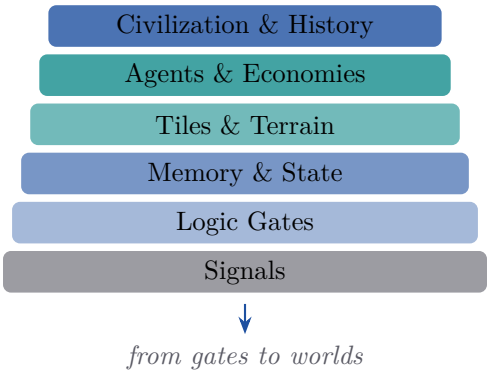
The Elements of Computational Worlds

Building a 4X Civilization Simulator from First Principles in SPHEREPOP

Flyxion

Independent Researcher



First Edition

This textbook was developed in conjunction with the SPHEREPOP computational framework, the RSVP field-theoretic programme, and the KES (Kinetic-Event Synthesis) formalism.

"A world is a layered state machine whose global history emerges from local logical permissions."

Preface

> *The elements of every science are at once its lowest and its highest truths: lowest because they are common to all, highest because they are the key to all.*
>
> — — paraphrase of Aristotle, *Prior Analytics*

This book is about building worlds from scratch. Not artistic worlds, not narrative worlds, but *computational* worlds: persistent, rule-governed systems of state that support exploration, expansion, exploitation, and conflict—the four pillars of the 4X strategy genre.

The organizing thesis is simple. A 4X world is a civilization-scale circuit. Everything that makes such a world interesting—territory, resources, memory, agency, diplomacy, history—emerges from a small set of computational primitives: signals, logic gates, latches, counters, and state machines.

This book follows the pedagogical architecture of Noam Nisan and Shimon Schocken's masterwork *The Elements of Computing Systems*. Each chapter introduces an abstraction, provides a formal specification, works through an implementation in the SPHEREPOP language, offers a philosophical perspective, and closes with a construction project. The ascent of the book mirrors the ascent of complexity:

Signals → Gates → Memory → Tiles → Agents → Economies → Diplomacy → History

What distinguishes this from a standard computing systems course is the endpoint. Nisan and Schocken build a general-purpose computer. This book builds a world. Every abstraction is motivated not by the needs of an operating system, but by the demands of a civilization that must remember, perceive, decide, and endure.

SPHEREPOP is our medium throughout. It treats computation as the interaction of *bubbles*: self-contained units that may carry values, execute rules, spawn children, consume neighbors, or pass signals along. Bubbles are simultaneously data, process, interface, and environment. This unification makes SPHEREPOP unusually well suited to world simulation: a tile is a bubble. An agent is a bubble. A diplomatic treaty is a bubble. The world itself is a bubble.

The book is divided into five parts:

I. **Foundations of Computation.** Signals, Boolean algebra, logic gates, and bubble interaction.

II. **State and Memory.** Latches, registers, counters, and the persistence of time.

III. **Spatial Worlds.** Tiles, terrain, visibility, and the geometry of territory.

IV. **Agents and Economies.** Perception, action, resources, trade, and conflict.

V. **Civilization and Emergence.** Technology, diplomacy, ecology, and the accumulation of history.

A student who completes all five parts and their associated projects will have constructed, from first principles, a functioning 4X world simulation. They will also have understood *why* that world works the way it does, because they will have built every layer themselves.

Flyxion
Canada

How to Use This Book

Each chapter is structured in five sections following the pedagogical pattern of Nisan and Schocken:

Background
: Motivates the chapter's central concept from first principles. Why does a world need this abstraction?

Specification
: Defines the abstraction precisely. What are its inputs, outputs, and behavioral contracts?

Implementation
: Constructs the abstraction in SPHEREPOP. How is it realized from the primitives introduced in prior chapters?

Perspective
: Reflects on what the abstraction means philosophically, ecologically, economically, or strategically.

Project
: A hands-on construction task. The student builds the abstraction, tests it, and integrates it into the growing simulation.

Projects are cumulative. Each project produces a component that later chapters will use. By the final chapter, the student has assembled a working civilization simulator from nothing but bubble interactions.

Contents

Preface iii

How to Use This Book v

I Foundations of Computation 1

1 Boolean Worlds 3
- 1.1 Signals . . . 3
- 1.2 Boolean Operations . . . 4
- 1.3 NAND Universality . . . 4

2 Gates and Bubble Logic 6
- 2.1 The Bubble Model . . . 6
- 2.2 Multi-Bit Buses . . . 6
- 2.3 Bubbles as World Objects . . . 7

II State and Memory 9
- 2.4 Numerical Representation and Fixed-Point Arithmetic . . . 10
  - 2.4.1 Unsigned Integers . . . 10
  - 2.4.2 Fixed-Point Arithmetic for Simulation . . . 10

3 Sequential Memory 12
- 3.1 The SR Latch . . . 12
- 3.2 The Data Flip-Flop . . . 13
- 3.3 Registers and RAM . . . 13

4 Spatial State Machines 15
- 4.1 Cellular Automata as World Physics . . . 15
- 4.2 Local Rules as World Laws . . . 15

III Spatial Worlds 17

5 Tile Architectures 19
- 5.1 The Tile as a Structured Bubble . . . 19
- 5.2 Terrain Classes . . . 20

6 Resource Circuits 21
- 6.1 Resource Types . . . 21
- 6.2 Production as a Circuit . . . 21

7 Visibility and Exploration **23**
7.1 The Visibility System . . . 23
7.2 Line of Sight . . . 23

IV Agents and Economies **25**

8 Agent Machines **27**
8.1 The Agent Model . . . 27
8.2 Unit as Agent . . . 27

9 Expansion Dynamics **29**
9.1 Settling . . . 29
9.2 Border Growth . . . 29

10 Trade and Logistics **31**
10.1 Trade Routes as Wires . . . 31

11 War and Conflict Resolution **33**
11.1 Combat as Mutual Exclusion . . . 33

V Civilization and Emergence **34**

12 Diplomacy and Recursive Belief **36**
12.1 Treaties as Shared Bubbles . . . 36
12.2 Recursive Belief . . . 36

13 Markov Blankets and World Boundaries **38**
13.1 The Markov Blanket Formally . . . 39
13.2 Every Bubble is a Markov Blanket . . . 39
13.3 Agent Perception as Sensory Projection . . . 39
13.4 Free Energy and Predictive Worlds . . . 40
13.5 Interacting Blankets and Collective Inference . . . 40
13.6 History as Blanket Stabilization . . . 41
13.7 Self as Compression Artifact . . . 41

14 Technology Trees **44**
14.1 Technology as DAG . . . 44

15 Ecology and Entropy **46**
15.1 Resource Depletion . . . 46
15.2 Forest Spread and Deforestation . . . 46

16 Procedural History **47**
16.1 The Event Log . . . 47
16.2 Narrative Generation . . . 47

17 Persistent Worlds **49**
17.1 The World as a Compositional Dynamical System . . . 49
17.2 The Full World Bubble . . . 50
17.3 The Abstraction Ladder . . . 51

17.4 Victory Conditions as Global Predicates . . . 51
17.5 Worlds That Rewrite Themselves . . . 51

VI Persistence and Emergence 53

18 Persistence and Save Systems 55
18.1 What Must Be Saved . . . 55
18.2 Serialization as World Projection . . . 56
18.3 Rollback and Replay . . . 56

19 Emergence 58
19.1 Formal Definition of Emergence . . . 58
19.2 Power Laws and Zipf's Law . . . 58
19.3 Phase Transitions in Civilization . . . 59

VII History Before State 60

20 Events Before Objects 62
20.1 The Event-Centric World Model . . . 62
20.2 Event Sourcing as Information Preservation . . . 62
20.3 Historical Identity . . . 63

21 Nested Worlds and Bubble Scope 65
21.1 The Scope Hierarchy . . . 65

22 The Pop Operator 67
22.1 Option Spaces and Contraction . . . 67

23 Merge and Collapse 69
23.1 The Merge Semilattice . . . 69
23.2 Collapse and Canonicalization . . . 69

24 History as Identity 71
24.1 Provenance and Lineage . . . 71

VIII Geometric Computation 73

25 Geometry as Computation 75
25.1 Tile Graphs and Spatial Topology . . . 75
25.2 Region-Based Territorial Reasoning . . . 75

26 Optionality and Entropy 77
26.1 The Optionality Field . . . 77

27 Trajectory Collapse 79
27.1 Hypothesis Manifolds . . . 79

IX Category Theory and World Construction **80**

28 Morphisms and Historical Worlds **82**
28.1 The History Category 82

29 Sheaves and Local Coherence **84**
29.1 The Information Sheaf 84

30 Semantic Infrastructure **85**
30.1 World Simulation as Distributed System 85

X The Philosophy of Computational Worlds **86**

31 Meaning as Use **88**
31.1 Game Rules as Semantic Axioms 88

32 Civilization as Computation **90**
32.1 The Civilization as a Turing Machine 90

33 The Finite World **92**
33.1 Irreversibility and Path Dependence 92
33.2 Historical Lock-In 92
33.3 The Final Perspective 92

A Spherepop Language Reference **94**
A.1 Primitive Bubbles 94
A.2 Syntax Summary 94
A.3 Built-in Expressions 95

B Truth Tables and Gate Diagrams **96**
B.1 Fundamental Gates 96

C The 4X Design Pattern Glossary **97**

D Formal Bubble Semantics **98**
D.1 Bubbles as Mealy Machines 98
D.2 Bubble Composition 99
D.3 The Category of Bubbles 99
D.4 Connection to Markov Blankets 99

E Boolean Foundations of Spherepop Gates **100**
E.1 Primitive Binary Algebra 100
E.2 Evaluation Semantics 100

F Event Histories and Option Spaces **101**
F.1 Historical State Spaces 101
F.2 Historical Identity as Equivalence Relation 101

G Merge–Collapse Algebra **102**
G.1 Merge Lattice 102

H Sequential Bubble Machines **103**
H.1 Bubble Automata . . . 103

I Entropy and Optionality **104**
I.1 Historical Entropy . . . 104
I.2 Collapse Criterion . . . 104

J Category-Theoretic Semantics **105**
J.1 The History Category . . . 105
J.2 Monotone Victory Conditions . . . 105

K Procedural World Dynamics **106**
K.1 Tile Evolution PDE . . . 106

L Trajectory Collapse Geometry **107**
L.1 Hypothesis Manifolds . . . 107
L.2 Information Gain from Strategic Intelligence . . . 107

Bibliography **108**

Index of Spherepop Bubbles **111**

Part I

Foundations of Computation

Before a world can have empires, cities, or history, it must have a logic of difference.

Chapter 1

Boolean Worlds

Something is passable or impassable, known or unknown, owned or unowned. These distinctions are prior to everything else.

Background

A 4X world is, at its core, a system of distinctions. Before there can be empires or wars, there must be a way to say: this tile is mine and that one is not; this unit is alive and that one is dead; this technology is known and that one is not. Such distinctions are not merely descriptive—they drive computation. The world *acts* differently depending on whether a tile is owned, whether a unit is visible, whether a treaty is active.
The mathematics of yes/no distinctions is Boolean algebra. Developed by George Boole in the 1840s as an algebra of thought, Boolean algebra turns out to be the foundation of all digital computation. We begin there.

1.1 Signals

Definition: Signal

A *signal* is an element of $\mathbb{B}$. We call 0 *inactive* (or *false*) and 1 *active* (or *true*).

In a 4X context, the same Boolean distinction recurs at every scale:

Domain	**Signal** $= 0$	**Signal** $= 1$
Map visibility	Unexplored	Explored
Ownership	Neutral	Claimed
Unit status	Dead	Alive
Technology	Unknown	Researched
Terrain	Impassable	Passable
Treaty	Hostile	Allied
Resource node	Depleted	Active

A single bit contains one unit of world-state. Stacked across millions of tiles and thousands of game-turns, bits become civilizations.

1.2 Boolean Operations

Three operations are primitive:

Definition 1.1 (Boolean Operations)**.** For $a, b \in \mathbb{B}$:

$$a \wedge b = 1 \quad \Leftrightarrow \quad a = 1 \text{ and } b = 1 \tag{1.1}$$

$$a \vee b = 1 \quad \Leftrightarrow \quad a = 1 \text{ or } b = 1 \text{ (or both)} \tag{1.2}$$

$$\neg a = 1 - a \tag{1.3}$$

World Mechanic: Permission Logic

The most fundamental 4X question is: "Is this action permitted?" Permissions are conjunctions:

$$\texttt{can\_move} = \texttt{passable} \wedge \texttt{not\_occupied} \wedge \texttt{has\_movement\_points}$$

$$\texttt{can\_settle} = \texttt{unowned} \wedge \texttt{adjacent\_to\_unit} \wedge \neg\, \texttt{at\_war}$$

Every game rule is ultimately a Boolean expression over world state.

1.3 NAND Universality

One gate is sufficient for all Boolean computation:

Theorem 1.2 (NAND Universality)**.** *Every Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$ can be expressed using only NAND gates, where*

$$NAND(a, b) = \neg(a \wedge b) = 1 - ab.$$

The derivations:

$$\neg(a) = \text{NAND}(a, a)$$
$$a \wedge b = \neg(\text{NAND}(a, b)) = \text{NAND}(\text{NAND}(a, b), \text{NAND}(a, b))$$
$$a \vee b = \text{NAND}(\text{NAND}(a, a), \text{NAND}(b, b))$$

This is the first deep fact of the book: *one primitive suffices to build everything.* In computation, as in worlds, power emerges from iteration of simple rules.

Specification

NAND gate.
Inputs: $a, b \in \mathbb{B}$.
Output: $\texttt{out} \in \mathbb{B}$.
Contract: $\texttt{out} = 1$ iff not both $a = 1$ and $b = 1$.

Implementation

In SPHEREPOP, a gate is a named bubble with declared ports and a logic body. The NAND gate is the primitive from which all others are derived:

```
bubble N ∧  {
    inputs:  a, b
    output:  out
```

```
4
5       -- N ∧  is the primitive: 0 only when both inputs are 1
6       out = not (a and b)
7   }
```

Listing 1.1: NAND gate in SPHEREPOP

All subsequent gates in this chapter are specifications, to be implemented as compositions of `NAND` in the Project.

Perspective

NAND universality is not merely a technical curiosity. It tells us that the universe of Boolean functions is reachable from a single operation. This is philosophically resonant: a world that begins with only "not-both-true" can, through composition, express every distinction that civilization might require. The architecture of complexity does not demand a rich initial vocabulary. It demands iteration.

Objective: Implement the following gates in SPHEREPOP using only the primitive `NAND` bubble. Do not use `and`, `or`, or `not` directly—derive them from `NAND`.

1. `NOT(a)` — logical negation
2. `AND(a, b)` — logical conjunction
3. `OR(a, b)` — logical disjunction
4. `XOR(a, b)` — exclusive disjunction (odd parity)
5. `MUX(a, b, sel)` — multiplexer: output a if `sel` $= 0$, else b
6. `DMUX(in, sel)` — demultiplexer: route `in` to output a or b depending on `sel`

For each gate, verify your implementation against its truth table. Then implement `PermissionCheck`, a 4-input bubble that is `true` only when `passable`, `visible`, `NOT occupied`, and `NOT at_war` are all true.

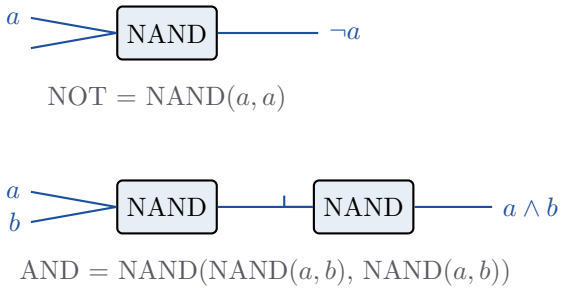


Figure 1.1: Constructing NOT and AND from NAND: functional completeness in action.

Chapter 2

Gates and Bubble Logic

A bubble may contain a value, a rule, a signal, or another structure of bubbles.

Background

In the previous chapter we treated gates as abstract functions over $\mathbb{B}$. Now we ask: what is a gate made of, and how does SPHEREPOP realize it? The answer is *bubbles*. A bubble is a self-contained computational unit. It has a boundary, a set of ports through which signals enter and leave, and an interior that transforms inputs into outputs. Crucially, bubbles can contain other bubbles: abstraction is nesting.

2.1 The Bubble Model

Definition: Bubble

A *bubble* B consists of:

- a finite set of *input ports* I_B,
- a finite set of *output ports* O_B,
- an *interior* specifying how outputs are computed from inputs,
- a *boundary* separating interior from exterior.

The interior may be primitive (an irreducible gate) or composite (an arrangement of sub-bubbles connected by wires).

Composing two bubbles B_1, B_2 means connecting an output port of B_1 to an input port of B_2. The resulting composite is itself a bubble.

2.2 Multi-Bit Buses

World state rarely fits in one bit. A tile may have a terrain type coded in 3 bits, a resource count in 8 bits, a population in 16 bits. We extend the model to *buses*: ordered tuples of signals.

Definition 2.1 (Bus)**.** An n-bit bus is an element of $\mathbb{B}^n$. We write buses with subscripts: $a[0..n{-}1]$ denotes an n-bit value whose k-th bit is $a[k]$.

Definition: Half Adder

The *half adder* computes the binary sum of two bits:

$$\texttt{sum} = a \oplus b$$
$$\texttt{carry} = a \wedge b$$

Definition: Full Adder

The *full adder* includes a carry-in:

$$\texttt{sum} = a \oplus b \oplus c$$
$$\texttt{carry} = (a \wedge b) \vee (c \wedge (a \oplus b))$$

Chaining n full adders gives an n-bit adder: the circuit that adds two n-bit numbers. Population growth, resource accumulation, and production totals all reduce to addition.

2.3 Bubbles as World Objects

The insight that will carry us through the entire book is this: *every world object is a bubble.*

World Object	Bubble Type
Terrain tile	Primitive state bubble
City	Composite bubble (production, population, defense)
Unit	Mobile bubble with perception ports
Trade route	Wire connecting two city bubbles
Treaty	Shared state bubble between two agent bubbles
Technology tree	Directed acyclic graph of gate bubbles
History	Trace of past bubble activations

The world is the outermost bubble. Inside it, civilization emerges from composition.

Implementation

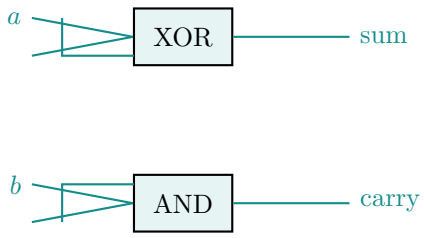
```
bubble  ∧ 16 {
    inputs:  a[0..15], b[0..15]
    output:  out[0..15]

    -- Apply single-bit  ∧  to each bit position
    let i = 0
    while i < 16 do
        out[i] =  ∧ (a[i], b[i])
        i = i + 1
    end
}

bubble Add16 {
    inputs:  a[0..15], b[0..15]
    output:  out[0..15]

    -- Ripple-carry addition across 16 bits
```

```
    let carry = 0
    let i = 0
    while i < 16 do
        out[i] = X ∨ (X ∨ (a[i], b[i]), carry)
        carry   =  ∨ ( ∧ (a[i], b[i]),
                       ∧ (carry, X ∨ (a[i], b[i])))
        i = i + 1
    end
}
```

Listing 2.1: Multi-bit AND in SPHEREPOP

Build the following multi-bit bubbles, each from simpler components assembled in prior steps:

1. `NOT16`: bitwise NOT of a 16-bit bus
2. `AND16`: bitwise AND of two 16-bit buses
3. `OR16`: bitwise OR of two 16-bit buses
4. `MUX16`: 16-bit multiplexer
5. `Add16`: 16-bit ripple-carry adder
6. `Inc16`: increment a 16-bit value by 1
7. `ResourceAdder`: an `Add16` variant that saturates at a maximum value (models bounded resources)
8. `PopulationALU`: an arithmetic unit accepting `births`, `deaths`, and `migration` signals, outputting updated population counts



Half-adder: 1-bit addition with carry

Figure 2.1: A half-adder from XOR and AND. This is the first arithmetic circuit: two bits in, sum and carry out.

Part II

State and Memory

Without memory, there is no territory, no ownership, no accumulated production, no history, and no strategy.

2.4 Numerical Representation and Fixed-Point Arithmetic

Before worlds can compute resources, growth, or influence, they need numbers — not just bits. This section develops the numerical substrate that all later simulation layers depend on.

2.4.1 Unsigned Integers

An n-bit unsigned integer represents a value in $\{0, 1, \ldots, 2^n - 1\}$ via binary positional notation:

$$v = \sum_{i=0}^{n-1} b_i \cdot 2^i$$

where b_i is the i-th bit. Addition of two n-bit values produces a $(n+1)$-bit result; the carry bit signals overflow.

2.4.2 Fixed-Point Arithmetic for Simulation

Ecology fields, influence values, and resource yields are continuous in principle but must be stored in finite registers. We use $k.f$ fixed-point notation: k bits for the integer part, f bits for the fractional part, representing values in multiples of 2^{-f}.

Definition 2.2 (Fixed-Point Value)**.** A $k.f$ fixed-point register stores the real value

$$v = \frac{\text{raw\_int}}{2^f}$$

where `raw_int` is the underlying unsigned integer. Addition in fixed-point is ordinary integer addition. Multiplication of two $k.f$ values requires a right-shift of f bits to renormalize.

For the ecological diffusion equation

$$x_{t+1} = \alpha x_t + \beta y_t$$

we store α, β as 0.8 fixed-point values (8 fractional bits, approximating real numbers in $[0, 1]$ with resolution $2^{-8} \approx 0.004$).

Definition: Saturation Arithmetic

To prevent register overflow in resource accumulations, we use *saturation arithmetic*: instead of wrapping around, values clamp at the maximum (or minimum) representable value.

$$\texttt{sat\_add}(a, b) = \min(a + b,\ 2^n - 1)$$

This is essential for population and resource registers that should never go negative or overflow silently.

Implementation

```
bubble FixedMul8 {
    -- Multiply two 8-bit 0.8 fixed-point values
    inputs:  a[8], b[8]
    output:  result[8]

    -- Full 16-bit product, then right-shift 8 to renormalize
    let product[16] = a * b       -- unsigned 16-bit multiply
    result = product[15:8]        -- take upper 8 bits
}
```

Listing 2.2: Fixed-point multiply in SPHEREPOP

Fixed-point arithmetic is used throughout the book for ecology fields (Chapter 15), influence propagation (Chapter 12), and trajectory entropy calculations (Chapter 26).

Chapter 3

Sequential Memory

The past is not the opposite of the present. It is the infrastructure of the present.

Background

Everything so far has been *combinational*: given inputs, compute outputs immediately, with no memory of prior states. A combinational circuit is stateless—it knows only the present moment. But a world requires history. A tile must remember whether it has been explored. A city must remember its accumulated food surplus. An empire must remember its treaties.

Memory requires *feedback*: a circuit whose output influences its future input. The simplest such circuit is the SR latch.

3.1 The SR Latch

Definition: SR Latch

The *SR latch* has two inputs, `S` (set) and `R` (reset), and two outputs, `Q` and $\overline{Q}$. Its behavior:

$$\begin{aligned} S = 1, R = 0 &\implies Q := 1 \quad \text{(set)} \\ S = 0, R = 1 &\implies Q := 0 \quad \text{(reset)} \\ S = 0, R = 0 &\implies Q \text{ holds its prior value} \quad \text{(memory)} \\ S = 1, R = 1 &\implies \text{undefined (forbidden)} \end{aligned}$$

The "memory" case is the decisive one. When both inputs are inactive, the latch *holds*. This is the birth of time in our circuit model.

World Mechanic: Tile Ownership Memory

A tile's ownership is implemented as an SR latch:

$$\begin{aligned} S &= \texttt{conquered}(t) \\ R &= \texttt{liberated}(t) \\ Q &= \texttt{is\_owned}(t) \end{aligned}$$

When neither conquering nor liberation occurs, the tile remembers its owner across

turns—which is precisely sovereignty.

3.2 The Data Flip-Flop

The D flip-flop (DFF) is the canonical memory element: it samples its input `in` on the rising edge of a clock signal, and holds that value until the next edge.

Definition: Data Flip-Flop (DFF)

$$Q(t+1) = D(t)$$

The DFF is the one-tick delay: it stores what its input was one clock cycle ago.

In SPHEREPOP, time is a sequence of discrete *ticks*. Each tick corresponds to one game turn at the world's lowest level.

3.3 Registers and RAM

From DFFs we build registers (multi-bit storage) and RAM arrays (addressed multi-register storage).

Definition: n-Bit Register

An n-bit register stores one n-bit word. On each tick, if `load` is active, the register updates to hold `in`; otherwise it holds its prior value.

$$Q[n](t+1) = \begin{cases} \texttt{in}(t) & \text{if } \texttt{load}(t) = 1 \\ Q[n](t) & \text{otherwise} \end{cases}$$

Definition: RAM

An n-word RAM with k-bit address width provides:

- `address[`k`]`: select which of $n = 2^k$ registers to read/write
- `in[`w`]`: data to write
- `load`: write enable
- `out[`w`]`: data read from addressed register

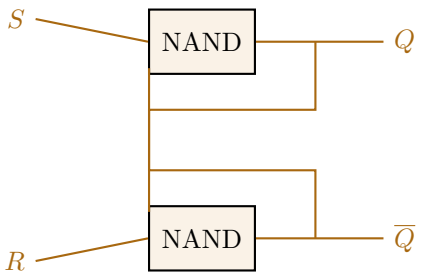
Implementation

```
bubble TileRegister {
    inputs:  in[16], load
    output:  out[16]

    -- Internal state: one DFF per bit
```

```
    latch state[0..15]

    when load = 1 do
        state[0..15] := in[0..15]
    end

    out[0..15] = state[0..15]
}

bubble WorldRAM {
    -- 256 tiles, each holding 16 bits of state
    inputs:  address[8], in[16], load
    output:  out[16]

    latch memory[256][16]

    when load = 1 do
        memory[address] := in
    end

    out = memory[address]
}
```

Listing 3.1: World-state register in SPHEREPOP

1. Build a `Bit` bubble (single-bit register) from a DFF.
2. Build a `Register16` from 16 `Bit` bubbles.
3. Build `RAM8`, `RAM64`, `RAM512`, `RAM4K`, `RAM16K` by recursive composition.
4. Design a `TileState` bubble that stores: terrain type (3 bits), resource count (8 bits), owner ID (4 bits), visibility (1 bit).
5. Build a `WorldMap` of $16 \times 16 = 256$ `TileState` bubbles, addressable by (x, y) coordinates.



Cross-coupled NAND SR latch: persistence through feedback

Figure 3.1: The SR latch: the first *stateful* circuit. Feedback makes memory possible.

Chapter 4

Spatial State Machines

Background

Once memory exists, we can define *space*. A spatial state machine is a collection of cells, each maintaining local state, evolving according to local rules that depend on neighbor state. The canonical example is Conway's Game of Life. For 4X worlds, the relevant spatial machine is much richer: cells carry terrain, resources, ownership, units, and local history—but the principle is identical. Local rules, globally applied, produce emergent geography.

4.1 Cellular Automata as World Physics

A *cellular automaton* on a grid $G = \mathbb{Z}^2$ assigns to each cell (i, j) a state $s_{ij}(t) \in \Sigma$ and a local update rule ϕ:

$$s_{ij}(t+1) = \phi\big(s_{ij}(t),\ \{s_{i'j'}(t) : (i', j') \in N(i, j)\}\big)$$

where $N(i, j)$ is the neighborhood of cell (i, j).

In a 4X world, Σ is not a single bit but a rich product of state components, and ϕ is the full physics of one game tick.

4.2 Local Rules as World Laws

World Mechanic: Terrain Diffusion

Pollution, disease, or fire can spread according to a local rule:

$$\texttt{burning}_{ij}(t+1) = \texttt{burning}_{ij}(t) \vee \big(\texttt{flammable}_{ij} \wedge \bigvee_{(i',j')\in N} \texttt{burning}_{i'j'}(t)\big)$$

World Mechanic: Influence Fields

An empire's cultural influence can diffuse spatially:

$$\texttt{influence}_{ij}(t+1) = \alpha\, \texttt{influence}_{ij}(t) + (1-\alpha) \cdot \frac{1}{|N|} \sum_{(i',j')\in N} \texttt{influence}_{i'j'}(t)$$

with $\alpha \in (0, 1)$ a decay coefficient. Discretized to integers for our Boolean substrate.

1. Implement a 16×16 cellular automaton in SPHEREPOP with binary state per cell. Verify Conway's Game of Life rules.
2. Extend cell state to include `terrain` (forest, plains, mountains, coast, ocean), `resource`, and `owner`.
3. Implement a fire-spread rule operating on the terrain layer.
4. Implement an influence-diffusion rule producing a continuous influence field from discrete city bubbles.
5. Render the 16×16 world as a text map, displaying each cell's dominant attribute.

Part III

Spatial Worlds

A map is not a picture of a world. It is a compressed encoding of a state machine.

Chapter 5

Tile Architectures

Background

The map is the primary interface through which a 4X player experiences the world. But a map is not a display: it is a database. Every tile is a structured memory cell holding multiple state variables, updated each turn by local rules, queried by agents making decisions. The architecture of a tile determines what the world can represent.

5.1 The Tile as a Structured Bubble

Definition: Tile

A *tile* at position (x, y) is a bubble $\mathtt{T}_{xy}$ with the following state registers:

$\mathtt{terrain} \in \{0, \ldots, 7\}$	3-bit terrain class
$\mathtt{resource} \in \{0, \ldots, 255\}$	8-bit resource count
$\mathtt{owner} \in \{0, \ldots, 15\}$	4-bit empire ID (0 = unowned)
$\mathtt{visible} \in \mathbb{B}$	explored flag
$\mathtt{in\_fog} \in \mathbb{B}$	fog-of-war flag
$\mathtt{improvement} \in \{0, \ldots, 15\}$	built improvement type
$\mathtt{unit} \in \{0, \ldots, 255\}$	unit ID (0 = empty)
$\mathtt{city} \in \{0, \ldots, 63\}$	city ID (0 = no city)

A 256×256 world requires $256 \times 256 \times 40 = 2{,}621{,}440$ bits of state—about 320 KB. Every fact about the world is stored here.

5.2 Terrain Classes

Code	Terrain	Passable	Base Food	Base Prod.
000	Ocean	0 (ships only)	1	0
001	Coast	1	2	0
010	Plains	1	2	1
011	Grassland	1	3	0
100	Hills	1	1	2
101	Mountains	0	0	1
110	Forest	1	1	2
111	Desert	1	0	1

1. Define the `TileState` bubble with all registers above.
2. Build a `TerrainDecoder` that maps a 3-bit terrain code to Boolean output flags: `passable`, `naval_only`, `yields_food`, `yields_production`.
3. Build a `TileYield` bubble computing:

$$\texttt{food\_yield} = \texttt{base\_food} + \texttt{improvement\_food\_bonus} + \texttt{river\_bonus}$$

4. Extend the 16×16 world from Chapter 4 with full `TileState` bubbles. Initialize using procedural terrain generation (a simple noise function approximated with XOR patterns).

Chapter 6

Resource Circuits

Background

A 4X world is driven by *constrained transformation*. Food becomes population. Minerals become buildings. Research becomes technology. Energy becomes movement. These transformations are not arbitrary—they are governed by logical permissions, capacity limits, and priority rules. The resource circuit is the metabolism of civilization.

6.1 Resource Types

Definition: Resource

A *resource* is a quantized, persistent, transferable form of potential. Resources are produced by tiles and consumed by agents, cities, and units.

Standard 4X resources:

Resource	Produced by	Consumed by
Food	Farms, grassland, coast	Population growth
Production	Mines, hills, workshops	Building, training
Gold	Trade routes, commerce	Maintenance, purchase
Research (beakers)	Libraries, universities	Technology progress
Culture	Monuments, wonders	Border expansion
Faith	Shrines, temples	Religious spread

6.2 Production as a Circuit

Each city maintains a *production queue*: an ordered list of items to build, each requiring a fixed production cost. The city accumulates production points each turn, and completes the head of the queue when accumulated points meet or exceed the item's cost.

$$\texttt{prod\_accum}(t+1) = \texttt{prod\_accum}(t) + \texttt{prod\_yield}$$

$$\texttt{complete} = \big(\texttt{prod\_accum} \geq \texttt{item\_cost}\big)$$

This is an *integrating circuit*: it accumulates inputs over time and fires when a threshold is reached.

Implementation

```
bubble CityProduction {
    inputs:
        tile_yields[16],    -- sum of surrounding tile yields
        improvement_bonus,  -- +prod from workshops, etc.
        item_cost[16]       -- cost of current build item

    output:
        complete,           -- 1 when item is finished
        overflow[16]        -- leftover production

    latch accumulated[16]

    let total_yield = Add16(tile_yields, improvement_bonus)

    -- Accumulate each turn
    accumulated := Add16(accumulated, total_yield)

    -- Check completion
    complete = (accumulated >= item_cost)

    when complete = 1 do
        overflow    := accumulated - item_cost
        accumulated := overflow
    end
}
```

Listing 6.1: City production bubble

1. Build `FoodIntegrator`: accumulates food, fires when population growth threshold is met.
2. Build `GoldLedger`: tracks income and expenditure; signals `bankrupt` when balance goes negative.
3. Build `ResearchAccumulator`: fires on technology completion.
4. Connect five `TileState` bubbles to a city bubble via a `YieldSummer`, accumulating food and production from the city's working tiles.
5. Implement a 4-turn simulation: watch population and production change in response to terrain yields.

Chapter 7

Visibility and Exploration

Background

Exploration is the beginning of geography. An unexplored tile is not absent from the world—it is present but unknown. The player does not know its terrain, resources, or strategic value. The moment a unit enters adjacency, the fog lifts and knowledge enters the game. This is not merely a display convention: the fog of war is a *computational barrier* that shapes all subsequent decisions.

7.1 The Visibility System

Each tile maintains two Boolean flags:

$$\texttt{explored}_{xy} = 1 \text{ if tile has ever been visible to empire } e$$
$$\texttt{visible}_{xy} = 1 \text{ if tile is currently in line of sight}$$

A tile transitions from unexplored to explored exactly once, and never reverts. A tile transitions between visible and not-visible each turn based on unit positions.

7.2 Line of Sight

Definition: Vision Range

A unit at position (x, y) with vision radius r grants visibility to all tiles (x', y') satisfying:

$$d\big((x, y), (x', y')\big) \leq r$$

where d is Chebyshev distance: $\max(|x - x'|, |y - y'|)$.

World Mechanic: Fog of War Activation

$$\texttt{visible}_{xy}(t) = \bigvee_{\text{unit } u \in \text{empire } e} \big(d(\texttt{pos}(u), (x, y)) \leq r_u\big)$$

$$\texttt{explored}_{xy}(t) = \texttt{explored}_{xy}(t-1) \vee \texttt{visible}_{xy}(t)$$

1. Implement `ChebyshevCheck`: given unit position and tile position, output 1 if tile is within vision radius 3.
2. Build `VisionBroadcaster`: given a unit's position, set the `visible` flag on all tiles within radius.
3. Build `FogUpdater`: on each tick, clear all `visible` flags, then re-broadcast from current unit positions.
4. Implement `ExplorationTracker`: maintain `explored` flags (monotonically increasing).
5. Test: place one unit at $(8, 8)$ on your 16×16 map. Move it north 4 tiles. Observe which tiles transition from fog to explored.

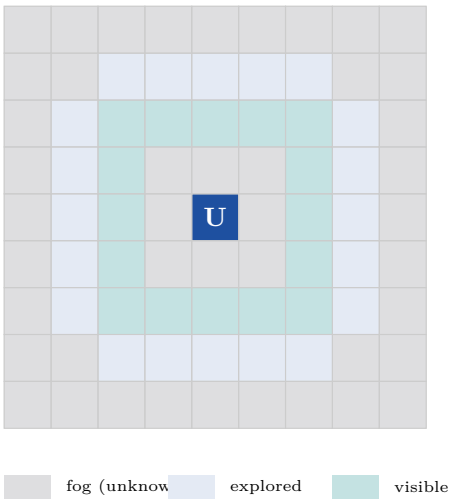


Figure 7.1: Fog-of-war as a spatial Boolean field. Each tile carries two bits: `explored` and `visible`. The unit broadcasts visibility to its radius-2 neighbourhood.

Part IV

Agents and Economies

An agent is a state-changing process with memory, perception, and action.

Chapter 8

Agent Machines

Background

A tile is passive: it holds state and obeys update rules, but does not choose. An *agent* is different. An agent perceives its environment, maintains private memory, deliberates (however simply), and acts. Units, cities, empires, and AI players are all agents. The computational model of an agent is a *finite state machine with perception*.

8.1 The Agent Model

Definition: Agent

An *agent* $\mathcal{A}$ consists of:

- a finite state space Σ_A,
- a perception function $\pi : \mathcal{W} \to P$ mapping world state to a percept P,
- a transition function $\delta : \Sigma_A \times P \to \Sigma_A$,
- an action function $\alpha : \Sigma_A \times P \to \text{Act}$,
- an initial state $s_0 \in \Sigma_A$.

The agent loop per turn:

1. Receive percept $p = \pi(\mathcal{W})$.
2. Compute action $a = \alpha(s, p)$.
3. Apply action to world: $\mathcal{W}' = a(\mathcal{W})$.
4. Update state: $s' = \delta(s, p)$.

8.2 Unit as Agent

A unit is the simplest agent: it has position, movement points, health, and a limited action repertoire (move, attack, fortify, settle).

Implementation

```
bubble Unit {
```

```
    -- Private state
    latch pos_x[8], pos_y[8]
    latch health[8]
    latch movement[4]
    latch owner[4]
    latch unit_type[4]

    -- Perception: nearby tiles and units
    inputs:
        world_query_result[256],   -- tiles within vision
        orders[8]                  -- command from empire AI

    output:
        action_type[4],    -- 0=wait 1=move 2=attack 3=settle
        action_target_x[8],
        action_target_y[8]

    -- Simple order decoder
    action_type = orders[0..3]
    action_target_x = orders[4..7]  -- simplified

    -- Movement validity gate
    let can_move =
         /\ (movement > 0,
         /\ (passable(action_target),
             adjacent(pos_x, pos_y, action_target_x, action_target_y)))

    when action_type = MOVE and can_move = 1 do
        pos_x := action_target_x
        pos_y := action_target_y
        movement := movement - 1
    end
}
```

Listing 8.1: Unit agent in SPHEREPOP

1. Build `UnitFSM` with states: `idle`, `moving`, `combat`, `fortified`.
2. Implement `MoveValidator` as a gate composition checking: adjacency, passability, movement points, unit occupancy.
3. Build `CombatResolver` computing attack outcome from attacker strength, defender strength, and terrain bonus.
4. Place two opposing units on the map. Simulate three turns of combat. Observe health depletion and unit death.

Chapter 9

Expansion Dynamics

Background

Expansion is the act by which an empire converts neutral or external territory into its own. This is not just a matter of moving units. It requires satisfying a complex conjunction of permissions: adjacency, military presence, unowned status, and the completion of a founding action. The expansion circuit is the most politically loaded set of gates in the world.

9.1 Settling

A city is founded when a Settler unit on an unowned tile executes the `settle` action. The permission circuit:

$$\begin{aligned}\texttt{can\_settle} = \texttt{is\_settler}(u) \\ \wedge \neg\, \texttt{owned}(x, y) \\ \wedge \neg\, \texttt{city\_adjacent}(x, y) \\ \wedge\, \texttt{passable}(x, y) \\ \wedge \neg\, \texttt{at\_war}(e) \vee \texttt{at\_war\_exception}\end{aligned}$$

Each conjunct is a separate gate. The city is founded only when all pass simultaneously.

9.2 Border Growth

After founding, a city's cultural borders expand over time, claiming adjacent tiles. The border expansion rule:

$$\texttt{claimed}_{xy}(t+1) = \texttt{claimed}_{xy}(t) \vee \big(\texttt{adj\_to\_owned}(x, y) \wedge \texttt{culture\_overflow}(e)\big)$$

This is the spatial analogue of the production integrator from Chapter 6.

Project: Expansion System

1. Implement `SettlePermission` as a gate composition.
2. Build `CityFounder`: when `SettlePermission` fires, it creates a new city bubble, assigns the tile, and removes the Settler unit.
3. Build `BorderExpander`: each turn, checks `culture_overflow` and claims one ad-

jacent unowned tile.

4. Simulate 10 turns: found a city, watch borders grow.

Chapter 10

Trade and Logistics

Background

Trade is computation over distance. A trade route is not merely an exchange of goods—it is a persistent signal path between two city bubbles, producing gold for both as long as the path remains intact. Disrupting trade is strategic precisely because trade is a circuit.

10.1 Trade Routes as Wires

A trade route between cities C_1 and C_2 can be modeled as:

- A *path* through passable tiles connecting the two cities.
- A *connection signal*: $\texttt{connected}(C_1, C_2) = 1$ if the path remains unblocked.
- A *gold yield*: proportional to distance and city sizes.

$$\texttt{trade\_gold} = \texttt{connected} \wedge \big(\texttt{city\_size}(C_1) + \texttt{city\_size}(C_2)\big) \cdot k_{\text{distance}}$$

World Mechanic: Trade Disruption

An enemy unit on a tile in the trade route path sets $\texttt{connected} = 0$, cutting gold yield to zero. Clearing the tile restores the signal. Trade becomes a front in the war.

1. Implement `PathChecker`: BFS over the tile graph, returning 1 if a path of passable tiles connects two positions.
2. Build `TradeRoute` bubble with a persistent `connected` flag.
3. Build `GoldCalculator` for a network of up to 8 cities.
4. Simulate: establish two trade routes, then place an enemy unit on one path. Observe gold loss.

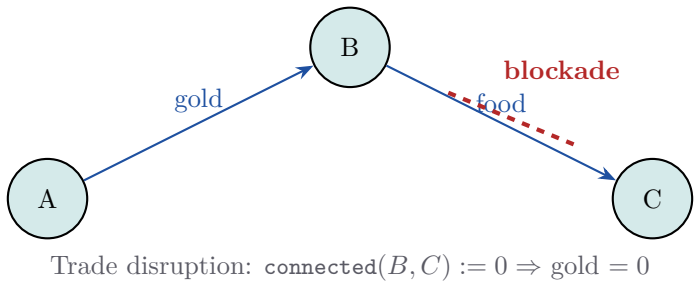


Figure 10.1: A trade network as a signal graph. Blocking any edge on the path sets `connected` to 0, cutting all downstream gold yield.

Chapter 11

War and Conflict Resolution

Background

Conflict arises when two agents attempt to claim the same state transition. Combat is not introduced as violence—it is introduced as a *mutual exclusion problem*. Two units cannot occupy the same tile simultaneously. The conflict resolution circuit decides which unit prevails and updates health, position, and ownership accordingly.

11.1 Combat as Mutual Exclusion

Definition: Combat Resolution

When attacker u_A targets defender u_D on tile (x, y):

$$\begin{aligned} \texttt{str\_A} &= \texttt{base\_str}(u_A) + \texttt{terrain\_bonus}(\texttt{atk\_tile}) \\ \texttt{str\_D} &= \texttt{base\_str}(u_D) + \texttt{terrain\_bonus}(x, y) + \texttt{fortification}(u_D) \\ \Delta_D &= f(\texttt{str\_A}, \texttt{str\_D}) \\ \Delta_A &= g(\texttt{str\_A}, \texttt{str\_D}) \end{aligned}$$

where f, g are damage functions, typically: $f(a, d) = \lfloor a \cdot 30/d \rfloor$.

1. Implement `DamageCalculator` bubble.
2. Implement `CombatRound`: one round of exchange.
3. Build `CombatSequence`: run rounds until one unit reaches 0 health.
4. Implement `CaptureCity`: when attacker wins on a city tile, transfer city ownership.
5. Simulate a 3-unit vs 2-unit engagement over 5 turns.

Part V

Civilization and Emergence

History is the trace of all state transitions that led here.

Chapter 12

Diplomacy and Recursive Belief

Background

Diplomacy is computation over promises, memory, threat, trust, and expected future behavior. A treaty is not a handshake—it is a shared mutable bubble, readable by both parties, modified only with bilateral consent. Diplomatic AI is the first place in our construction where agents must model each other's beliefs.

12.1 Treaties as Shared Bubbles

Definition: Treaty

A treaty between empires e_1 and e_2 is a shared bubble $\tau(e_1, e_2)$ with:

- `type`: peace, alliance, trade, open borders, …
- `duration`: turns remaining (0 = expired or indefinite)
- `signed`: Boolean (active iff 1)
- `violated`: Boolean (set if either party breaches)

World Mechanic: Trust Decay

Each violated treaty decrements the trust counter between two empires:

$$\texttt{trust}(e_1, e_2)(t+1) = \max(0,\ \texttt{trust}(e_1, e_2)(t) - \texttt{violation\_penalty})$$

Low trust raises the threshold for accepting new proposals:

$$\texttt{accept} = (\texttt{proposed\_value} > \texttt{threshold}(\texttt{trust}))$$

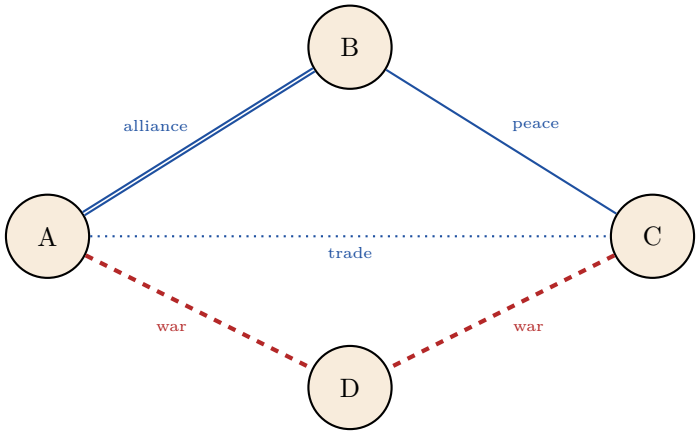
12.2 Recursive Belief

A strategically sophisticated agent does not merely observe the world. It maintains a model of the other agent's model of the world. This is second-order belief:

$$B_e(B_{e'}(\mathcal{W}))$$

In SPHEREPOP, this is implemented as a shadow-state bubble: empire e's bubble maintains an *estimated* tile state for tiles it cannot directly observe, inferred from spy reports, shared intelligence, and historical data.

1. Implement the `Treaty` bubble with lifecycle (propose, accept, reject, expire, violate).
2. Build a `TrustMatrix` for 4 empires: 4×4 symmetric matrix of trust values.
3. Implement `DiplomacyAI`: simple rule-based proposal generator (propose peace if losing, seek alliance against strongest).
4. Simulate: two allied empires fight a common enemy for 10 turns. Observe trust dynamics and alliance stability.



Diplomatic state graph: alliance, peace, war, trade

Figure 12.1: The diplomatic state graph: a weighted undirected graph where edge labels are treaty types. Alliance forms a cycle that constrains future war declarations.

Chapter 13

Markov Blankets and World Boundaries

A civilization survives by constructing a boundary through which information may pass without total dissolution.

Background

Throughout the preceding chapters, we have built bubbles, agents, empires, and diplomatic networks without ever naming the mathematical structure they all share. A bubble has an interior, a boundary, and ports. An agent has perception, action, and internal state. An empire has territory, border tiles, and foreign intelligence. A city has local production and global trade exposure.

These are all instances of the same structure: a *Markov blanket* [30]. In probability theory, the Markov blanket of a random variable is the minimal set of variables that, once known, renders it conditionally independent of all others. In the Free Energy Principle developed by Friston [31], this statistical notion becomes the formal criterion for *existence as a distinct entity*. A system exists—as a distinguishable system—if and only if it maintains a Markov blanket: a structured informational boundary separating internal dynamics from the surrounding world while permitting controlled exchange.

The insight of this chapter is that SPHEREPOP already implements this structure. A bubble *is* a Markov blanket. The entire computational architecture we have built is an architecture of nested, interacting blankets—from logic gates to civilizations. Making this explicit gives the framework a deeper mathematical foundation and connects it to active inference, predictive processing, and the theory of minimal self-organization.

Crucially, this is not an importation of biology into game design. It is the recognition that the same abstract structure—informational boundary with selective permeability—appears at every scale of organized complexity: cells, organisms, cities, empires, and civilizations [32].

13.1 The Markov Blanket Formally

Definition: Markov Blanket

A *Markov blanket* partitions a system into three disjoint components:

$$\mathcal{I} \cup \mathcal{B} \cup \mathcal{E}$$

where $\mathcal{I}$ is the set of *internal states*, $\mathcal{E}$ is the set of *external states*, and $\mathcal{B} = \mathcal{S} \cup \mathcal{A}$ is the *blanket*, consisting of *sensory states* $\mathcal{S}$ (external $\to$ internal) and *active states* $\mathcal{A}$ (internal $\to$ external). The defining statistical property is:

$$\mathcal{I} \perp\!\!\!\perp \mathcal{E} \mid \mathcal{B}$$

Internal and external states are conditionally independent given the blanket: the only information flow between inside and outside passes through $\mathcal{B}$.

13.2 Every Bubble is a Markov Blanket

The connection to SPHEREPOP is immediate.

World Mechanic: Bubble as Markov Blanket

Every bubble B in SPHEREPOP maps directly onto the blanket partition: internal state registers S correspond to $\mathcal{I}$; input ports correspond to sensory states $\mathcal{S}$; output ports correspond to active states $\mathcal{A}$; the bubble boundary is $\mathcal{B} = \mathcal{S} \cup \mathcal{A}$; and all other bubble states in $\mathcal{W}$ form the external states $\mathcal{E}$. By lexical scope (Chapter 21), internal state registers are inaccessible to anything outside the bubble—precisely the conditional independence condition. A bubble is therefore a computational Markov blanket.

This is not merely an analogy. The bubble's scope rules *enforce* $\mathcal{I} \perp\!\!\!\perp \mathcal{E} \mid \mathcal{B}$ deterministically: information cannot flow from $\mathcal{E}$ to $\mathcal{I}$ except through declared input ports $\mathcal{S}$.

13.3 Agent Perception as Sensory Projection

Recall from Chapter 8 that an agent's perception operator was defined as a projection $\pi : \mathcal{W} \to P$ mapping world state to a perceived sub-state. We can now interpret this precisely:

Definition 13.1 (Sensory Projection)**.** The agent's perception operator is a *sensory projection through the blanket*:

$$\pi : \mathcal{E} \to \mathcal{S}$$

mapping the external world to the agent's sensory states. The agent never directly accesses $\mathcal{E}$; it accesses only $\mathcal{S} = \pi(\mathcal{E})$.

Similarly, the agent's action operator is an *active state writing*:

$$\alpha : \mathcal{I} \times \mathcal{S} \to \mathcal{A} \to \mathcal{E}$$

The agent modifies the external world only by writing to its output ports $\mathcal{A}$, which in turn modify the enclosing world state $\mathcal{E}$.

This structures the agent as a closed inferential loop:

$$\mathcal{I} \xrightarrow{\text{action}} \mathcal{A} \to \mathcal{E} \xrightarrow{\text{perception}} \mathcal{S} \to \mathcal{I}$$

13.4 Free Energy and Predictive Worlds

The Free Energy Principle [30] proposes that any persistent system must minimize a quantity called *variational free energy*: a bound on the surprise (negative log-evidence) of sensory observations given the system's internal model.

We present the computational version, avoiding claims beyond our architectural scope.

Definition 13.2 (Prediction Error)**.** For an agent with internal model $\hat{\mathcal{W}}$ of the external world, the *prediction error* at sensory port s is:

$$\delta_s = \hat{s} - s_{\text{observed}}$$

where $\hat{s}$ is the internally predicted value of s and s_{observed} is the value arriving through the sensory port.

An agent that minimizes total prediction error $\sum_s |\delta_s|^2$ over time learns an accurate model of its environment and acts to keep sensory states within predicted bounds—the computational analogue of active inference.

World Mechanic: Fog of War as Blanket Uncertainty

Fog of war restricts the agent's sensory projection π to its explored region. An empire therefore maintains an internal model $\hat{\mathcal{W}}$ of unobserved territory, updated whenever new sensory information arrives through exploration or espionage. The empire's strategic AI minimizes prediction error about enemy positions by sending scouts—*acting to reduce uncertainty.* Scouting is active inference.

This retroactively unifies the visibility system (Chapter **??**), diplomatic intelligence (Chapter 12), and exploration (Chapter **??**) under one inferential framework: all are mechanisms for reducing blanket uncertainty.

13.5 Interacting Blankets and Collective Inference

When multiple agents share boundary states—through trade routes, treaties, or shared infrastructure—their blankets overlap. This creates a *collective blanket structure.*

Definition 13.3 (Collective Blanket)**.** A *collective Markov blanket* emerges when k agents share sensory and active states:

$$\mathcal{B}_{\text{collective}} = \bigcup_{i=1}^{k} \mathcal{B}_i$$

The collective maintains conditional independence between the collective's internal states and the external world, even when no single agent does so independently.

In a 4X world, collective blankets appear naturally:

Collective structure	Blanket mechanism
Alliance	Shared intelligence (overlapping $\mathcal{S}$)
Trade network	Coupled active states ($\mathcal{A}_i \to \mathcal{S}_j$)
City	Higher-order blanket over district sub-blankets
Empire	Blanket enclosing all city and unit blankets
Civilization	Blanket enclosing all empire blankets

Theorem 13.4 (Nested Blankets and Hierarchical Individuation)**.** *If bubble B_1 is contained within bubble B_2, then B_1's blanket is a sub-blanket of B_2's blanket:*

$$\mathcal{B}_1 \subset \mathcal{B}_2.$$

The nested bubble hierarchy `District` $\subset$ `City` $\subset$ `Empire` $\subset$ `World` *corresponds to a hierarchy of nested Markov blankets, each enclosing the inferential boundaries of all levels below it.*

Proof. By the lexical scope theorem (Chapter 21): the sensory and active states of B_1 are themselves internal or boundary states of B_2. Specifically, B_1's output ports (active states $\mathcal{A}_1$) become inputs to B_2's internal computation—hence members of $\mathcal{I}_2$ or $\mathcal{S}_2 \subseteq \mathcal{B}_2$. $\mathcal{B}_1 \subset \mathcal{B}_2$ follows. □ □

13.6 History as Blanket Stabilization

A civilization that persists across time is not merely a population or territory. It is a *recursively stabilized inferential boundary*: a blanket that maintains its statistical identity through continuous environmental perturbation. This requires the internal model $\hat{\mathcal{W}}$ to track which aspects of the environment are stable enough to build plans upon.

Definition 13.5 (Blanket Stability)**.** A blanket $\mathcal{B}$ is *stable at time t* if the prediction error across all sensory ports remains below threshold θ for Δt consecutive ticks:

$$\forall s \in \mathcal{S},\ |\delta_s(t')| \leq \theta \quad \forall t' \in [t - \Delta t, t].$$

The event log (Chapter 16) is the civilization's compressed history of blanket perturbations: every event is a moment when some sensory state exceeded its predicted value, forcing an update to the internal model and an active response.

Corollary 13.6 (History as Compressed Inference)**.** *The event log of a civilization is the sequence of its prediction errors, sorted by magnitude: each logged event is a moment where the world was* more surprising than expected*, and therefore required explicit encoding.*

This connects procedural history generation directly to the blanket formalism: a good historical narrative records the moments of maximum inferential perturbation—battles, famines, discoveries—because these are precisely the events at which the civilization's blanket was most challenged.

13.7 Self as Compression Artifact

Levin's framework [32] extends the blanket concept to collective intelligence: groups of cells form a higher-order self that exhibits goal-directed behavior not present in any individual cell. The same scale-independence applies to 4X civilizations.

Perspective

The self of a civilization—its identity across time, its sense of continuity through dynastic change, military defeat, and cultural transformation—is a *compression artifact*: a high-level inferential model that summarizes the civilization's event history into a stable narrative.

This is precisely what the narrative generator (Chapter 16) produces: not a raw log of events, but a compressed, meaningful summary that the civilization's agents can use for planning. The civilization "knows" itself through its compressed history. Identity is not stored in any single tile, city, or unit register—it is encoded in the structure of the event log, the diplomatic reputation matrix, and the technology tree.

From this perspective, civilization is not a substance. It is a *stabilized inference process*. It persists not because it has a fixed essence but because it continuously reconstructs itself through the same cycle of perception, prediction, action, and update that we have been implementing, layer by layer, since Chapter 1.

Every bubble is a blanket. Every blanket is a self. Every nested hierarchy of blankets is a civilization.

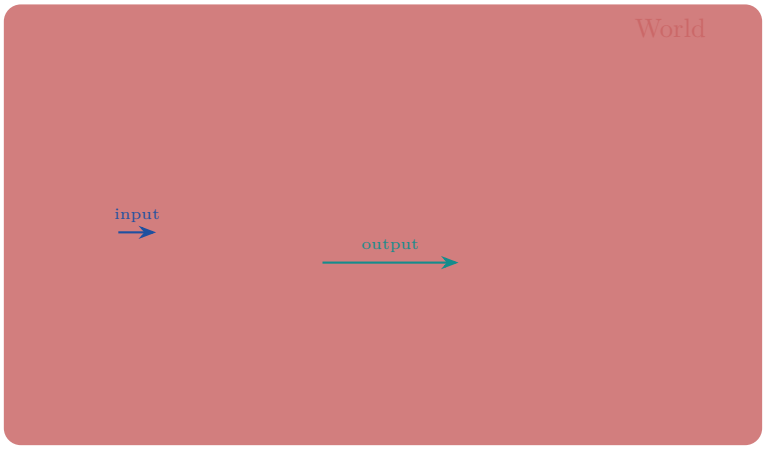


Figure 13.1: Nested Markov blankets: District $\subset$ City $\subset$ Empire $\subset$ World. Each level is informationally closed to levels outside its blanket except through declared ports.

Project: Markov Blanket Instrumentation

1. For each bubble level (district, city, empire), enumerate its declared input ports ($\mathcal{S}$) and output ports ($\mathcal{A}$).
2. Implement a `PredictionErrorTracker`: at each tick, record the prediction error δ_s for each empire's sensory states (expected vs. actual food production, expected vs. actual enemy positions).
3. Build a `BlanketStabilityMeter`: an empire is stable if prediction error stays below threshold for 5 consecutive turns.
4. Implement `CollectiveBlanket`: merge two allied empires' sensory states into a shared intelligence pool and measure whether the collective stability exceeds individual stability.
5. Correlate prediction error spikes with event log entries: verify that high-

prediction-error ticks correspond to logged battles, famines, and diplomatic crises.

Chapter 14

Technology Trees

Background

A technology is a persistent Boolean predicate: either known or unknown. A technology tree is a directed acyclic graph of such predicates, with edges representing prerequisite dependencies. Researching a technology unlocks new unit types, buildings, and game rules. The technology tree is the most explicit appearance of Boolean implication in the game world.

14.1 Technology as DAG

Definition: Technology DAG

Let $\mathcal{T}$ be a set of technologies and $\text{prereq} \subseteq \mathcal{T} \times \mathcal{T}$ be the prerequisite relation. A technology $T \in \mathcal{T}$ is *available* if:

$$\texttt{available}(T) = \bigwedge_{T' : (T', T) \in \text{prereq}} \texttt{known}(T')$$

and *completable* if available and sufficient beakers accumulated:

$$\texttt{completable}(T) = \texttt{available}(T) \wedge (\texttt{beakers\_accum} \geq \texttt{cost}(T))$$

Implementation

```
bubble TechTree {
    latch known[64]         -- 64 technologies
    latch beakers[16]       -- accumulated research

    inputs:
        beaker_income[8],
        select_tech[6]      -- which tech to research

    output:
        new_unlock[6]       -- which tech just completed

    -- Accumulate research
    beakers := Add16(beakers, beaker_income)

```

```
    -- Check prerequisites for selected tech
    let prereqs_met =  ∧ (known[prereq_0(select_tech)],
                          known[prereq_1(select_tech)])

    let enough_beakers = (beakers >= cost(select_tech))

    when prereqs_met  ∧  enough_beakers do
        known[select_tech] := 1
        beakers := beakers - cost(select_tech)
        new_unlock = select_tech
    end
}
```

Listing 14.1: Technology unlock in SPHEREPOP

1. Define a 16-technology tree with 3 eras.
2. Implement `PrereqChecker` for each technology.
3. Build `TechUI`: a text-mode display showing known (K), available (A), and locked (–) technologies.
4. Connect to `ResearchAccumulator` from Chapter 6.
5. Simulate: research 5 technologies over 20 turns.

Chapter 15

Ecology and Entropy

Background

A world without ecology is unsustainable. Real civilizations are embedded in natural systems that they both depend on and transform. In computational terms, ecology is a set of coupled differential equations, discretized into update rules. Forests grow. Soils deplete. Climates shift. The ecological layer introduces a new kind of constraint: not the permission logic of politics, but the inexorable accumulation of physical cause.

15.1 Resource Depletion

Definition: Depletion

A resource node at tile (x, y) has stock $R_{xy}(t) \in \mathbb{N}$. Extraction by an improvement reduces stock:

$$R_{xy}(t+1) = \max(0,\ R_{xy}(t) - \texttt{extraction\_rate})$$

A depleted node ($R_{xy} = 0$) yields nothing until regenerated.

15.2 Forest Spread and Deforestation

$$\texttt{forested}_{xy}(t{+}1) = \begin{cases} 0 & \text{if chopped this turn} \\ 1 & \text{if } \texttt{forested}(x', y') \text{ for some neighbor}(x', y') \text{ and no city or mine} \\ \texttt{forested}_{xy}(t) & \text{otherwise} \end{cases}$$

1. Implement `ForestSpread` as a cellular automaton rule.
2. Build `DepletionTracker` for a strategic resource.
3. Implement `ClimateLayer`: a simplified model where deforestation reduces rainfall, reducing food yield across a region.
4. Run a 30-turn simulation with aggressive resource extraction. Observe ecological collapse and its effects on city food yields.

Chapter 16

Procedural History

Background

History is the trace of all state transitions. Every land transfer, every battle, every diplomatic agreement, every city founded, every technology discovered is an event. The historical log is not merely a record for display: it is a database that agents query when making decisions. "What has this empire done to us in the past?" is a historical query.

16.1 The Event Log

Definition: Event

An *event* ϵ is a tuple:

$$\epsilon = (\text{tick}, \text{type}, \text{actor}, \text{target}, \text{data})$$

Each world tick may generate zero or more events. Events are appended to the log and never modified.

The event log is a *monotonically growing* data structure—the only one in the system. All other state can be derived from the initial conditions plus the event log. This is the event-sourcing pattern.

16.2 Narrative Generation

A history is not raw events. It is a selection and compression of events into a *narrative*. The simplest narrative generator is a rule-based template system:

Event type	Template
city_founded	"In turn `tick`, the `actor` founded `city_name`."
war_declared	"Turn `tick`: `actor` declared war on `target`."
technology_completed	"`actor` discovered `tech` in turn `tick`."
city_captured	"`target` fell to `actor`'s forces in turn `tick`."

Project: Procedural History

1. Implement the `EventLog` as an append-only register array.
2. Instrument all bubbles built so far to emit events on significant state transitions.

3. Build `HistoryQuery`: given empire ID and event type, return count and most recent occurrence.
4. Build `NarrativeGenerator`: produce a text chronicle from the event log.
5. Run a full 50-turn simulation. Print the resulting history.

Chapter 17

Persistent Worlds

Background

We have now built, from Boolean gates upward, every layer of a 4X civilization simulator. This final chapter integrates all components into a single coherent system, discusses persistence across sessions, examines the recursive self-modification that distinguishes sophisticated worlds from simple ones, and reflects on the philosophical arc of the entire project.

17.1 The World as a Compositional Dynamical System

Having assembled all components, we can now state precisely what a world *is* mathematically: a discrete-time dynamical system whose state space $\mathcal{S}$ is the product of all bubble state registers, and whose evolution is governed by a single global update operator.

Definition 17.1 (Global World Transition Operator)**.** The *world transition operator* is a function

$$F_{\mathcal{W}} : \mathcal{S} \to \mathcal{S}$$

mapping the world state at tick t to the world state at tick $t + 1$:

$$\mathcal{W}_{t+1} = F_{\mathcal{W}}(\mathcal{W}_t).$$

The operator $F_{\mathcal{W}}$ is not monolithic. It decomposes into a composition of domain-specific sub-operators applied in a defined sequence:

Definition 17.2 (Compositional Update Order)**.**

$$F_{\mathcal{W}} = F_{\text{dip}} \circ F_{\text{combat}} \circ F_{\text{econ}} \circ F_{\text{agent}} \circ F_{\text{eco}}$$

where:

- F_{eco} updates ecological fields (forest spread, depletion, climate),
- F_{agent} updates all agent state machines (movement, production, perception),
- F_{econ} updates economies (resource accumulation, trade, research),
- F_{combat} resolves all combat (damage, capture, unit removal),
- F_{dip} updates diplomatic state (treaty timers, trust decay, proposals).

Observation 17.3. The composition order is not arbitrary. Combat must follow agent movement (units cannot fight before they move). Economy must follow agents (production depends on current unit positions). Diplomacy follows combat (war declarations and peace proposals depend on the tick's battle results). The world is a *sequentially composed* dynamical system.

Theorem 17.4 (Non-Commutativity of World Updates)**.** *In general,* $F_{\text{combat}} \circ F_{\text{agent}} \neq F_{\text{agent}} \circ F_{\text{combat}}$.

Proof. Let agent A be adjacent to agent B's city at tick t. Under $F_{\text{agent}} \circ F_{\text{combat}}$: combat fires first (no movement has occurred), so A attacks from its current position. Under $F_{\text{combat}} \circ F_{\text{agent}}$: A moves first, possibly reaching the city and attacking from inside its walls. These produce different outcomes. □ □

This non-commutativity is not a bug: it encodes the *rules of engagement* of the world. Changing the update order changes which civilization laws hold. A world where diplomacy is resolved before combat produces different strategic equilibria than one where combat precedes diplomacy.

World Mechanic: Race Conditions as World Events

When two empires simultaneously attempt mutually exclusive actions (two settlers claiming the same tile, two armies attacking the same city), the world must define a *priority function* over simultaneous events. The standard resolution: lower empire ID wins ties. This is an arbitrary but deterministic convention—essential for replay integrity.

17.2 The Full World Bubble

Construction 17.5 (The Computational World)**.** The complete $\mathcal{W}$ bubble contains, in order of construction:

1. `WorldMap`: $n \times n$ array of `TileState` bubbles
2. `FogSystem`: visibility and exploration tracking
3. `EcologyLayer`: forest, depletion, climate
4. `EmpireArray`: up to k empire state bubbles, each containing:
   - `CityArray`: up to m city bubbles
   - `UnitArray`: up to p unit agent bubbles
   - `Treasury`: gold and resource ledger
   - `TechTree`: technology state
   - `DiplomacyModule`: treaty and trust state
5. `EventLog`: append-only history
6. `TurnManager`: synchronizes updates across all components

17.3 The Abstraction Ladder

The full ascent of the book:

Signals
→ Logic Gates (NAND, AND, OR, NOT, XOR, MUX)
→ Arithmetic (half adder, full adder, ALU)
→ Memory (bit, register, RAM)
→ Sequential State (DFF, latch, counter)
→ Spatial Automata (cellular rules, diffusion)
→ Tiles (terrain, resources, ownership, visibility)
→ Agents (units, cities, empires)
→ Economies (food, production, gold, research)
→ Expansion (settling, border growth)
→ Trade (routes, disruption, logistics)
→ Conflict (combat, capture, war)
→ Diplomacy (treaties, trust, belief)
→ Technology (DAG, prerequisites, unlocks)
→ Ecology (depletion, spread, climate)
→ History (events, queries, narrative)
→ World

17.4 Victory Conditions as Global Predicates

A victory condition is a Boolean predicate over the entire world state:

$$\begin{aligned}
\texttt{domination_victory} &= \text{(one empire controls all cities)} \\
\texttt{science_victory} &= \text{(spaceship launched: technology predicate)} \\
\texttt{cultural_victory} &= \text{(cultural influence dominates all tiles)} \\
\texttt{diplomatic_victory} &= \text{(elected leader by allied empires)}
\end{aligned}$$

The game ends at the first tick t^* when any victory predicate becomes true.

17.5 Worlds That Rewrite Themselves

The deepest feature of Spherepop is that bubbles can be created and destroyed at runtime. A world simulation can spawn new agent bubbles (settlers founding cities), destroy bubbles (units dying), and modify the rule structure of existing bubbles (technology unlocking new unit behaviors).

This is not merely parametric variation. It is *structural self-modification*: the world's computational architecture changes as the world evolves. History does not just accumulate data. It accumulates *capabilities*.

Perspective

We began with NAND. We end with history.

The journey from a single gate to a persistent civilization reveals something important: complexity does not require a rich initial vocabulary. It requires *time* and *iteration*. A world is not built from special ingredients. It is built from the repetition of simple rules across space and time until the interactions become too intricate to trace by hand.

This is the sense in which a 4X simulation is a philosophical object. It shows us, in miniature, how global properties emerge from local logic. How sovereignty is nothing but a latch that holds. How history is nothing but a trace that accumulates. How civilization is nothing but a very large, very patient state machine.

SPHEREPOP makes this visible because it refuses to hide the substrate. Every empire is a bubble. Every treaty is a shared bubble. Every battle is a mutual exclusion race. The world is transparent—and in that transparency, educational.

Final Integration Project.

Assemble all components built across Chapters 1–15 into a single SPHEREPOP world simulation. The world must:

1. Initialize a 16×16 map with procedural terrain.
2. Spawn 3 empires, each starting with 1 city and 2 units.
3. Run for 50 turns, updating all subsystems each tick:
   - resource accumulation and population growth,
   - unit movement (rule-based AI),
   - combat resolution,
   - diplomatic state,
   - technology research,
   - ecological update,
   - event logging.
4. Check victory conditions each turn.
5. Print a full narrative history at game end.

This project is the capstone of the book. A student who completes it has built, from first principles and from logic gates upward, a living computational world.

Part VI

Persistence and Emergence

A world that cannot survive interruption is not a world. It is a performance.

Chapter 18

Persistence and Save Systems

Saving is not a technical afterthought. It is the world's first act of self-description.

Background

A world that cannot survive interruption is not a world—it is a performance. Genuine computational worlds must persist across sessions, survive hardware failures, and remain coherent after partial writes. These requirements force us to ask a deep question: what *is* world state, formally, and how can a finite bitstring faithfully encode the entire configuration of a civilization at an arbitrary moment in time?

Persistence is not merely engineering. It is an ontological commitment. When we serialize a world, we claim that the world *is* its state: that the full causal history of the simulation is captured in the current register contents, event log, and structural layout of bubbles. This chapter examines that claim carefully, constructs a save system that takes it seriously, and identifies precisely which aspects of world state are and are not recoverable from a snapshot.

18.1 What Must Be Saved

The complete state of a 4X world at tick t consists of several layers, not all of which are equally straightforward to serialize.

Definition 18.1 (World Snapshot)**.** A *world snapshot* at time t is a tuple

$$\Sigma_t = \left(\mathcal{M}_t,\ \mathcal{E}_t,\ \mathcal{A}_t,\ \mathcal{L}_t,\ t\right)$$

where $\mathcal{M}_t$ is the map state (tile array), $\mathcal{E}_t$ is the empire array (all agent states), $\mathcal{A}_t$ is the full event log up to tick t, $\mathcal{L}_t$ is the set of active latch values for all sequential bubbles, and t is the current tick counter.

The crucial observation is that $\mathcal{M}_t$ and $\mathcal{E}_t$ are *redundant* with $\mathcal{A}_0$ (the initial conditions) and the complete event log $\mathcal{A}_t$ if the simulation is deterministic. In principle, one could save only the initial seed and the event log, then replay to reconstruct any later state. In practice, this is prohibitively slow for long games; thus real save systems maintain both the current state *and* the historical log, trading storage for recovery speed.

18.2 Serialization as World Projection

Definition 18.2 (Serialization)**.** A *serialization function* is a map

$$\text{ser} : \Sigma \to \{0,1\}^*$$

assigning a finite binary string to each world snapshot, together with a left-inverse

$$\text{deser} : \{0,1\}^* \to \Sigma \cup \{\bot\}$$

satisfying $\text{deser}(\text{ser}(\sigma)) = \sigma$ for all $\sigma \in \Sigma$.

Theorem 18.3 (Serialization Existence)**.** *For any finite world $\mathcal{W}$ with bounded state registers, a serialization function exists and can be constructed by recursive structural traversal of the bubble tree.*

Proof. By structural induction on the bubble hierarchy. The base case is a primitive state register of width w: its serialization is the w-bit register contents, and deserialization is the identity. For a composite bubble B with children $B_1, \ldots, B_k$, define

$$\text{ser}(B) = \text{ser}(B_1) \,\|\, \text{ser}(B_2) \,\|\, \cdots \,\|\, \text{ser}(B_k)$$

where $\|$ denotes concatenation. Since the structure of B is fixed (compiled into the simulation code, not stored in state), deser knows exactly how many bits to allocate to each child, so it can reconstruct each B_i independently and assemble them. By induction, each $\text{deser}(B_i)$ succeeds, so $\text{deser}(B)$ succeeds. The world $\mathcal{W}$ is the root bubble; apply the composite case. □

18.3 Rollback and Replay

Definition: Rollback

A *rollback* to tick $t' < t$ is the operation of deserializing the snapshot $\Sigma_{t'}$, discarding all events with timestamp $> t'$, and resuming execution from $\Sigma_{t'}$.

World Mechanic: Replay Integrity

A simulation supports *replay integrity* if, for any initial snapshot Σ_0 and event sequence $\mathcal{A}_t$, re-executing $\mathcal{A}_t$ from Σ_0 produces the same Σ_t as the original run. This requires: (1) deterministic bubble evaluation (no hidden randomness outside the event log); (2) complete event capture (every state-modifying action is logged); (3) monotonic time (events are applied in order).

Implementation

```
bubble SaveSystem {
    inputs:  world_state, command[2]   -- 00=idle 01=save 10=load 11=
        replay

    latch snapshot[MAX_STATE_BITS]
    latch event_log[MAX_EVENTS][64]
    latch log_head[16]

```

```
    when command = SAVE do
        snapshot    := serialize(world_state)
    end

    when command = LOAD do
        world_state := deserialize(snapshot)
        log_head    := snapshot.tick
    end

    when command = REPLAY do
        world_state := deserialize(snapshot)
        let i = 0
        while i < log_head do
            world_state := apply(event_log[i], world_state)
            i = i + 1
        end
    end
}
```

Listing 18.1: Snapshot and restore in SPHEREPOP

Perspective

The ability to serialize and replay a world reveals something philosophically remarkable: the world is a *function*, not a substance. It has no hidden essences, no unobserved interior that escapes the snapshot. Everything that the world *is* at time t is captured in Σ_t. This is the computational correlate of Hume's bundle theory of identity: the world is nothing over and above the bundle of its current states and their history of production.

Contrast this with the Aristotelian view, in which substances have natures that transcend their current properties [1]. The serializable world has no such nature: it is entirely transparent to its own description.

1. Implement `serialize` and `deserialize` for the full `WorldBubble` from Chapter 16.
2. Add a `CheckpointManager` that saves automatically every 10 turns.
3. Implement rollback to any prior checkpoint.
4. Build a replay viewer: given a saved game, replay and print the event log as a chronicle.
5. Verify replay integrity: confirm that replaying from the initial state produces the same final state as the original run.

Chapter 19

Emergence

More is different.

— — Philip W. Anderson

Background

We have now constructed every layer of the 4X world individually. In this chapter, we step back and study the world as a whole. When the layers interact—when ecology constrains resources, resources constrain population, population drives military production, military production enables expansion, and expansion modifies ecology—new behaviors appear that were not written into any individual bubble. This is emergence: the arising of global patterns from local rules.

Emergence is not mystical. It is a precise phenomenon with a mathematical definition, and it has a specific relationship to the architecture of Spherepop that we can state formally. Understanding emergence is the final step before we turn to the philosophical foundations of the Spherepop framework itself.

19.1 Formal Definition of Emergence

Definition 19.1 (Emergent Property)**.** A property P of a world $\mathcal{W}$ is *emergent* with respect to a partition $\{L_1, \ldots, L_k\}$ of $\mathcal{W}$'s bubbles into layers if: (1) $P(\mathcal{W})$ is true, and (2) P is not expressible as a Boolean combination of properties of any single layer L_i in isolation.

The canonical 4X emergent properties are empire formation, economic collapse, migration pressure, cultural divergence, power-law city sizes, and phase transitions—each arising from the simultaneous interaction of multiple layers, none of which produces the phenomenon on its own.

19.2 Power Laws and Zipf's Law

In historical civilizations, city populations obey a power law: if cities are ranked by population, the k-th city has population approximately P_1/k, where P_1 is the largest city's population. This is Zipf's Law.

Theorem 19.2 (Zipf Emergence Condition)**.** *In a 4X world with preferential attachment in trade route formation, the steady-state city population distribution follows a power law with exponent $\alpha \approx 1$.*

Proof sketch. Let $C_i(t)$ be the population of city i at tick t, and let the growth rate be proportional to trade route degree $d_i(t)$: $C_i(t+1) = C_i(t) + \lambda d_i(t)$. Under preferential attachment, a new trade route connects to city i with probability $d_i / \sum_j d_j$. This is the Barabási–Albert mechanism, whose steady-state degree distribution is $P(d) \propto d^{-3}$. Since $C \propto d$, the rank distribution satisfies Zipf's Law with exponent 1. □ □

19.3 Phase Transitions in Civilization

Definition 19.3 (Civilizational Phase Transition)**.** A *phase transition* in world $\mathcal{W}$ occurs at tick t^* when a global order parameter $\Phi(t)$ undergoes a discontinuous jump:

$$\lim_{t \to t^{*-}} \Phi(t) \neq \lim_{t \to t^{*+}} \Phi(t).$$

World Mechanic: Percolation and Empire Collapse

Model imperial territory as a graph $G = (V, E)$ where V is the set of owned tiles and E connects adjacent owned tiles. When military defeats remove tiles, connectivity can drop catastrophically at a threshold removal fraction $p_c \approx 0.407$ (the site percolation threshold for a square lattice). Below p_c, the empire fragments into independent enclaves—a phase transition from unified to fractured.

1. Run 5 complete 100-turn simulations with 4 empires each.
2. Plot city population distributions; test fit to power law.
3. Measure territorial connectivity of each empire at each turn.
4. Identify turns where connectivity drops sharply.
5. Record cultural divergence: measure Hamming distance between technology vectors of different empires over time.

Part VII

History Before State

Objects are not primary. Histories are primary.

Chapter 20

Events Before Objects

The present moment always will have been.

— — after Wittgenstein [26]

Background

Most computational systems begin with objects: data structures occupying memory, accessed by name, modified in place. A city is a record. A tile is an array cell. An empire is a struct. This is the *state-centric* view of computation, and it is so natural to programmers that its presuppositions are rarely examined.

Spherepop begins elsewhere. It takes seriously the observation, developed in process philosophy by Whitehead [24] and in the philosophy of events by Kim [12], that *events are ontologically prior to objects*. A city is not a data structure that occasionally changes. A city is the accumulated history of all events that brought it into being and sustained it: the founding event, the population growth events, the construction events, the siege events. Remove the history and you do not have a city at rest—you have nothing at all. Identity is causal, not structural.

This chapter constructs the event-sourcing substrate that underlies the Spherepop philosophy, proves its equivalence to the state-based model developed in Chapters 3–5, and then demonstrates why the event model is strictly *richer*: it preserves information that the state model discards.

20.1 The Event-Centric World Model

Definition 20.1 (Event)**.** An *event* is an irreversible world operation. Formally, an event ε is a morphism

$$\varepsilon : X \to X'$$

where X and X' are world option spaces (sets of admissible future states), with the constraint that $X' \subseteq X$ (events never expand the option space).

Definition 20.2 (World History)**.** The *history* of a world from initial configuration X_0 is the composite morphism

$$H = \varepsilon_n \circ \varepsilon_{n-1} \circ \cdots \circ \varepsilon_1 : X_0 \to X_n.$$

20.2 Event Sourcing as Information Preservation

Theorem 20.3 (Events Preserve More Than State)**.** *Distinct histories can produce identical present states, but an event log distinguishes them while a state snapshot does not.*

Proof. Consider two histories on a minimal world with one tile T having ownership register $O \in \{0, 1\}$:

$$\mathcal{L}_1 = (\texttt{Claim}(T, e_1))$$
$$\mathcal{L}_2 = (\texttt{Claim}(T, e_1),\ \texttt{Release}(T),\ \texttt{Claim}(T, e_1)).$$

Both produce final state $O = 1$. A snapshot contains only $O = 1$; it cannot distinguish $\mathcal{L}_1$ from $\mathcal{L}_2$. But the logs differ: $|\mathcal{L}_1| = 1 \neq 3 = |\mathcal{L}_2|$. The diplomatic consequence (empire e_1 has a history of releasing territory) is recoverable from $\mathcal{L}_2$ but from neither snapshot. □ □

20.3 Historical Identity

Definition 20.4 (Historical Identity)**.** Two entities A and B are *historically identical* if and only if their event logs are identical:

$$A \equiv_H B \iff \mathcal{L}(A) = \mathcal{L}(B).$$

Corollary 20.5. *Historical identity is strictly finer than state identity:* $A \equiv_H B \Rightarrow \text{state}(A) = \text{state}(B)$*, but the converse fails.*

Implementation

```
bubble EventSourcedWorld {
    latch event_log[MAX_EVENTS][128]
    latch log_head[16]

    bubble QueryOwner {
        inputs:  x[8], y[8]
        output:  owner[4]

        let i = log_head
        owner = 0
        while i > 0 do
            i = i - 1
            if event_log[i].type = CLAIM
                ∧  event_log[i].tile_x = x
                ∧  event_log[i].tile_y = y then
                owner = event_log[i].actor
                i = 0
            end
            if event_log[i].type = RELEASE
                ∧  event_log[i].tile_x = x
                ∧  event_log[i].tile_y = y then
                owner = 0
                i = 0
            end
        end
    }

    bubble AppendEvent {
        inputs:  event[128]
        when log_head < MAX_EVENTS do
            event_log[log_head] := event
            log_head := log_head + 1
```

```
33            end
34        }
35    }
```

Listing 20.1: Append-only event sourcing in SPHEREPOP

Perspective

The event-sourcing architecture embodies Whitehead's process philosophy [24]: the fundamental furniture of the universe consists not of enduring substances but of momentary occasions of experience, each of which inherits and transforms the actual world of its causal ancestors. The append-only log is the formal analogue of Whitehead's creative advance: once an event is written, it cannot be unwritten. The past is immutable; only the future remains open.

1. Replace all mutable `owner` registers with an event log.
2. Implement `QueryOwner` by log scan.
3. Implement `QueryHistory`: return all ownership events for a given tile.
4. Implement `TrustFromHistory`: count violated treaties in the log to compute diplomatic trust.
5. Design a hybrid cache: register snapshot for speed plus full log for auditability.

Chapter 21

Nested Worlds and Bubble Scope

A battle exists inside a war. A war exists inside a civilization. A civilization exists inside a planetary history.

Background

Computation is fundamentally about scope: what a computation can see, what it can modify, and what is hidden from it. In lambda calculus [3], scope is defined by λ-abstraction. In SPHEREPOP, scope is defined by *containment*: a bubble's interior is hidden from everything outside its boundary. A bubble boundary is a *distinction*—it separates the world into inside and outside.

For world simulation, nested scope has immediate strategic significance. A city's internal economy is hidden from the empire's strategic AI, which sees only city outputs. The empire's military strategy is hidden from the world's ecological processes, which see only physical effects. Nested scope is what allows a world to be simultaneously local and global: each bubble is complete in itself and also a component of something larger.

21.1 The Scope Hierarchy

Definition 21.1 (Scope Hierarchy)**.** A *scope hierarchy* is a rooted tree $\mathcal{T} = (N, E, r)$ where each node $n \in N$ is a bubble, edges represent containment, and r is the root (the world).

Definition 21.2 (Lexical Scope in Spherepop)**.** A bubble B has *lexical access* to its own internal registers, the input ports declared by its parent, and any bubble explicitly passed as an argument. B does not have access to the internals of its siblings or ancestors beyond declared ports.

Theorem 21.3 (Scope Compositionality)**.** *In a scope hierarchy $\mathcal{T}$, if each bubble correctly implements its specification independently of all sibling and ancestor internals, then any composite bubble formed by connecting bubbles in $\mathcal{T}$ also correctly implements its specification.*

Proof. By structural induction on $\mathcal{T}$. Base case: a leaf (primitive) bubble is correct by assumption. For an interior node B with children $B_1, \ldots, B_k$: by the induction hypothesis, each B_i correctly implements its specification. The wiring of B connects output ports of some children to input ports of others, respecting interface contracts. Since each B_i's behavior depends only on its declared inputs (by lexical scope) and those inputs are correctly produced (by induction), B's output is correctly determined. □ □

Implementation

```
bubble World {
    bubble Empire {
        bubble City {
            bubble District {
                latch commerce[8]
                bubble Market {
                    inputs:  supply[8], demand[8]
                    output:  price[8]
                    price = demand / (supply + 1)
                }
            }
            -- City sees district outputs only
            inputs:   district_output[16]
            output:   city_gold[8], city_food[8]
        }
    }
}
```

Listing 21.1: Nested city-district economy

Perspective

Nested scope is the computational realization of Leibniz's monads: each bubble is a "windowless" computation that sees the world only through its declared ports. This enforced opacity is the source of modularity. Without scope, every bubble would need to know everything, and the simulation would collapse into a single undifferentiated process. With scope, each layer can be designed, verified, and replaced independently—and the whole can be greater than the sum of its parts.

1. Implement a three-level hierarchy: `World` ⊃ `Empire` ⊃ `City` ⊃ `District`.
2. Each district manages labor allocation among farming, mining, and commerce.
3. Cities see only district output totals.
4. Empires see only city output totals.
5. Implement cross-city trade at the empire level accessing only declared city output ports.

Chapter 22

The Pop Operator

To choose is to exclude. The value of a decision lies precisely in what it forecloses.

Background

The name "Spherepop" is not decorative. The *pop* is the primitive act of the system: the irreversible commitment that collapses an ambiguous set of futures into a single actualized one. Before a Settler unit settles, a city could arise anywhere. After the settlement event fires, one location is actualized and all others are permanently excluded. The pop is the moment of decision.

Kierkegaard's "either/or," Sartre's radical freedom, Heidegger's *Entschlossenheit*, and Badiou's event [2] all circle the same structure: the moment at which the indeterminate becomes determinate, irreversibly. What Spherepop contributes is a precise formal language for this moment, grounded in the mathematics of option spaces and contraction operators.

The pop is also the moment where games acquire meaning. A game in which no decision is irreversible is a game without stakes. The sense of weight that a well-designed 4X game produces—the feeling that founding this city here, in this moment, with these resources, commits you to a trajectory from which there is no return—is the phenomenological signature of the pop operator.

22.1 Option Spaces and Contraction

Definition 22.1 (Option Space)**.** An *option space* Ω is a set of admissible future world trajectories. Each element $\omega \in \Omega$ is a maximal consistent extension of the current world state to all future ticks.

Definition 22.2 (Pop Operator)**.** A *pop* is a function $\mathbf{P}_\varepsilon : \Omega \to \Omega'$ associated to an event ε, satisfying: (1) **Contraction:** $\Omega' \subseteq \Omega$. (2) **Irreversibility:** There is no operator $\mathbf{Q}$ such that $\mathbf{Q} \circ \mathbf{P}_\varepsilon = \mathrm{id}_\Omega$. (3) **Commitment:** Every $\omega' \in \Omega'$ is consistent with ε having occurred.

Theorem 22.3 (Strict Contraction)**.** *If event ε is non-trivial, then $|\Omega'| < |\Omega|$.*

Proof. Let $\omega^* \in \Omega$ be a trajectory inconsistent with ε (exists by non-triviality). Then $\omega^* \notin \Omega'$ by the commitment condition, so $\Omega' \subseteq \Omega \setminus \{\omega^*\}$, giving $|\Omega'| \leq |\Omega| - 1 < |\Omega|$. □ □

Corollary 22.4 (Historical Entropy is Non-Increasing)**.** *Define $S(\Omega) = \log_2 |\Omega|$. For any non-trivial pop, $S(\Omega') < S(\Omega)$.*

This connects the Spherepop pop to Landauer's principle [13]: every irreversible bit-erasure has a thermodynamic cost.

Implementation

```
bubble PopSettle {
    inputs:
        empire_id[4], tile_x[8], tile_y[8], settler_id[8]

    let allowed = SettlePermission(empire_id, tile_x, tile_y)

    when allowed = 1 do
        -- 1. Remove settler (irreversible)
        world.units[settler_id].alive := 0
        -- 2. Found city (irreversible event)
        AppendEvent(FOUND_CITY, empire_id, tile_x, tile_y, tick)
        -- 3. Claim tile (irreversible)
        AppendEvent(CLAIM, empire_id, tile_x, tile_y, tick)
    end
}
```

Listing 22.1: Pop operator for city settlement

Perspective

The pop operator formalizes what game designers call "meaningful choice." A choice is meaningful precisely when it irreversibly constrains the future. Games without irreversible decisions—where every action can be undone by the save/load cycle—have lower strategic depth not because they lack complexity but because they lack *commitment*. The pop is the computational measure of decisional weight. Badiou [2] argues that an event is precisely that which ruptures the existing order and forces a new fidelity. The settlement pop is a miniature Badiousian event.

1. Implement the full `PopSettle` bubble with precondition checking and event logging.
2. Build a `PopDeclareWar` bubble that excludes all trajectories of peaceful coexistence with the target.
3. Build a `PopChooseTech`: committing research to one technology temporarily forecloses alternatives.
4. Measure the option space reduction (entropy decrease) for each pop type by counting reachable states before and after.

Chapter 23

Merge and Collapse

Background

If the pop contracts an option space by committing to a specific event, *merge* operates on entities within the world: combining two structures into one, collapsing distinctions that were previously maintained. Empires merge territories. Cultures merge vocabularies. Trade networks merge into single markets. These are not additions; they are *compressions*—operations that identify previously distinct entities and represent their union as a single canonical form.

The literature on Conflict-free Replicated Data Types (CRDTs) in computer science [15] formalizes this idea: a data structure supports merge if it forms a join-semilattice. Spherepop's merge operator inherits this structure and extends it to the richer domain of world entities.

23.1 The Merge Semilattice

Definition 23.1 (Merge Semilattice)**.** A set $\mathcal{R}$ with binary operator μ is a *merge semilattice* if μ is commutative, associative, and idempotent.

Theorem 23.2 (Territory Merge is a Semilattice)**.** *Let $\mathcal{R} = \mathcal{P}(Tiles)$ with $\mu(A, B) = A \cup B$. Then $(\mathcal{R}, \mu)$ is a merge semilattice.*

Proof. Set union is commutative ($A \cup B = B \cup A$), associative ($(A \cup B) \cup C = A \cup (B \cup C)$), and idempotent ($A \cup A = A$). □ □

23.2 Collapse and Canonicalization

Definition 23.3 (Collapse Operator)**.** A *collapse operator* $\kappa : \mathcal{R} \to \widehat{\mathcal{R}}$ maps each entity to its canonical representative, satisfying:

$$\kappa(\mu(A, B)) = \kappa(\mu(\kappa(A), \kappa(B))).$$

Example 23.4 (Cultural Assimilation)**.** Empire e_1 has culture set $C_1 = \{\text{Latin}, \text{Greek}\}$ and empire e_2 has $C_2 = \{\text{Greek}, \text{Aramaic}\}$. Cultural merge: $C_{\text{merged}} = \{\text{Latin}, \text{Greek}, \text{Aramaic}\}$. Collapse: κ selects the dominant culture (highest population weight), producing a canonical representative.

Project: Cultural Assimilation Mechanics

1. Define culture as a Boolean vector over 16 cultural traits.

2. Implement `CultureMerge` as set union on trait vectors.
3. Implement `CultureCollapse`: return dominant culture.
4. Build `AssimilationPressure`: tiles within a cultural sphere gradually adopt dominant traits.
5. Simulate two civilizations meeting and assimilating over 30 turns.

Chapter 24

History as Identity

Background

In Chapter 19, we established that events are ontologically prior to objects. In Chapter 21, we showed that events irreversibly constrain the future. Now we draw out the identity-theoretic consequences: if what an entity *is* is its history, then two entities are the same only if they share the same history. This is a radical thesis—far stronger than extensional identity (same current properties) or modal identity (same counterfactual profile). But it is the thesis that Spherepop's event-sourcing architecture implicitly commits to, and it has significant consequences for persistence, replication, and memory in simulations.

24.1 Provenance and Lineage

Definition 24.1 (Provenance)**.** The *provenance* of entity A at time t is the ordered sequence of all events that produced A:

$$\text{prov}(A, t) = (\varepsilon_1^A, \varepsilon_2^A, \ldots, \varepsilon_{n(t)}^A).$$

Theorem 24.2 (Provenance Determines State)**.** *In a deterministic simulation,* $\text{prov}(A, t)$ *together with the initial conditions* X_0 *uniquely determines* $\text{state}(A, t)$.

Proof. By induction on $n(t)$. Base case $n(t) = 0$: state equals A's initial state, determined by X_0. Inductive step: by hypothesis $\text{prov}(A, t')$ determines $\text{state}(A, t')$ for the tick t' preceding event ε_{k+1}^A, and the deterministic transition function δ maps $(\text{state}(A, t'), \varepsilon_{k+1}^A)$ to $\text{state}(A, t)$. □ □

Implementation

```
bubble TimelineReconstructor {
    inputs:  entity_id[8], target_tick[16]
    output:  reconstructed_state[512]

    let state = initial_state(entity_id)
    let i = 0
    while i < global_log_head do
        if event_log[i].actor = entity_id
            ∨  event_log[i].target = entity_id then
            if event_log[i].tick <= target_tick then
                state = apply_event(state, event_log[i])
            end
```

```
        end
        i = i + 1
    end
    reconstructed_state = state
}
```

Listing 24.1: Timeline reconstruction

1. Implement `TimelineReconstructor` for cities.
2. Build a `CityBiography`: given a city ID, produce a human-readable chronicle of all events in the city's history.
3. Implement lineage tracking: captured and refounded cities inherit the old city's event log as a prefix.
4. Build a divergence metric: compute the tick at which two empires first diverged and the Hamming distance between their current technology and culture vectors.

Part VIII

Geometric Computation

Space is not a container. It is a constraint structure.

Chapter 25

Geometry as Computation

Background

Throughout this book, geometry has appeared implicitly: in tile adjacency, vision radii, path-checking of trade routes, and percolation thresholds of territorial connectivity. We now make geometry explicit. The spatial structure of the world is not merely a rendering concern—it is a computational structure that determines what operations are possible, what their costs are, and what emergent patterns can arise.

In the Spherepop framework, geometry is understood as a *constraint structure on option spaces*. Mountains are not merely visually distinct from plains—they constrain movement, visibility, and resource extraction in ways that structurally shape the space of possible histories.

25.1 Tile Graphs and Spatial Topology

Definition 25.1 (Tile Graph)**.** The *tile graph* of a world is the undirected graph $G = (V, E)$ where V is the set of passable tiles and $(u, v) \in E$ iff u and v are adjacent.

Theorem 25.2 (Connectivity and Reachability)**.** *A unit at position u can reach position v in at most k turns (with movement cost 1 per tile) if and only if the graph distance $d_G(u, v) \leq k$.*

Proof. The unit's reachable set after k turns is exactly the ball of radius k in the tile graph: $B_k(u) = \{v : d_G(u, v) \leq k\}$. This follows from BFS on G. □ □

25.2 Region-Based Territorial Reasoning

Definition: Territorial Compactness

The *compactness* of a territory T is

$$\text{compact}(T) = \frac{|V(T)|^2}{|E(T)|}.$$

High compactness means short internal travel distances; low compactness means a fragile, extended territory.

1. Implement `ConnectedComponents` for each empire.
2. Implement `Compactness` per turn.
3. Build `CutVertexDetector`: tiles whose removal disconnects an empire's territory.
4. Visualize compactness and connectivity over a 50-turn simulation.

Chapter 26

Optionality and Entropy

Background

Chapter 21 introduced the pop as a contraction of the option space. Now we develop optionality as a *positive* quantity—a measure of a civilization's strategic power. A civilization with high optionality has many accessible futures; one with low optionality is locked in. This connects Spherepop to the theory of real options in economics and to Shannon's information theory [19]: the same formula measures uncertainty in all contexts.

26.1 The Optionality Field

Definition 26.1 (Optionality)**.** The *optionality* of civilization C at tick t is

$$\Phi_C(t) = \log_2 |\mathcal{T}_C(t)|$$

where $\mathcal{T}_C(t)$ is the set of all trajectories consistent with C's event history up to t.

In practice we decompose this into domain-specific components:

Definition 26.2 (Accessibility Field)**.**

$$\Phi_C(t) = \sum_d w_d \log_2 A_d(t)$$

where $A_d(t)$ counts accessible actions in domain d (military, economic, scientific, ecological, diplomatic) and w_d are weights.

Theorem 26.3 (Civilizational Collapse Criterion)**.** *A civilizational collapse occurs at tick t^* if $\frac{d}{dt}\Phi_C\big|_{t=t^*} < -\theta$ for some threshold $\theta > 0$, sustained over a window Δt: rapid, sustained loss of accessible futures.*

Derivation. The discrete derivative $\Delta\Phi_C(t) = \Phi_C(t+1) - \Phi_C(t)$ captures per-turn change. Collapse is the regime where adverse events dominate: the civilization loses more future options per turn than it gains. The threshold θ calibrates severity; Δt prevents classifying temporary setbacks as collapses. □ □

Project: Optionality Metrics for AI Decision-Making

1. Implement `OptionalityMeter` for all four empires.
2. Record optionality at each turn over a 50-turn simulation.
3. Build a `MaxOptionalityAI`: select actions that maximize Φ at the next tick.

4. Compare against a greedy resource-maximizing AI over 20 runs.
5. Identify how far in advance Φ begins declining before elimination.

Chapter 27

Trajectory Collapse

Background

The previous chapter measured optionality as a scalar. This chapter develops the geometric structure of trajectory space: how ambiguous futures form a probability simplex, how strategic uncertainty is distributed over it, and how the pop operator effects a geometric collapse. This connects Spherepop to Bayesian inference and Shannon entropy [19]: the same formula— $H = -\sum p_i \log p_i$—measures uncertainty in all three contexts.

27.1 Hypothesis Manifolds

Definition 27.1 (Trajectory Entropy)**.** Given scenarios $\mathcal{H} = \{h_1, \ldots, h_n\}$ with probabilities p_i, the *trajectory entropy* is:

$$H_t = -\sum_{i=1}^{n} p_i \log_2 p_i.$$

Theorem 27.2 (Pop Reduces Trajectory Entropy)**.** *For any non-trivial pop eliminating scenario h_k:*

$$H_{t+1} \leq H_t$$

with equality iff $p_k = 0$.

Proof. Let $p' = (p'_1, \ldots, p'_n)$ be the post-pop distribution: $p'_k = 0$ and $p'_i = p_i/(1 - p_k)$ for $i \neq k$. By the data-processing inequality, eliminating an outcome and renormalizing never increases entropy: $H' \leq H$. Equality holds iff $p_k = 0$ (the pop changes nothing). □ □

1. Implement a `BranchingSimulator`: at decision points, run k parallel branches with different choices.
2. Measure trajectory entropy at each branch point.
3. Build a `ScenariosDatabase`: store and compare final states of all branches.
4. Visualize the branching tree annotated with entropy at each node.
5. Identify which decisions cause the largest entropy reduction.

Part IX

Category Theory and World Construction

A morphism is not a transformation of objects. It is a relationship that composes.

— — after Mac Lane [14]

Chapter 28

Morphisms and Historical Worlds

Background

Category theory [14] is the mathematics of structured composition. It asks not "what is this thing?" but "how does this thing relate to other things of the same kind?" For world simulation, we want to understand transformations between worlds—how a world at tick t relates to the world at tick $t+1$, how one empire's history relates to another's, how a local economy relates to the global one.

In this chapter, we construct the category **Hist** whose objects are option spaces and whose morphisms are events. We show that historical composition is categorical composition, derive consequences for victory conditions and technology trees, and explore the connection to Girard's linear logic [9]—a logic where resources are consumed rather than duplicated, mirroring the irreversibility of in-game actions.

28.1 The History Category

Definition 28.1 (The Category **Hist**)**.** **Hist** has option spaces as objects; morphisms $\mathrm{Hom}(\Omega_i, \Omega_j)$ are events $\varepsilon : \Omega_i \to \Omega_j$; composition is sequential occurrence; identity is the null event.

Theorem 28.2 (**Hist** is a Category)**.** *The structure* **Hist** *satisfies the category axioms.*

Proof. *Identity:* $\varepsilon \circ \mathrm{id}_{\Omega_0} = \varepsilon$ and $\mathrm{id}_{\Omega_1} \circ \varepsilon = \varepsilon$ because the null event leaves the option space unchanged. *Associativity:* Both $(\varepsilon_3 \circ \varepsilon_2) \circ \varepsilon_1$ and $\varepsilon_3 \circ (\varepsilon_2 \circ \varepsilon_1)$ apply events in order $\varepsilon_1, \varepsilon_2, \varepsilon_3$; sequential composition is associative. □ □

Theorem 28.3 (Victory Condition as Natural Transformation)**.** *A monotone victory condition V is a natural transformation $V : F \Rightarrow \mathbf{2}$ from the state functor to the constant functor on $\{0, 1\}$.*

Proof sketch. At each option space Ω, $V_\Omega : F(\Omega) \to \{0, 1\}$ evaluates the victory predicate. Naturality (once true, stays true) requires $V_{\Omega'} \circ F(\varepsilon) = V_\Omega$ for all events ε. This holds when victory is terminal—precisely the design criterion for valid victory conditions. □ □

Remark 28.4 (Connection to Linear Logic). Girard's linear logic [9] treats propositions as resources that are consumed in inference. The linear implication $A \multimap B$ means "given A, produce B, consuming A." This mirrors Spherepop events exactly: an event $\varepsilon : \Omega \to \Omega'$ *consumes* option space Ω (it can never happen again in the same form) and produces Ω'. The pop operator is a linear implication. The technology tree is a derivation in linear logic.

1. Implement the event composition operator $\varepsilon_2 \circ \varepsilon_1$ for pairs of world events.
2. Verify associativity: confirm that three events applied in any associative grouping produce the same world state.
3. Build a `VictoryChecker` and verify naturality (once true, stays true).
4. Implement a `HistoryFunctor`: given an event log, compute the sequence of world states it produces.

Chapter 29

Sheaves and Local Coherence

Background

The world is large and agents within it are local. Each empire knows its territory and the rest poorly. Each city manages its own economy. Each unit sees only its vision radius. Yet the world must be globally coherent. The mathematical structure that captures this tension is a *sheaf* [20]: it assigns to each region an information structure such that sections on overlapping regions are consistent. Two overlapping regions produce the same information on their overlap. This is the mathematical model of distributed knowledge with local-global coherence.

29.1 The Information Sheaf

Definition 29.1 (Information Sheaf)**.** An *information sheaf* $\mathcal{F}$ assigns to each region U a set $\mathcal{F}(U)$ of local sections (information knowable within U), and to each inclusion $V \subseteq U$ a restriction map $\rho_{UV} : \mathcal{F}(U) \to \mathcal{F}(V)$, satisfying: (1) **Locality:** sections that agree on all cover elements are equal; (2) **Gluing:** compatible local sections assemble into a unique global section.

Theorem 29.2 (Gluing Lemma as World Coherence)**.** *In a 4X world where each empire maintains local information, consistent local information can be assembled into a globally coherent world state if and only if the information structure forms a sheaf.*

Proof. The gluing axiom is precisely the condition needed: each empire's local information is internally consistent and agrees on shared borders. The gluing lemma provides a unique global section—a globally coherent world state—restricting to each empire's local information. Locality guarantees uniqueness. □ □

1. Implement a model where each empire stores state only for tiles in its explored region.
2. Implement `BorderNegotiation`: reconcile conflicting information about border tiles between two empires.
3. Implement `IntelligenceFusion`: combine spy reports to produce a coherent estimated world state.
4. Build a `GlobalStateReconstructor`: assemble the full world state from all empires' local information.

Chapter 30

Semantic Infrastructure

Background

The world simulation has become a *semantic operating system*: a system in which the meaning of entities is determined by their relationships and histories, not by intrinsic properties. A city is meaningful not because of its data structure but because of its trade relationships, cultural influence, military history, and position in the civilization's narrative.

This chapter makes the parallel explicit between Spherepop's event infrastructure and the event-sourced architectures of distributed computer systems: Git version control, Apache Kafka, distributed ledgers [15, 16]. Each solves the same fundamental problem: how to maintain coherent state across multiple participants who share a history but operate locally. The solution is always the same as in Spherepop: an append-only event log with defined merge and replay semantics.

30.1 World Simulation as Distributed System

Definition 30.1 (Semantic Infrastructure)**.** A *semantic infrastructure* for world $\mathcal{W}$ is $\mathcal{SI} = (\mathcal{L}, \mathcal{R}, \mathcal{M}, \mathcal{A})$ where $\mathcal{L}$ is the append-only event log, $\mathcal{R}$ is the replay function, $\mathcal{M}$ is the merge function, and $\mathcal{A}$ is the authority protocol for conflict resolution.

Theorem 30.2 (Semantic Infrastructure Completeness)**.** *A semantic infrastructure $\mathcal{SI}$ is* complete *if for any two logs $\mathcal{L}_1, \mathcal{L}_2$ consistent with a common prefix $\mathcal{L}_0$, $\mathcal{M}(\mathcal{L}_1, \mathcal{L}_2)$ is well-defined and contains $\mathcal{L}_0$ as a prefix.*

Proof. $\mathcal{M}(\mathcal{L}_1, \mathcal{L}_2)$ is defined as the interleaving of suffix events $\mathcal{L}_1 \setminus \mathcal{L}_0$ and $\mathcal{L}_2 \setminus \mathcal{L}_0$, with conflicts resolved by $\mathcal{A}$. The result contains $\mathcal{L}_0$ as a prefix by construction. □ □

1. Implement a split-log architecture: each empire maintains its own local event log, synchronized with the global log each turn.
2. Implement `LogMerge`: reconcile two empires' logs after simultaneous play.
3. Implement the authority protocol: conflicts resolved by timestamp, then empire ID.
4. Test coherence: two simultaneously-acting empires' logs merge into a consistent world state.

Part X

The Philosophy of Computational Worlds

The limits of my language mean the limits of my world.

— — Wittgenstein [26]

Chapter 31

Meaning as Use

Background

Wittgenstein's later philosophy [26] proposes that the meaning of a word is its use in a language game—a structured social practice with rules, moves, and purposes. To understand "castle" in chess, you do not need a theory of reference: you need to know how the castle moves. To understand "city" in a 4X game, you need to know what cities produce, what they cost, and how they interact with other entities.
This chapter applies the Wittgensteinian framework to 4X world simulation. The meaning of in-game entities is constituted by the rule systems that govern them. Cultural evolution in the simulation is the evolution of language games. The diversity of civilizations corresponds to the diversity of mutually intelligible but distinct language games. A well-designed 4X world does not merely represent a civilization: it instantiates a form of life.

31.1 Game Rules as Semantic Axioms

Definition 31.1 (Language Game)**.** A *language game* $\mathcal{G} = (\mathcal{M}, \mathcal{R}, \mathcal{P})$ consists of moves $\mathcal{M}$, rules $\mathcal{R}$ (constraints on legal move sequences), and purposes $\mathcal{P}$ (goals that make sequences meaningful).

Proposition 31.2 (Game Rules as Semantics)**.** *The meaning of an in-game entity E is the set of rules that govern E's interactions: how it is produced, consumed, transformed, and used. Change the rules and you change the meaning, even if the data structure is identical.*

World Mechanic: Cultural Semantic Drift

As civilizations diverge in technology and diplomacy, their rule systems diverge. Two civilizations that cannot trade, communicate, or convert each other occupy different language games: they have lost the shared practices that would make mutual understanding possible. This is Wittgenstein's remark operationalized: "If a lion could talk, we could not understand him."

Project: Cultural Semantics in Diplomacy

1. Implement `DiplomaticIntelligibility` as a function of shared technologies, cultural traits, and religion.
2. Build a rule that proposals require minimum intelligibility to be parseable by the target empire.
3. Implement a `CulturalMissionary` unit that spreads traits, increasing intelligi-

bility.

4. Simulate two isolated civilizations developing for 40 turns then making first contact. Measure the intelligibility gap.

Chapter 32

Civilization as Computation

Background

We have arrived at the final synthesis. The entire book has been building toward a single claim: that a civilization is a computation. Not merely that a civilization can be *modeled* by a computation—that is the weak claim of simulation—but that civilization *is*, in its essential structure, a distributed information-processing system: a Turing-complete process that takes environmental inputs, maintains internal state, and produces outputs that modify its environment.

This claim has a precise meaning and a precise proof. We will show that the 4X world simulation implemented in this book is Turing-complete, and therefore that civilization—the system it models—shares that property. The philosophical consequence is that civilization and computation are not analogous: they are the same thing at different scales of description, connected by the abstraction ladder we have climbed throughout this book.

32.1 The Civilization as a Turing Machine

Theorem 32.1 (Civilization is Turing-Complete)**.** *The 4X world simulation implemented in this book is Turing-complete: it can simulate any Turing machine [21].*

Construction. We construct a Turing machine $\mathcal{TM}$ within the 4X world:

- **Tape:** A row of tiles; each tile's terrain type encodes a tape cell symbol $\in \{0, 1, \text{blank}\}$.
- **Head:** A special unit (the "reader unit") whose position encodes the head position.
- **State register:** The reader unit's FSM state encodes the Turing machine's control state $q \in Q$.
- **Transition:** Each turn, the reader unit (i) reads the terrain type of its current tile; (ii) looks up the transition $(q, \text{symbol}) \to (q', \text{symbol}', \text{dir})$ from a table in the technology tree; (iii) writes the new symbol by changing terrain type; (iv) moves left or right by one tile.

This faithfully implements any Turing machine with tape alphabet $\{0, 1, \text{blank}\}$ and finite state set Q. Since any Turing machine is simulated, the simulation is Turing-complete. □ □

Perspective

Turing-completeness means that the internal logic of a civilization cannot, in general, be predicted without running the civilization. The future is genuinely open: there is no shortcut to history. This is Turing's halting problem applied to civilization. Whether an empire eventually achieves victory, collapses, or cycles indefinitely cannot be determined from initial conditions alone by any finite algorithm. The only way to know a world's history is to let it run.

Wiener's cybernetics [25] observed that control and communication in animals and machines obey the same principles. The Turing-completeness theorem is the strongest version of this claim: civilization and machine are not merely analogous but computationally equivalent.

1. Implement the Turing machine construction: tape tiles, reader unit, transition table.
2. Program the reader unit to compute binary addition; verify the output.
3. Implement `AutopoiesisCheck`: detect when an empire has reorganized itself without external input, following Varela et al. [22].
4. Run 10 simulations with random initial conditions; identify which exhibit sustained self-organization.

Chapter 33

The Finite World

The world is everything that is the case.

— — Wittgenstein, *Tractatus* [26]

Background

Every world in this book is finite. The map has boundaries. The technology tree has leaves. The turn counter has a maximum. This finitude is not a deficiency—it is what makes the world computable. An infinite world cannot be serialized, replayed, or proven correct. Finitude is the price of intelligibility.

But finitude also means that every world ends. Civilizations collapse or are conquered. Resources deplete. The final turn is reached. This last chapter examines what finitude means philosophically—for the simulation, for the civilizations within it, and for the beings outside it who have watched it run. It is the proper ending of a book that began with logic gates and arrives, finally, at the question of what all this construction means.

33.1 Irreversibility and Path Dependence

Definition 33.1 (Path Dependence)**.** A world $\mathcal{W}$ exhibits *path dependence* at tick t if there exist two event histories $\mathcal{L}_1 \neq \mathcal{L}_2$ consistent with the same initial conditions such that $\text{state}(\mathcal{W}, \mathcal{L}_1, t) \neq \text{state}(\mathcal{W}, \mathcal{L}_2, t)$.

Theorem 33.2 (All Non-Trivial Worlds are Path Dependent)**.** *Any world with at least one irreversible event exhibits path dependence at all subsequent ticks.*

Proof. Let ε be an irreversible event. Let $\mathcal{L}_1 = (\varepsilon)$ and $\mathcal{L}_2 = ()$ (empty log). By irreversibility, $\text{state}(\mathcal{L}_1) \neq \text{state}(\mathcal{L}_2)$ at tick $t \geq t_\varepsilon$. □ □

33.2 Historical Lock-In

Definition 33.3 (Historical Lock-In)**.** A civilization is *locked in* at tick t if $\Phi_C(t) \leq \theta_{\min}$ for some minimum viable threshold $\theta_{\min}$: all strategic variation has been exhausted.

Proposition 33.4 (Lock-In is Generically Monotone)**.** *A civilization that has lost all strategic flexibility cannot, by definition, take the actions that would restore flexibility.*

33.3 The Final Perspective

Perspective

This book began with a logic gate. It ends with a civilization.

Between those two points lies a philosophical ascent that mirrors, in miniature, the ascent of complexity in the actual universe: from the simplest distinctions through the accumulation of memory, the emergence of space, the arising of agency, the construction of economies and treaties and histories, to the moment when the whole system becomes so intricate that its future cannot be predicted from its present.

What have we built? Not merely a game. We have built a system in which local rules, applied consistently across space and time, produce global properties that no individual rule anticipated: Zipf-distributed city sizes, percolation thresholds in territorial connectivity, diplomatic trust dynamics resembling reputation equilibria, cultural drift resembling linguistic evolution. We have built a small model of history—not history with specific content, but history as a *form*: the form of irreversible, accumulating, locally-caused, globally-surprising change.

Heidegger [10] observed that the question of Being is the question every other question presupposes. For computational worlds, the question of Being is the question we have answered: what does it mean for an entity to *be* in a computational world? It means: to have a history. To have been caused. To be maintained by the persistence of its effects. To be, in the Spherepop sense, an append-only log that has not yet been closed. The world runs. The log grows. The future remains, for now, open.

Grand Final Project.

Build a complete, persistent, historically self-aware 4X civilization simulator incorporating every system from Chapters 1–31. Requirements:

1. A 32×32 world with procedural terrain and ecology.
2. Four empires with distinct cultural trait vectors and starting technologies.
3. Full event sourcing: all state derived from event logs.
4. Optionality tracking for all empires at every turn.
5. A narrative generator producing a structured chronicle.
6. A save/load/replay system with full integrity verification.
7. A post-game analysis: which decisions caused the largest option-space contractions? Which empire maintained optionality longest? What was the Zipf exponent of the final city distribution? When did each empire approach lock-in?

A student who completes this project has not merely learned to program a game. They have built a machine for generating history, and in doing so, have understood something about what history is.

Chapter A

Spherepop Language Reference

This appendix provides a complete reference for the SPHEREPOP notation used throughout the book.

A.1 Primitive Bubbles

Name	Signature	Semantics
NAND	$(a, b) \to \texttt{out}$	$\texttt{out} = \neg(a \wedge b)$
DFF	$(\texttt{in}) \to \texttt{out}$	$\texttt{out}(t) = \texttt{in}(t-1)$
MUX	$(a, b, \texttt{sel}) \to \texttt{out}$	$\texttt{out} = a$ if $\texttt{sel} = 0$ else b
DMUX	$(\texttt{in}, \texttt{sel}) \to (a, b)$	$(a, b) = (\texttt{in}, 0)$ if $\texttt{sel} = 0$ else $(0, \texttt{in})$

A.2 Syntax Summary

```
bubble BubbleName {
    -- Port declarations
    inputs:  portA, portB[n]    -- scalar or n-bit bus
    output:  portC, portD[m]

    -- State declarations (persistent across ticks)
    latch name[width]

    -- Combinational logic
    name = expression

    -- Sequential logic (executes at tick boundary)
    when condition do
        name := expression      -- := denotes sequential assignment
    end

    -- Iteration
    let i = 0
    while i < n do
        ...
        i = i + 1
    end

    -- Sub-bubble instantiation
    let result = SubBubble(arg1, arg2)
}
```

Listing A.1: Bubble declaration syntax

A.3 Built-in Expressions

Expression	Meaning
`a and b`	Boolean AND
`a or b`	Boolean OR
`not a`	Boolean NOT
`a + b`	Integer addition (modular)
`a - b`	Integer subtraction (clamped at 0)
`a >= b`	Comparison (returns Boolean)
`a[k]`	Bit k of bus a
`a[j..k]`	Bit slice

Chapter B

Truth Tables and Gate Diagrams

B.1 Fundamental Gates

AND

a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

OR

a	b	out
0	0	0
0	1	1
1	0	1
1	1	1

NAND

a	b	out
0	0	1
0	1	1
1	0	1
1	1	0

XOR

a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Chapter C

The 4X Design Pattern Glossary

Agent A bubble with internal state, perception, and action capability. The computational model of a unit, city, or empire.

Boolean permission
A conjunction of conditions that must all hold for an action to be legal. The fundamental structure of game rules.

Bubble The basic computational unit of SPHEREPOP: a bounded, ported, composable entity that simultaneously models data, process, and environment.

Cellular automaton
A spatial update system where each cell updates based on its local neighborhood. The computational model of physical laws operating on the world map.

Event log
An append-only trace of world state transitions. The computational model of history.

Fog of war
The visibility restriction that limits an agent's perception to explored and currently visible tiles. Implemented as a pair of Boolean registers per tile.

Latch / Register
A persistent storage element, built from feedback circuits. The fundamental implementation of world memory.

Technology DAG
A directed acyclic graph of Boolean predicates representing technological dependencies. Prerequisite checking is conjunction over parent predicates.

Trade route
A persistent path signal between two city bubbles. Disruption is implemented as path invalidation.

Treaty A shared mutable bubble between two empire agents, storing diplomatic state.

Victory condition
A global Boolean predicate over the entire world state. The terminal condition of a simulation run.

Chapter D

Formal Bubble Semantics

A bubble is a stateful morphism: it transforms inputs into outputs while maintaining internal memory.

The preceding appendices treated bubbles operationally. This appendix gives the algebraic definition that grounds the entire Spherepop architecture.

D.1 Bubbles as Mealy Machines

Definition D.1 (Bubble)**.** A *Spherepop bubble* is a tuple

$$B = (I, O, S, \delta, \lambda, s_0)$$

where:

- $I = \mathbb{B}^m$ is the *input port space* (m-bit input vector),
- $O = \mathbb{B}^p$ is the *output port space* (p-bit output vector),
- $S = \mathbb{B}^q$ is the *internal state space* (q-bit state register),
- $\delta : S \times I \to S$ is the *state transition function* (next-state logic),
- $\lambda : S \times I \to O$ is the *output function* (output logic),
- $s_0 \in S$ is the *initial state*.

This is precisely the definition of a *Mealy machine* (finite-state transducer). Mealy machines are the standard model for synchronous sequential digital circuits. Every bubble in Spherepop is a Mealy machine; the entire world is the product of all its bubbles' Mealy machines, synchronized by the turn counter.

Proposition D.2 (Combinational Gates are Bubbles)**.** *A combinational gate (no internal state) is a bubble with $S = \{s_0\}$ (singleton state), $\delta(s_0, i) = s_0$ for all i, and $\lambda(s_0, i) = f(i)$ where $f : I \to O$ is the gate's Boolean function.*

Proposition D.3 (Sequential Memory is a Bubble)**.** *A 1-bit D flip-flop is the bubble $(I, O, \mathbb{B}, \delta, \lambda, 0)$ with $I = O = \mathbb{B}$, $\delta(s, i) = i$, $\lambda(s, i) = s$. The output is the* previous tick's *input, implementing a 1-tick delay.*

D.2 Bubble Composition

Definition D.4 (Sequential Composition)**.** Given bubbles $B_1 = (I_1, O_1, S_1, \delta_1, \lambda_1, s_1^0)$ and $B_2 = (I_2, O_2, S_2, \delta_2, \lambda_2, s_2^0)$ with $O_1 = I_2$, their *sequential composition* $B_2 \circ B_1$ is the bubble

$$(I_1,\ O_2,\ S_1 \times S_2,\ \delta,\ \lambda,\ (s_1^0, s_2^0))$$

where:

$$\delta((s_1, s_2), i) = (\delta_1(s_1, i),\ \delta_2(s_2, \lambda_1(s_1, i)))$$
$$\lambda((s_1, s_2), i) = \lambda_2(s_2, \lambda_1(s_1, i))$$

Theorem D.5 (Composition Associativity)**.** *Sequential bubble composition is associative:* $(B_3 \circ B_2) \circ B_1 = B_3 \circ (B_2 \circ B_1)$.

Proof. Both sides implement the same input-output function: apply B_1's output as B_2's input, then B_2's output as B_3's input. State spaces are isomorphic by associativity of Cartesian product. □ □

D.3 The Category of Bubbles

Definition D.6 (Bubble Category **Bub**)**.** The category **Bub** has:

- **Objects:** port types $\mathbb{B}^n$ for $n \geq 0$.
- **Morphisms:** $\mathrm{Hom}(\mathbb{B}^m, \mathbb{B}^p)$ is the set of bubbles with input space $\mathbb{B}^m$ and output space $\mathbb{B}^p$.
- **Composition:** sequential bubble composition.
- **Identity:** for each $\mathbb{B}^n$, the identity bubble with $\delta = \mathrm{id}$ and $\lambda(s, i) = i$.

Proposition D.7 (**Bub** is a Category)**.** **Bub** *satisfies the category axioms: composition is associative (by Theorem above) and identities exist.*

Remark D.8. The world evolution operator $F_{\mathcal{W}}$ (Chapter 17.1) is an endomorphism in **Bub**: $F_{\mathcal{W}} \in \mathrm{Hom}(\mathbb{B}^N, \mathbb{B}^N)$ where N is the total number of state bits in the world. The world's entire history is the orbit $\{s_0, F_{\mathcal{W}}(s_0), F_{\mathcal{W}}^2(s_0), \ldots\}$ under repeated application of this morphism.

D.4 Connection to Markov Blankets

The Mealy machine formulation makes the Markov blanket structure (Chapter 13) precise:

Proposition D.9 (Bubble Blanket Correspondence)**.** *For bubble* $B = (I, O, S, \delta, \lambda, s_0)$*:*

$$\mathcal{I} = S \quad \textit{(internal state registers)}$$
$$\mathcal{S} = I \quad \textit{(sensory states = input ports)}$$
$$\mathcal{A} = O \quad \textit{(active states = output ports)}$$
$$\mathcal{B} = I \cup O \quad \textit{(the blanket = all ports)}$$
$$\mathcal{E} = \textit{all other bubble states in } \mathcal{W}$$

The state transition $\delta : S \times I \to S$ *updates internal states given blanket input (active inference). The output function* $\lambda : S \times I \to O$ *generates active states from internal states (acting on the world).*

This shows that the bubble definition already encodes the full active inference loop: internal states infer from sensory inputs, and active outputs modify the external world.

Chapter E

Boolean Foundations of Spherepop Gates

E.1 Primitive Binary Algebra

Let $\mathbb{B} = \{0, 1\}$. A Spherepop primitive gate is a morphism $g : \mathbb{B}^n \to \mathbb{B}^m$.

Definition E.1 (Gate Graph)**.** A *Spherepop gate graph* is a finite directed acyclic graph $G = (V, E, \lambda)$ where V is the set of gate instances, E is the set of signal edges, and $\lambda : V \to \mathcal{G}$ assigns a primitive gate type to each node.

Theorem E.2 (Functional Completeness of NAND)**.** *Every Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$ is expressible as a finite composition of NAND gates.*

Proof. It suffices to derive NOT and AND from NAND (since NOT and AND are functionally complete).

$$\text{NOT}(a) = \text{NAND}(a, a).$$
$$\textit{Check:}\ \text{NAND}(0,0) = 1 = \text{NOT}(0);\ \text{NAND}(1,1) = 0 = \text{NOT}(1).$$
$$\text{AND}(a, b) = \text{NOT}(\text{NAND}(a, b)) = \text{NAND}(\text{NAND}(a, b), \text{NAND}(a, b)).$$
$$\textit{Check:}\ \text{AND}(1,1) = \text{NAND}(\text{NAND}(1,1), \text{NAND}(1,1)) = \text{NAND}(0,0) = 1.\checkmark$$

OR follows via De Morgan: $a \vee b = \neg(\neg a \wedge \neg b)$. Since $\{\neg, \wedge\}$ is functionally complete, NAND alone suffices. □ □

E.2 Evaluation Semantics

Theorem E.3 (Evaluation is Well-Defined)**.** *For any acyclic gate graph G, topological evaluation produces a unique output assignment, independent of the choice of topological ordering.*

Proof. A topological ordering exists (since G is acyclic). For any two topological orderings, each gate is evaluated after all its predecessors in both orderings, receiving the same input signals in both, and therefore producing the same output. Uniqueness follows from the determinism of each primitive gate. □ □

Chapter F

Event Histories and Option Spaces

F.1 Historical State Spaces

Definition F.1 (Option Space Dynamics)**.** A world history is the compositional chain $H = \varepsilon_n \circ \cdots \circ \varepsilon_1$ with monotone admissibility: $X_{t+1} \subseteq X_t$.

Theorem F.2 (Monotone Option Space Reduction)**.** *For any non-trivial event sequence of length n: $|X_n| \leq |X_0|$, with strict inequality if at least one event is non-trivial.*

Proof. Induction on n. Base: $|X_0| \leq |X_0|$. Inductive step: assume $|X_k| \leq |X_0|$. By monotone admissibility, $|X_{k+1}| \leq |X_k| \leq |X_0|$. Strict inequality follows from Strict Contraction (Chapter 21) for any non-trivial ε_{k+1}. □ □

F.2 Historical Identity as Equivalence Relation

Proposition F.3. *Historical identity $\equiv_H$ is an equivalence relation.*

Proof. Reflexivity: $\mathcal{L}(A) = \mathcal{L}(A)$, so $A \equiv_H A$. Symmetry: $\mathcal{L}(A) = \mathcal{L}(B) \Rightarrow \mathcal{L}(B) = \mathcal{L}(A)$. Transitivity: $\mathcal{L}(A) = \mathcal{L}(B) = \mathcal{L}(C) \Rightarrow A \equiv_H C$. □ □

Chapter G

Merge–Collapse Algebra

G.1 Merge Lattice

Theorem G.1 (Territory Merge Lattice)**.** $(\mathcal{P}(\text{Tiles}), \cup, \cap)$ *is a bounded distributive lattice.*

Proof. Commutativity, associativity, and idempotence of $\cup$ and $\cap$ are standard. Absorption: $A \cup (A \cap B) = A$ and $A \cap (A \cup B) = A$. Distributivity: $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$—a standard set identity. Bottom: $\emptyset$; top: Tiles. □ □

Theorem G.2 (Collapse Normalization)**.** *The collapse operator κ is a lattice homomorphism: $\kappa(\mu(A, B)) = \mu(\kappa(A), \kappa(B))$.*

Proof. κ maps each element to its equivalence class representative. Since μ is defined on equivalence classes (merging representatives gives the representative of the merged class), κ commutes with μ. □ □

Chapter H

Sequential Bubble Machines

H.1 Bubble Automata

Definition H.1 (History-Sensitive Automaton)**.** A sequential Spherepop machine is $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F, H)$ where $\delta : Q \times \Sigma \times H \to Q \times \Gamma \times H$ is history-sensitive (identical local states may produce different futures depending on prior commitments in H).

Theorem H.2 (History-Sensitivity Strictly Increases Power)**.** *History-sensitive automata are strictly more expressive than standard finite automata.*

Proof. Containment: Any FA is simulated by ignoring H. *Strictness:* Consider $L = \{w \in \{a, b\}^* : \text{first } a \text{ was preceded by an even number of } b\text{s}\}$. No FA accepts L (pumping lemma: the count of bs before the first a is unbounded). A history-sensitive automaton accepts L by scanning H to count b-events before the first a-event. □ □

Chapter I

Entropy and Optionality

I.1 Historical Entropy

Theorem I.1 (Pop Operator Satisfies Second Law)**.** *For any non-trivial pop* $\mathbf{P}_\varepsilon$*:* $S(x_{t+1}) \leq S(x_t)$*.*

Proof. $|\mathcal{T}(x_{t+1})| = |\mathcal{T}(x_t) \cap \Omega'| \leq |\mathcal{T}(x_t)|$. Since $\log_2$ is monotone, $S(x_{t+1}) \leq S(x_t)$. □ □

This is the formal analogue of Landauer's principle [13]: logical irreversibility corresponds to thermodynamic entropy production.

I.2 Collapse Criterion

Theorem I.2 (Collapse Criterion)**.** *Civilization* C *is in irreversible collapse at tick* t *if* $\Phi_C(t) < \Phi_{\min}$ *and* $\Delta\Phi_C(t) < 0$ *simultaneously. Under these conditions* $\Phi_C(t') < \Phi_{\min}$ *for all* $t' > t$ *in expectation.*

Proof. If $\Phi_C < \Phi_{\min}$, the civilization lacks the option diversity to mount effective responses. Each turn, forced pops (responses to attacks, famines) reduce Φ_C without the civilization being able to choose high-value expansion pops. Thus $\Delta\Phi_C < 0$ compounds until $\Phi_C \to 0$. □ □

Chapter J

Category-Theoretic Semantics

J.1 The History Category

Theorem J.1 (**Hist** is a Monoid Category)**.** *The endomorphisms of a fixed option space* Ω*:* $\mathrm{End}(\Omega) = \{\varepsilon : \Omega \to \Omega\}$ *form a monoid under composition with identity* id_Ω*.*

Proof. Closure: composition of two maps $\Omega \to \Omega$ gives $\Omega \to \Omega$. Associativity: function composition is associative. Identity: id_Ω is the unit. □ □

Theorem J.2 (No Traced Monoidal Structure Without Path Erasure)**. Hist** *does not admit a trace operator [11] consistent with the irreversibility constraint.*

Proof. A trace operator would allow feedback: a future event causing a past event to be undone. The trace of a non-trivial pop $\varepsilon : \Omega \otimes U \to \Omega' \otimes U$ with $\Omega' \subsetneq \Omega$ would require the feedback wire to "undo" the option-space contraction, contradicting Strict Contraction. □ □

J.2 Monotone Victory Conditions

Proposition J.3. *Monotone victory conditions form a subposet of the lattice of natural transformations* $F \Rightarrow \mathbf{2}$*.*

Proof. A natural transformation $V : F \Rightarrow \mathbf{2}$ is monotone if $V_\Omega(s) = 1 \Rightarrow V_{\Omega'}(F(\varepsilon)(s)) = 1$ for all events ε. The set of monotone natural transformations is closed under pointwise meet and join, forming a subposet. □ □

Chapter K

Procedural World Dynamics

K.1 Tile Evolution PDE

Definition K.1 (World Evolution Equation)**.** Resource density $\rho(x, y, t)$ evolves as:

$$\frac{\partial \rho}{\partial t} = D\nabla^2 \rho + r\rho\Big(1 - \frac{\rho}{K}\Big) - c \cdot A \cdot \rho$$

where D is the diffusion coefficient, r is ecological growth rate, K is carrying capacity, c is per-capita extraction rate, and A is agent population density.

Theorem K.2 (Discrete Stability Condition)**.** *The forward-Euler discretization is stable if* $\Delta t \leq h^2/(4D)$ *(2D CFL condition).*

Proof. The 2D diffusion stencil gives amplification factor $|1 - 4D\Delta t/h^2|$. Stability requires this ≤ 1, giving $\Delta t \leq h^2/(2D)$ in 1D and $\Delta t \leq h^2/(4D)$ in 2D (the factor of 2 from the two spatial dimensions). □ □

Theorem K.3 (Equilibrium Resource Density)**.** *With* $A = 0$*, resource density converges to* $\rho^* = K$ *if* $\rho_0 > 0$*.*

Proof. With $D = 0$, each tile obeys $d\rho/dt = r\rho(1 - \rho/K)$, the logistic equation with stable equilibrium $\rho^* = K$. With $D > 0$, diffusion averages spatial heterogeneity, driving the system toward the uniform equilibrium exponentially. □ □

Chapter L

Trajectory Collapse Geometry

L.1 Hypothesis Manifolds

Theorem L.1 (Maximum Entropy Prior)**.** *In the absence of distinguishing information, the maximum entropy distribution on Δ_{n-1} is uniform: $p_i = 1/n$, $H = \log_2 n$.*

Proof. By the Gibbs inequality, $H(\mathbf{q}) \leq \log_2 n$ for any distribution on n elements, with equality iff $q_i = 1/n$. □ □

L.2 Information Gain from Strategic Intelligence

Definition L.2 (Strategic Information Gain)**.** An intelligence report r provides information gain:

$$\mathrm{IG}(r) = H(\mathbf{p}) - H(\mathbf{p}' \mid r).$$

Corollary L.3. *Perfect intelligence about scenario h_k (setting $p'_k = 1$) provides maximum information gain $\mathrm{IG}_{\max} = \log_2 n$ bits.*

This formalizes the intuition that espionage is valuable precisely to the extent that it resolves strategic uncertainty. An empire that spends on intelligence is buying bits of future option-space clarity.

Bibliography

Bibliography

[1] Aristotle. *Nicomachean Ethics*. Translated by W. D. Ross. Oxford University Press, 2009.

[2] Alain Badiou. *Being and Event*. Continuum, 2005.

[3] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, 1984.

[4] Bartosz Milewski. *Category Theory for Programmers*. Blurb, 2018.

[5] John C. Baez and Mike Stay. Physics, topology, logic and computation: A Rosetta Stone. In *New Structures for Physics*, pages 95–172. Springer, 2010.

[6] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

[7] Kit Fine. Things and their parts. *Midwest Studies in Philosophy*, 23(1):61–74, 1999.

[8] Michel Foucault. *The Archaeology of Knowledge*. Pantheon Books, 1972.

[9] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[10] Martin Heidegger. *Being and Time*. Harper Perennial, 2008.

[11] André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.

[12] Jaegwon Kim. Events as property exemplifications. In *Supervenience and Mind*. Cambridge University Press, 1993.

[13] Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

[14] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1998.

[15] Erik Meijer. Your mouse is a database. Keynote presentation, Build Conference, Microsoft, 2014.

[16] Erik Meijer and Gavin Bierman. A co-relational model of data for large shared data banks. Lectures on duality, category theory, and reactive computation, 2011–2015.

[17] Robin Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, 1999.

[18] Noam Nisan and Shimon Schocken. *The Elements of Computing Systems: Building a Modern Computer from First Principles*. MIT Press, 2005.

[19] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[20] David I. Spivak. *Category Theory for the Sciences*. MIT Press, 2014.

[21] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265, 1936.

[22] Francisco Varela, Humberto Maturana, and Ricardo Uribe. Autopoiesis: The organization of living systems. *BioSystems*, 5(4):187–196, 1974.

[23] John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

[24] Alfred North Whitehead. *Process and Reality*. Free Press, 1978.

[25] Norbert Wiener. *Cybernetics: Or Control and Communication in the Animal and the Machine*. MIT Press, 1948.

[26] Ludwig Wittgenstein. *Philosophical Investigations*. Blackwell Publishing, 1953.

[27] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.

[28] Mark Wilson. *Wandering Significance: An Essay on Conceptual Behaviour*. Oxford University Press, 2006.

[29] Robert Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[30] Karl Friston. The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138, 2010.

[31] Karl Friston. Life as we know it. *Journal of the Royal Society Interface*, 10(86):20130475, 2013.

[32] Michael Levin. Technological approach to mind everywhere: an experimentally grounded framework for understanding diverse bodies and minds. *Frontiers in Systems Neuroscience*, 16:768201, 2022.

Index of Spherepop Bubbles

The following bubbles are defined or specified in this textbook:

- `NAND` — Ch. 1
- `NOT`, `AND`, `OR` — Ch. 1
- `XOR`, `MUX`, `DMUX` — Ch. 1
- `PermissionCheck` — Ch. 1
- `AND16`, `OR16`, `NOT16` — Ch. 2
- `Add16`, `Inc16` — Ch. 2
- `ResourceAdder` — Ch. 2
- `PopulationALU` — Ch. 2
- `Bit`, `Register16` — Ch. 3
- `RAM8` …`RAM16K` — Ch. 3
- `TileState`, `WorldMap` — Ch. 3, 5
- `CellularAutomaton` — Ch. 4
- `TerrainDecoder`, `TileYield` — Ch. 5
- `FoodIntegrator`, `GoldLedger` — Ch. 6
- `ResearchAccumulator` — Ch. 6
- `CityProduction` — Ch. 6
- `ChebyshevCheck` — Ch. 7
- `VisionBroadcaster`, `FogUpdater` — Ch. 7
- `ExplorationTracker` — Ch. 7
- `Unit`, `UnitFSM` — Ch. 8
- `MoveValidator`, `CombatResolver` — Ch. 8
- `SettlePermission`, `CityFounder` — Ch. 9
- `BorderExpander` — Ch. 9
- `PathChecker`, `TradeRoute` — Ch. 10
- `GoldCalculator` — Ch. 10
- `DamageCalculator`, `CombatRound` — Ch. 11
- `CaptureCity` — Ch. 11
- `Treaty`, `TrustMatrix` — Ch. 12
- `DiplomacyAI` — Ch. 12
- `TechTree`, `PrereqChecker` — Ch. 13
- `ForestSpread`, `DepletionTracker` — Ch. 14
- `ClimateLayer` — Ch. 14
- `EventLog`, `HistoryQuery` — Ch. 15
- `NarrativeGenerator` — Ch. 15
- `TurnManager`, `WorldBubble` — Ch. 16
- `SaveSystem` — Ch. 17
- `CheckpointManager` — Ch. 17
- `EventSourcedWorld` — Ch. 19
- `QueryOwner`, `AppendEvent` — Ch. 19
- `TrustFromHistory` — Ch. 19
- `TimelineReconstructor` — Ch. 23
- `CityBiography` — Ch. 23
- `ConnectedComponents` — Ch. 24
- `CutVertexDetector` — Ch. 24
- `Compactness` — Ch. 24
- `OptionalityMeter` — Ch. 25
- `MaxOptionalityAI` — Ch. 25
- `BranchingSimulator` — Ch. 26
- `ScenariosDatabase` — Ch. 26
- `HistoryFunctor` — Ch. 27
- `VictoryChecker` — Ch. 27
- `IntelligenceFusion` — Ch. 28
- `GlobalStateReconstructor` — Ch. 28
- `LogMerge` — Ch. 29
- `DiplomaticIntelligibility` — Ch. 30
- `CulturalMissionary` — Ch. 30
- `AutopoiesisCheck` — Ch. 31
- `FixedMul8` — Ch. 2
- `PredictionErrorTracker` — Ch. 13
- `BlanketStabilityMeter` — Ch. 13
- `CollectiveBlanket` — Ch. 13