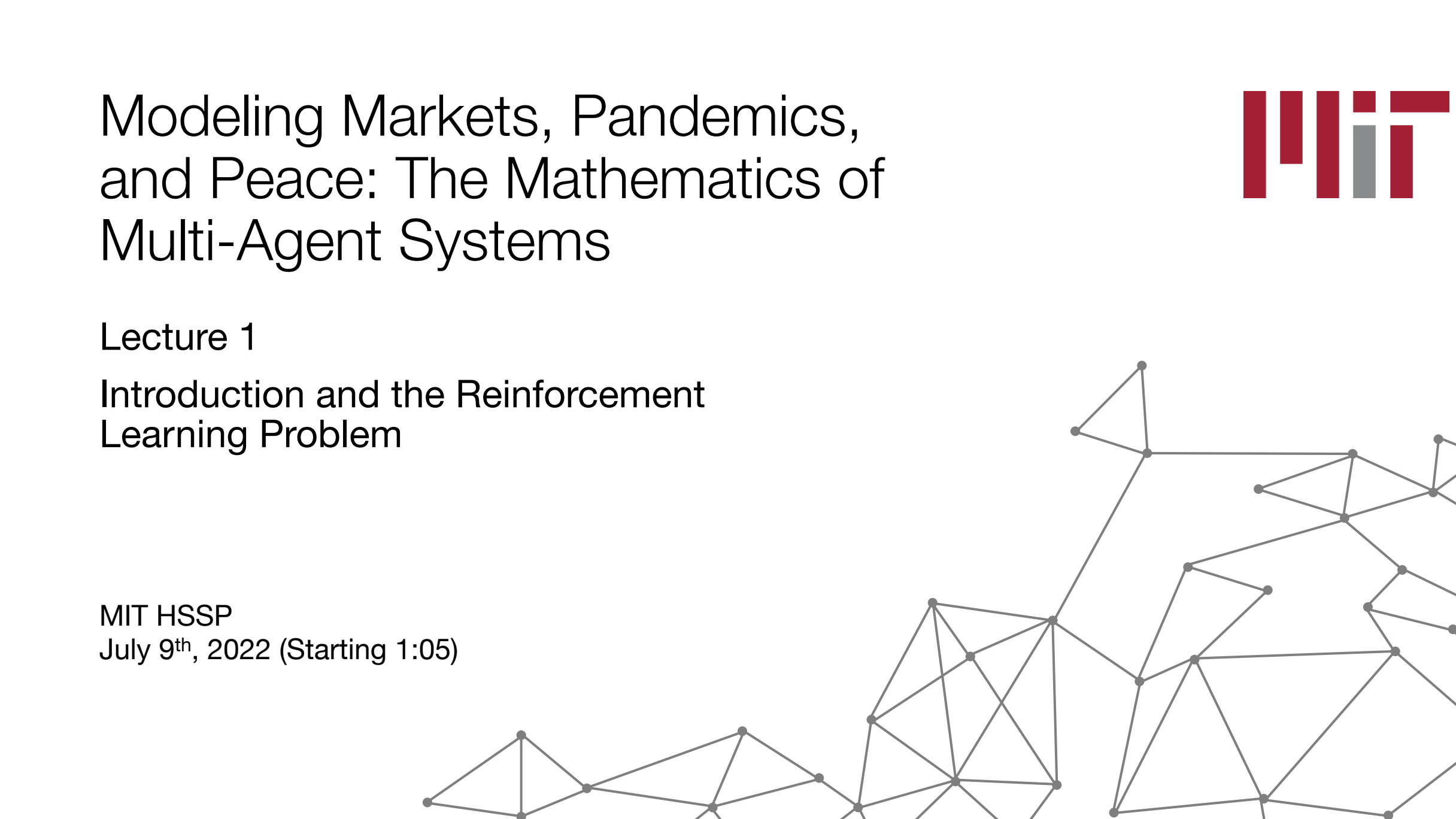
Modeling Markets, Pandemics, and Peace: The Mathematics of Multi-Agent Systems



Lecture 1

Introduction and the Reinforcement Learning Problem

MIT HSSP
July 9th, 2022 (Starting 1:05)

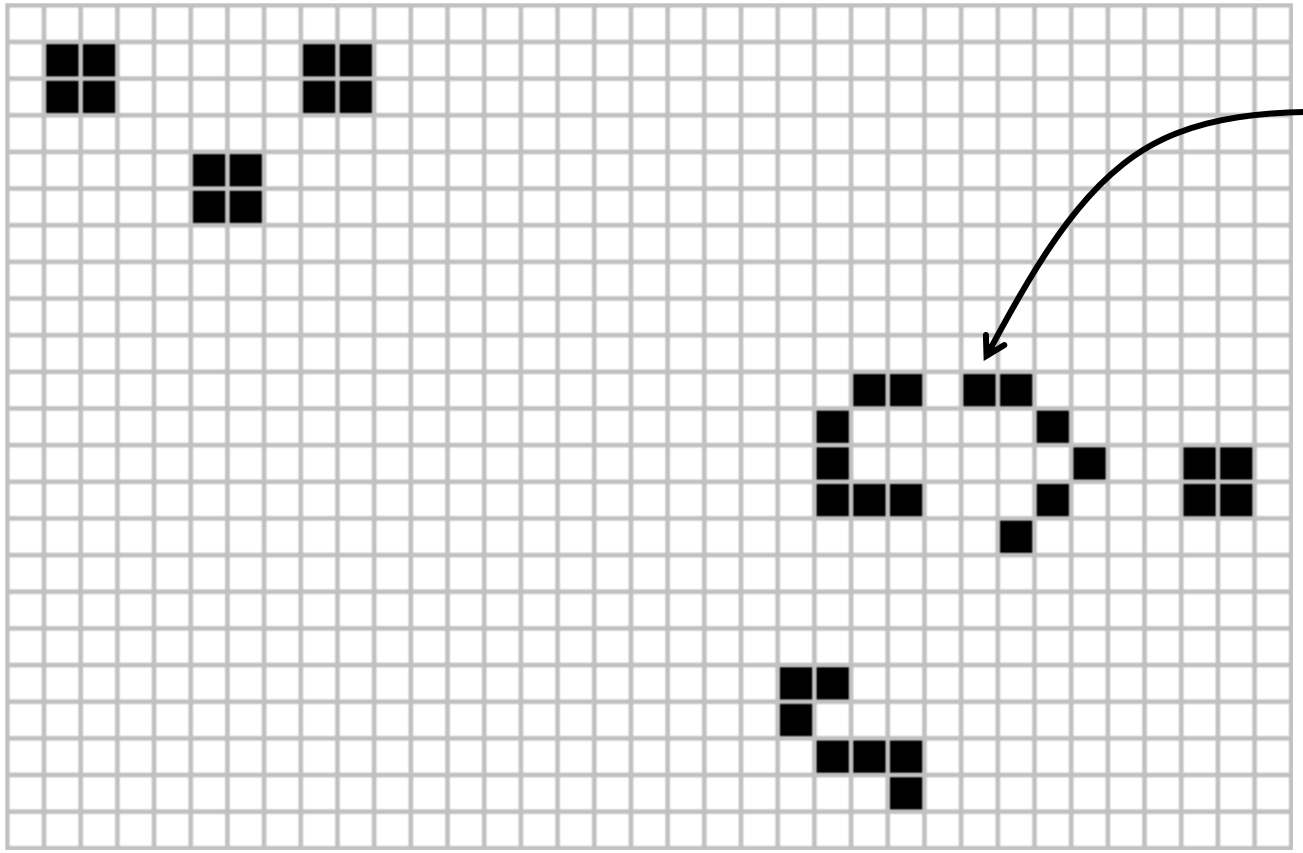
Introductions

Sihao Huang
Physics (8) and Electrical Engineering (6-1)
with a minor in Political Science

Julia Balla
Mathematics with Computer Science
(18-C) with a minor in Economics

Email: C15061-teachers@esp.mit.edu

What is a multi-agent system?

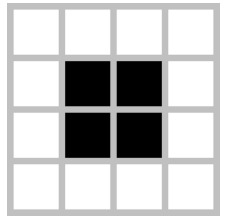


Conway's Game of Life

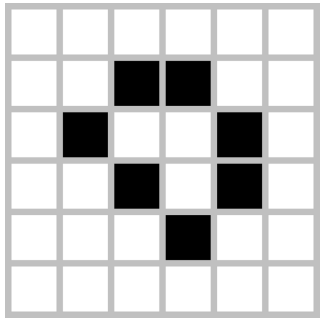
Each cell has two states: either dead (white) or alive (black).

1) Any live cell with two or three live neighbors survives.
2) Any dead cell with three live neighbors becomes a live cell.
3) All other live cells die in the next generation. Similarly, all other dead cells stay dead.

Still life

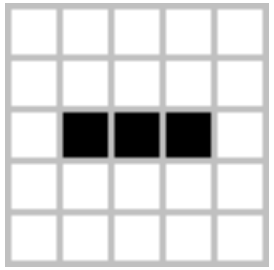


“Block”

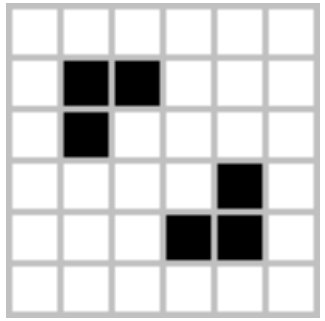


“Loaf”

Oscillators

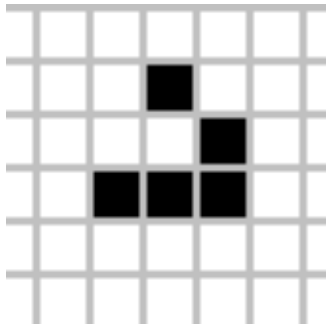


“Blinker”

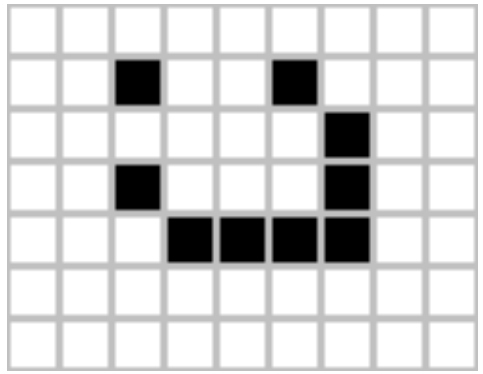


“Beacon”

Spaceships

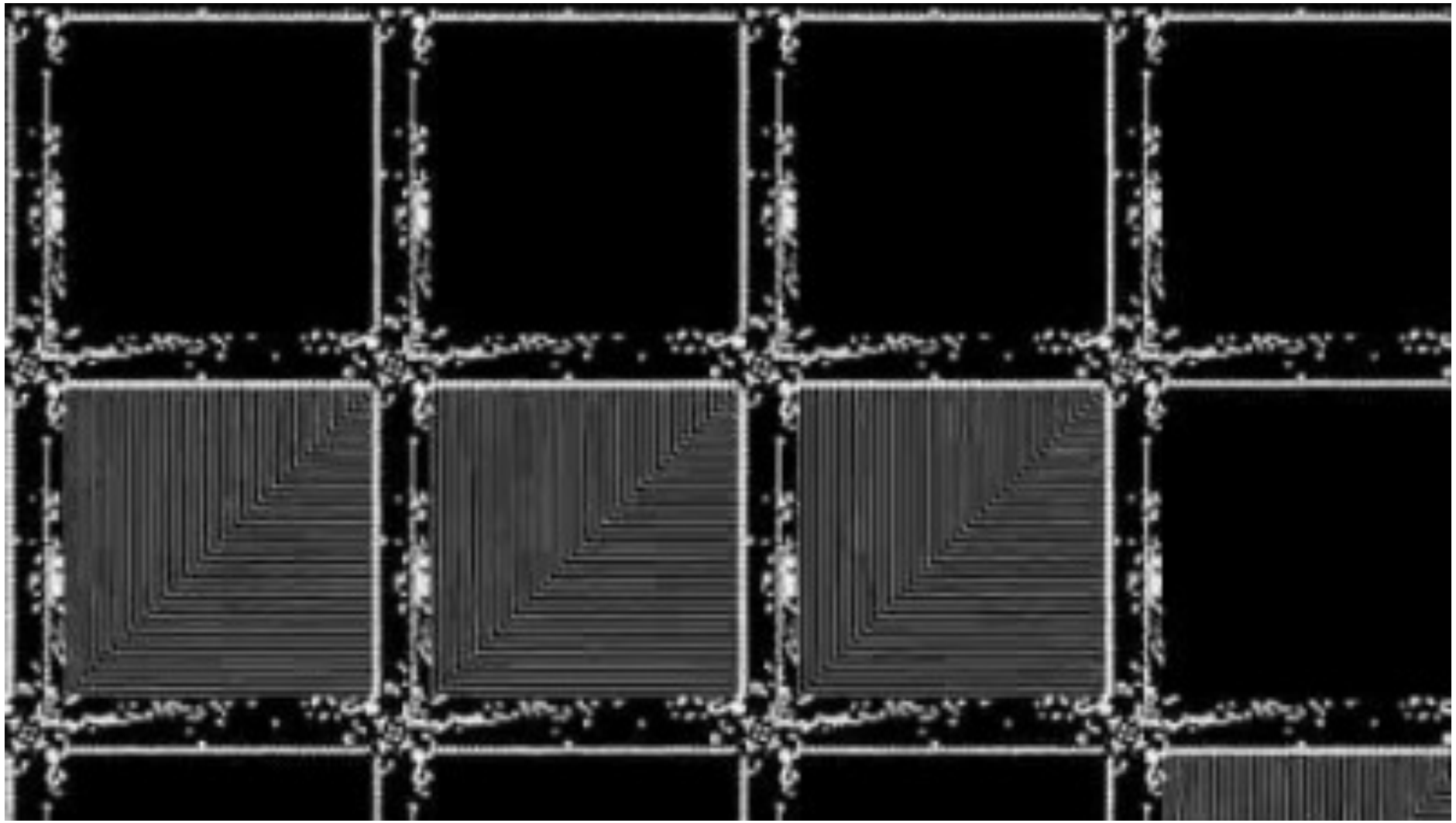


“Glider”



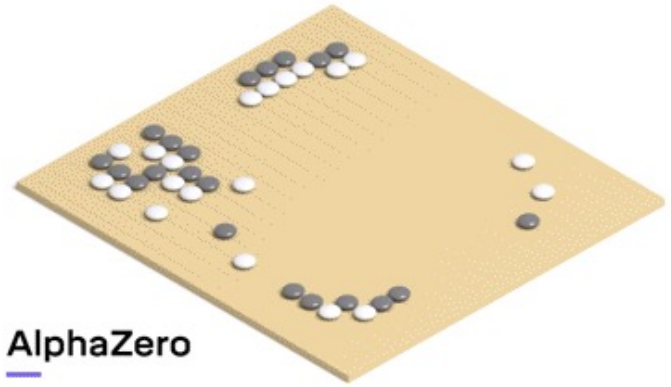
“Lightweight Spaceship”

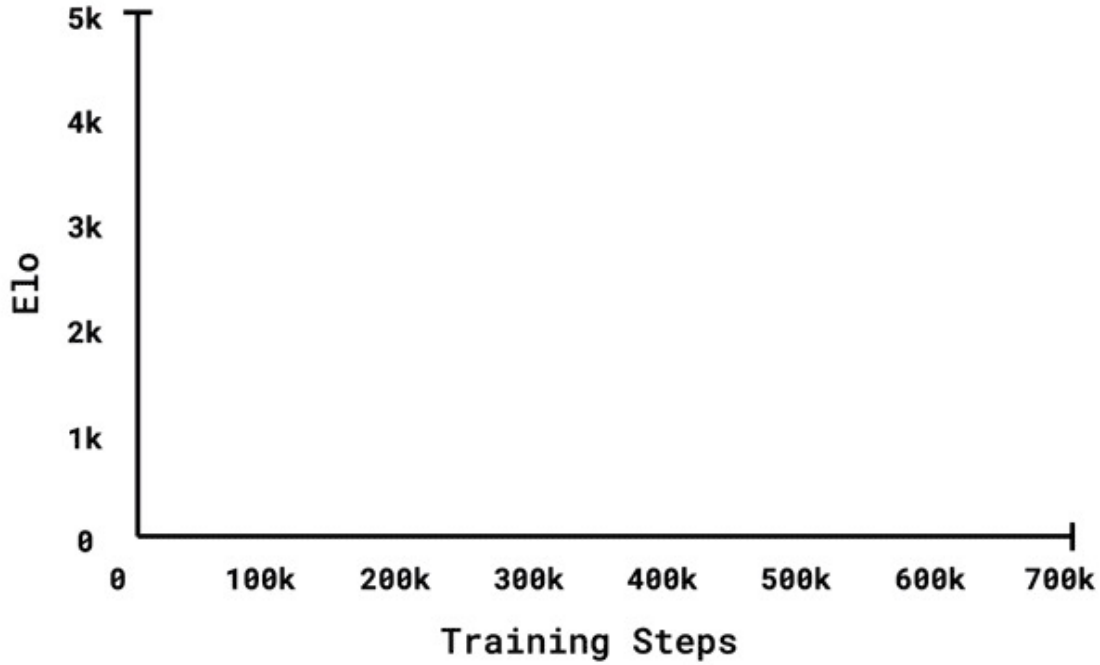
Life in Life



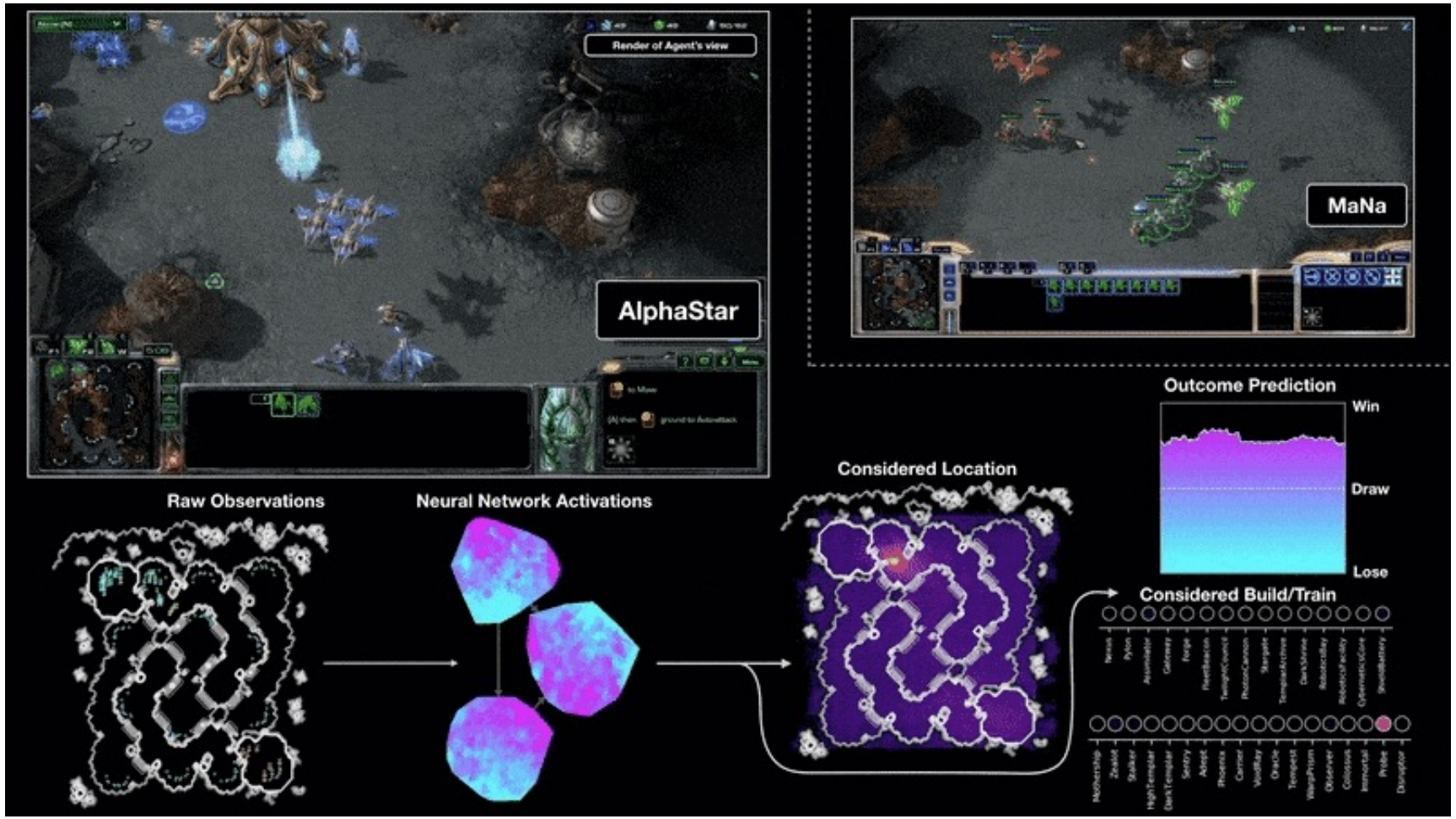
Deep reinforcement learning: (far) more sophisticated agents

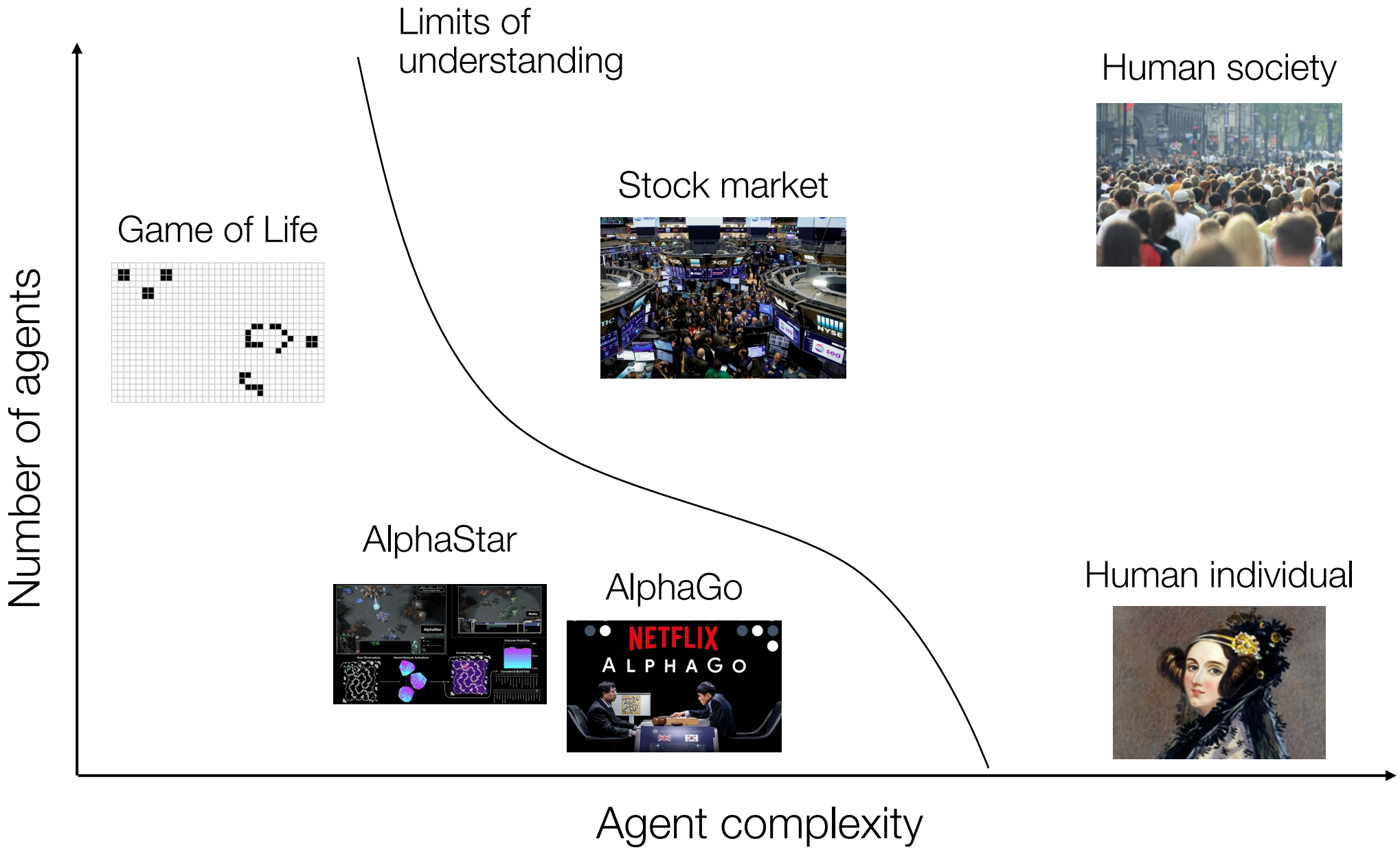
Instead of operating on fixed rules, RL agents are able to learn and adapt





Beating world champions at Starcraft




Limits of
understanding
Human society
Stock market
Game of Life
AlphaStar
AlphaGo
NETFLIX
ALPHAGO
Human individual
Number of agents
Agent complexity

A system with multiple intelligent agents

Outline of the Course

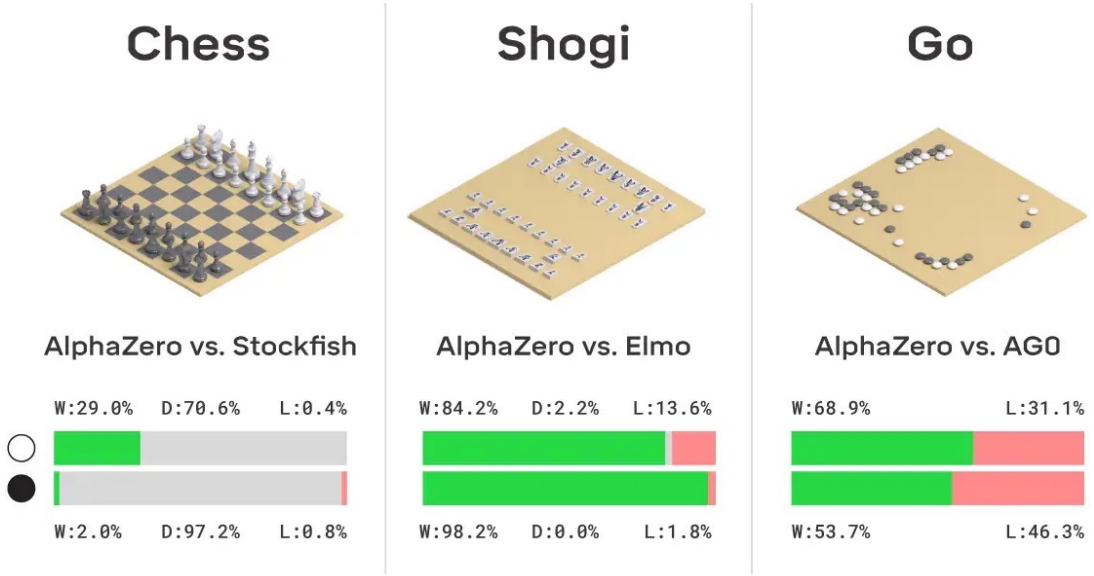
We want to find out how…

- Individual agents learn and interact with the environment
- Multiple agents come together to create emergent phenomena
- To relate this model to humans and social behavior

Lecture 1	Lecture 2	Lecture 3	Lecture 4	Lecture 5	Lecture 6
Introduction and the RL problem	How computers learn	How people learn	Multi-agent systems	Interactions on graphs	Complex systems science

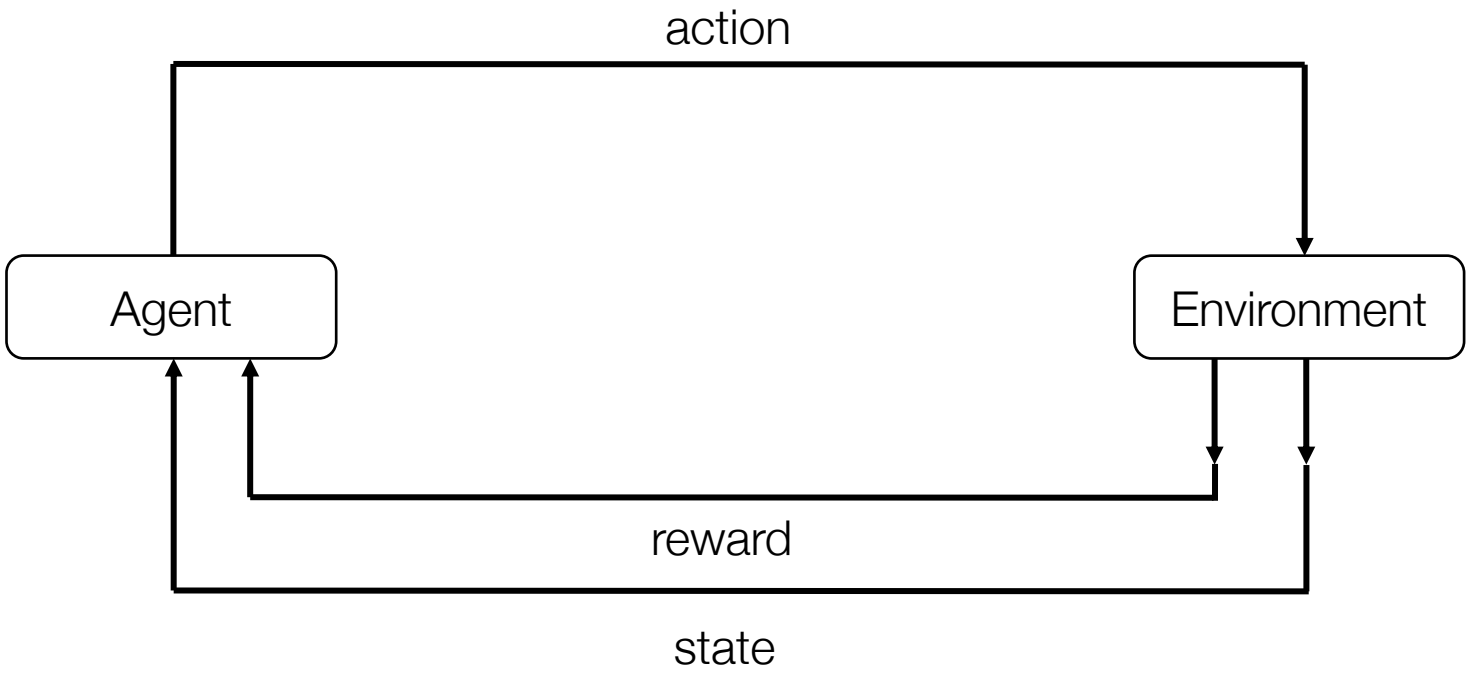
What is reinforcement learning?

Reinforcement learning (RL) is concerned with an agent's ability to learn by interacting with its environment.

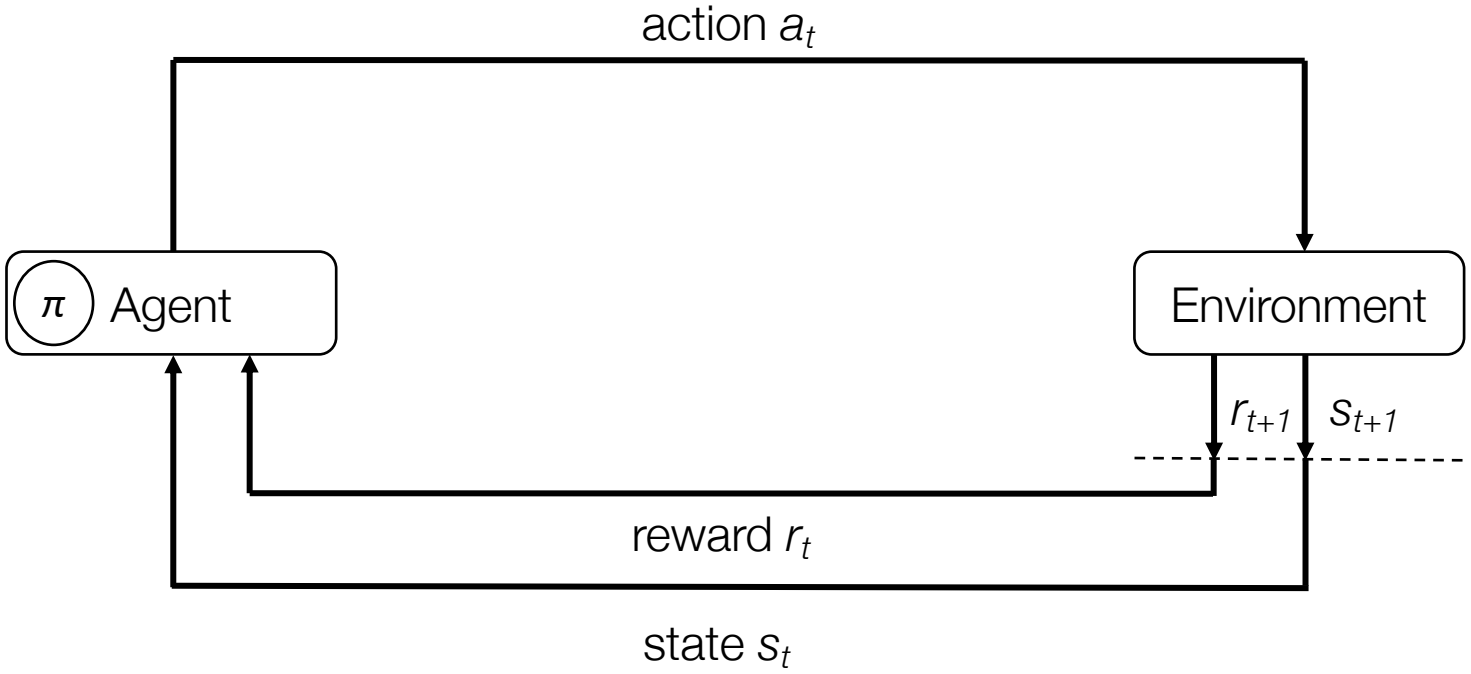


How do we mathematically model this learning process?

The agent-environment interaction loop



The agent-environment interaction loop

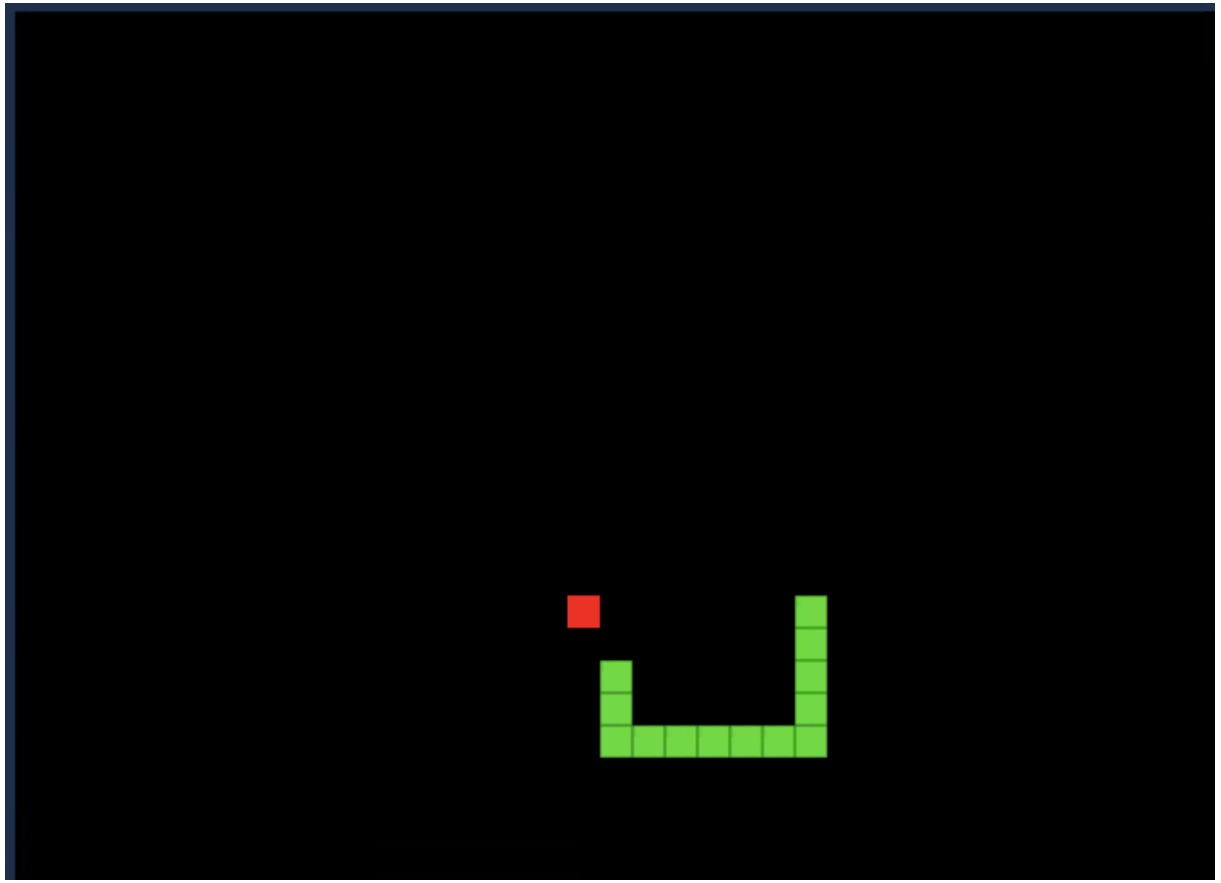


(s_0, a_0, r_1, s_1)
(s_1, a_1, r_2, s_2)
…
$(s_t, a_t, r_{t+1}, s_{t+1})$

Agent's goal: Find a **policy** $\pi: S \longrightarrow A$ mapping states to actions that maximizes reward (i.e. maximize $r_0 + r_1 + \ldots + r_T$)

A policy states that if the agent observes state s_t, it should take action $a_t = \pi(s_t)$

Example: RL to make AI Snake



The snake's goal is to eat as much food as possible without crashing into the walls or itself.

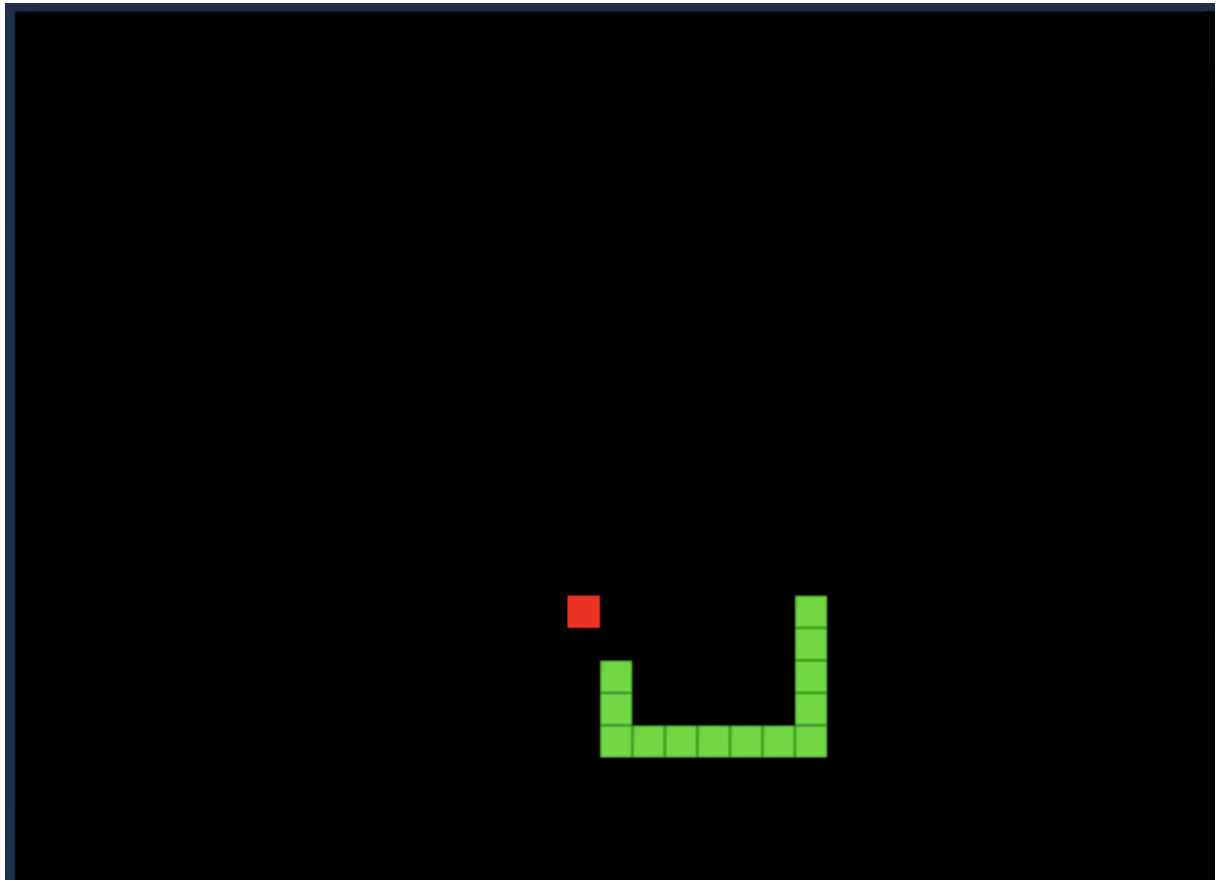
At each timestep *t*, the snake can move up, down, left, and right in the grid.

Each time it eats a food item, the length of the snake grows and a new food item spawns randomly.

If it crashes, the game is over.

We can use RL to train the AI to learn the rules and play on its own.

The RL model of Snake



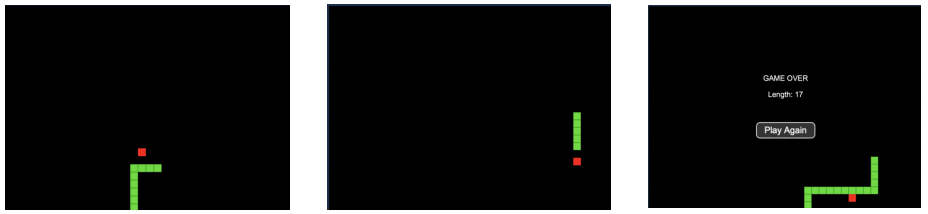
Components of the RL model:

Agent? Snake

Environment? Walls, apples, snake's body

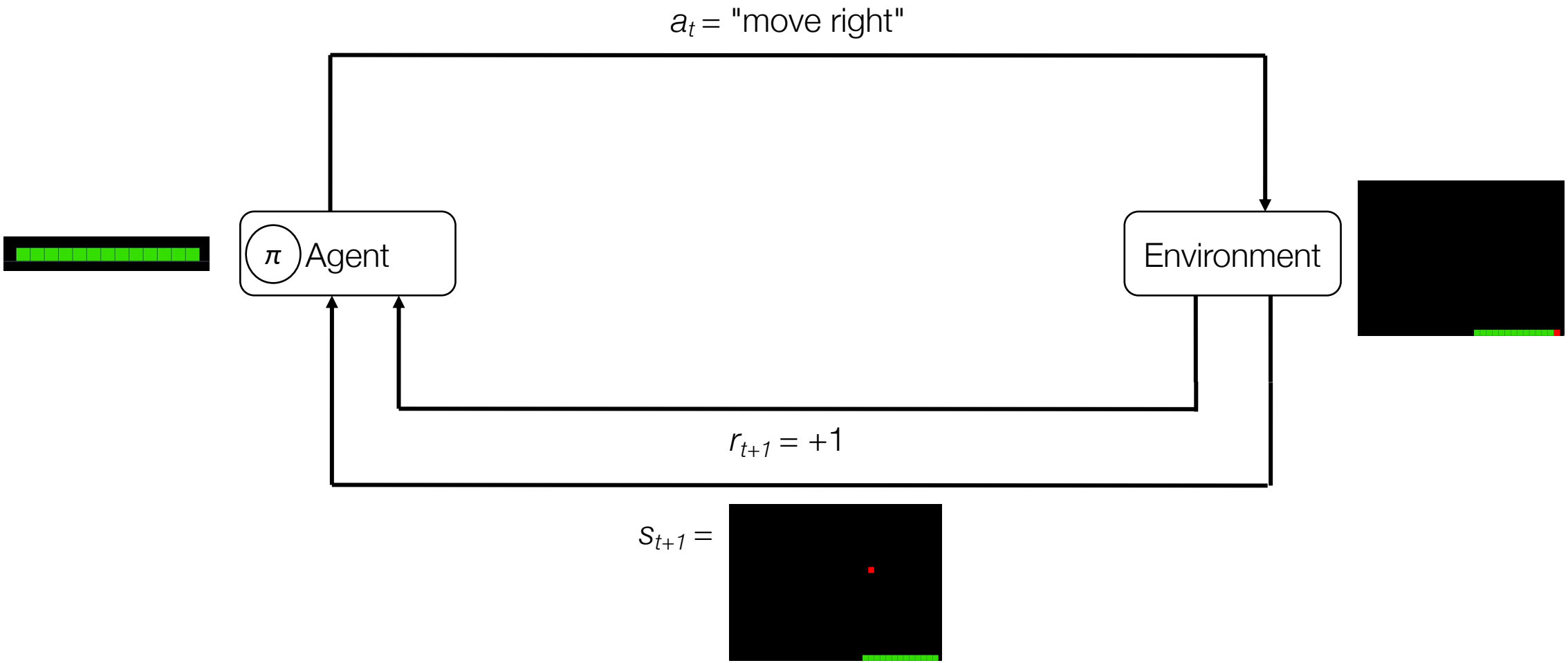
Actions? Moving up/down/left/right

States?

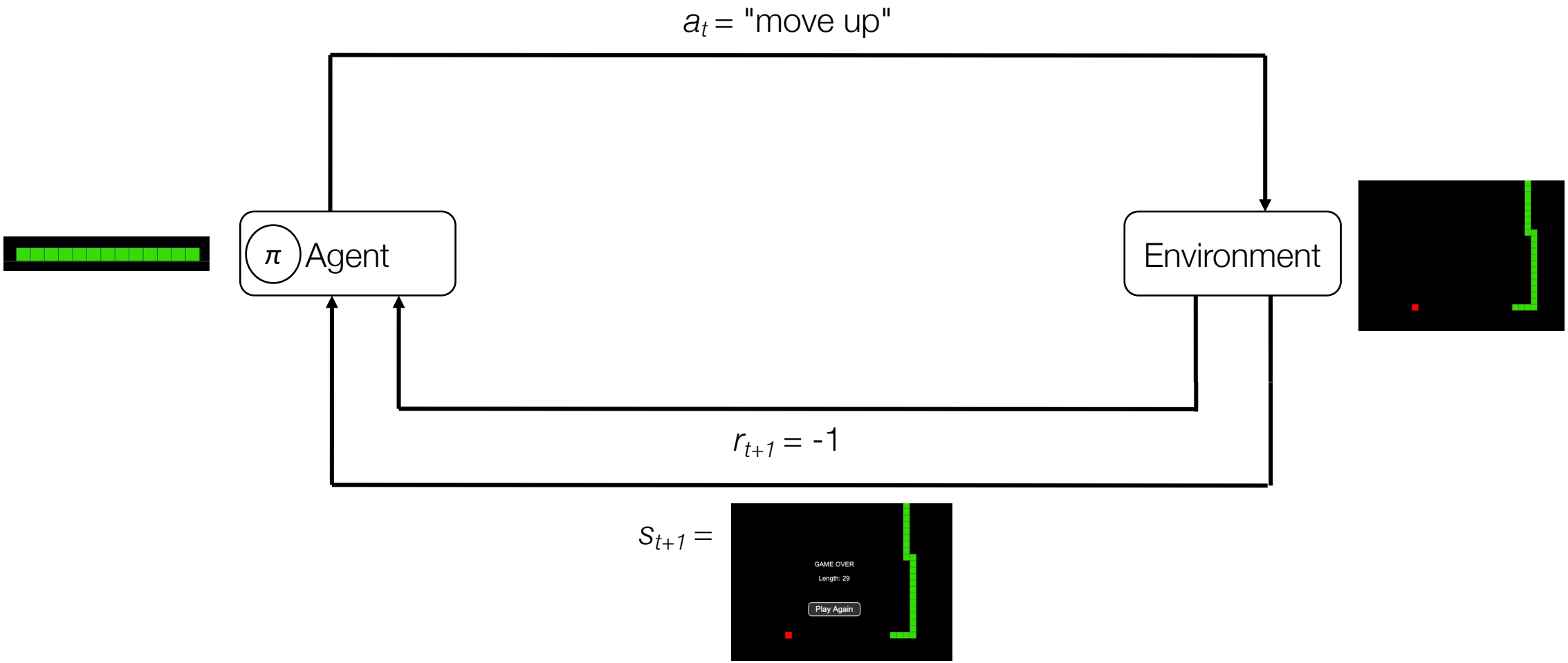


Rewards? (+1) for eating food
(-1) for crashing
0 for everything else

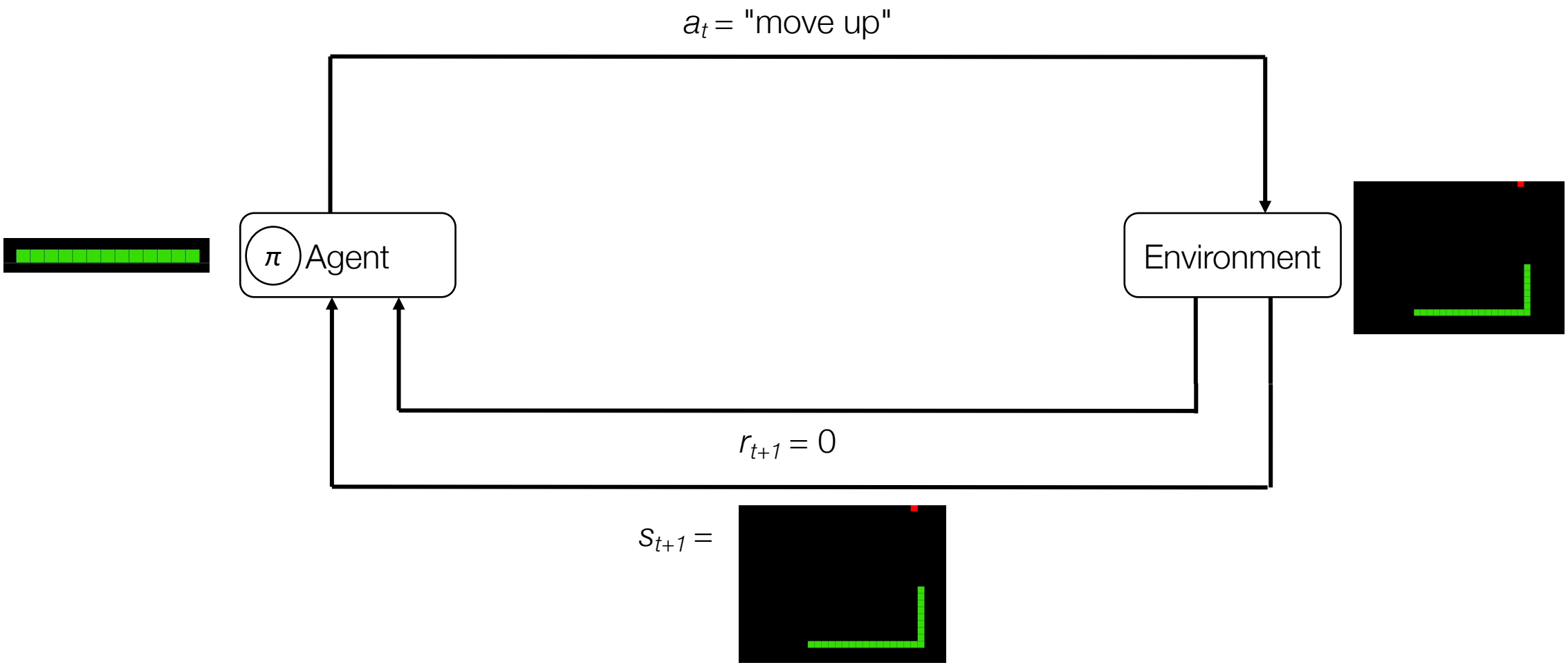
Example iteration (positive reward)



Example iteration (negative reward)



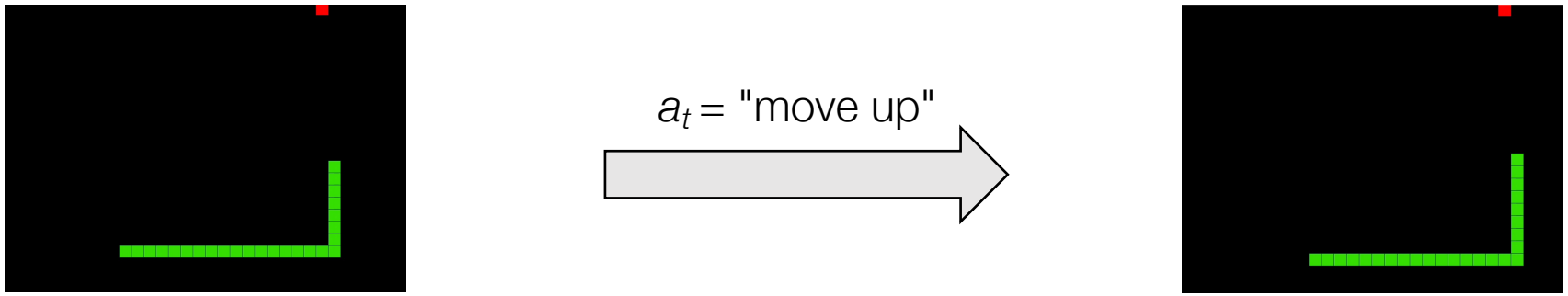
Example iteration (no reward)



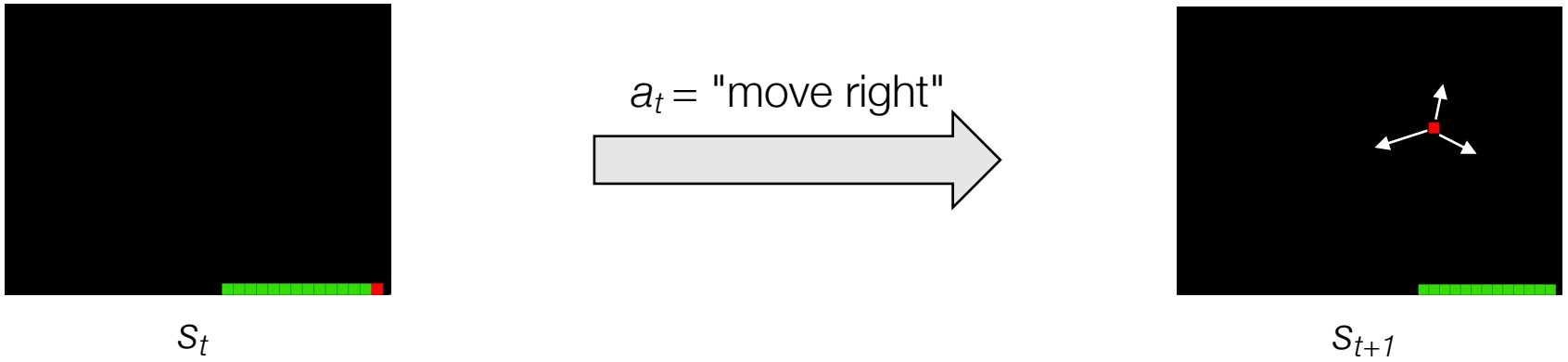
Through trial and error, the snake learns how to play!

Modelling uncertainty

In the Snake example, most states are fully determined by the snake's actions



However, there is uncertainty when new food spawns in a random location



We need to model the probability of the state transitioning from s_t to s_{t+1}

State transition probabilities

Ideally, we want the probability of transitioning to state s_{t+1} to depend only on:

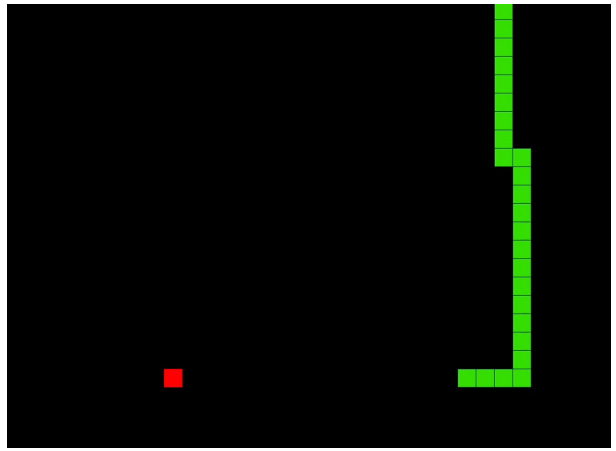
- Current state s_t
- Action a_t

s_t should summarize all immediate and previous information such that the agent doesn't have to keep track of the complete history (s_0, a_0, r_1, s_1, …, s_{t-1}, a_{t-1}, r_t, s_t)

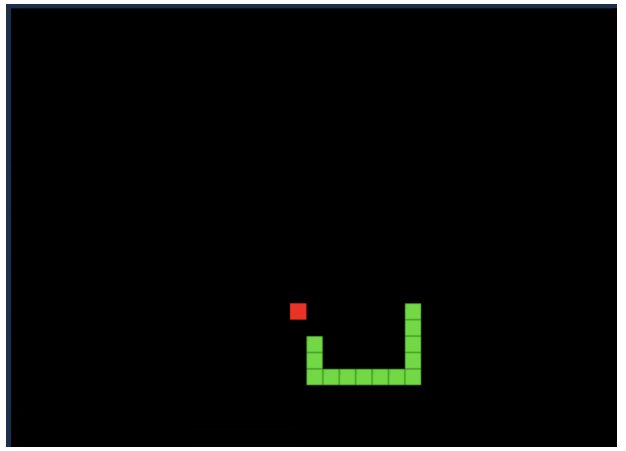
Such states are said to be **Markovian**, or have the **Markov Property.**

The value function

+1 reward (eating an apple)



-1 reward (death)

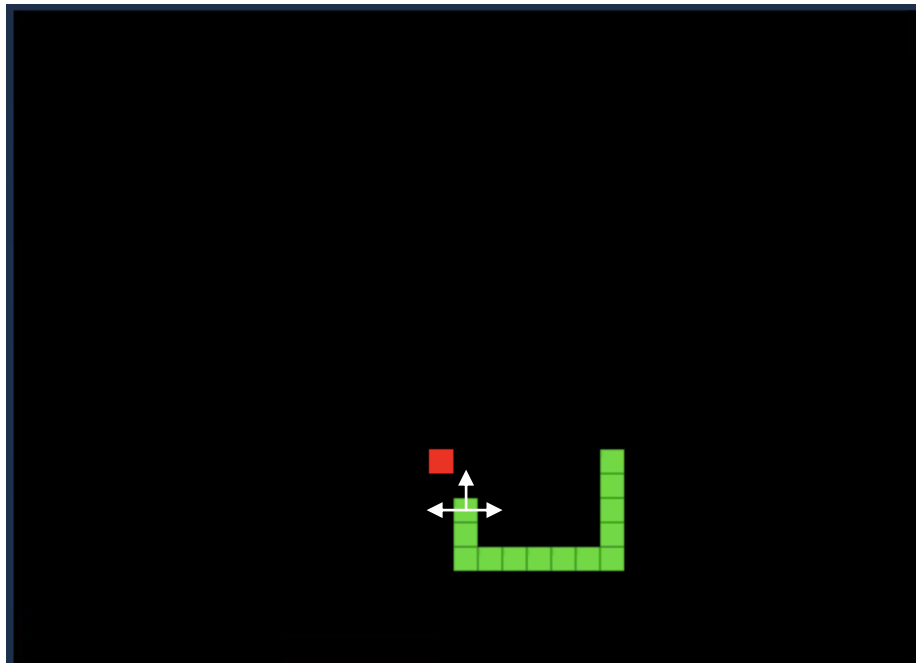


0 reward: what to do??

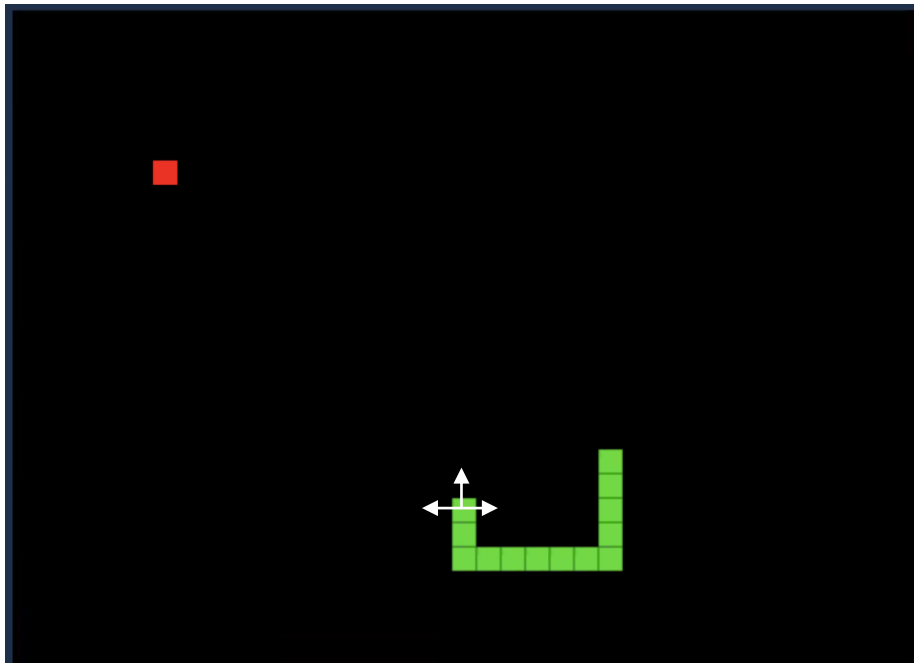
Even though the agent is getting zero reward, some states are better than others because they allow us to get future rewards. We assign these states higher **values** $v_\pi(s)$.

Note: the reward is computed by the environment, while the value is computed by the agent.

Some states are better than others



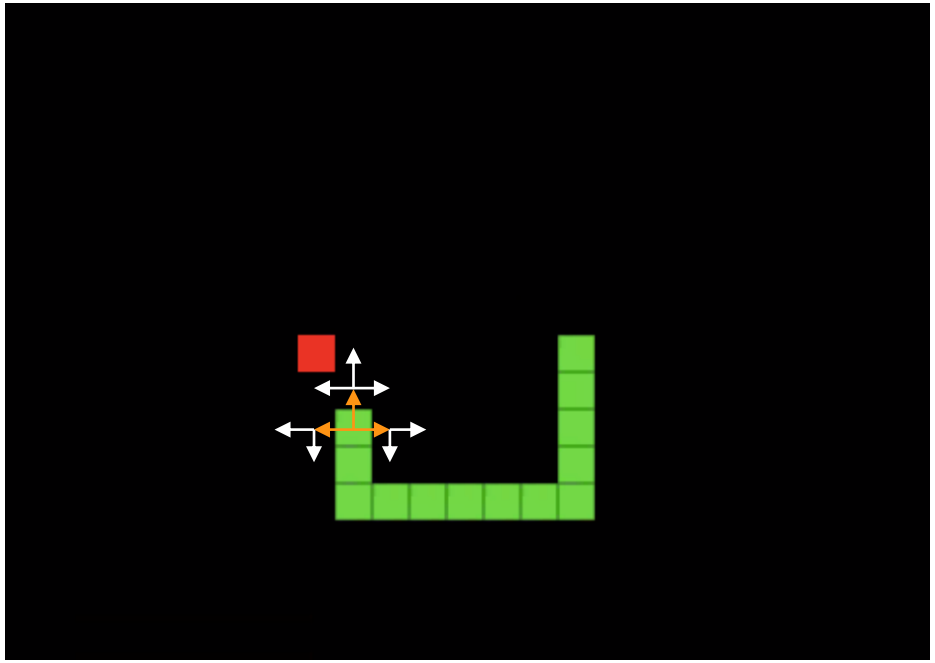
0 reward



0 reward

To determine the value of the current state, we want to **look ahead** to possible future states and see how close we are to getting the reward.

Estimating the value function



Since the state is Markovian, there is a unique value for each state and policy choice

$$v_\pi(s_t) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} R_{t+k+1} | S = s_t\right]$$

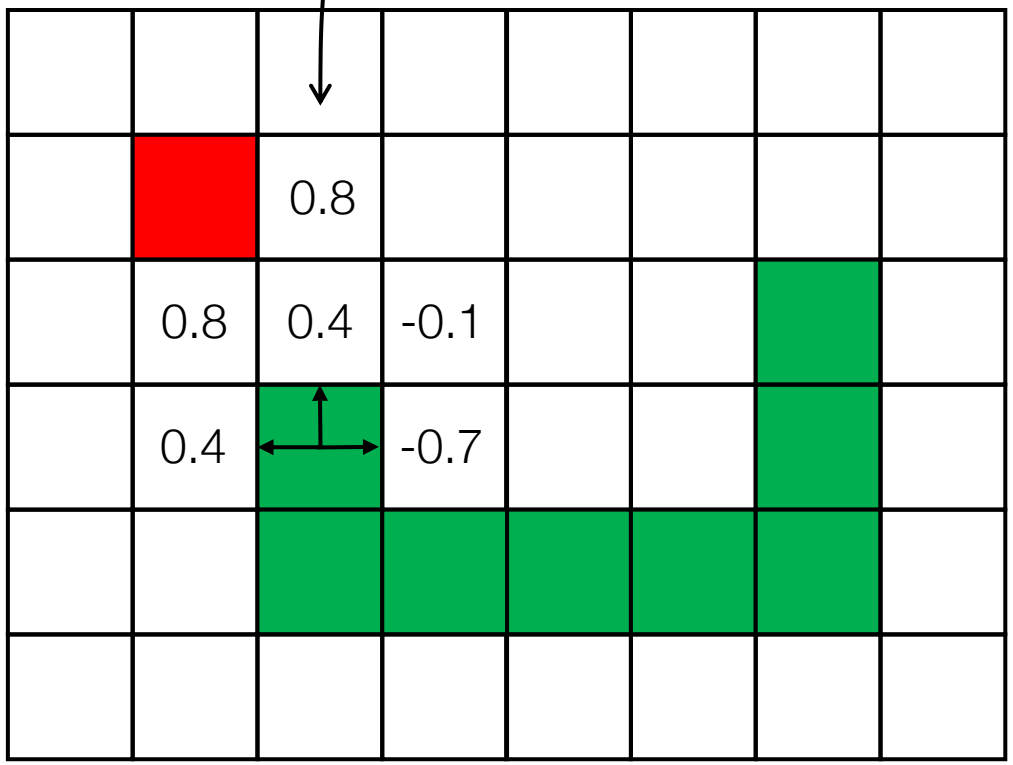
Average over all the possible trajectories under a given policy

Of all the future rewards obtained in that trajectory

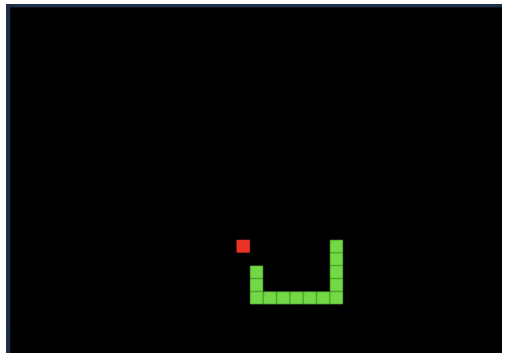
Starting from our current state

Estimating the value function

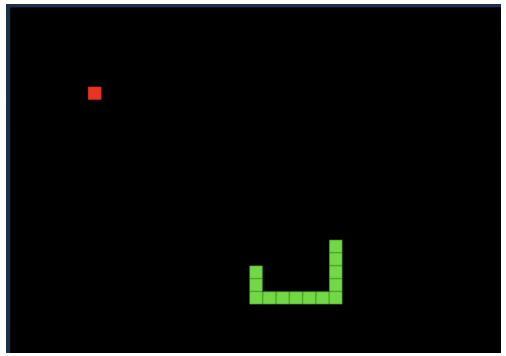
Each state is assigned a value $v_\pi(s_t) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} R_{t+k+1} | S = s_t\right]$



This solves our problem of not knowing where to go:



0 reward



0 reward

The optimal value function is given by

$v_*(s) = \text{maximize}_\pi(v_\pi(s))$

Bellman's equation

Since the state is Markovian, there is a unique value for each state and policy choice

Reward at the current state

Average reward from all the next states

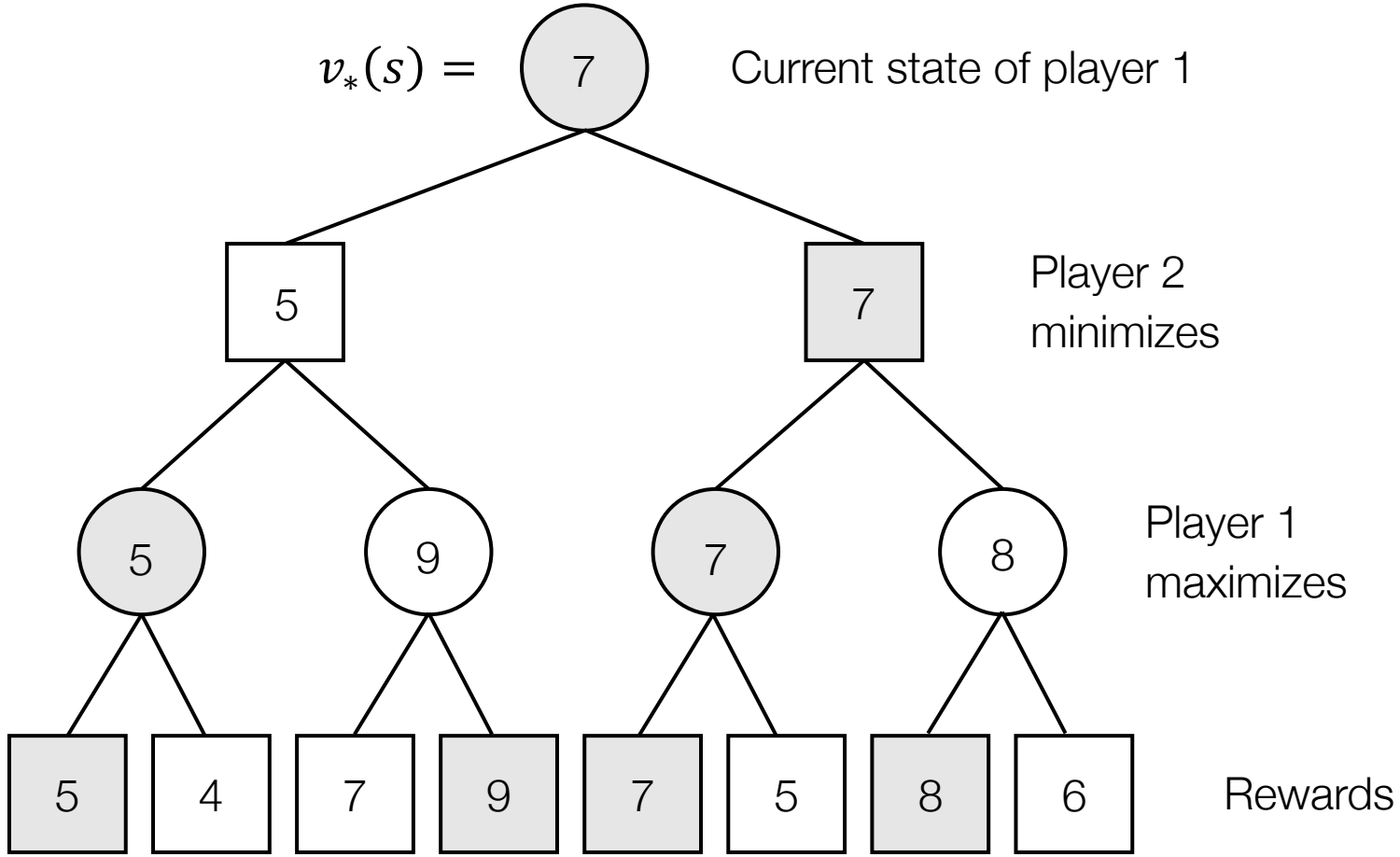
$$v_\pi(s_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} R_{t+k+1} | S = s_t \right] = r(s_t, a_t) + \mathbb{E}[v_\pi(s_{t+1})]$$

Bellman's equation	"If I know the shortest path from Boston to DC runs through New York, then once I get to New York, I should just follow the shortest path from New York to DC."

Estimating the optimal value function

Bellman's equation

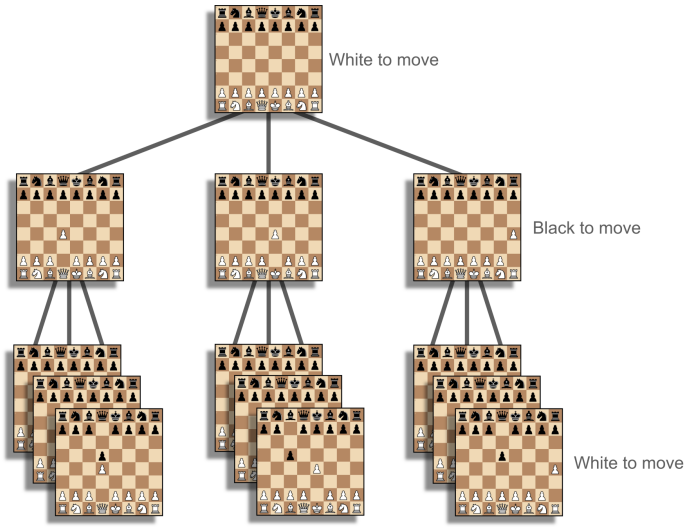
"If I know the shortest path from Boston to DC runs through New York, then once I get to New York, I should just follow the shortest path from New York to DC."



Combinatorial explosion

This fact helps us cut down our search space since we don't need to worry about everything that happens from Boston to D.C. once we get to New York.

But, computing the value function is still ***very hard***, particularly when we have **limited data.**



Current state

3 moves

9 moves

Grandmaster level: depth of 20

$$3^{20} = 3{,}486{,}784{,}401$$

Towards deep reinforcement learning

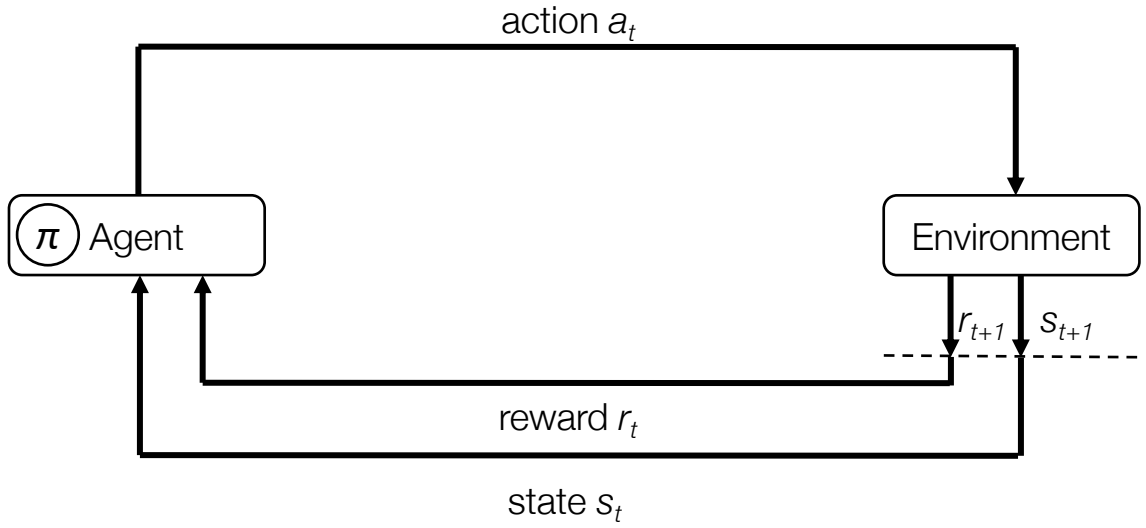
Grandmaster level: depth of 20

$$3^{20} = 3486784401$$

They're not supercomputers… they incorporate intuition.

Next lecture: deep reinforcement learning

Recap of RL



- **Policy** π: $S \longrightarrow A$ such that $a_t = \pi(s_t)$
- Agent's goal = find policy that maximizes total reward

- Our life is easier when states are **Markovian** (the future depends only on the current state and not the past)
- **Bellman's equation**: $v_\pi(s_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} R_{t+k+1} | S = s_t \right] = r(s_t, a_t) + \mathbb{E}[v_\pi(s_{t+1})]$
- Optimal value function: $v_*(s) = \text{maximize}_\pi(v_\pi(s))$

References and additional resources

- Reinforcement Learning: An Introduction by Richard S. Sutton and Andrew G. Barto
- Teach AI to Play Snake – Reinforcement Learning Tutorial video
- AlphaGo documentary
- OpenAI Spinning Up RL Tutorial