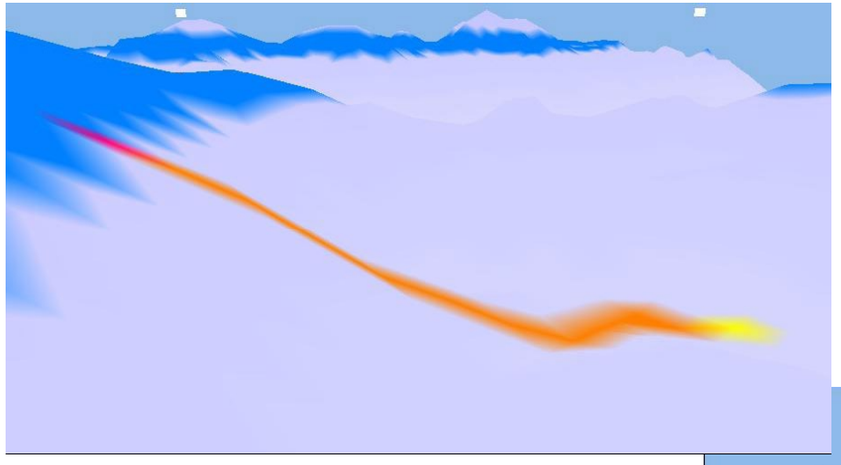
- The increment in the force is used in the next expression to determine if the snow continues its movement to other cell, or stops its movement, if the force is equal to zero.
- $F_{i,t} = \max(IF_{i,t} + \Delta F_{i,t}, 0)$

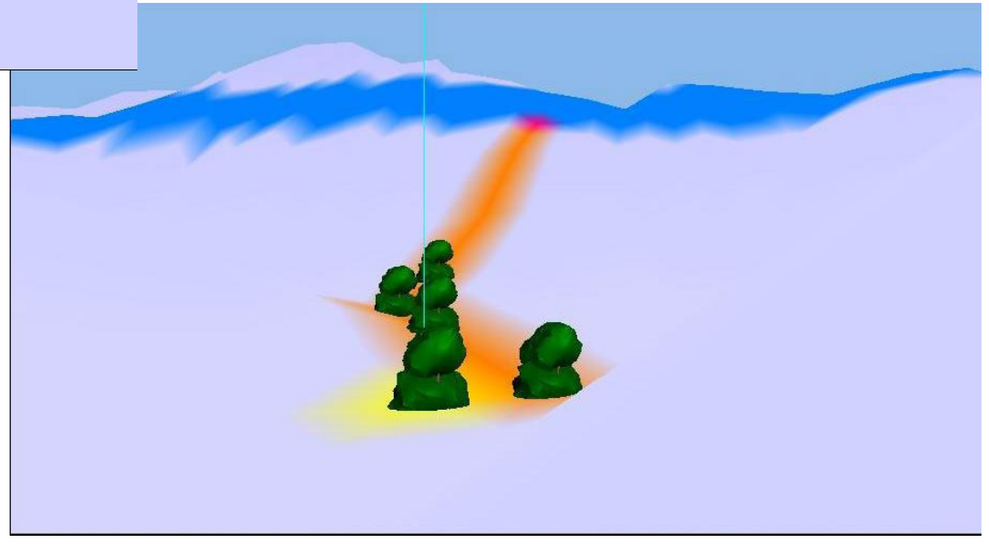
Evolution function

- $IF_{i,t}$=Impulse force, depends on the quantity and quality of the snow, and the slope.
- $SFF_{i,t}$= Sliding friction force between the avalanche and the underlying snow or ground.
- $IFF_{i,t}$= Internal dynamic shear resistance due to collisions and momentum exchange between particles and blocks of snow, (internal friction force).
- $ASFF_{i,t}$= Turbulent friction within the snow/air suspension, (air suspension friction force).
- $AFF_{i,t}$=Shear between the avalanche and the surrounding air, (air friction force).
- $FFF_{i,t}$= Fluid-dynamic drag at the front of the avalanche (front friction force).
- $OFF_{i,t}$=Obstacle friction force.
  - $\Delta F_{i,t} = IF_{i,t} - (SFF_{i,t} + IFF_{i,t} + ASFF_{i,t} + AFF_{i,t} + FFF_{i,t} + OFF_{i,t})$

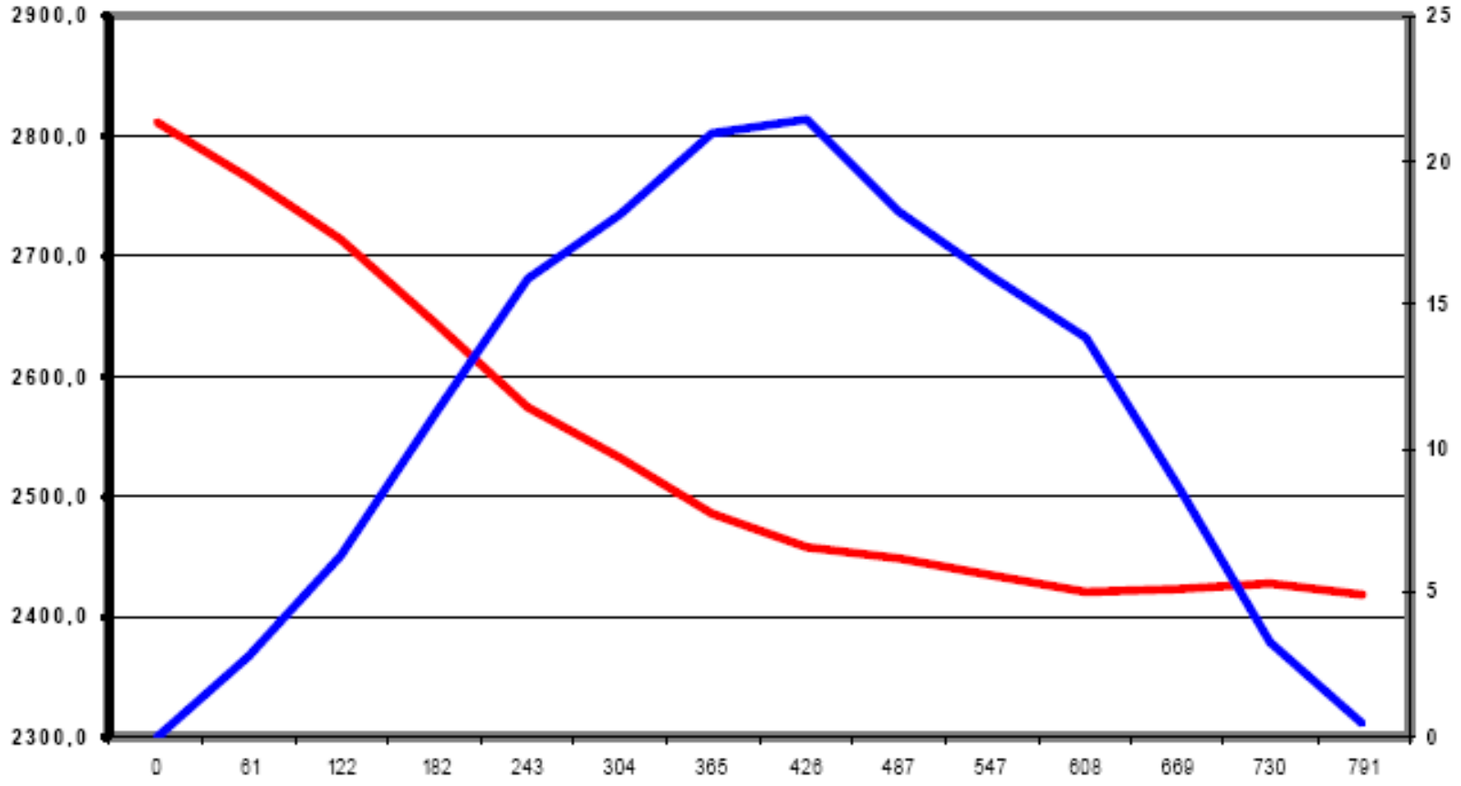
Slap avalanche

Results





Results


2900,0
2800,0
2700,0
2600,0
2500,0
2400,0
2300,0
25
20
15
10
5
0
0
61
122
182
243
304
365
426
487
547
608
669
730
791

Results (5)

Desencadenant:

Localització: 8 cel·les, de (108, 33) fins (115, 33)

Gruix de neu de placa: 50cm (per totes les cel·les fracturades)

Terreny subjacent: Neu dura

Obstacles: No

Característiques de l'allau:

Terreny subjacent del camí: Neu dura

Màxima distancia recorreguda: 1101,14m

Desnivell superat: 520,40m

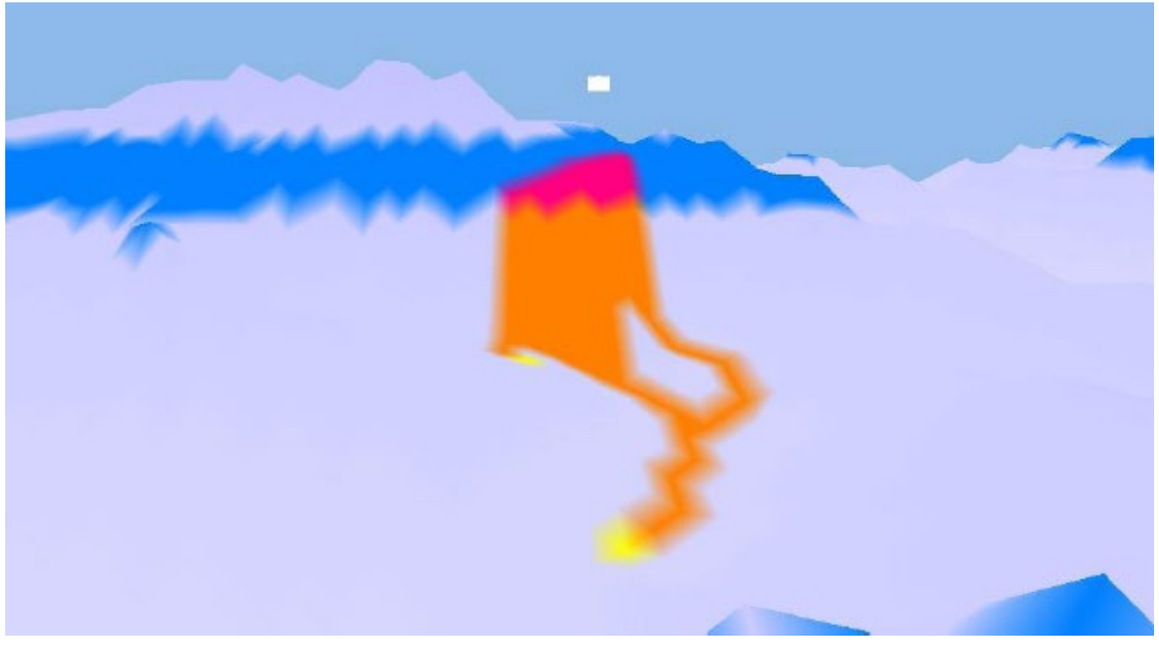
Massa transportada: 10625kg

Massa de neu en dipòsit: 9957,50kg

Massa de neu perduda pel camí: 667,5kg

Velocitat màxima: 67,23m/s

Results (5)



Results (5)

Results (1 vs 3)

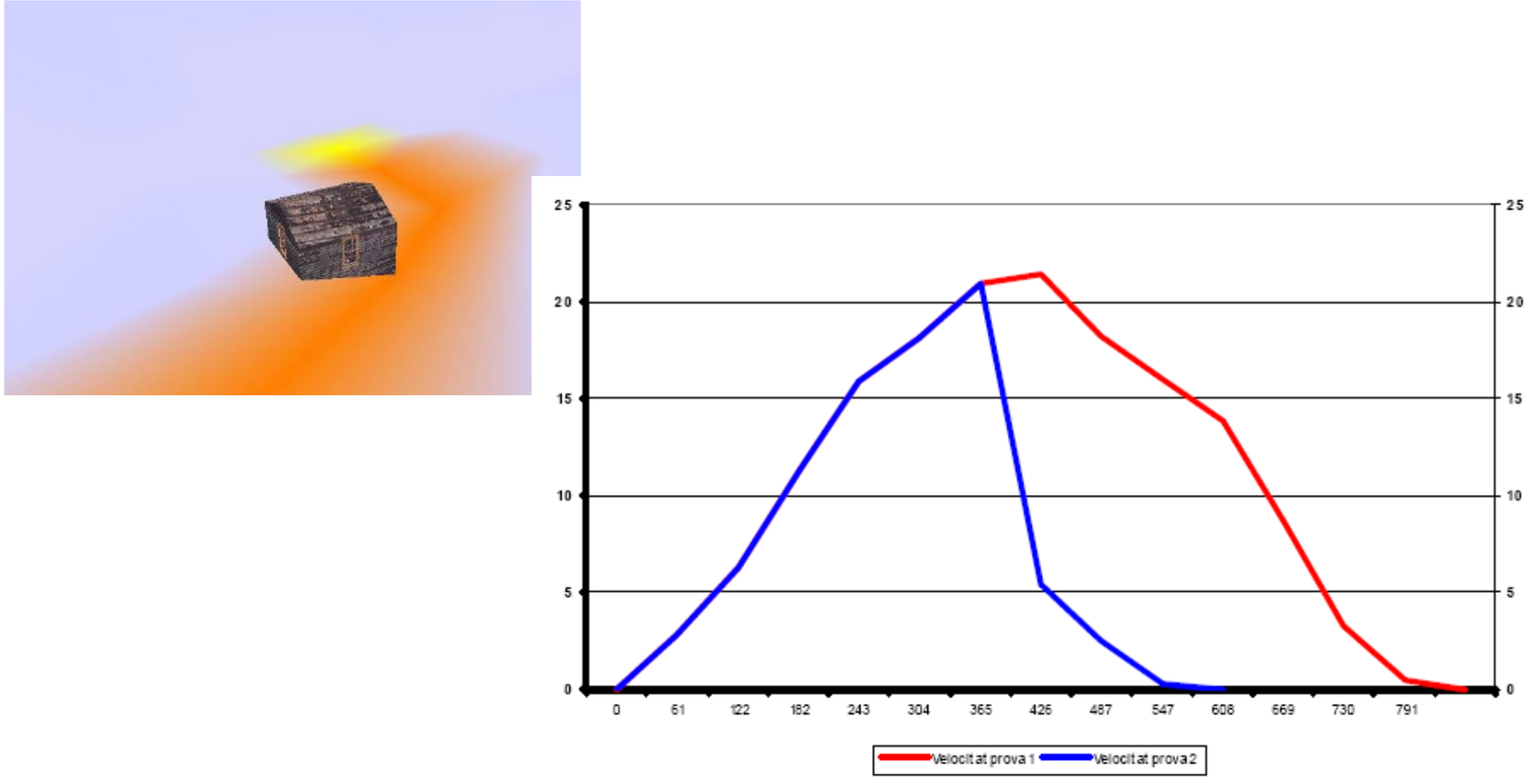
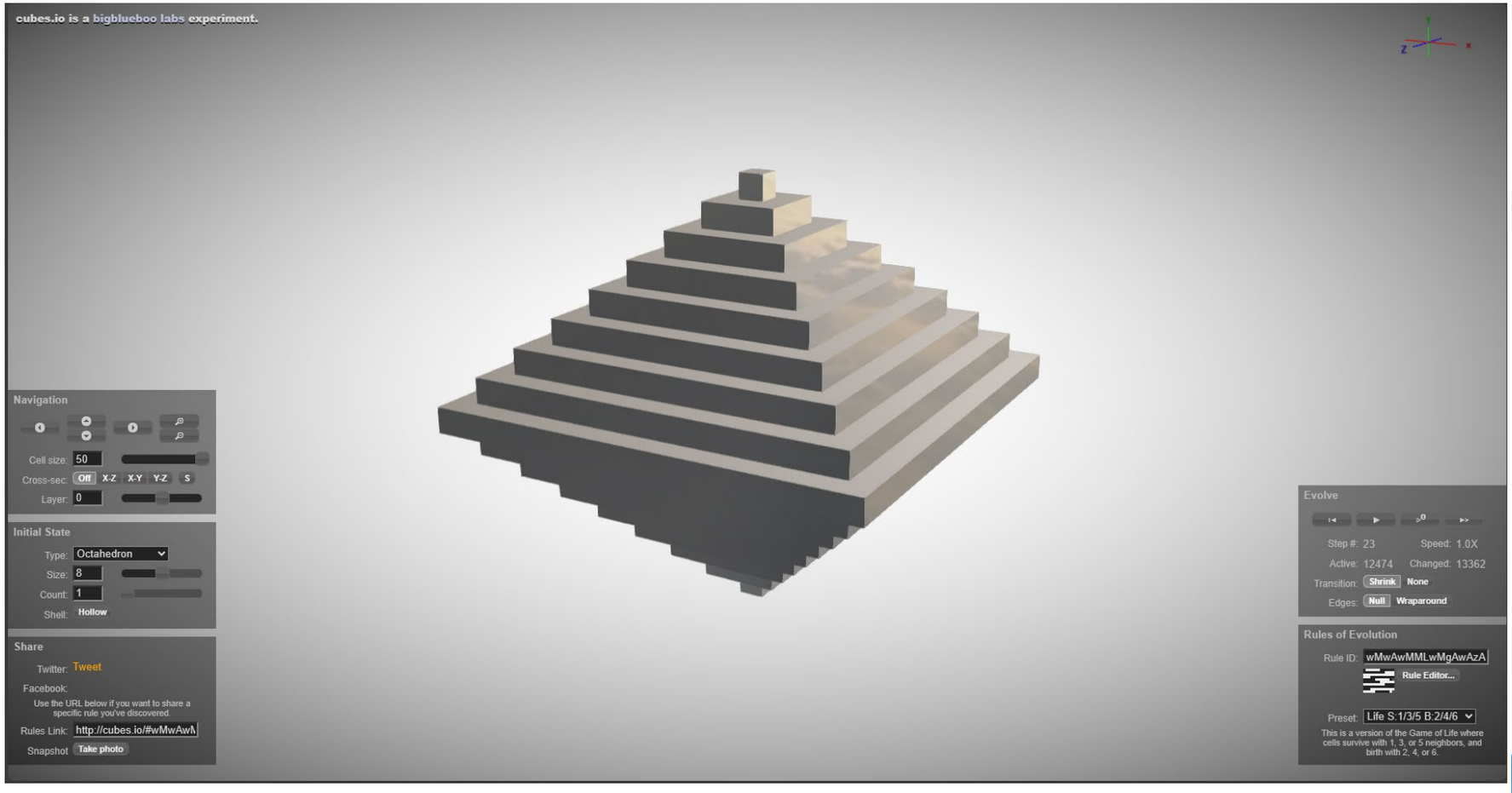


Figura 7.9 – Velocitat Sim.1 VS Velocitat Sim.3

Some tools

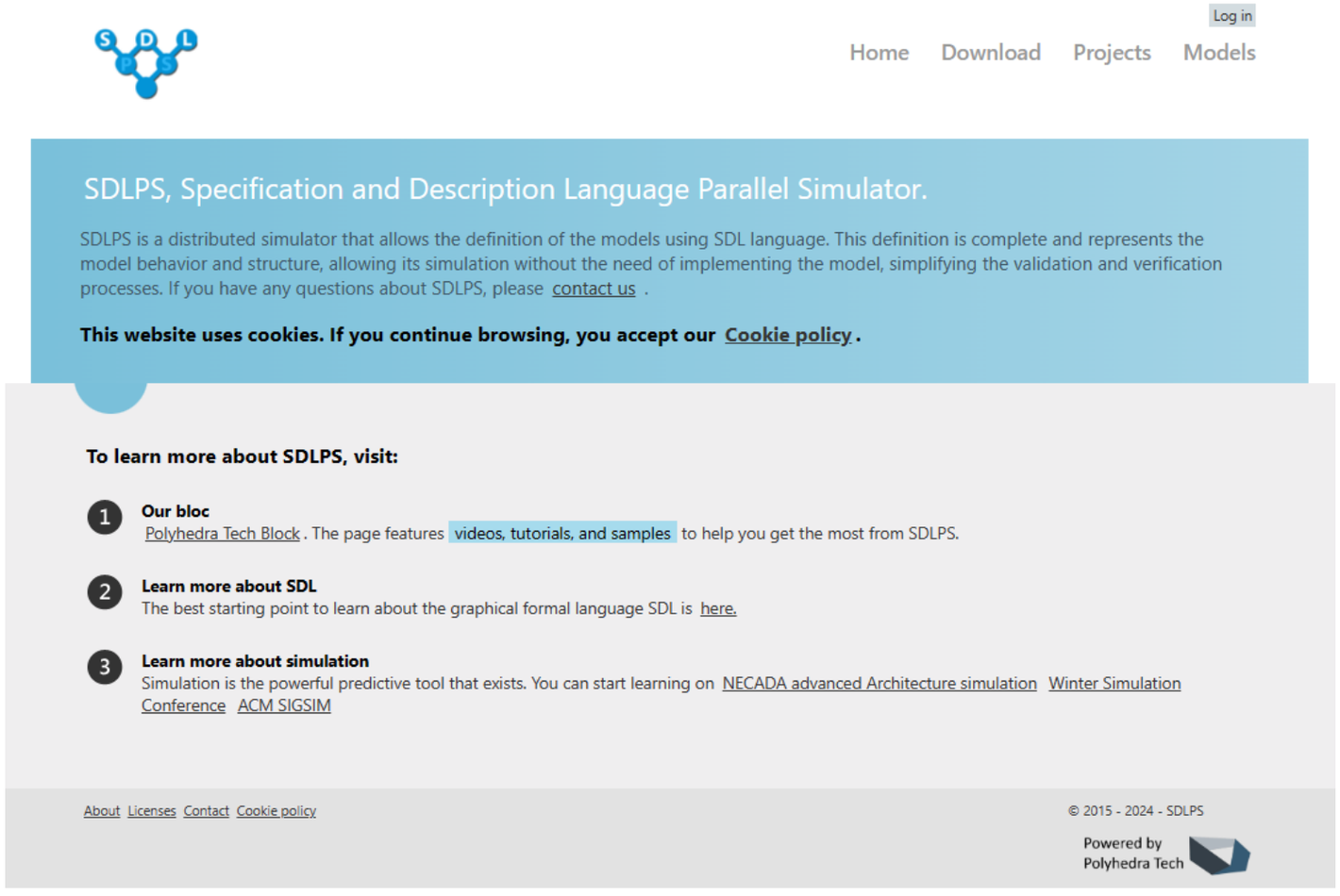
- cubes.io: 3d cellular automata



New!

Some tools

- SDLPS



More tools

- Play John Conway's Game of Life (playgameoflife.com)
- Cellular Automata Sim by AlexMulkerrin (itch.io)
- Cellarium by Benjamin Mastripolito (itch.io)
- Cellular Automaton Explorer - Wolfram Demonstrations Project
- Automata Ecosystem - Cellular Automata Simulation en Steam (steampowered.com)

Some references

- Doran, Jim. E. 2000, *HARD PROBLEMS IN THE USE OF AGENT-BASED MODELLING,* Proceedings of the Fifth International Conference on Logic and Methodology, Cologne, October 3-6
- Roher, G. 2000, *Can Simulation models Support Social Research? A Critical Discussion.*
- Itzhak Benenson, Itzhak Omer. 2001, *Agent-Based modeling of residential Distribution.* Departament of Geography and human Environment University Tel Aviv. Israel.
- Epstein, Joshua M. Axtell, Robert. 1996 *Growing Artificial Societies: Social Science from the Bottom Up.* MIT Press, Cambridge, MA, USA.

Some references

- Batty, Michael, 2005, *Cities and complexity*.The MIT Press.
- Torrens, Paul M. Benenson, Itzhak. 2004, Geosimulation, John Wiley & Sons Ltd., ISBN: 0-470-84349-7
- Wuensche, Andrew; Lesser, Mike. 1992, *The gobal dynamics of cellular automata*, Santa Fe institute, studies in the sciences of complexity, reference volume I, Addison Wesley, 1992
- G. 't Hooft, *The Cellular Automaton Interpretation of Quantum Mechanics*, **2015**, (2014).

Some references

- Fonseca i Casas, Pau; Casanovas, Josep; Montero, Jordi. 2004c. *Cellular automata and intelligent agents used to model natural disasters with discrete simulation*. Proceedings of EMS 2004.
- Fonseca P.; Casanovas J. 2005. ESS205, *Simplifying GIS data use inside discrete event simulation model through m:n-AC cellular automaton*; Proceedings ESS 2005.
- Fonseca i Casas P., Rodríguez Fontoba,S., 2007, Using GIS data in a m:n-ACk cellular automaton to perform an avalanche simulation. Geographical Information Science Research UK Conference 2007. National University of Ireland Maynooth. http://ncg.nuim.ie/gisruk/materials/proceedings/
- P. Fonseca i Casas, M. Colls, and J. Casanovas, A Novel Model to Predict a Slab Avalanche Configuration Using M : N-CA k Cellular Automata, Comput. Environ. Urban Syst. 35, 12 (2011).

More references

- P. Fonseca i Casas, Towards a Representation of Cellular Automaton Using Specification and Description Language, in System Analysis and Modeling. Languages, Methods, and Tools for Industry 4.0, edited by P. Fonseca i Casas, M.-R. Sancho, and E. Sherratt (Springer, 2019), pp. 163–179.
- P. Fonseca i Casas, J. Garcia i Subirana, V. Garcia i Carrasco, J. L. Silva de Barcellos, J. Roma, and X. Pi, SDL Cellular Automaton COVID-19 Conceptualization, in Proceedings of the 12th System Analysis and Modelling Conference (ACM, New York, NY, USA, 2020), pp. 144–153.