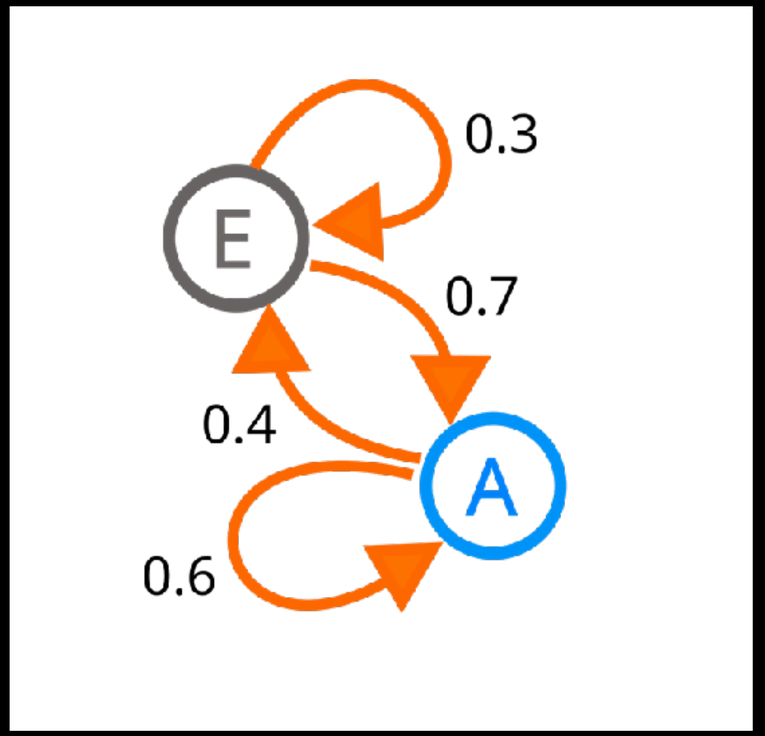
POMDPs, Swarms and gradient-free optimisation

Raoul Grouls, 3 November 2025

Markov process

A Markov process is

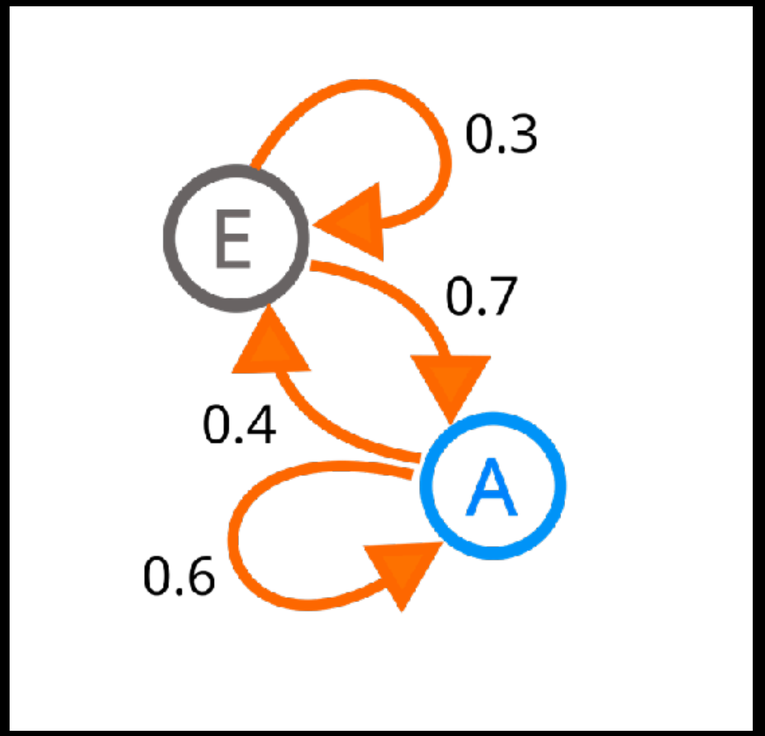
- A stochastic process that describes a sequence of events
- The probability of each event depends **only** on the state attained in the previous event.



Markov assumption

Informally:

“What happens next depends **only** on the current state”

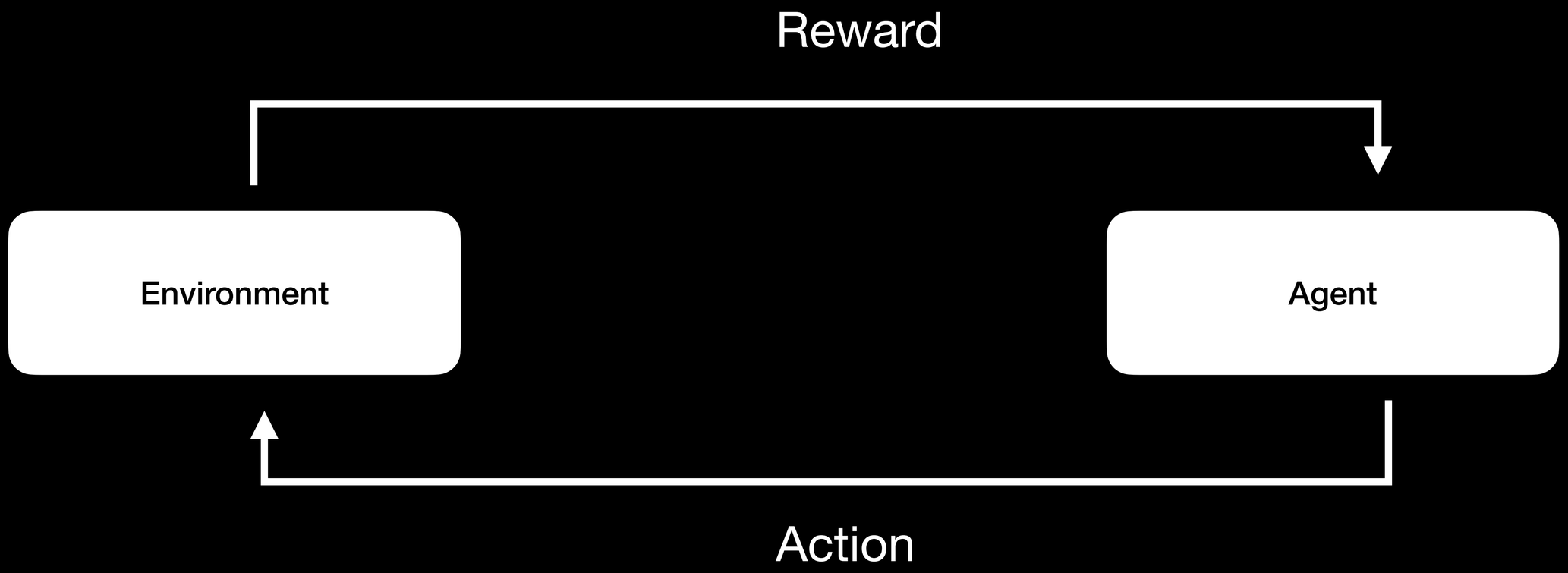


Markov Decision Process

- A MDP is a problem formulation; it is not an algorithm
- $MPD = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$
- state space $\mathcal{S}$
- action space $\mathcal{A}$
- transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow P(\mathcal{S})$
- reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
- discount factor $\gamma \in [0,1]$

Search spaces

- Assume we have a set of $\mathcal{S}$ parameters
- Continuous: $\mathcal{X} \subseteq \mathbb{R}^{|\mathcal{S}|}$
  - For example, [0.1 2.4 -1.4]
- Discrete: $\mathcal{X} \subseteq D_1 \times D_2 \times \ldots D_{|\mathcal{S}|}$
  - For example, $D_1 = \{0,1\}$ and $D_2 = \{\text{ReLU, Tanh}\}$
- Mixed:
  - For example $\mathcal{X} \subseteq \mathbb{R}^{S_c} \times \{0,1\}^{S_d}$ with S_c continuous parameters and S_d discrete binary parameters.


Reward
Environment
Agent
Action

Markov Decision Process

We want to find the action that maximizes the reward for the current state and action, plus the expected reward for future possible actions and new states.

$$U^*(s) = max_a \left(R(s,a) + \gamma \Sigma_{s'} \, T(s' \mid s,a) \; U^*(s') \right)$$

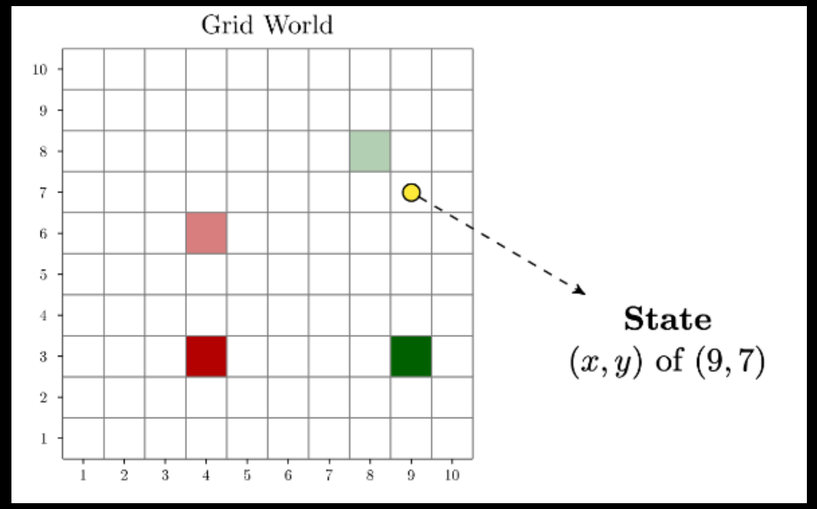
Markov Decision Process

The goal is to find an optimal policy π that tells us, given a state, what the most rewarding action will be.

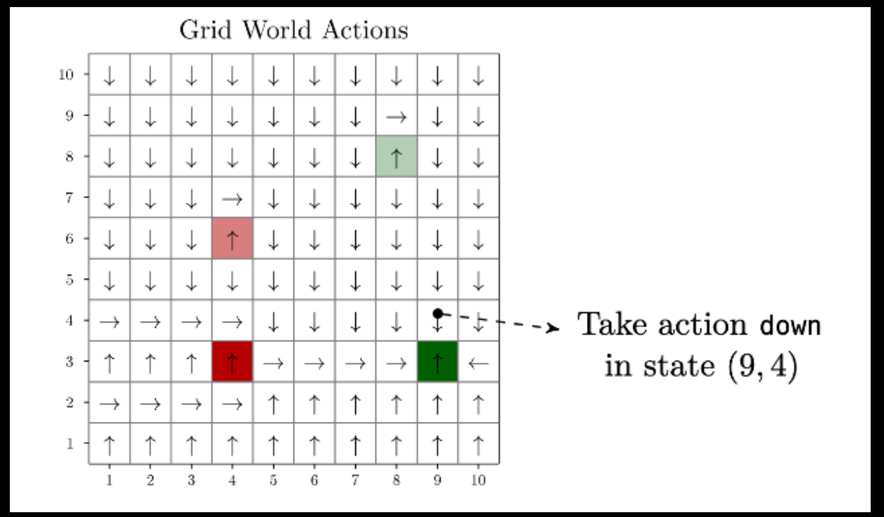
$$\pi^*(s) = a$$

State and Action space

- The set of all possible states an agent can be in

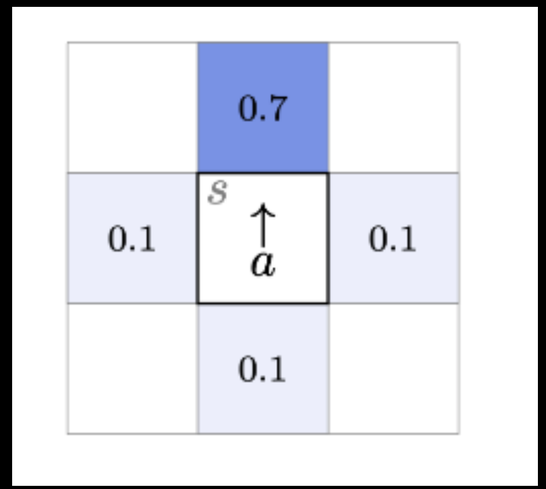


- The set of all possible actions an agent can take



Transition function

- transition function $T : \mathcal{S} \times \mathcal{A} \to P(\mathcal{S})$
- For continuous spaces: $P(\mathcal{S})$ is the space of probability measures on $\mathcal{S}$
- For discrete spaces:
  - $P(\mathcal{S}) = \Delta(\mathcal{S}) = \left\{p : \mathcal{S} \to [0,1] \,|\, \Sigma_{s \in \mathcal{S}} p(s) = 1\right\}$
  - $\Delta(\mathcal{S}) \in \mathbb{R}^{|\mathcal{S}|}$

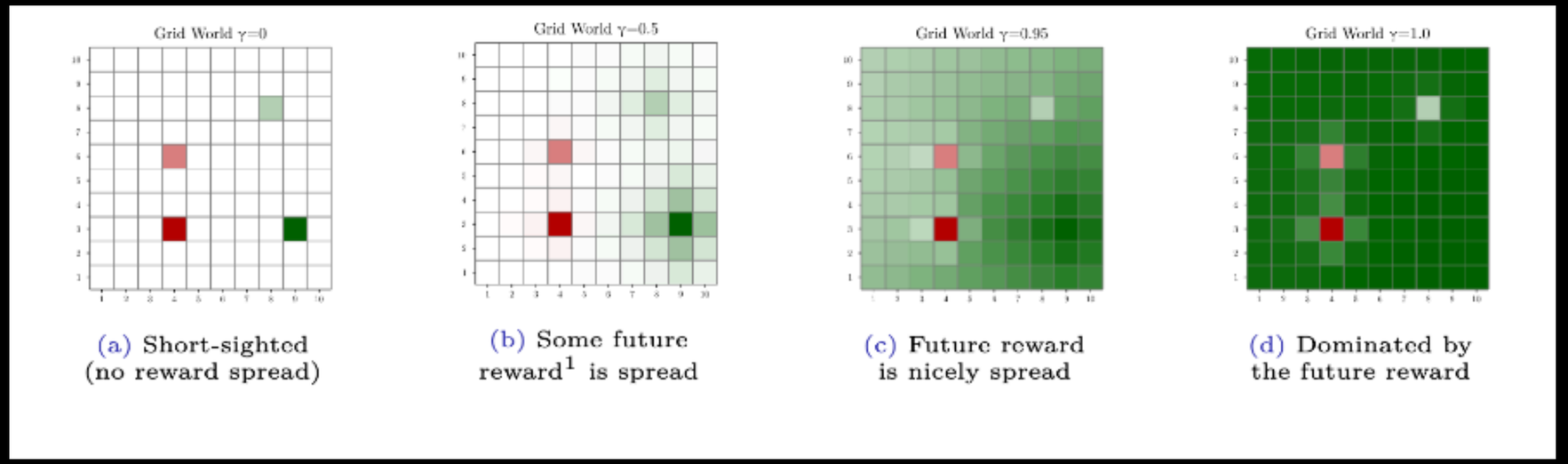


Reward function

- reward function $R\colon\ \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- Taking an action a in state s will give you some reward
- The reward can be any number, including 0

Discount factor

- discount factor $\gamma \in [0,1]$
- Controls how much future rewards are taken into account



(a) Short-sighted (no reward spread)

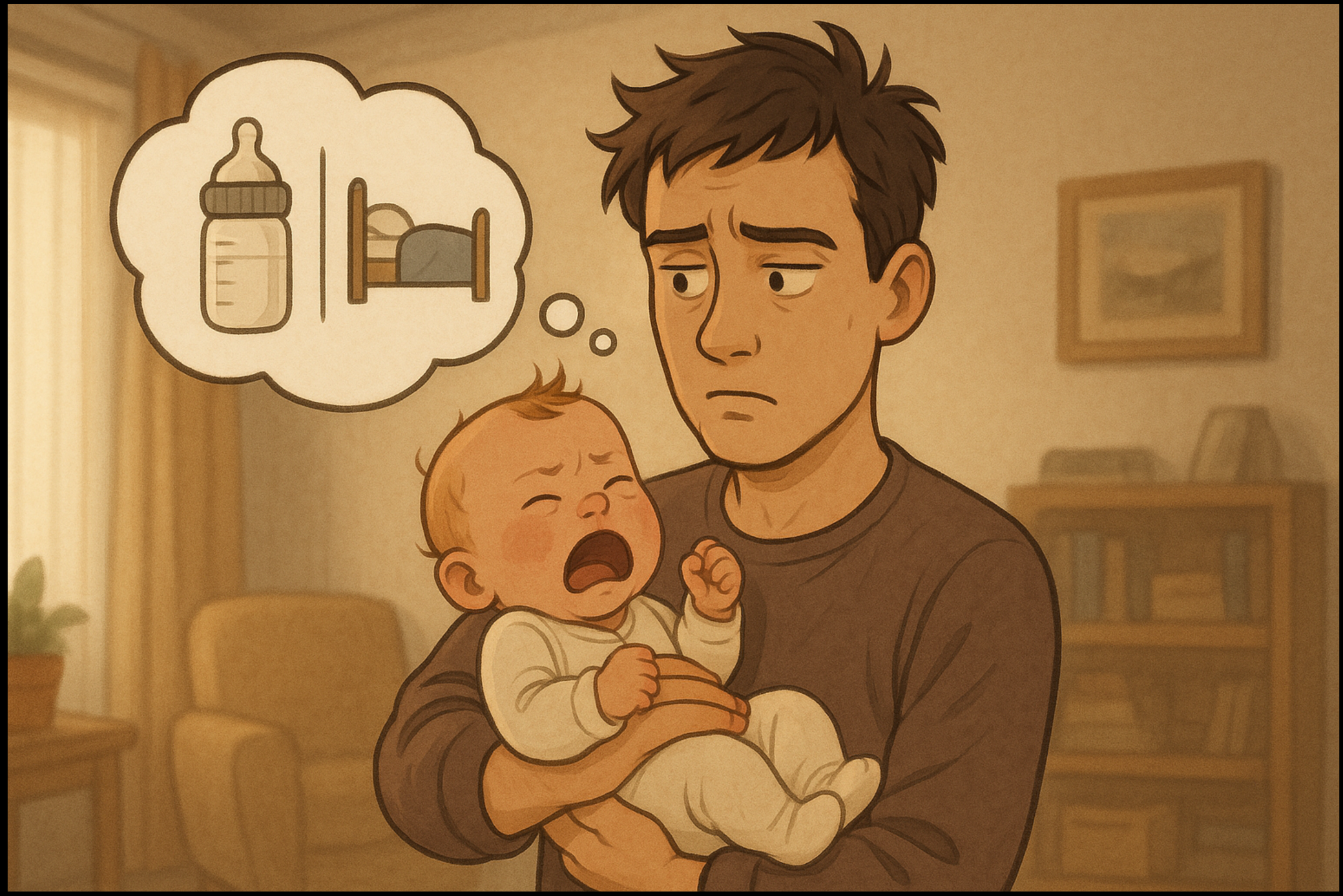
(b) Some future reward[1] is spread

(c) Future reward is nicely spread

(d) Dominated by the future reward

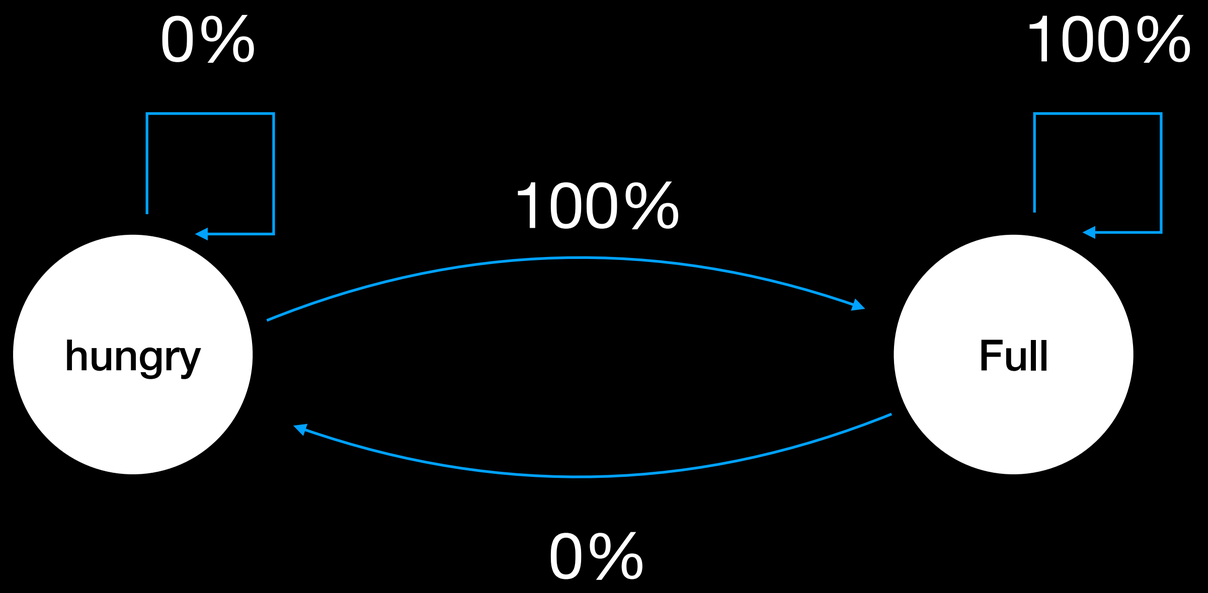
POMDP

- A *Partially observable* MDP
- We can not know the true state; we have observations that will reveal the true state with some probability

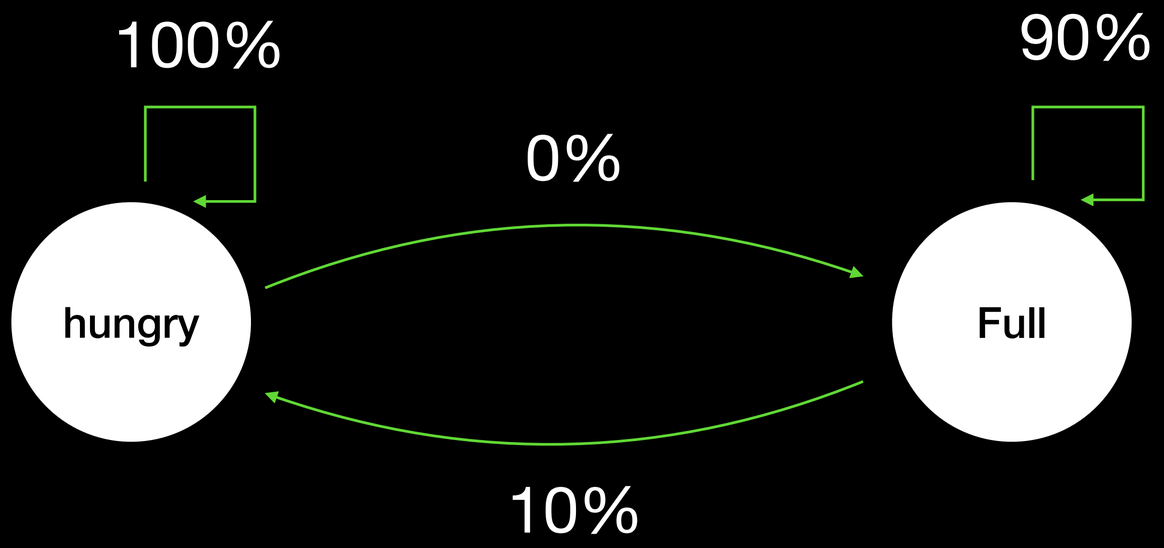


Crying baby problem

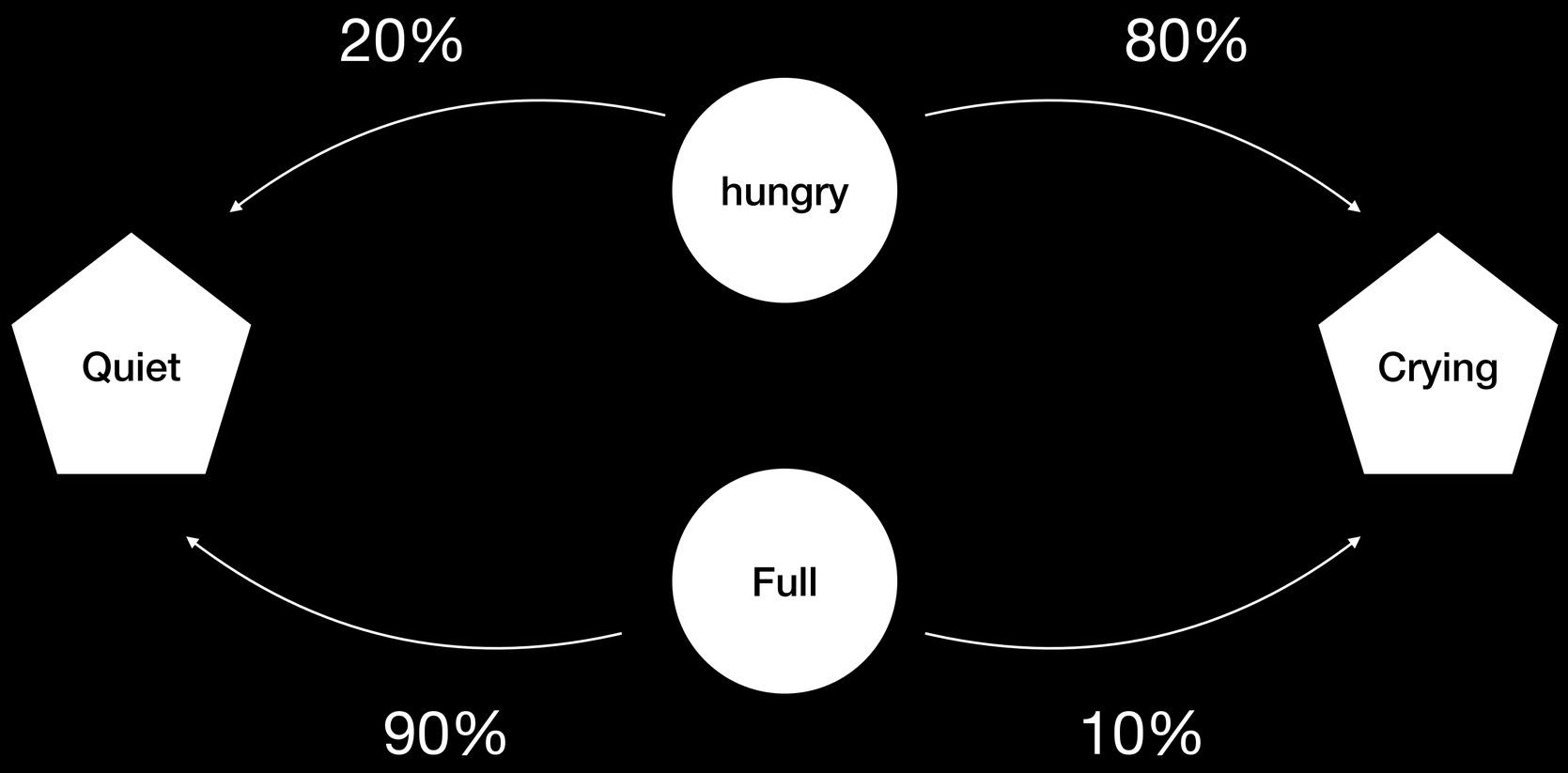
Feed



Ignore



Crying baby problem



Observation

State

Observation

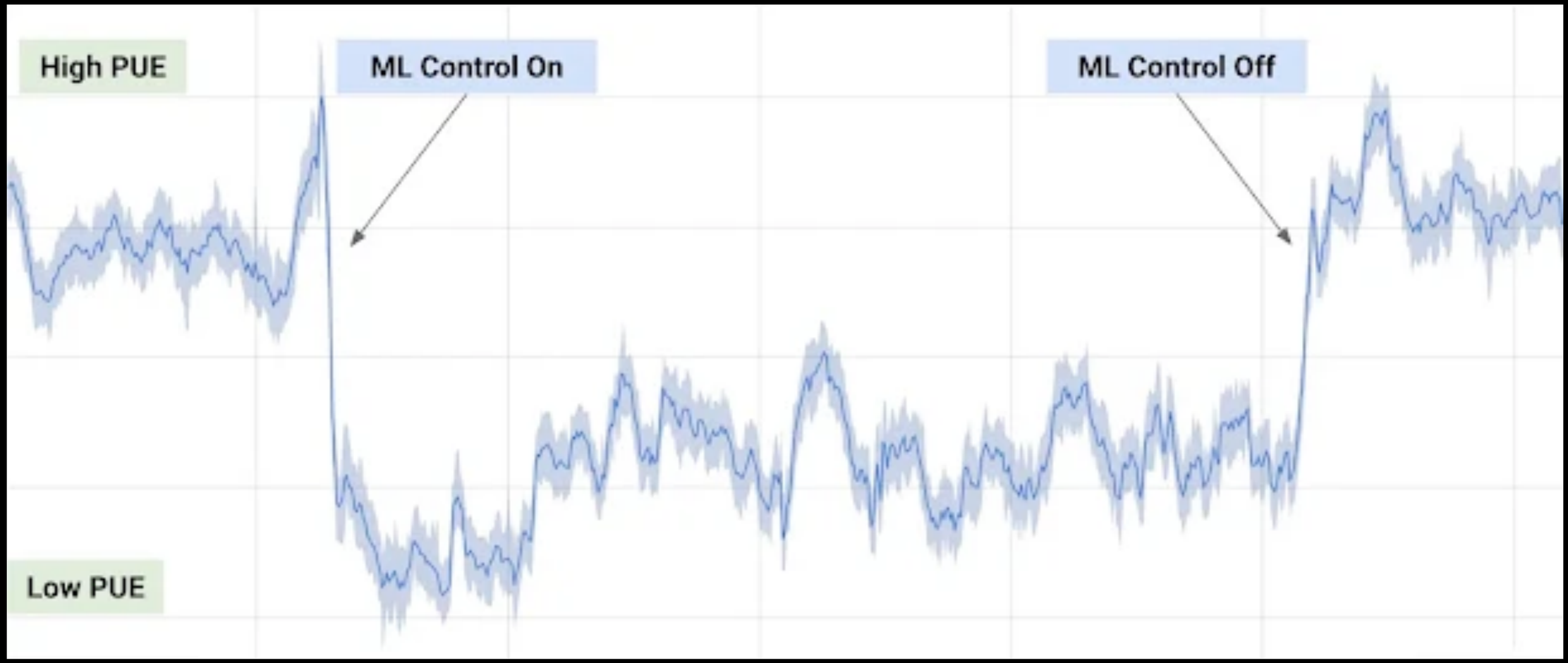
POMDP

- We need to add:
  - An observation space $\mathcal{O}$ that describes the state of possible observations
  - An observation function $O(o \mid s')$ that can give us for state s' the probability p that we see observation o
  - In the implementation, we use Bayes Theorem to flip this around to $P(s' \mid o)$ that gives us, given observation o, the probability we are actually in the unobservable state s'

Icecream in the summer

- Your data is sales of icecream & temperature in the summer
- What is happening when the sales goes up? And when the sales goes down?
- What is happening when the sales goes down, but the temperature is still very high?
- What is the state? What is the observation?

DeepMind AI Reduces Google Data Centre Cooling Bill by 40%



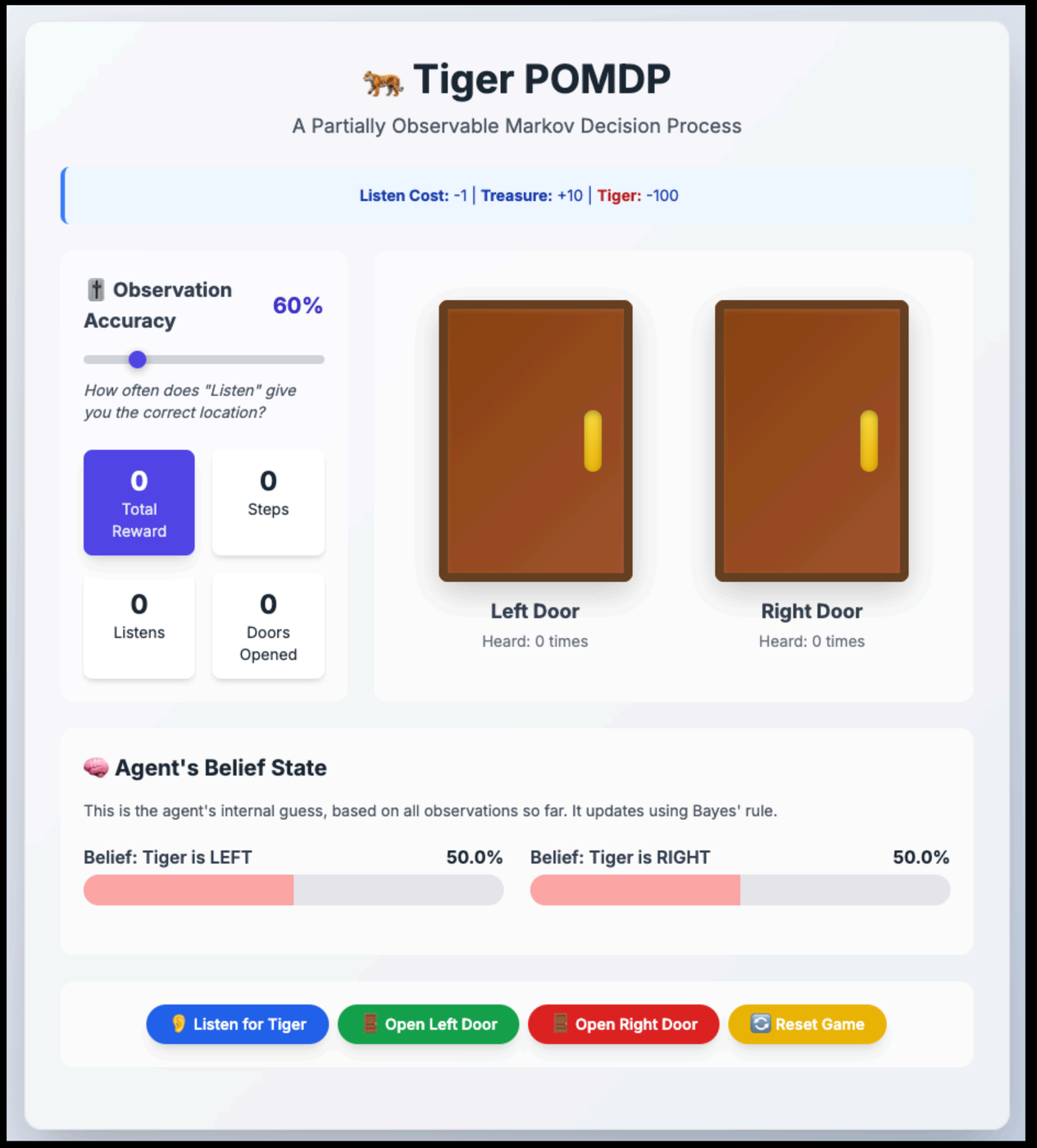
Tiger POMDP

- There are two doors
- One has a tiger (-100), one has a treasure (+10)
- You can listen (-1) which has a known accuracy of 60%
- Question: what is the best policy to win the most points over 10 rounds?

Tiger POMDP

Lets play:

- https://gemini.google.com/share/e004b8dd32ff



When gradient based methods fail

- NP-complete problems
- Discrete spaces
- Multi-modal landscapes

What are NP-complete problems

- It is very hard to FIND the optimal solution
- But it is relatively easy to check a solution
- Eg “find the shortest way from point A to B” is computationally hard
- “Check the distance from A to B” is relatively easy

What are NP-complete problems

- Karp defined 21 NP-complete problems. Two examples are:
  - Traveling Salesman: given a graph, what is the shortest possible route to visit each city once and return to the start
  - Knapsack problem: Given a set of items with weight and value, which items to include so that the total weight is low , and the value as high as possible.

What are NP-complete problems

- Many problems turn out to follow one of these 21 patterns.
- Eg knapsack follows the pattern: you have a limit (the capacity of the knapsack) and some items, each with a “cost” (the weight) and something you want to maximize (the value)
  - Cargo loading (eg even packing for a flight)
  - Planning a diet
  - Project management budgeting
  - Investment portfolios

AlphaZero

Go Game

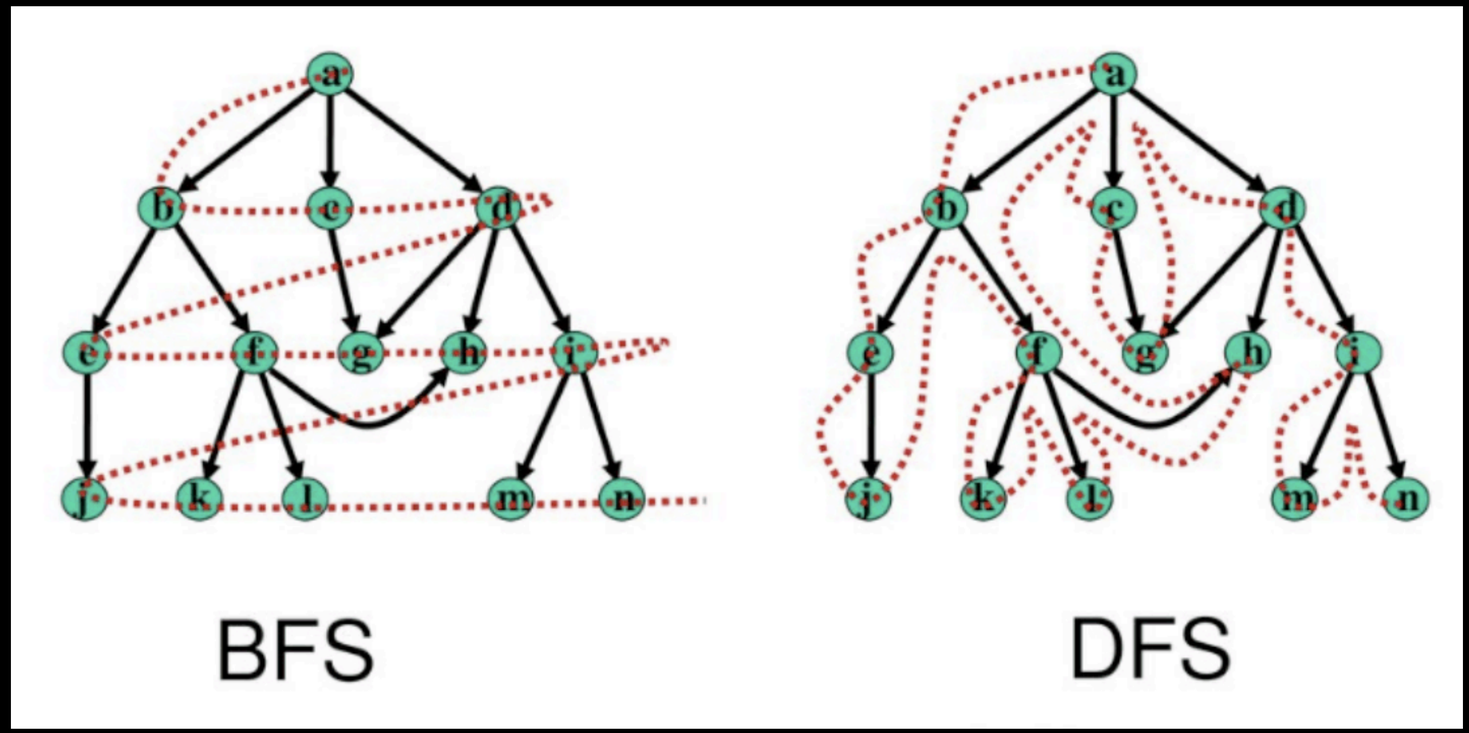
- Standard board size 19x19
- This gives 361 options for the first stone
- $361 \times 360 \times 359 \times \ldots$ possible games
- This is estimated to be around 10^{170} possible games

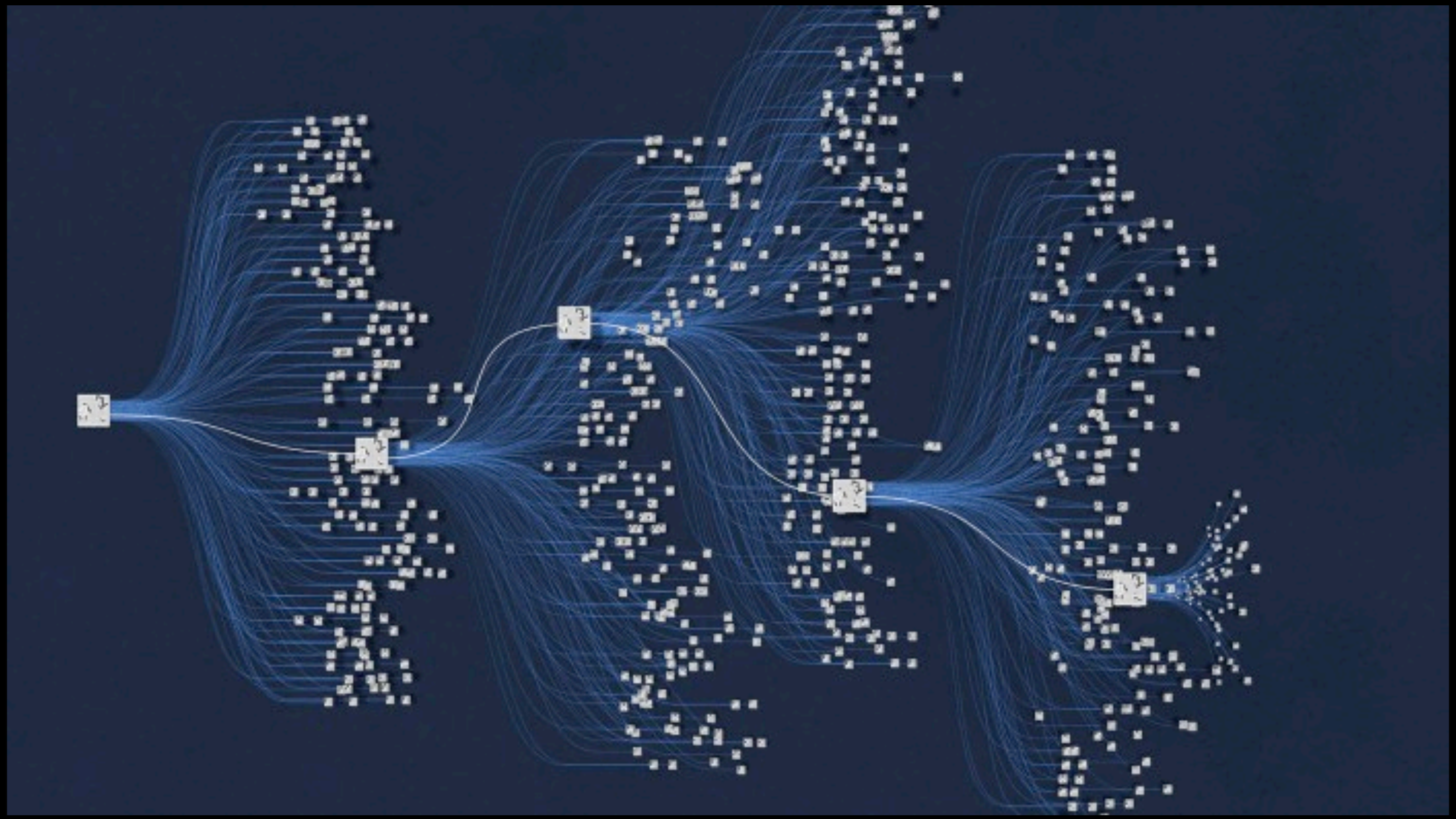
Move 37

AlphaZero

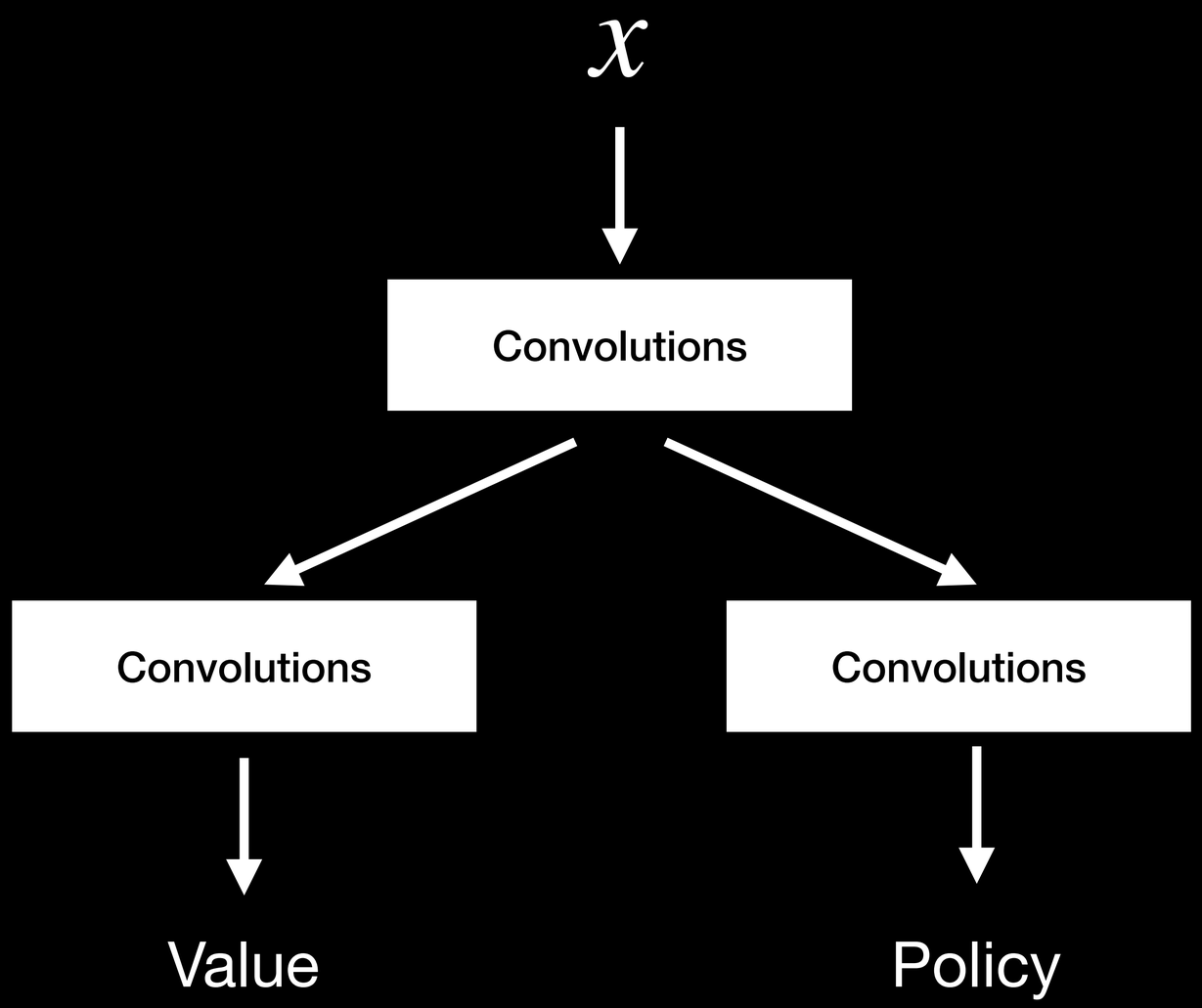
- A network that calculates which action gives the best expected value
- Monte-Carlo tree search for the best play

Tree search





Network



X		
	O	

State

0	0.1	0.4	0.1	0	0.1	0.1	0.1	0.1

Policy

0.7

Value

X		O
	O	

State

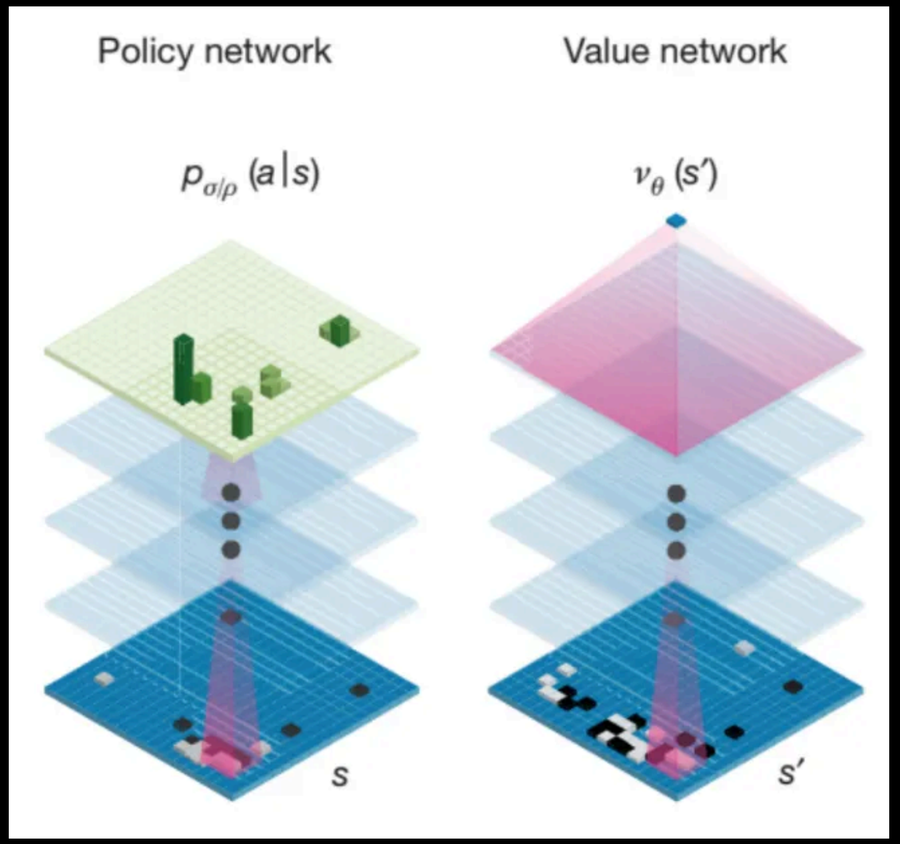
0	0.1	0.4	0.1	0	0.1	0.1	0.1	0.1

Policy

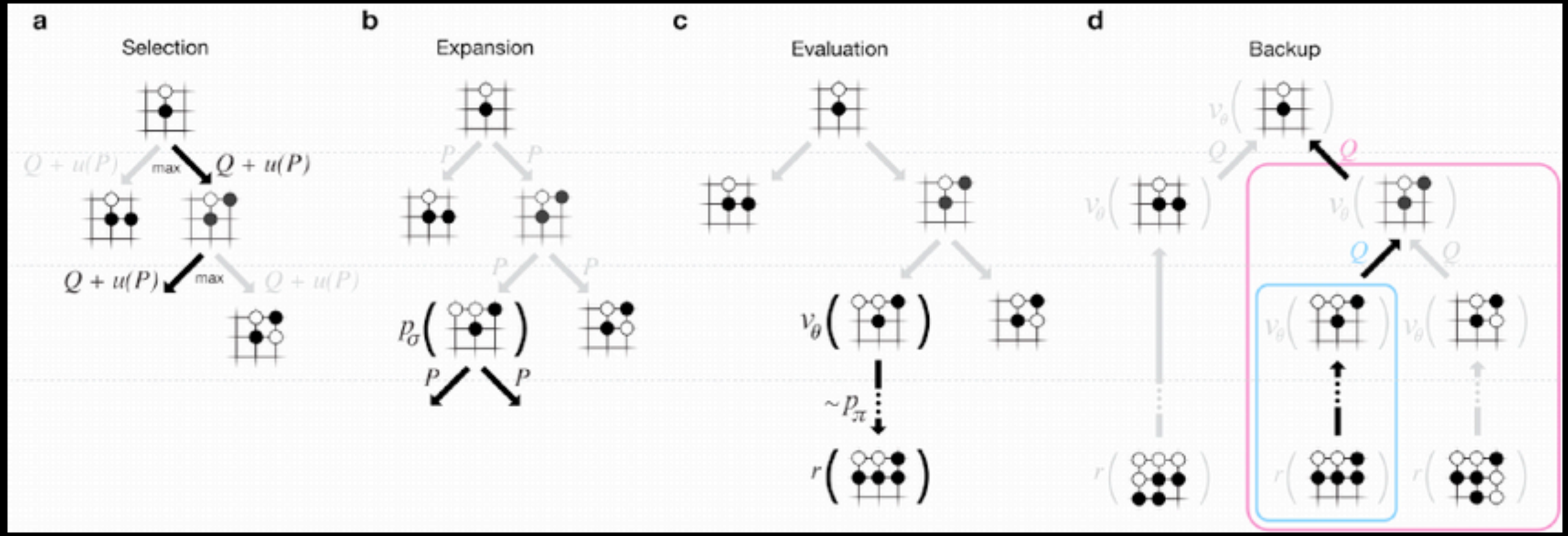
0.8

Value

Feature Category	Concise Description	# of Planes
Stone State	Current player's stones	1
	Opponent's stones	1
	Empty points	1
Board Constant	All 1s (bias/board extent)	1
Liberties	Stone string liberties (1 to 7, 8+; one-hot)	8
Recency of Moves	How many turns ago point was played (up to T; one-hot)	8
Capture Size	Opponent stones captured by move (1 to 7, 8+; one-hot)	8
Self-Atari Size	Own stones in atari if opponent plays here (1-7, 8+; one-hot)	8
Liberties After	New string's liberties after move (1 to 7, 8+; one-hot)	8
Ladder Properties	Is a ladder capture?	1
	Is a ladder escape?	1
Game Legality	Legal and not self-eye filling move?	1
Player's Turn	Current color to play (e.g., Black=1)	1
Total		**48**


Policy network
Value network
$p_{\sigma/\rho}(a|s)$
$v_\theta(s')$
s
s′

Monte Carlo Tree Search



AlphaZero

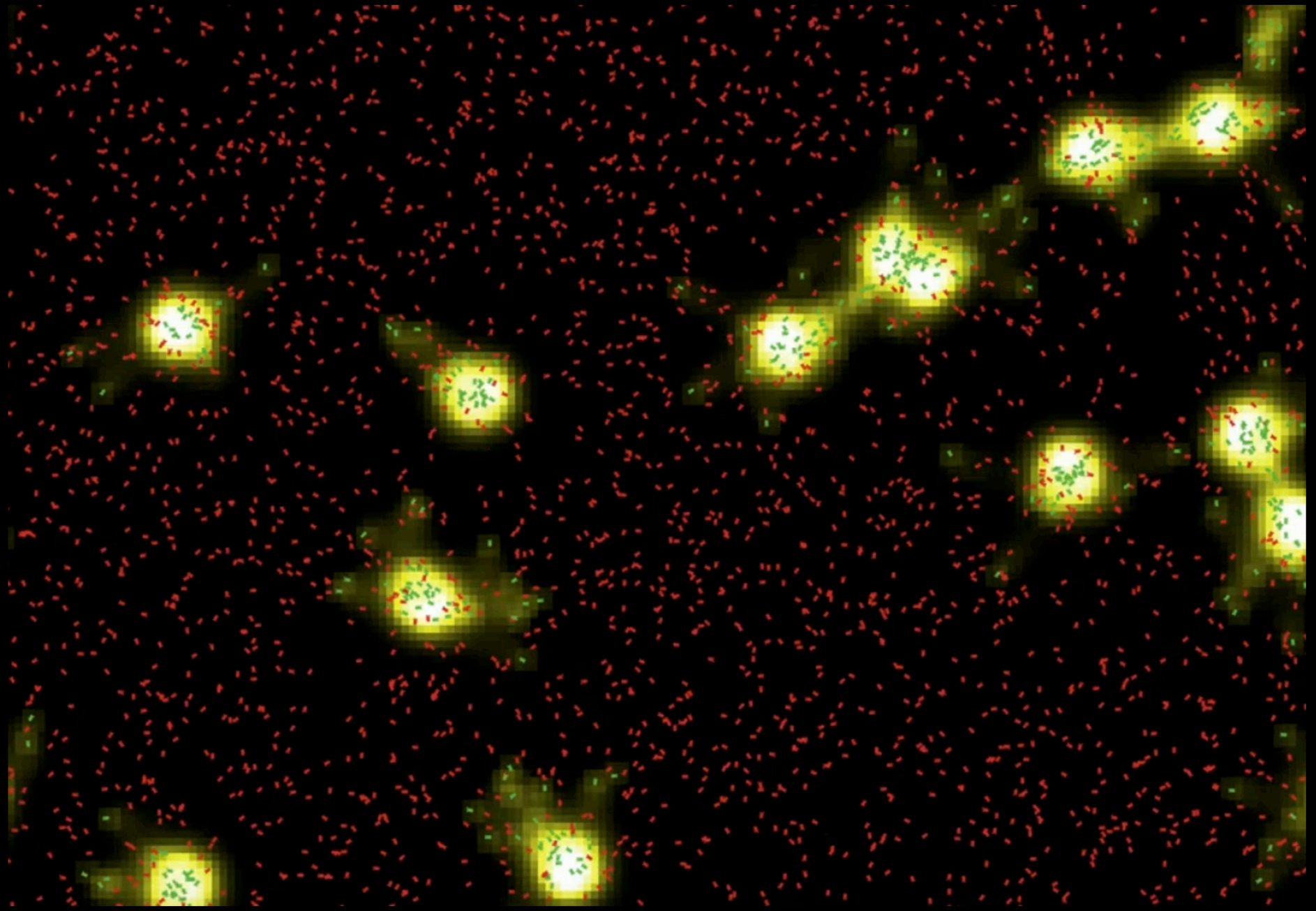
- Stone positions (1)
- History (last 7 moves)
- Current player color (1)

AlphaGo vs AlphaZero

0:100

Swarm intelligence


מכון ויצמן למדע



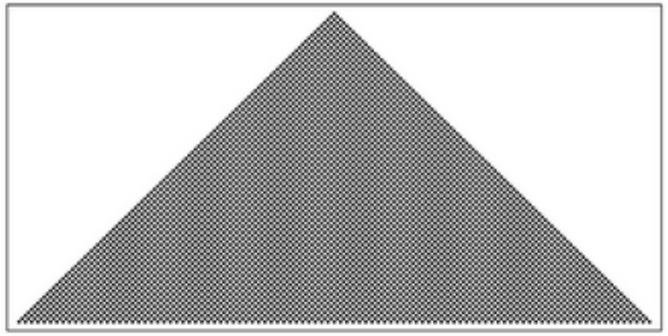
Complexity

- Much of the emphasis in existing science is on breaking systems down to find their underlying parts
- When a phenomenon is encoutered that seems complex, it is taken for granted that the underlying mechanism itself is complex
- Wrong assumption: if the rules of a program are simple, this means that the behavior must also be correspondingly simple
- Instead: simple rules can generate complex behavior
- Some systems are computationally irreducible: the only way to find their behavior is to trace each step. The only approach here is simulation.

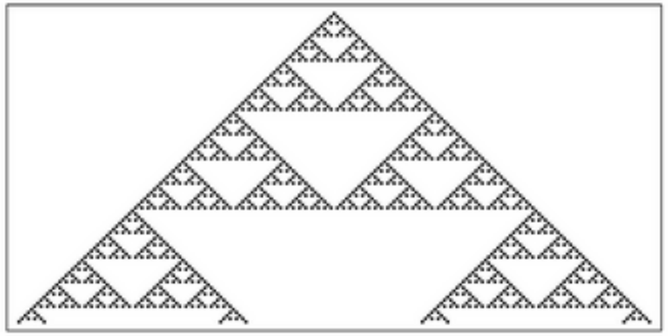
Cellular Automata

- Recursive computational functions
- A grid world, where each “cell” takes some of the environment as input and gives a new output, based on simple rules
- Simple rules, complex behaviors
- You can’t always “jump” to the end of an evolution: you will need to simulate the full program to see the end.

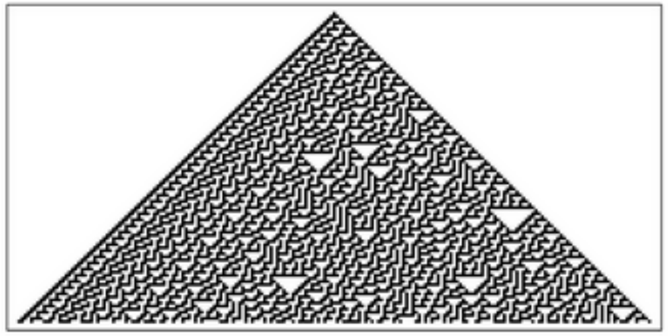
Cellular Automata



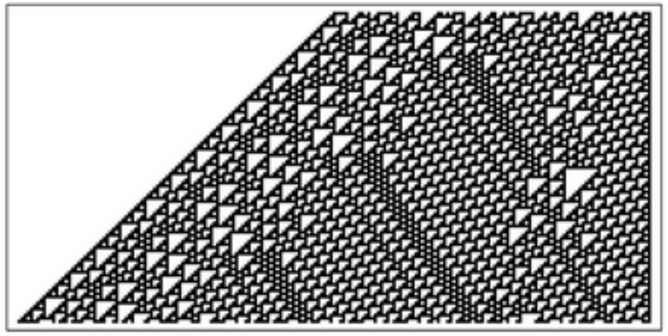
repetition (rule 250)



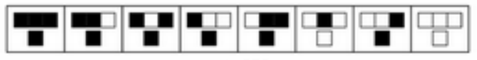
nesting (rule 90)



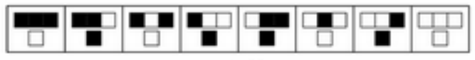
randomness (rule 30)



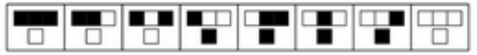
localized structures (rule 110)



rule 250



rule 90

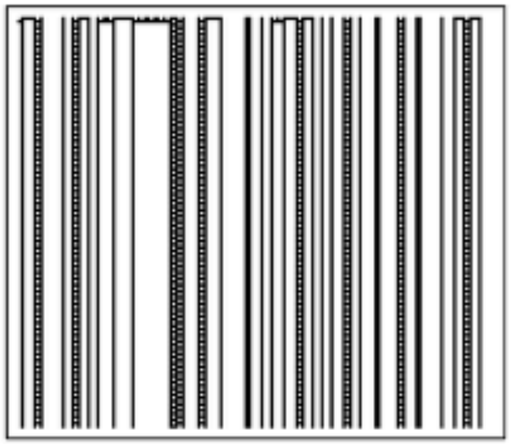


rule 30

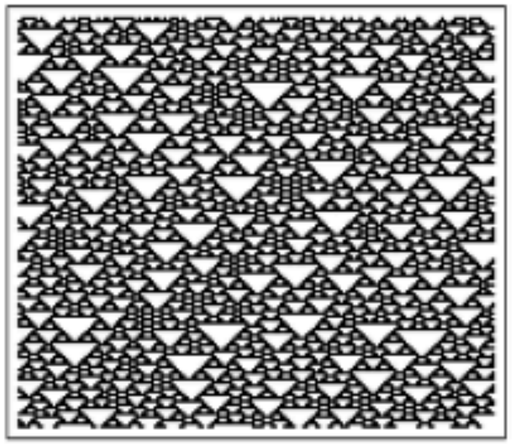
rule 110

Wolframs ‘four classes of behavior’

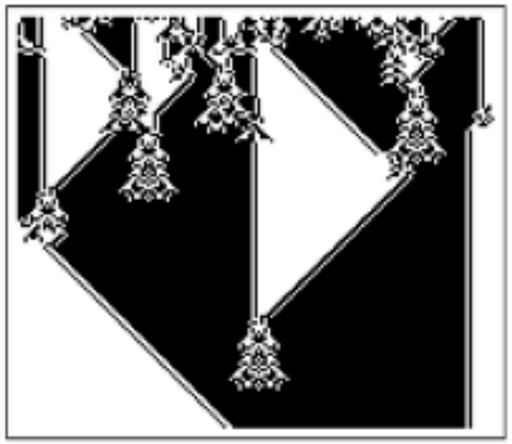
class 1



class 2



class 3



class 4

Simple

Repeating

Random

Localized structures

Wolframs ‘four classes of behavior’

- If a system is universal, it is effectively capable of emulating any other computational system
- At first glance, it seems quite impossible that simple systems can be universal
- However, simple systems like “game of life” and cellular automata rule 110 are Turing complete. This means that, in principle, one could build a computer with them.

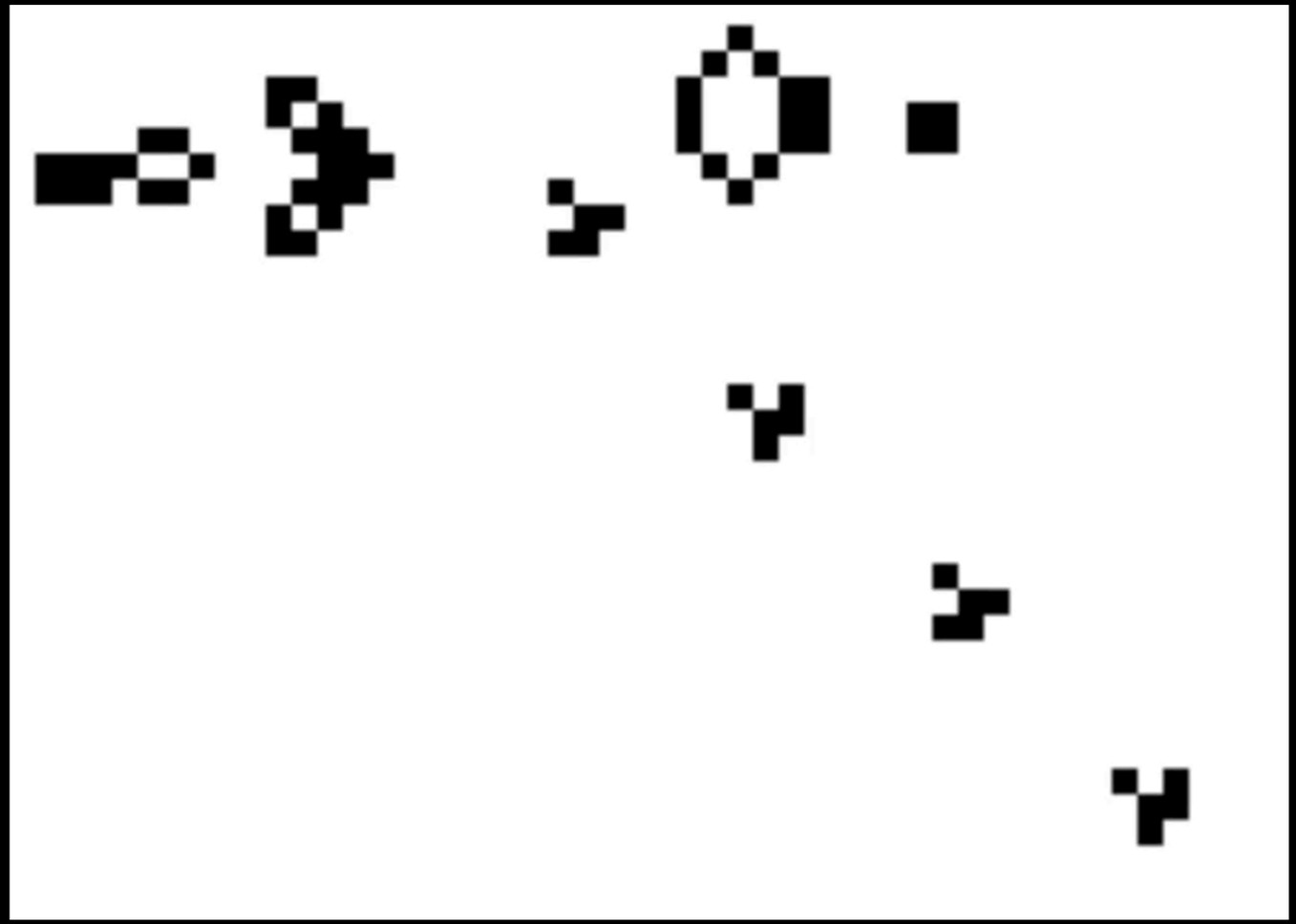
Game of Life

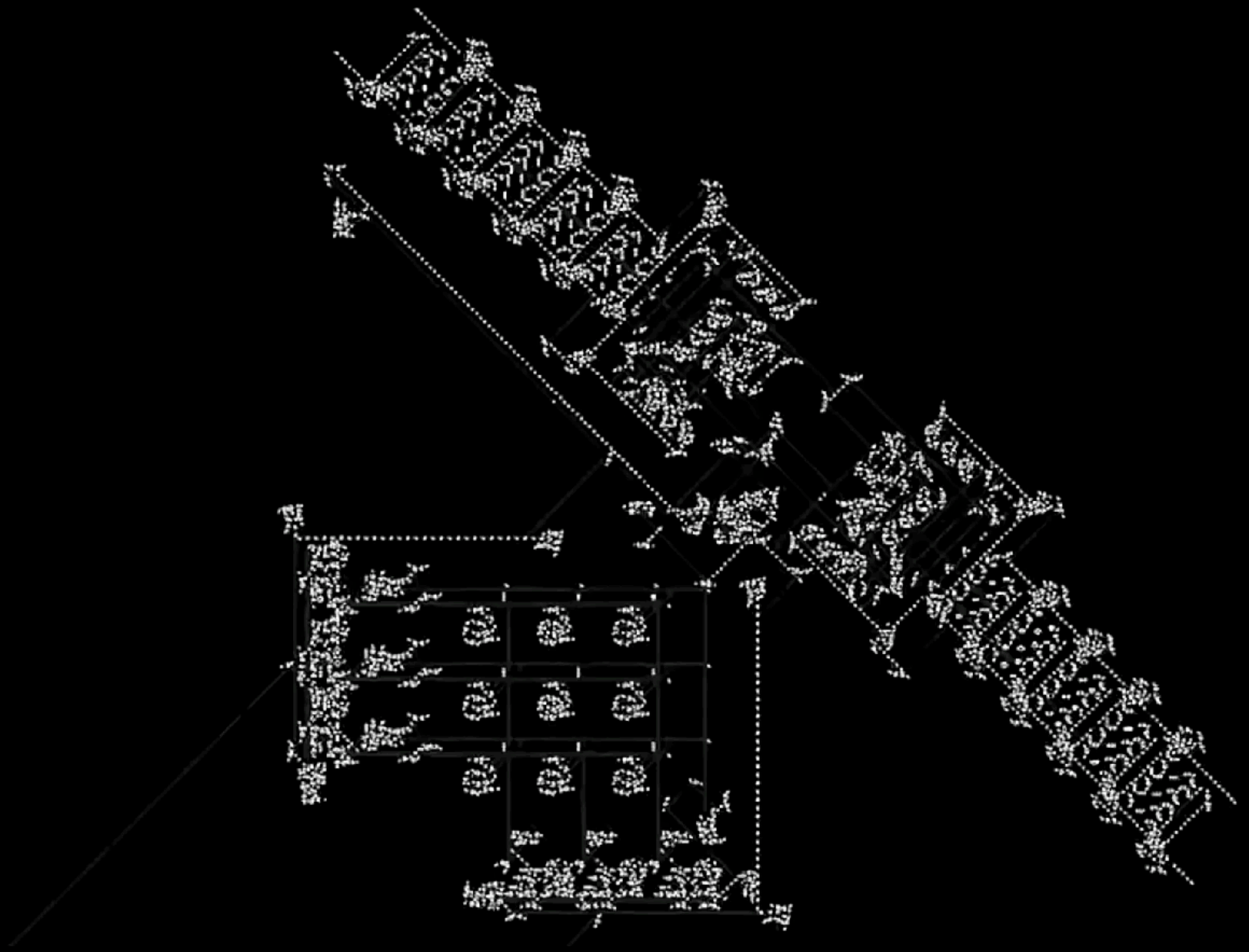
Any live cell with:

- Less than 2 neighbours -> die
- 2 or 3 neighbours -> live
- More than 3 neighbours -> die

Any dead cell:

- Exactly 3 neighbours -> live





https://youtu.be/Kk2MH9O4pXY

Swarm intelligence

- Swarm intelligence is an emergent phenomenon
- We use a swarm of simple agents, that often follow very simple rules
- Swarm intelligence utilizes randomness and the explore-exploit balance to obtain emergent intelligence of the swarm to solve a problem

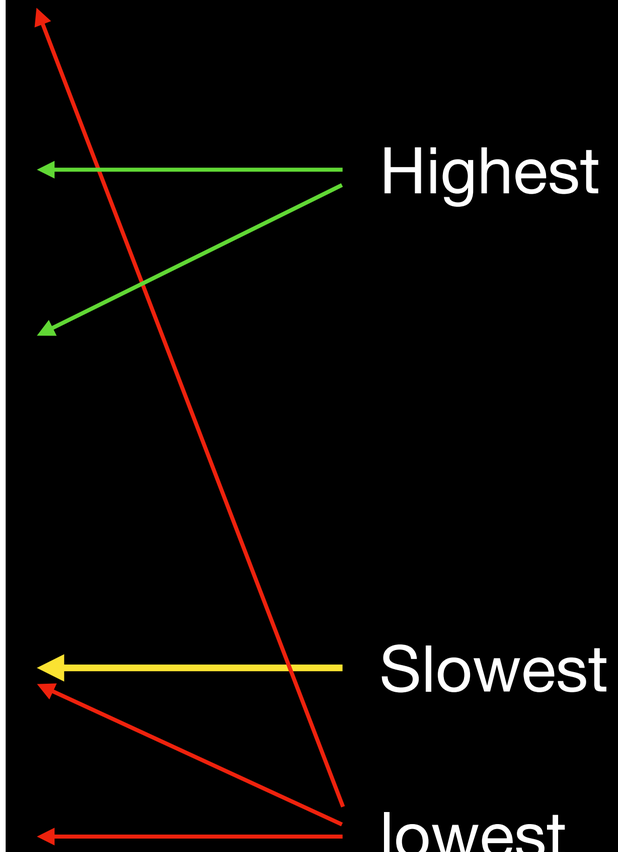
Swarm intelligence

Some examples

- Simulated Annealing : in the start the temperature is “high” and there is a lot of “bouncing around”. When everything “cools down” the randomness reduces
- Particle Swarm Optimisation: every particle has a personal best and is informed about the global best location. The direction of the search is a combination of local knowledge and global knowledge
- Artificial Bee Colony: there are different “types” of bees. Some are in “scout phase” and focus on exploring, some are in “worker bee” modus and focus on a promising “spot”
- Grey Wolf Optimisation: there is a hierachical structure (alpha, beta wolf) and a lot of “followers”. The alpha and beta wolves hunt the prey, the rest “encircles” the prey.

Table III
RESULT OF BENCHMARK-I

Dataset	Method	Score	Time
dna	HyperOpt Random	0.9538178	366
	HyperOpt TPE	0.9579369	322
	Optuna Random	0.9580583	94
	Optuna TPE	0.9602619	521
	Optunity PSO	0.925369	200
	SMAC	0.9595218	807
electricity	HyperOpt Random	0.6969229	546
	HyperOpt TPE	0.7011213	341
	Optuna Random	0.6942903	494
	Optuna TPE	0.7011213	167
	Optunity PSO	0.9074971	190
	SMAC	0.7011213	393
gas drift	HyperOpt Random	0.850121	23869
	HyperOpt TPE	0.88168	14105
	Optuna Random	0.868639	14549
	Optuna TPE	0.882114	4403
	Optunity PSO	0.982105	11923
	SMAC	0.825001	12966
nomao	HyperOpt Random	0.926516	1711
	HyperOpt TPE	0.931569	1441
	Optuna Random	0.93677	1064
	Optuna TPE	0.937144	1124
	Optunity PSO	0.938943	1554
	SMAC	0.937003	2622
pendigits	HyperOpt Random	0.979188	1783
	HyperOpt TPE	0.987569	1012
	Optuna Random	0.982508	336
	Optuna TPE	0.983886	897
	Optunity PSO	0.958028	2980
	SMAC	0.984226	860
semeion	HyperOpt Random	0.901575	762
	HyperOpt TPE	0.915246	458
	Optuna Random	0.92832	895
	Optuna TPE	0.934616	1989
	Optunity PSO	0.700295	538
	SMAC	0.929794	1647



A Comparative study of Hyper-Parameter Optimization Tools. Shekar S. et al., 2021, https://arxiv.org/abs/2201.06433

L-BFGS vs Differential Evolution

Algorithm	L-BFGS-B (Local)	Differential Evolution (Global)
Analogy	A single, blindfolded hiker.	A *population* of 50 hikers.
How it moves	Feels the *gradient* (slope) and takes the steepest downhill step.	Creates *new* hikers by combining three *other* random hikers.
Exploration	**None.** It only exploits the local area.	**High.** It creates new "trial" solutions all over the map.
Best For	Smooth, convex "bowl" problems (Sphere).	Bumpy, multi-modal "egg carton" problems (Rastrigin).
Fails On	Rastrigin (gets stuck in first pothole).	Sphere (is very slow and inefficient compared to L-BFGS-B).

Particle Swarm Optimization

- Goal: $\min_x f(x), x \in \mathbb{R}^d$
- We use a swarm of N particles, each particle has:
  - Position $x_i(t)$
  - Velocity $v_i(t)$
  - Personal best $p_i = \arg\min_t f(x_i(t))$
  - Global best $g = \arg\min_{j \in N} f(p_j)$

Particle Swarm Optimization

Update:

$$v_i(t+1) = \omega v_i(t) + c_1 r_1 \odot (p_i - x_i(t)) + c_2 r_2 \odot (g - x_i(t))$$

Where:

- $\omega \in [0,1]$ inertia weight (explore vs exploit)
- $c_1, c_2 > 0$ personal and global weights
- $r_1, r_2 \sim Uniform(0,1)^d$ a random vector

Grey Wolf Optimization

- The algorithm has a control parameter a that decreases from 2 to 0 over time

$$a = 2 - t\left(\frac{2}{T_{max}}\right)$$

With t the current iteration and T_{max} the total number of iterations.

Grey Wolf Optimization

- There are three leaders:
  - X_α Alpha (first-best)
  - X_β beta (second-best)
  - X_δ delta (third-best)
- $X_i(t)$ The location of each wolf in the pack i at time t, with $i = 1,\ldots,N$

Grey Wolf Optimization

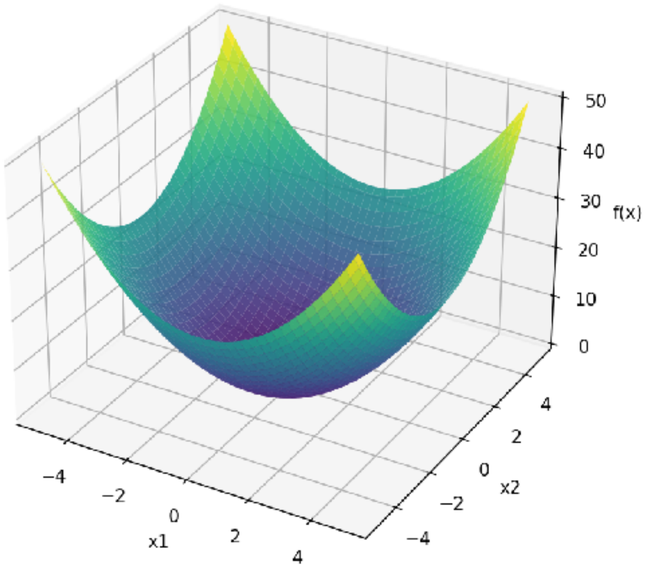
- For every wolf , for every leader, eg X_α, we calculate:
  - r_1, r_2 random vectors [0, 1]
  - $D_\alpha = |C_1 \cdot X_a - X_i(t)|$ the direction that wolf i would follow if it follows alpha
  - $X_1 = X_\alpha - A_1 \cdot D_\alpha$ the new location, had wolf i followed alpha
  - With explore parameters $A_1 = 2a \cdot r_1 - a$ and $C_1 = 2 \cdot r_2$

- Finally, we average the new direction of every wolf over the three positions, had it followed the three leaders:

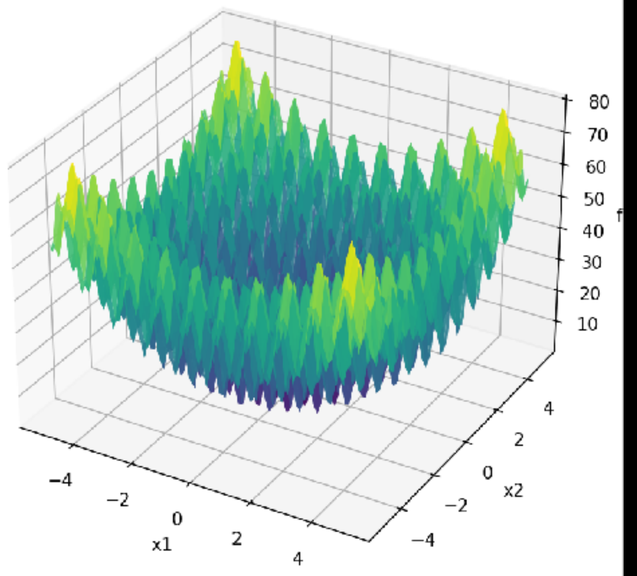
$$X_i(t+1) = \frac{X_1 + X_2 + X_3}{3}$$

2D Benchmark Function Visualizations

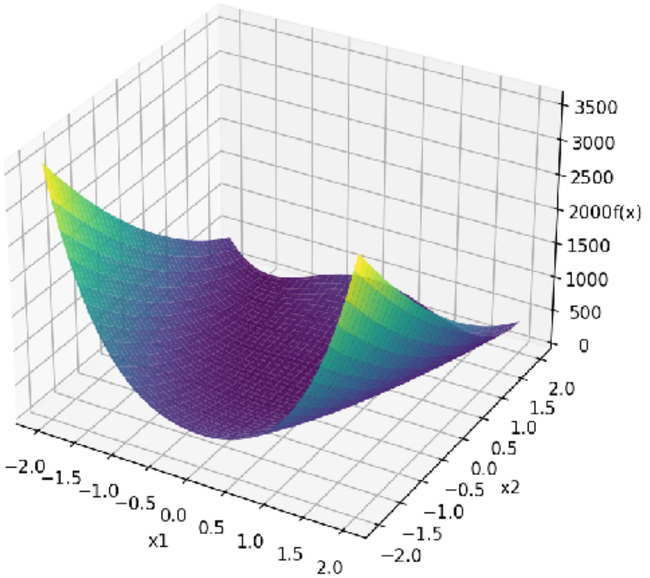
Sphere



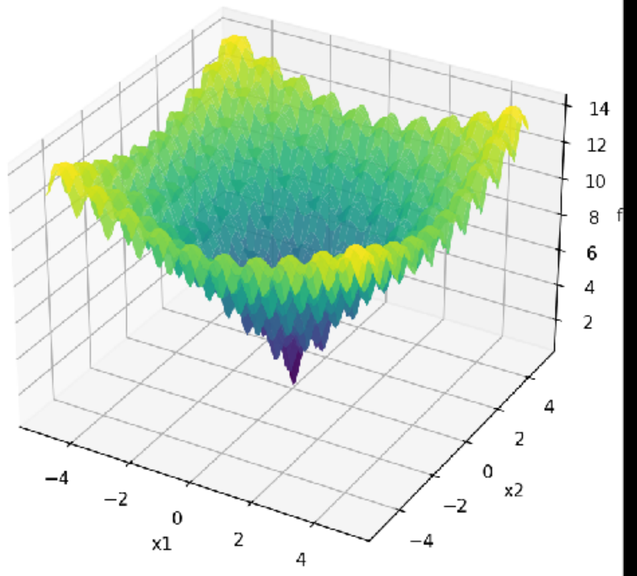
Rastrigin



Rosenbrock



Ackley



Algorithm Convergence Comparison (Dimensions=10)