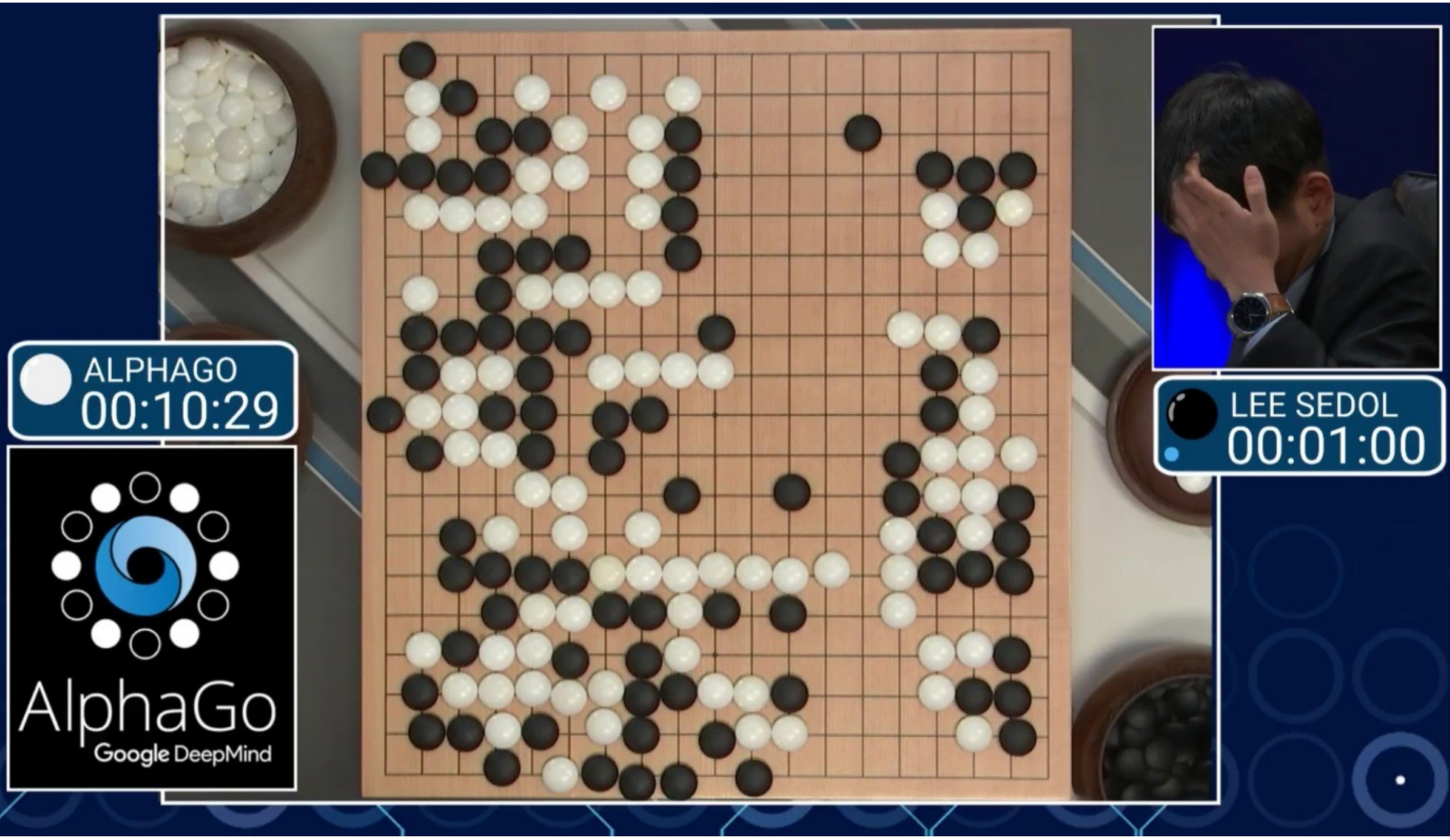
Lecture 11:
Reinforcement Learning

Artificial Intelligence

Julian Togelius


ALPHAGO
00:10:29
AlphaGo
Google DeepMind
LEE SEDOL
00:01:00

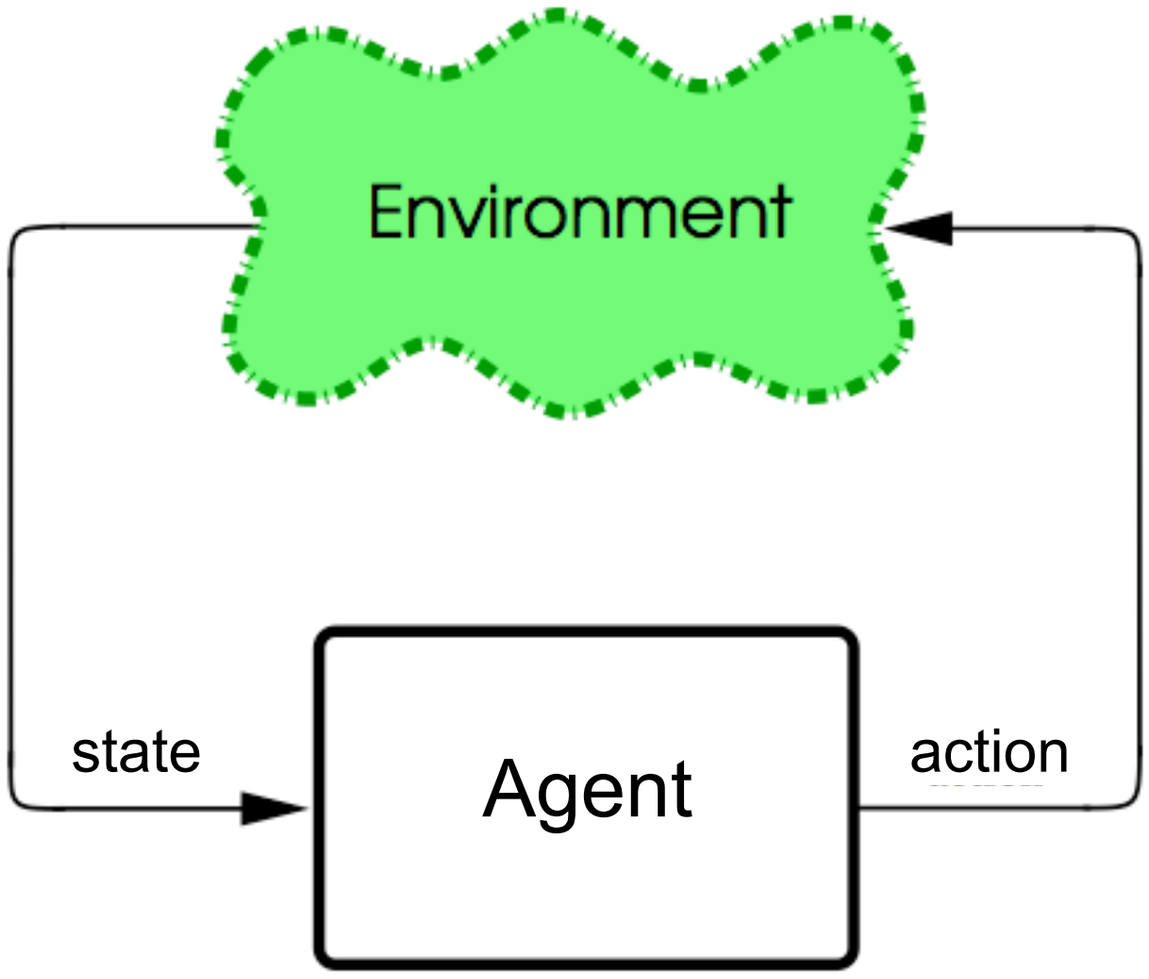

ima...
217 4 1


PHILADELPHIA
ZOO

Limits of tree search

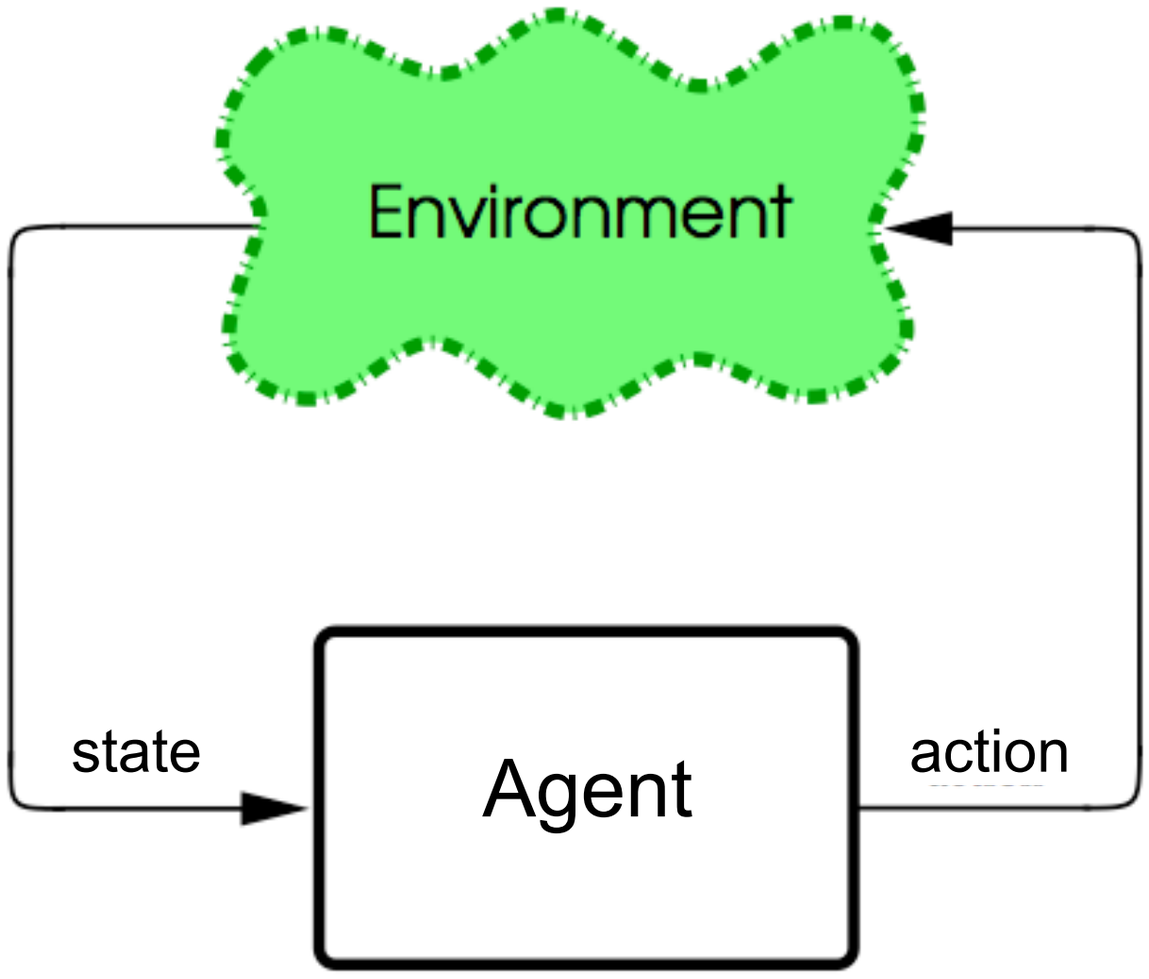
- Need a proper forward model
- Trees might be huge
- State evaluation might be hard
- Each action selection takes a lot of time

Sequential Decision Tasks



Agent sees state of the environment at each time-step and must select best action to achieve a goal. How can we learn what the best actions are?

Sequential Decision Tasks



s_t : state of the environment at time *t*

a_t : action taken by agent at time *t* after seeing s_t

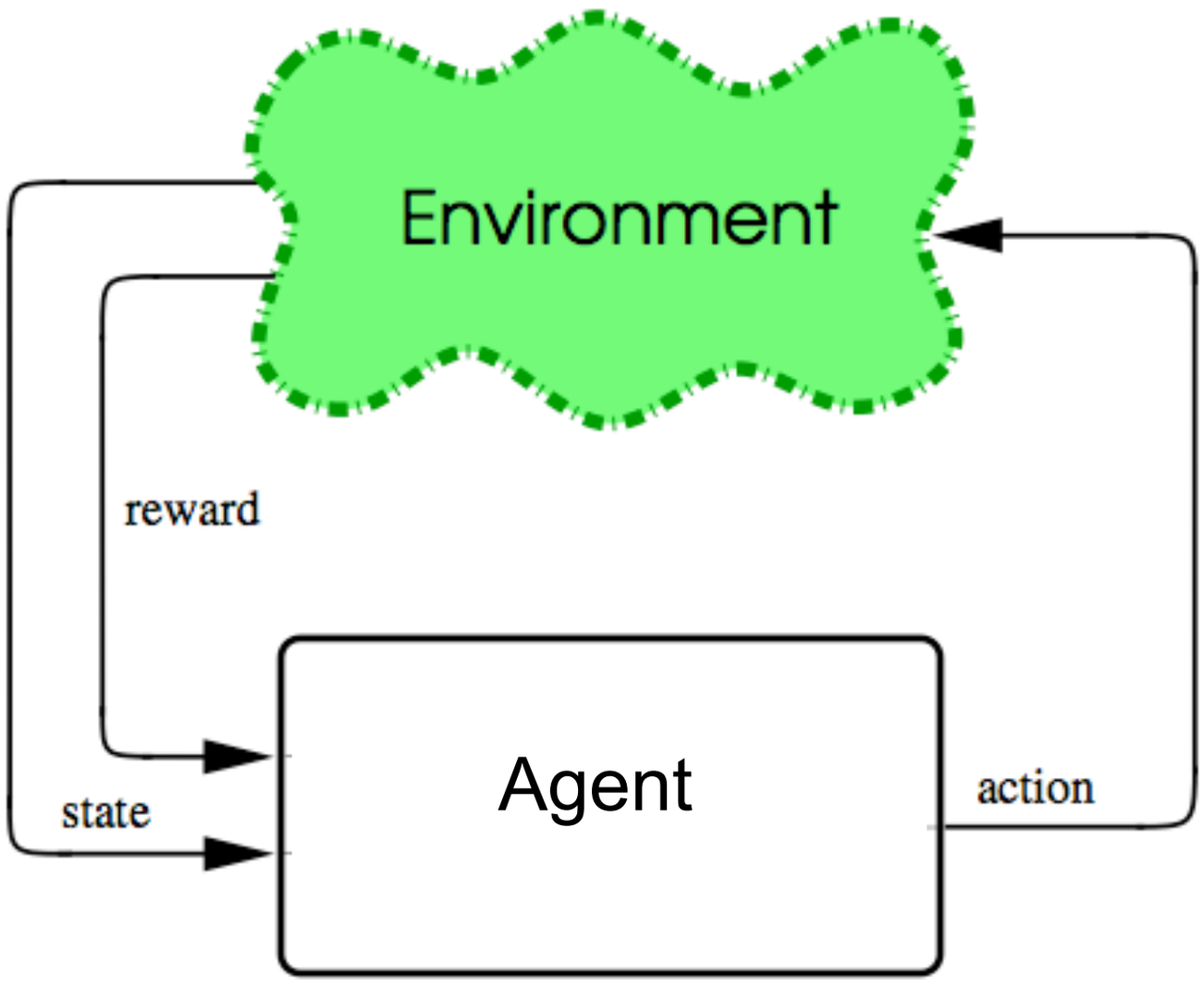
Decision sequence:

$$s_t \xrightarrow{a_t} s_{t+1} \xrightarrow{a_{t+1}} s_{t+2} \xrightarrow{a_{t+2}} s_{t+3} \xrightarrow{a_{t+3}} \ldots$$

Sequential Decision Tasks

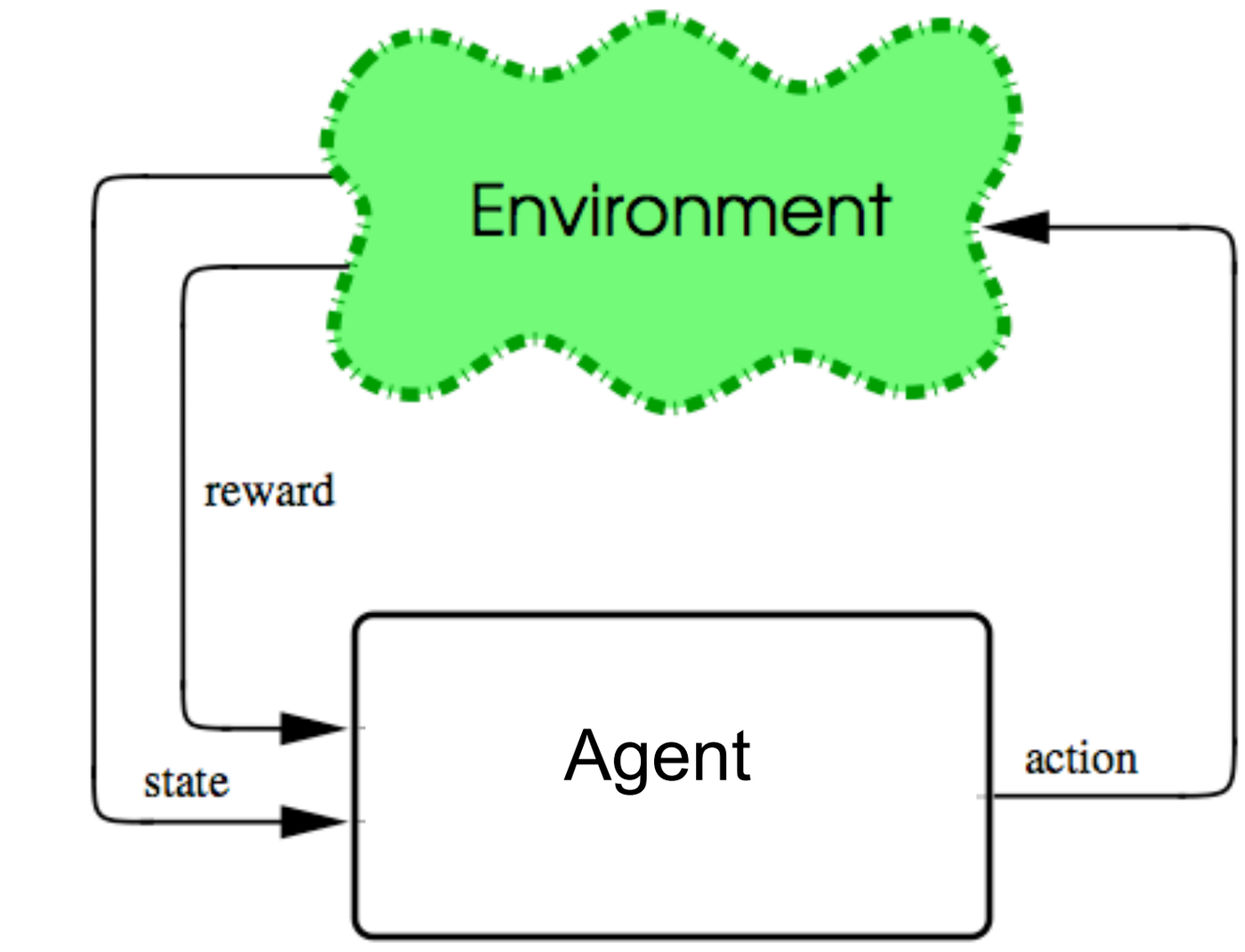
- Autonomous robotics
- Controlling chemical processes
- Network routing
- Game playing
- Stock trading

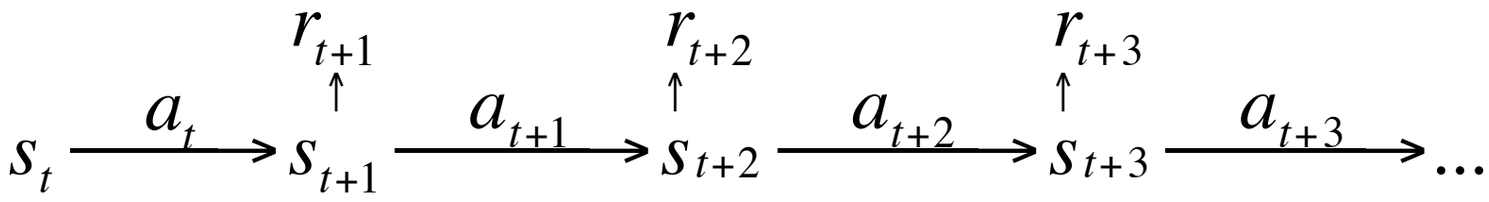
Reinforcement Learning Problem



Agent receives a *reinforcement* (“good” or “bad” behavior)

Reinforcement Learning Problem





Pac-Man

- One example:
  - 1 if you eat a pill
  - -10 if you get caught by a ghost
  - 2 if you eat a power pill or eat a ghost
  - 0 otherwise
- Another example:
  - -1 every time step
  - 1000000 if you win the level

Pac-man Rewards

		10
		

Actions: a & b

Model

a = down b = right	a = down b = right	10
a = up b = right	a = down b = left	a = down b = left
a = up b = right	a = up b = right	a = up b = left

Actions: a & b

State Utilities

a = down b = right	a = down b = right	10
a = up b = right	a = down b = left	a = down b = left
a = up b = right	a = up b = right	a = up b = left

8.1 9

7.29 6.56 5.9

6.56 5.9 5.3

Discount (γ): 0.9

Pac-man Outcome

b	b	10
a		
a		

Actions: a & b

Bellman Equation

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} p(s' \mid s, \pi(s)) V^{\pi}(s')$$

- R(s) - The reward for the current state
- γ - How much to discount future rewards ($0 \leq \gamma \leq 1$)
- P(s’ | s, a) - model of future states
- π(s) - action taken given state s
- Sum up the utilities of all future states

Agent Policy

- The policy implements the agents behavior
- In general, the policy could be stochastic:

$$\pi(s,a) = \Pr\{a_t = a \mid s_t = s\}$$

policy gives the probability of taking action *a* in state *s*

- Often the policy is deterministic and we can write:

$$\pi(s) \rightarrow a$$

for each state the policy says which action to use

Markov Decision Processes

a finite set of states : $s \in S$ also known as the *state-space*

a finite set of actions : $a \in A$ also known as the *action-space*

state transition probabilities : $P^{a}_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$

reward function : $R^{a}_{ss'} = \mathrm{E}\{r_{t+1} \mid s_t = s, a_t = a\}$

a policy : $\pi(s,a) = \Pr\{a_t = a \mid s_t = s\}$

…and the Markov property must hold.

Markov Decision Processes

a finite set of states : $s \in S$

a finite set of actions : $a \in A$

state transition probabilities : $P^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$

reward function : $R^a_{ss'} = \mathrm{E}\{r_{t+1} \mid s_t = s, a_t = a\}$

model

a policy : $\pi(s,a) = \Pr\{a_t = a \mid s_t = s\}$

…and the Markov property must hold.

The Markov Property

$$P^a_{ss'} = \Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_0, a_0, r_0\}$$

$$\equiv$$

$$P^a_{ss'} = \Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}$$

This just means that the probability of the next state and reward only depend on the immediately preceding state and action!

It doesn't matter what the happened before that!

Example Transition Matrix

Each action will have a transition matrix $P^a_{ss'}$

From: \ To:	s_1	s_2	s_3	s_4
s_1	0	0.1	0.3	0.6
s_2	0.5	0	0	0.5
s_3	0	0.4	0	0.6
s_4	0	0.2	0.3	0.5

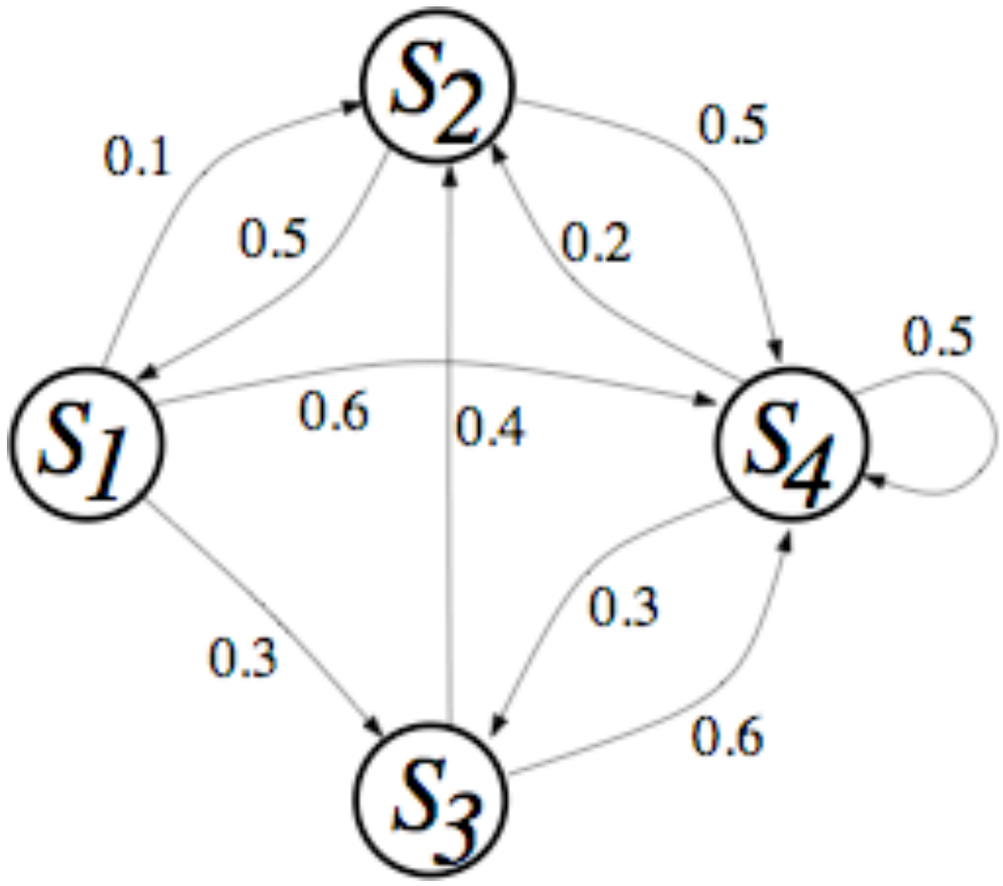
- example with four states
- each entry gives the probability of going from one state to another *if* the action is taken
- each row sums to 1.0

e.g. $P^a_{s_3 s_4} = 0.6$, there is a 60% chance of going to state 4 when action *a* is taken in state 3

State Transitions (cont.)

	s_1	s_2	s_3	s_4
s_1	0	0.1	0.3	0.6
s_2	0.5	0	0	0.5
s_3	0	0.4	0	0.6
s_4	0	0.2	0.3	0.5

=



transition graph

All edges leaving state add to 1.0

More About R

$$R_t = \sum_{k=0}^{T} \gamma^k \overbrace{r_{t+k+1}}^{\text{reward}}, \quad 0 \le \gamma \le 1$$

- R is also known as the return. It is how much reward the agent will receive from time t into the future.

- If γ is close to 0, then the agent cares more about selecting actions that maximize immediate reward: shortsighted

- if γ is close to 1, then the agent takes future rewards into account more strongly: farsighted

Policy

The goal is to learn a *policy* that maximizes the reward r over the long term:

$$R_t = \sum_{k=0}^{T} \gamma^k \underbrace{r_{t+k+1}}_{\text{reward}}, \quad 0 \le \gamma \le 1$$

where γ is the discount rate. γ=1 means the all rewards received matter equally. γ<1 means rewards further in the future are less important.

Why Are Reinforcement Learning Problems Hard to Solve?

- Have to discover behavior from scratch
- Only have scalar reinforcement to guide learning
- Reinforcement may be infrequent
- Credit assignment problem: How much credit should each action in the sequence of actions get for the outcome

Solving Reinforcement Learning Problems

- "Classical" approaches:
    - based on approximate dynamic programming (this and next lecture)
    - based on policy gradients (not covered here)
- Evolutionary approaches:
    - based on evolution, or other stochastic optimization methods (early part of the course)

Why Are Reinforcement Learning Problems Hard to Solve?

- Have to discover behavior from scratch
- Only have scalar reinforcement to guide learning
- Reinforcement may be infrequent
- Credit assignment problem: How much credit should each action in the sequence of actions get for the outcome

Solving Reinforcement Learning Problems

- "Classical" approaches:
    - based on approximate dynamic programming (this lecture)
    - based on policy gradients (not covered here)
- Evolutionary approaches:
    - based on evolution, or other stochastic optimization methods (early part of the course)

Solving Reinforcement Problems ("Classical" Approach)

- If the problem can be formulated as a Markov Decision Process, we can use a *value function* to represent how "good" each state is in terms of providing reward
- Use various methods to learn value function:
  - Dynamic Programming
  - Temporal Difference Learning (e.g. Q-learning)
  - Policy Search

Temporal Difference Methods (TD)

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

- Use the difference between the value the current state and the next visited state to update current state
- Don’t need values of **all** next states, which is good because in the real world we can only visit one at a time

Value Functions

Value function tells the agent how "good" it is to be in a given state

agent's policy

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} \qquad \text{where} \quad R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}, \quad 0 \le \gamma \le 1$$

reward

This says how much reward the agent can expect to receive in the future if it continues with its policy from state *s*

Value-function vs. Q-function

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

$$\mathrm{Q}^{\pi}(s,a) = \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

The Q-function implements one step of “lookahead”

It caches the value of taking each action in a given state

TD Control: Q-learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

- Uses the Q-value of the “best” action in the next state
- Version of TD using Q-function
- Because we have value for each action it can be used for on-line learning/control

Q-learning Algorithm

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
        $s \leftarrow s'$;
    until s is terminal

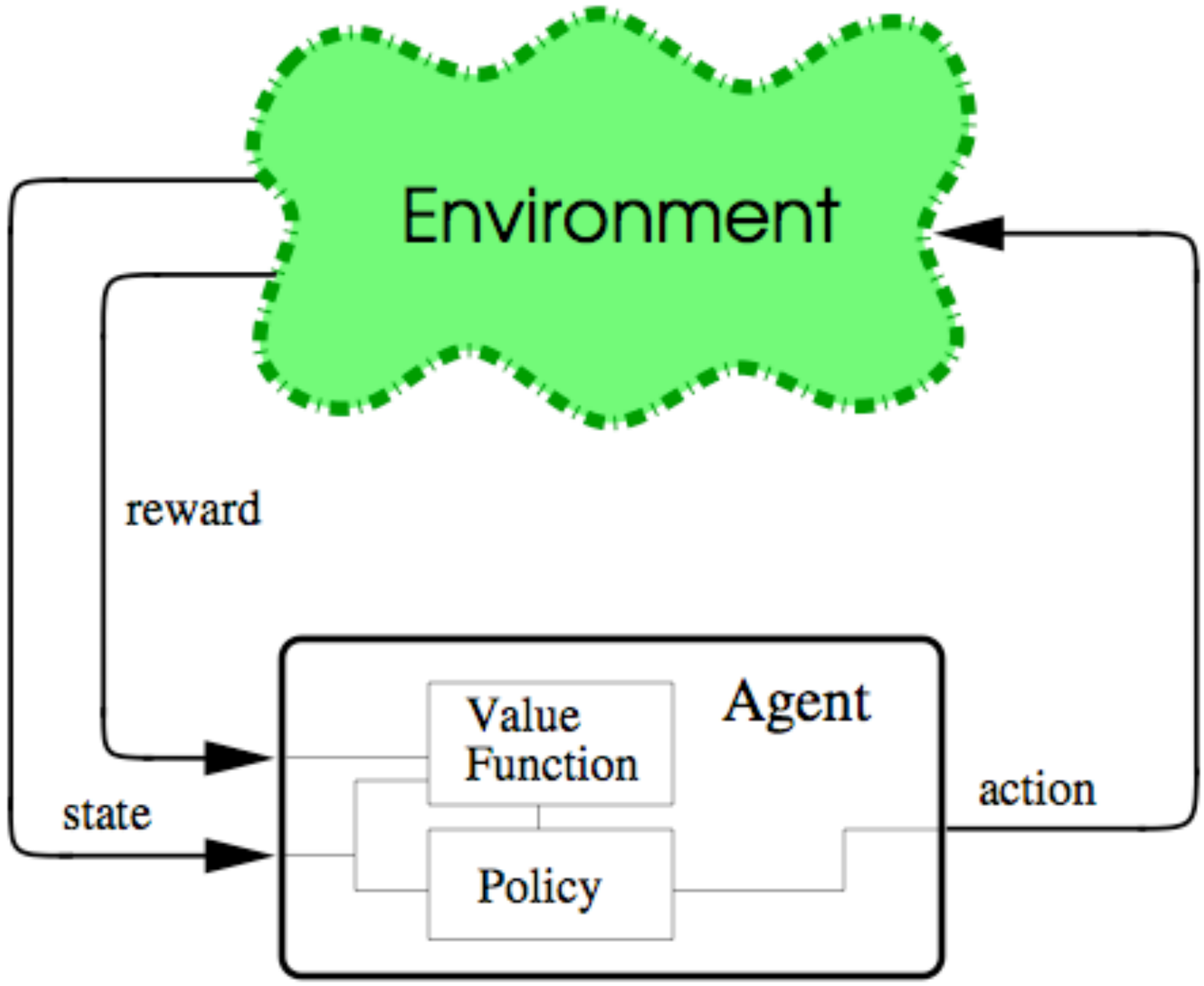
SARSA

- State-Action-Reward-State-Action
- Q-Learning: $Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$
- SARSA: $Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma Q(s',a') - Q(s,a))$

Exploration vs. Exploitation

- Exploitation: take good actions in each state already taken before to maximize reward
- Exploration: take a chance on actions that may have lower value in order to learn more, and maybe find true best action to later exploit

Need to balance the two!



Agent now consists of two components:

1. Value-function (Q-function)
2. Policy

A policy can be computed from the values

Reinforcement Learning Problem

A policy is the agent's function that maps states to actions:

$$\pi(s_t) \rightarrow a_t$$

For each state the agent encounters, the policy tells it what action to take.

The best policy is the one that selects the action in each state that leads to the highest long-term reward.

How to compute policy from *Q?*

- Greedy policy:

select action in each state with highest value

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$$

- ε–greedy policy:

select greedy action 1-$\varepsilon\%$ of the time and some other, random action $\varepsilon\%$ of the time (this will be useful later)

- Stochastic Policy:

Use action values to select actions probabilistically (more on this later)

OK, time to learn that value function!

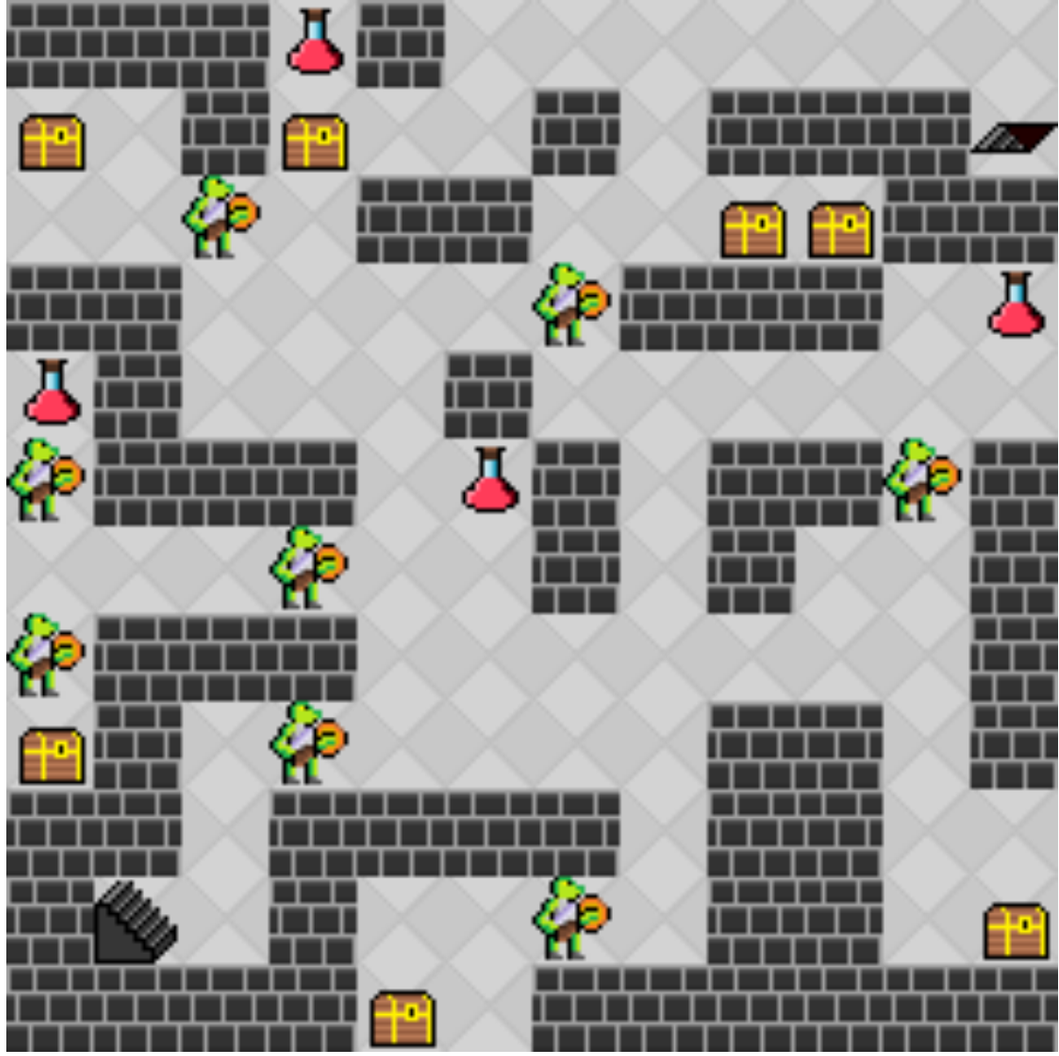
Limitations of Standard RL Methods

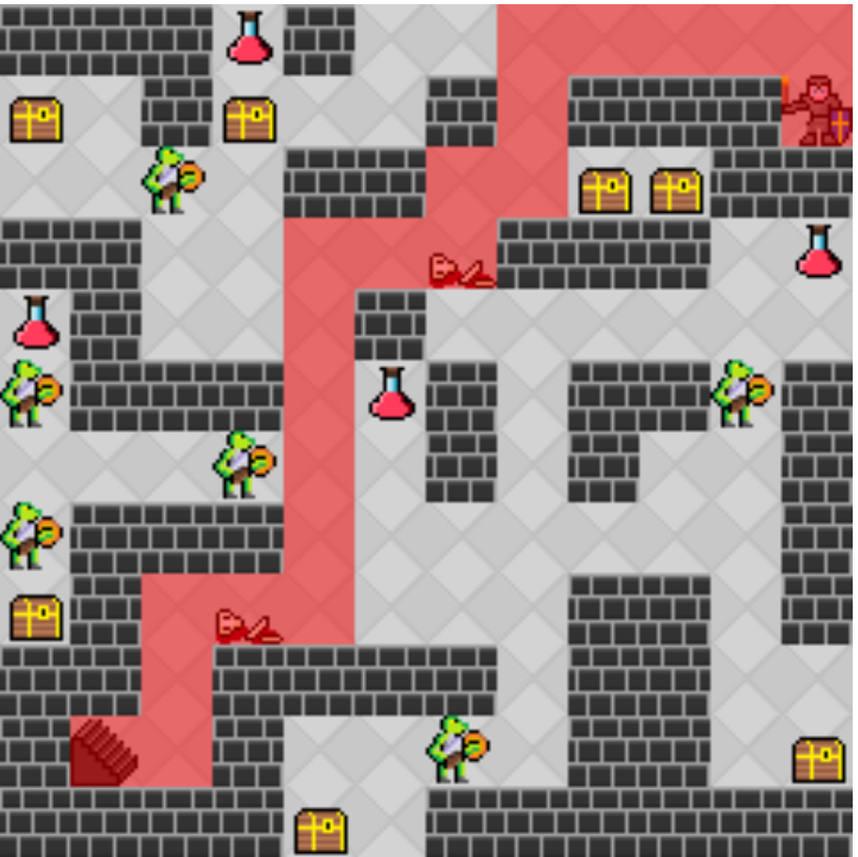
- Continuous state/action spaces
  - Cannot represent value-function with table
  - Need some kind of function approximator to represent V
  - Loss of convergence guarantees
- Partial Observability
  - Agent no longer sees underlying state
  - From the agent's perspective the Markov Property does not hold
  - Need to estimate underlying state

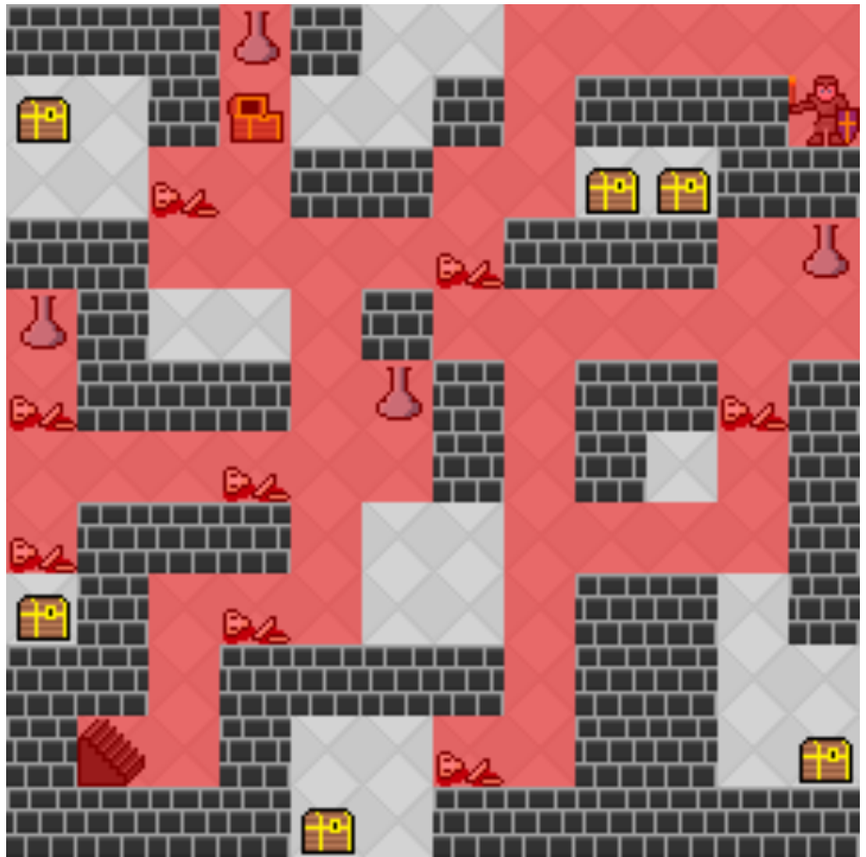
What if the state space is very large or continuous?

- Cannot represent value-function with table
- Need some kind of function approximator to represent V or Q
- Loss of convergence guarantees

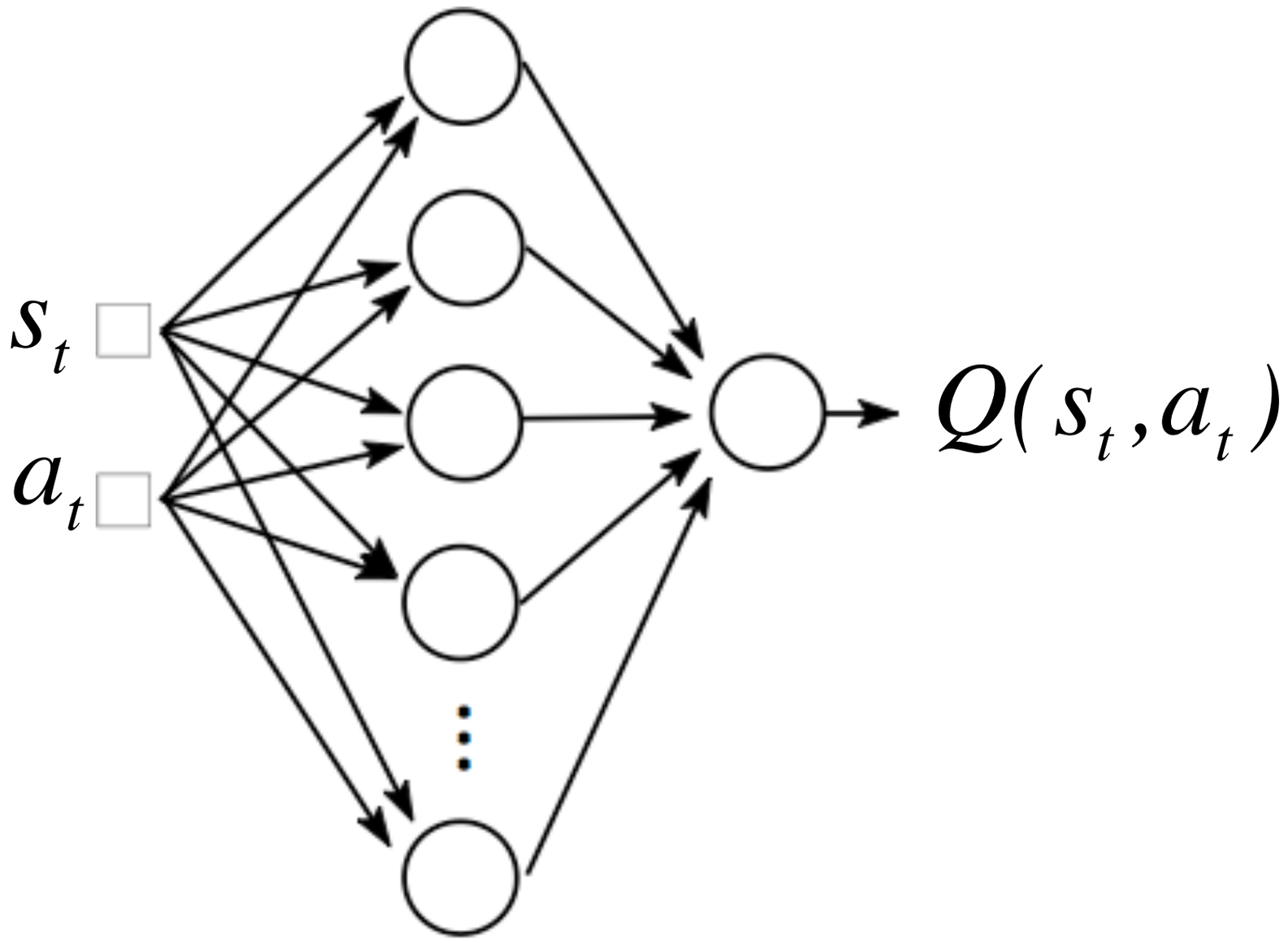
Problems with Q-learning?







Representing Q(s,a) with an NN



Training the NN Q-function

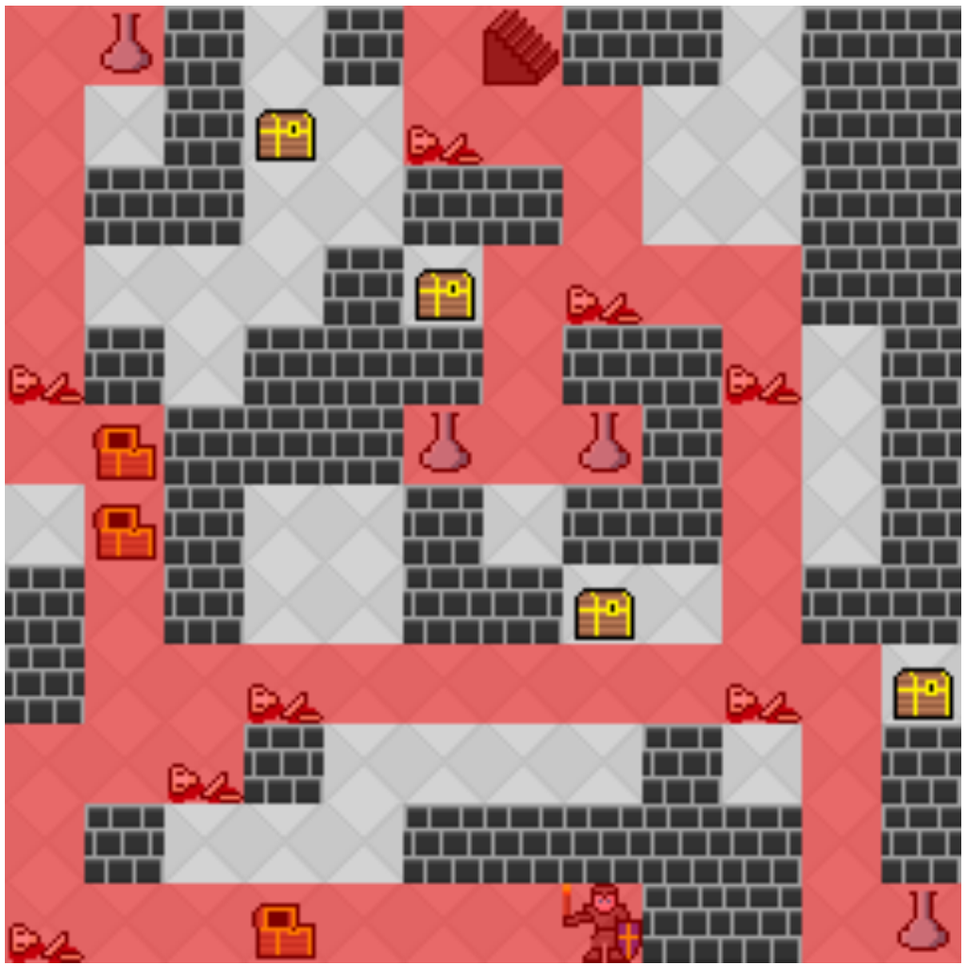
Training is performed on-line using the Q-values from the agent's state transitions

For Q-learning:

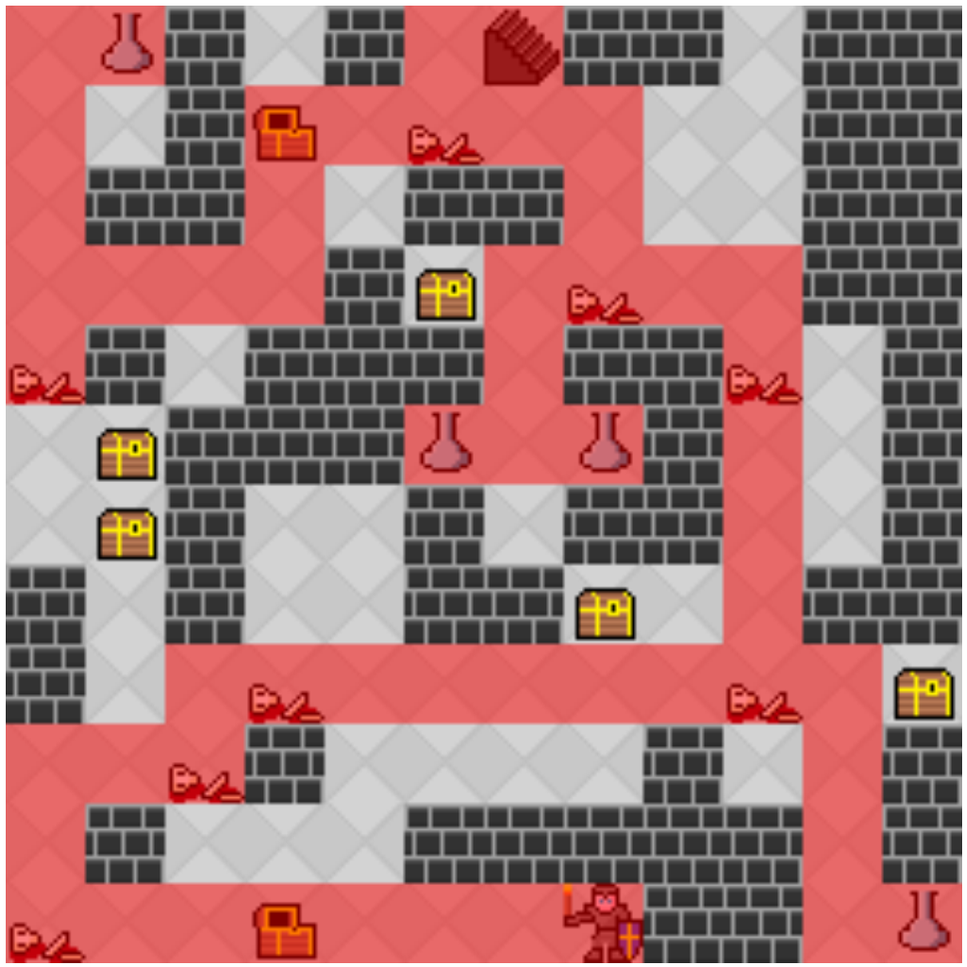
input: $s_t, \ a_t$

target: $r_t + \gamma \max_a Q(s_{t+1}, a)$

Function Approximators



Learned this map directly



Never learned from this map

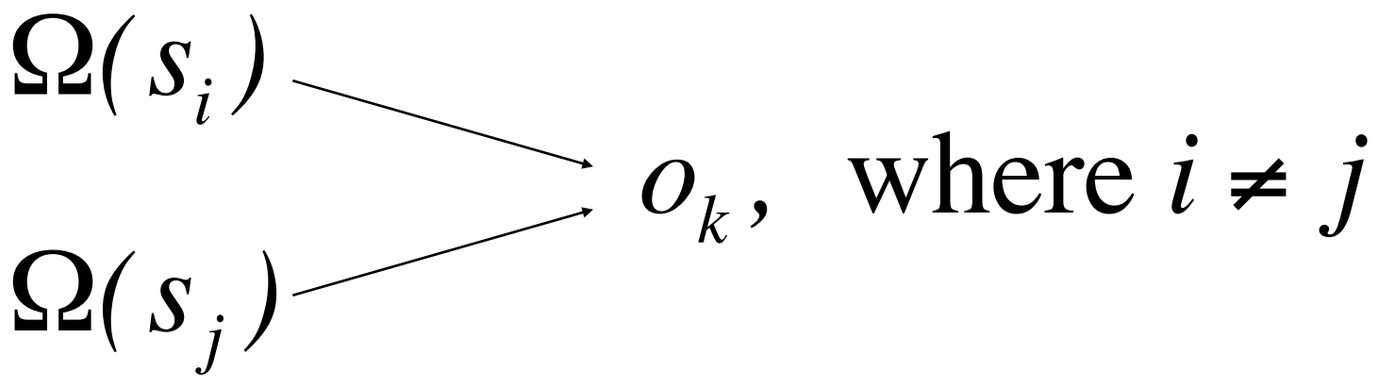
What if agent can't completely "see" the state?

- The environment is said to be *partially observable*
- The agent only receives an "observation" (O) of the state provided by its sensory system

Think of O as the agent's sensory system

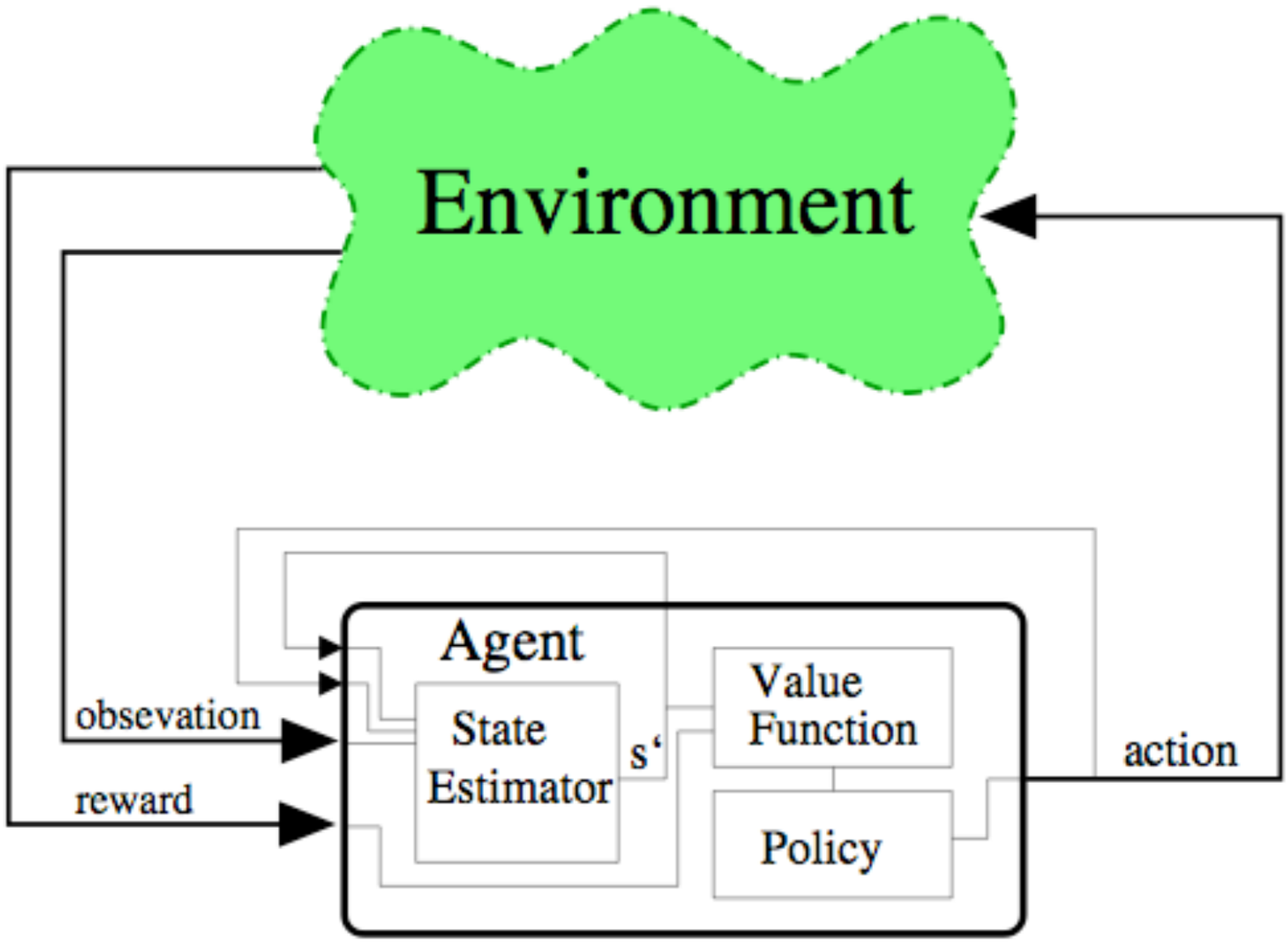
Perceptual Aliasing

- Different states can look the same or similar



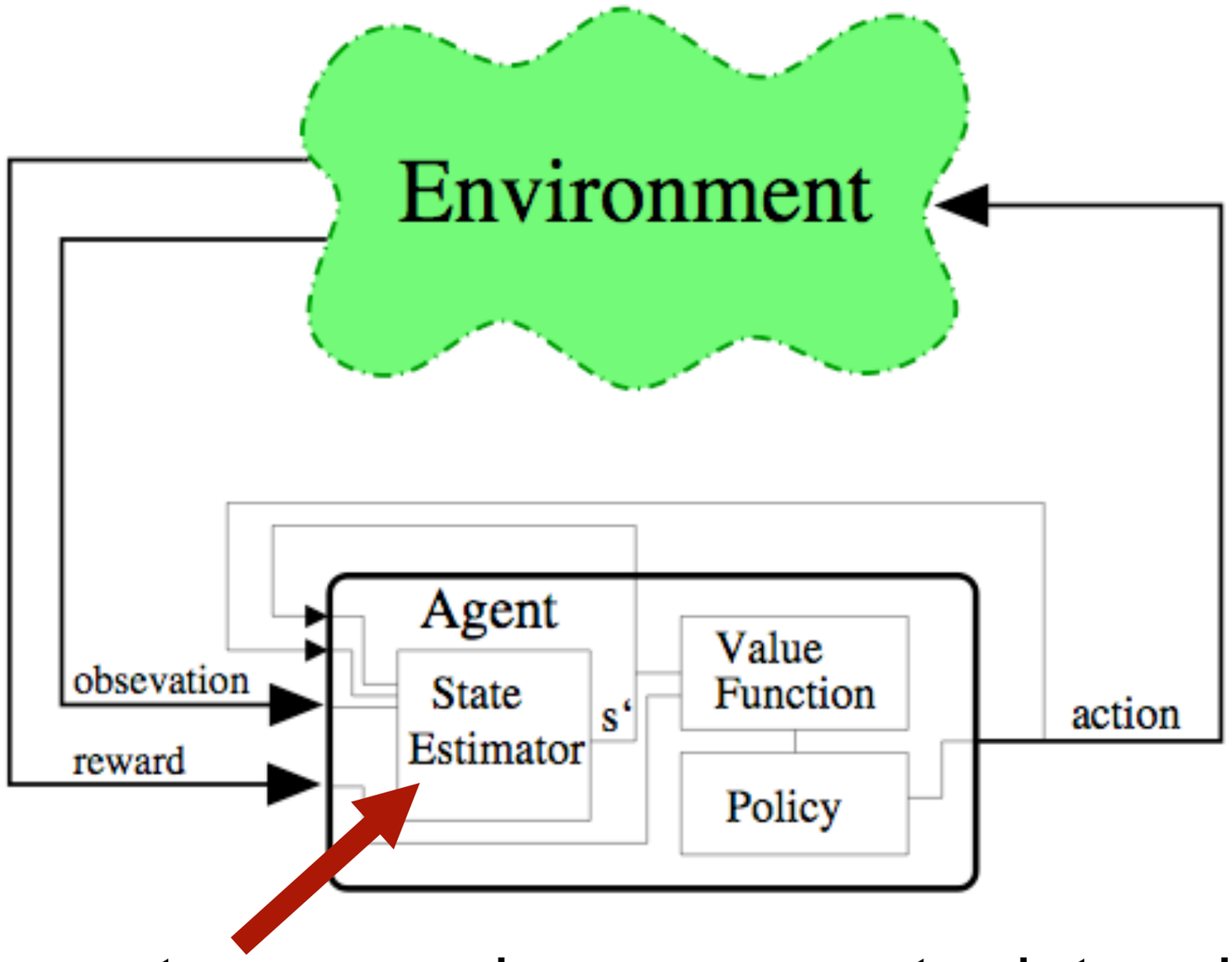
- If the action that is best for s_i is not good for s_j, we have a problem because agent will take the same action in both

RL under Partial Observability



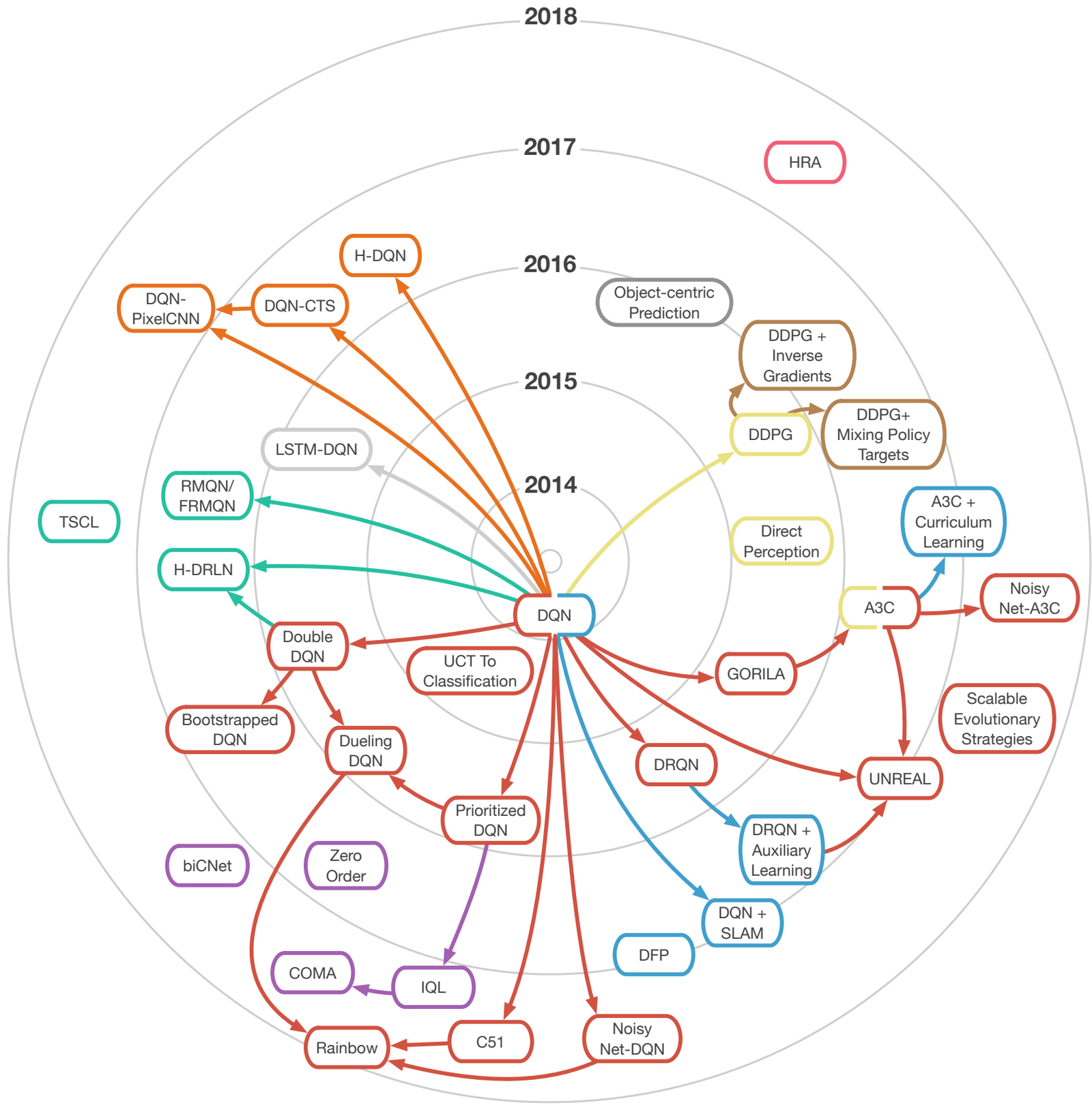
- Now, from the agent’s perspective, previous inputs (the observations) are important
- The current input is not enough to determine what state the environment is in!

RL under Partial Observability



The agent now needs some way to determine the underlying state; this means we need **memory**

One Solution: use RNN to represent V or Q-function


2018
2017
2016
2015
2014
HRA
H-DQN
DQN-PixelCNN
DQN-CTS
Object-centric Prediction
DDPG + Inverse Gradients
DDPG+ Mixing Policy Targets
DDPG
LSTM-DQN
RMQN/ FRMQN
TSCL
H-DRLN
Direct Perception
A3C + Curriculum Learning
A3C
Noisy Net-A3C
DQN
Double DQN
UCT To Classification
GORILA
Bootstrapped DQN
Dueling DQN
Scalable Evolutionary Strategies
DRQN
UNREAL
Prioritized DQN
DRQN + Auxiliary Learning
biCNet
Zero Order
DQN + SLAM
DFP
COMA
IQL
Rainbow
C51
Noisy Net-DQN
Ms Pac-Man
Atari
Montezuma's Revenge
Racing
Doom
Minecraft
StarCraft
2D Billiard
Text-based Game
RoboCup

Recap

- Reinforcement learning: learning from interacting with the environment
- Reward functions (utility)
- State transitions
- Markov property: everything in the state
- Bellman equation: reward taking into account future rewards
- Q-learning: model-free reinforcement learning, actions instead of states, utility of taking an action and then following the optimal policy
- Function approximators: When the number of states is too d… high

Neuroevolution

- Training neural networks using evolutionary computation
- Representation: the weights of the neural network
- Fitness function: how well does the network perform its task?
- Works really well - when the network is small…

A tale from Ancient History

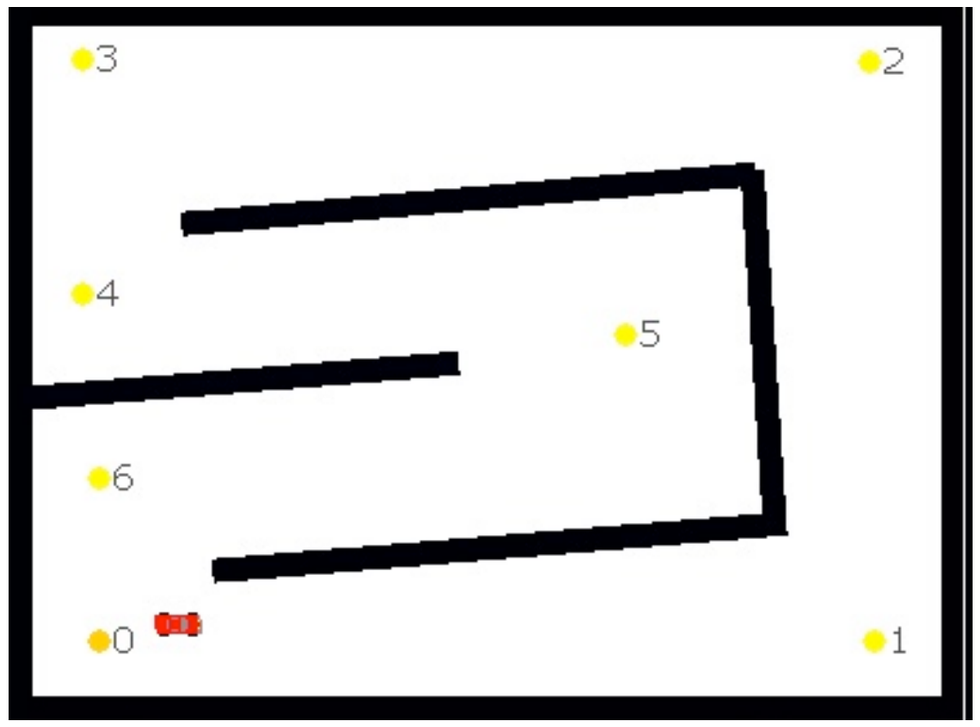
Togelius and Lucas (2006): Evolving Robust and Specialized Car Racing Skills. IEEE CEC.

The challenge: car racing

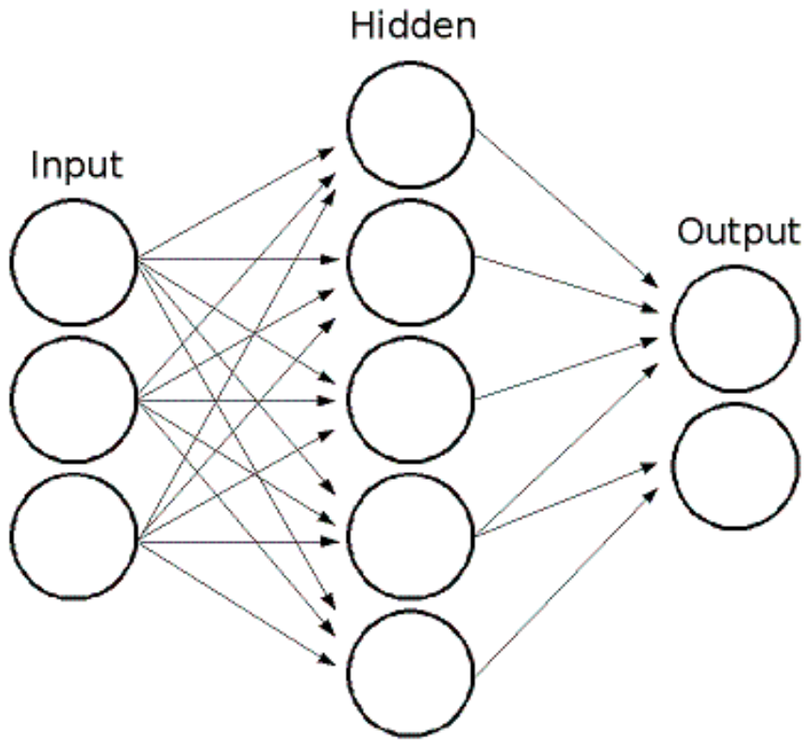
- Driving a car fast requires fine motor control (in both senses)
- Optimizing lap times requires planning
- Overtaking requires adversarial planning

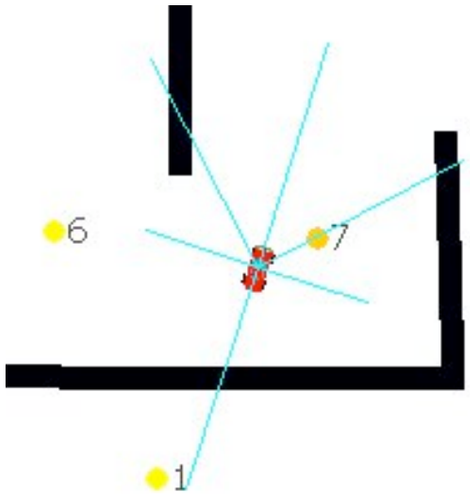
A simple car game

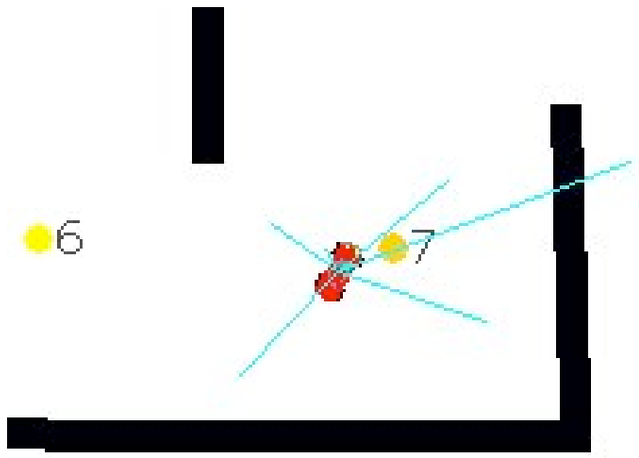
- Walls are solid
- Waypoints must be passed in order
- Fitness: continuous approximation of waypoints passed in 700 time steps

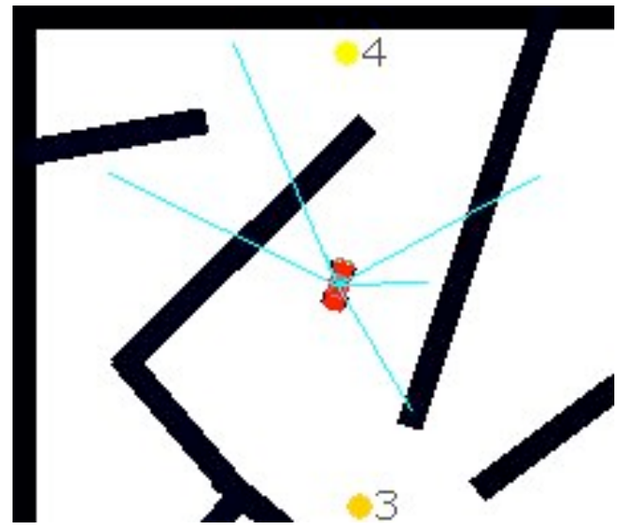


- Inputs
  - Six range-finder sensors (evolvable pos.)
  - Waypoint sensor, Speed, Bias
- Networks
  - Standard multi-layer perceptron, 9:6:2
  - Outputs interpreted as thrust/steering




6
7
1


6
7


4
3

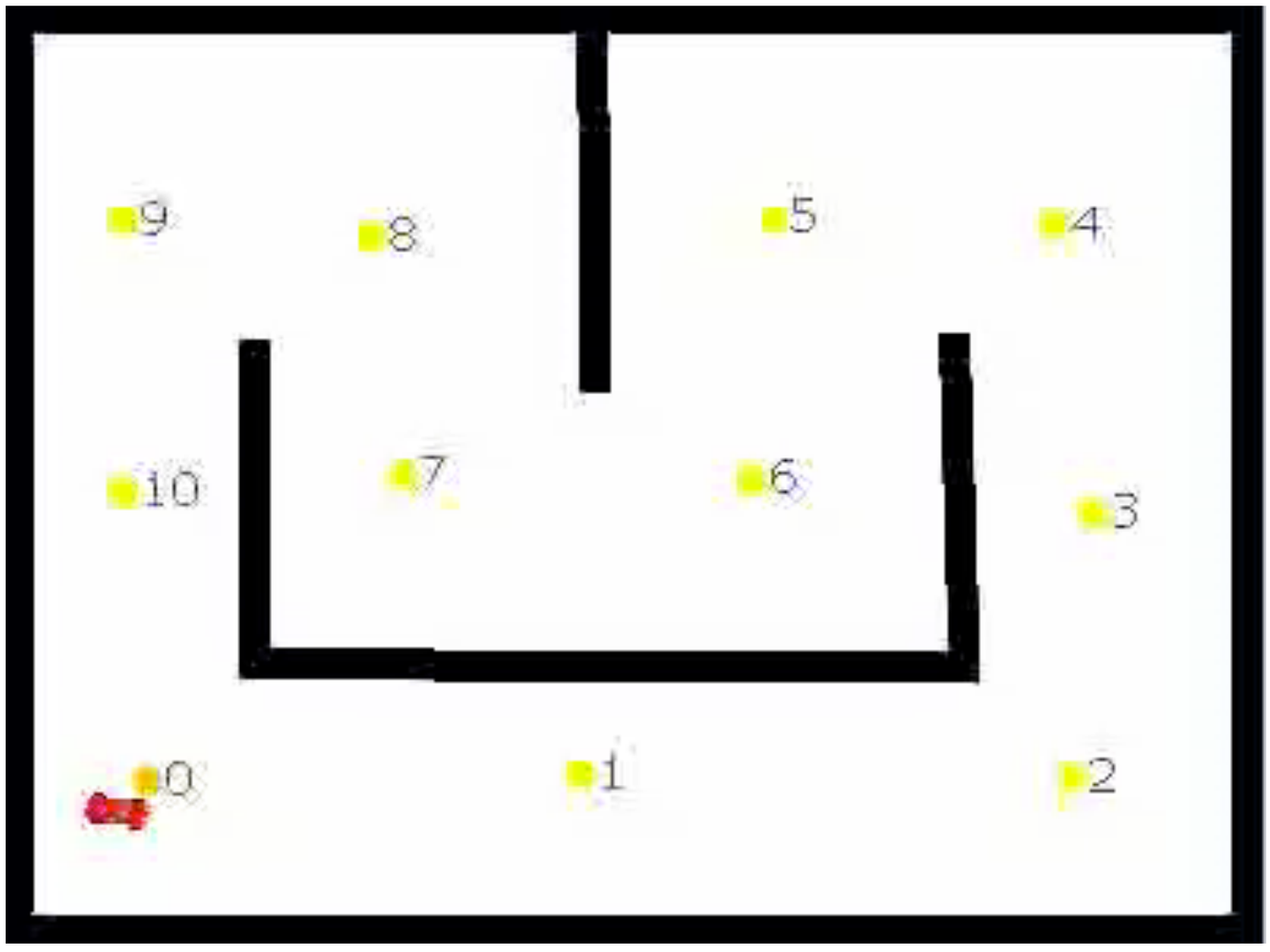
Algorithm 2: Evolution Strategy(μ,λ,n)

```
1  INITIALIZE (Population, μ + λ individuals)
2  for i=1 to n do
3      for j=1 to (μ + λ) do
4          EVALUATE (Population[j])
5      end
6      PERMUTE (Population)
7      SORTONFITNESS (Population)
8      for j=μ to (μ + λ) do
9          Population[j] ← COPY (Population[j-λ])
10         WEIGHTMUTATE (Population[j])
11     end
12 end
```

Mutation: add Gaussian noise with sd 1 to each connection

Fitness: progress around the track

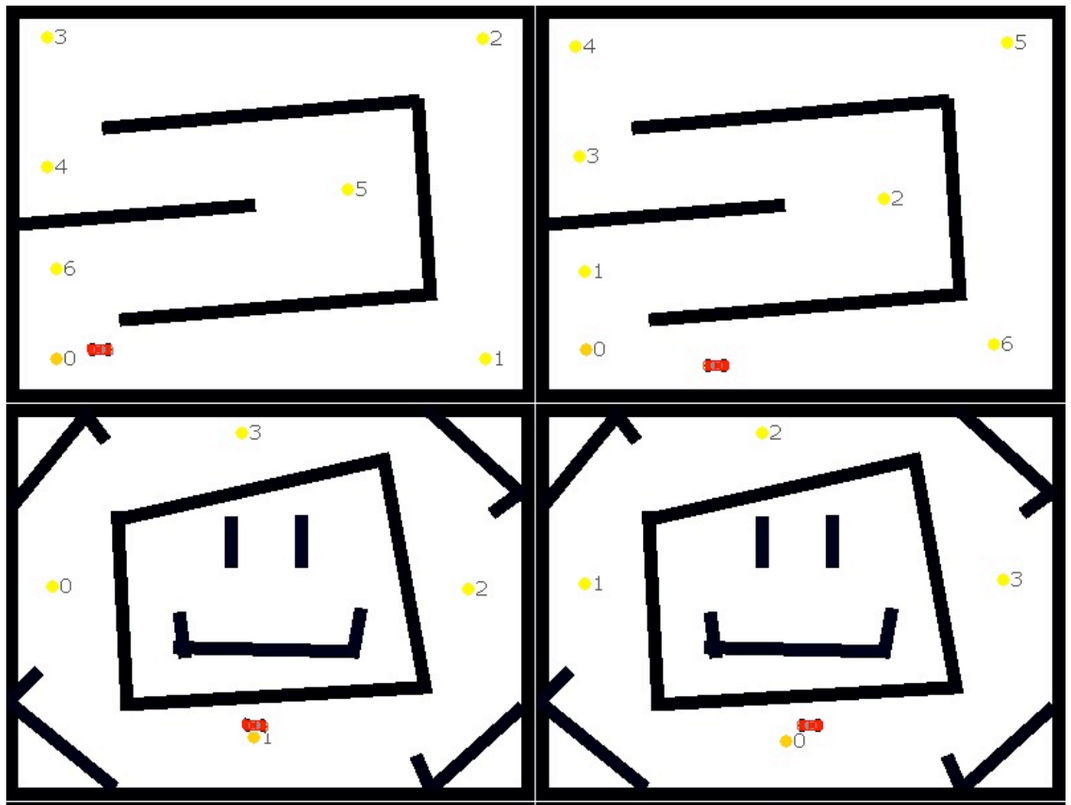
Example video

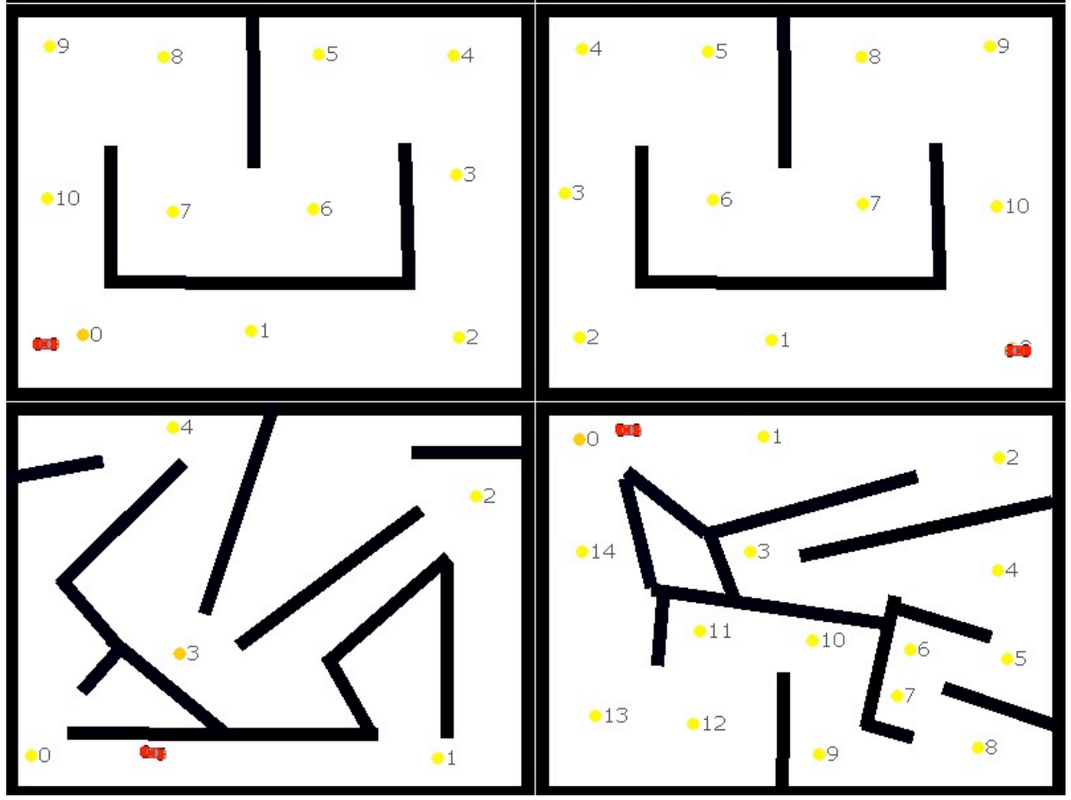


Evolved with 50+50 ES, 100 Generations

Generalization and specialization

- A controller evolved for one track does not necessarily perform well on other tracks
    - In fact, a controller evolved to drive a track clockwise will not be able to drive *the same track* counterclockwise
- How do we achieve more general game-playing skills?
    - Is there a tradeoff between generality and performance?

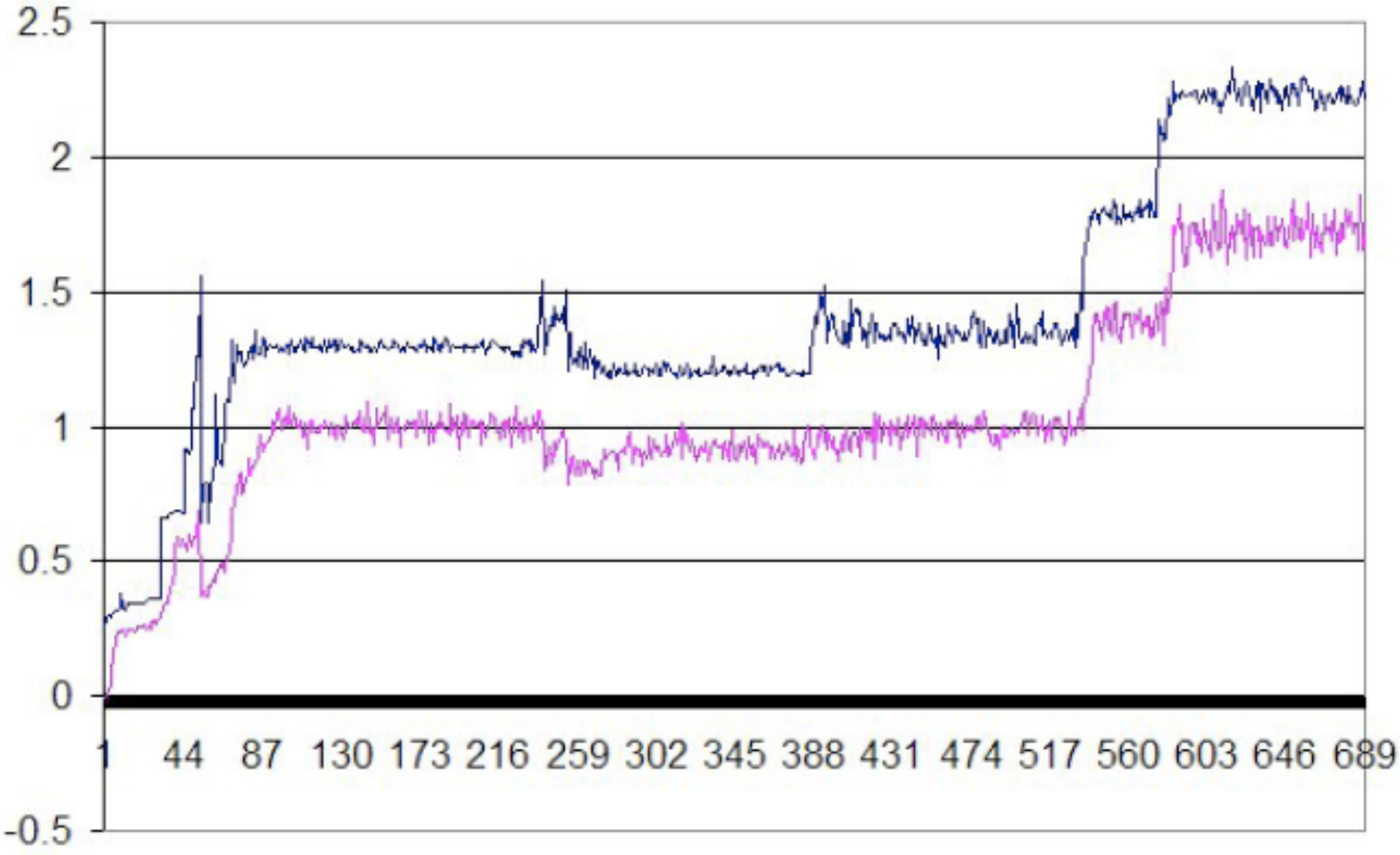

3
2
4
5
6
0
1
4
5
3
2
1
0
6
3
0
2
1
2
1
3
0


9
8
5
4
3
10
7
6
0
1
2
4
5
8
9
3
6
7
10
2
1
6
4
2
3
0
1
0
1
2
14
3
4
11
10
6
5
7
13
12
9
8

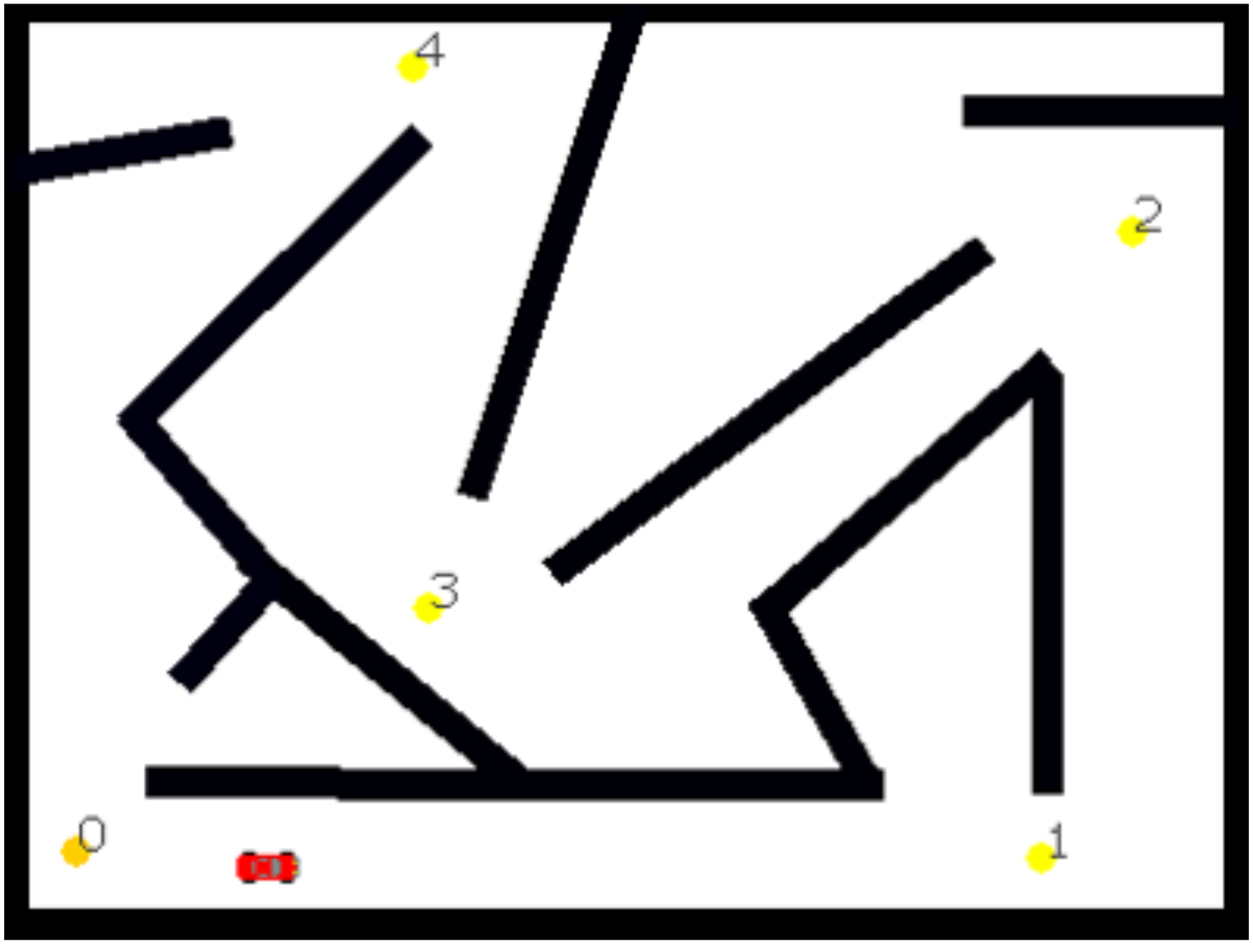
Incremental evolution

- Introduced by Gomez & Mikkulainen (1997)
  - Later also called curriculum learning
    - Change the fitness function *f* (to make it more demanding) as soon as a certain fitness is achieved
    - In this case, add new tracks to *f* as soon as the controller can drive 1.5 rounds on all tracks currently in *f*

Incremental evolution



Video: navigating a complex track



Observations

- Controllers evolved for specific tracks perform poorly on other tracks
- General controllers, that can drive almost any track, can be incrementally evolved
- Starting from a general controller, a controller can be further evolved for *specialization* on a particular track
  - drive faster than the general controller
  - works even when evolution from scratch did not work!

Two cars on a track

- Two car with solo-evolved controllers on one track: disaster
  - they don't even see each other!
- How do we train controllers that take other drivers into account? (avoiding collisions or using them to their advantage)
- Solution: car sensors (rangefinders, like the wall sensors) and *competitive coevolution*

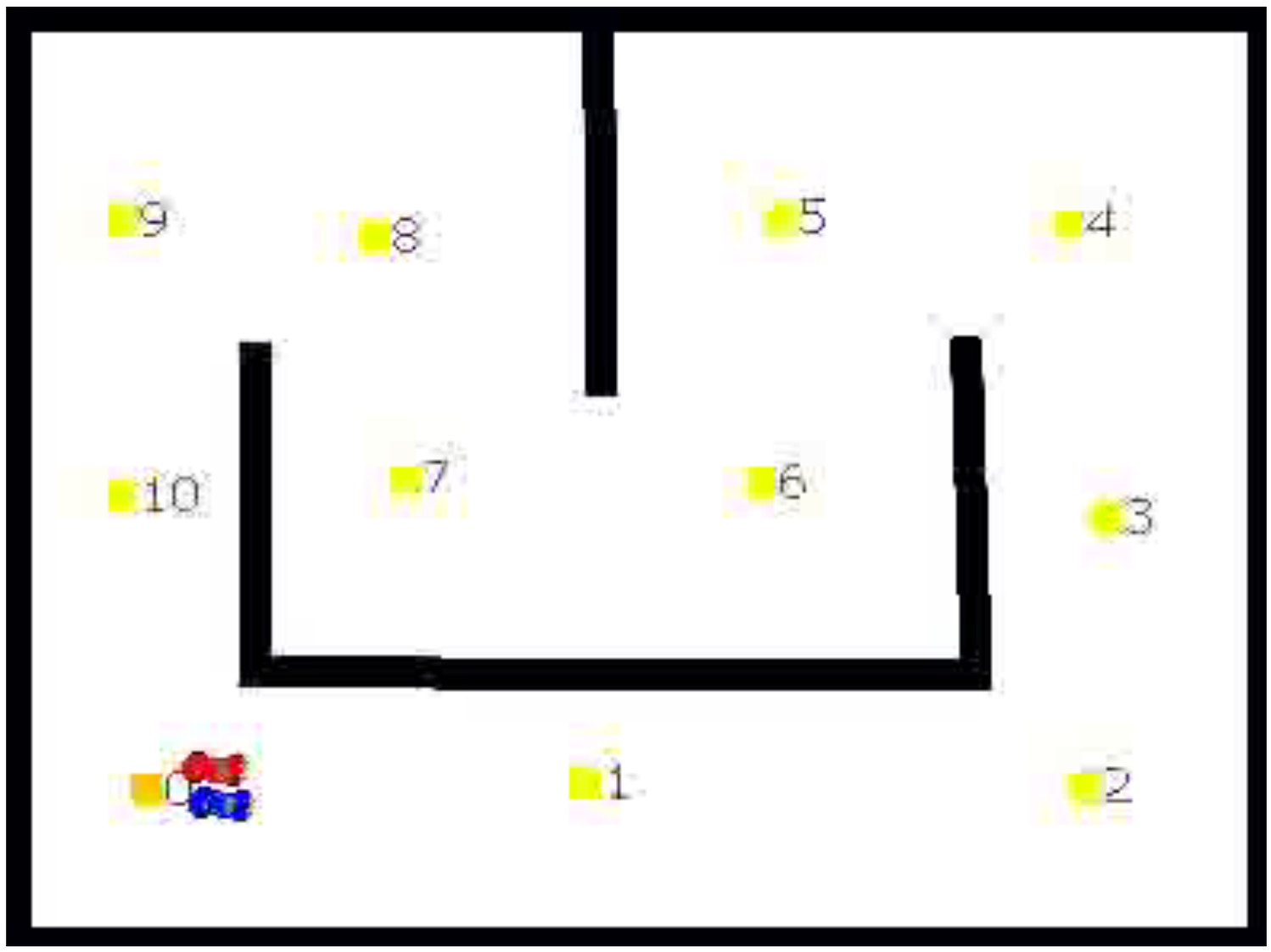
Competitive coevolution

- The fitness function evaluates at least two individuals
- One individual's success is *adversely* affected by the other's (directly or indirectly)
- Very potent, but seldom straightforward; e.g. Hillis (1991), Rosin and Belew (1996)

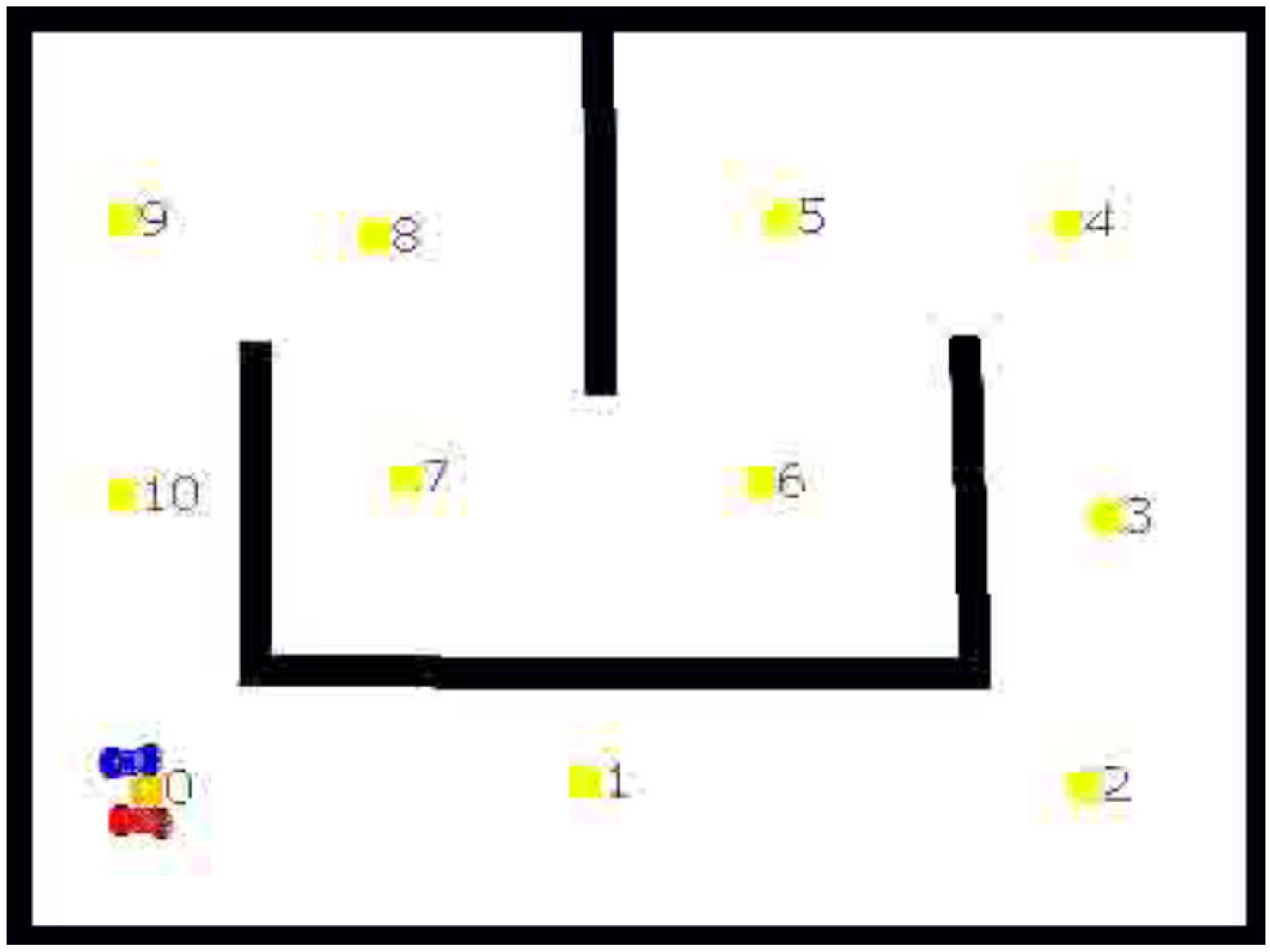
Competitive coevolution

- Standard 15+15 ES; each individual is evaluated through testing against the current best individual in the population
- Fitness function a mix of...
  - Absolute fitness: progress in *n* time steps
  - Relative fitness: distance ahead of or behind the other car after *n* time steps

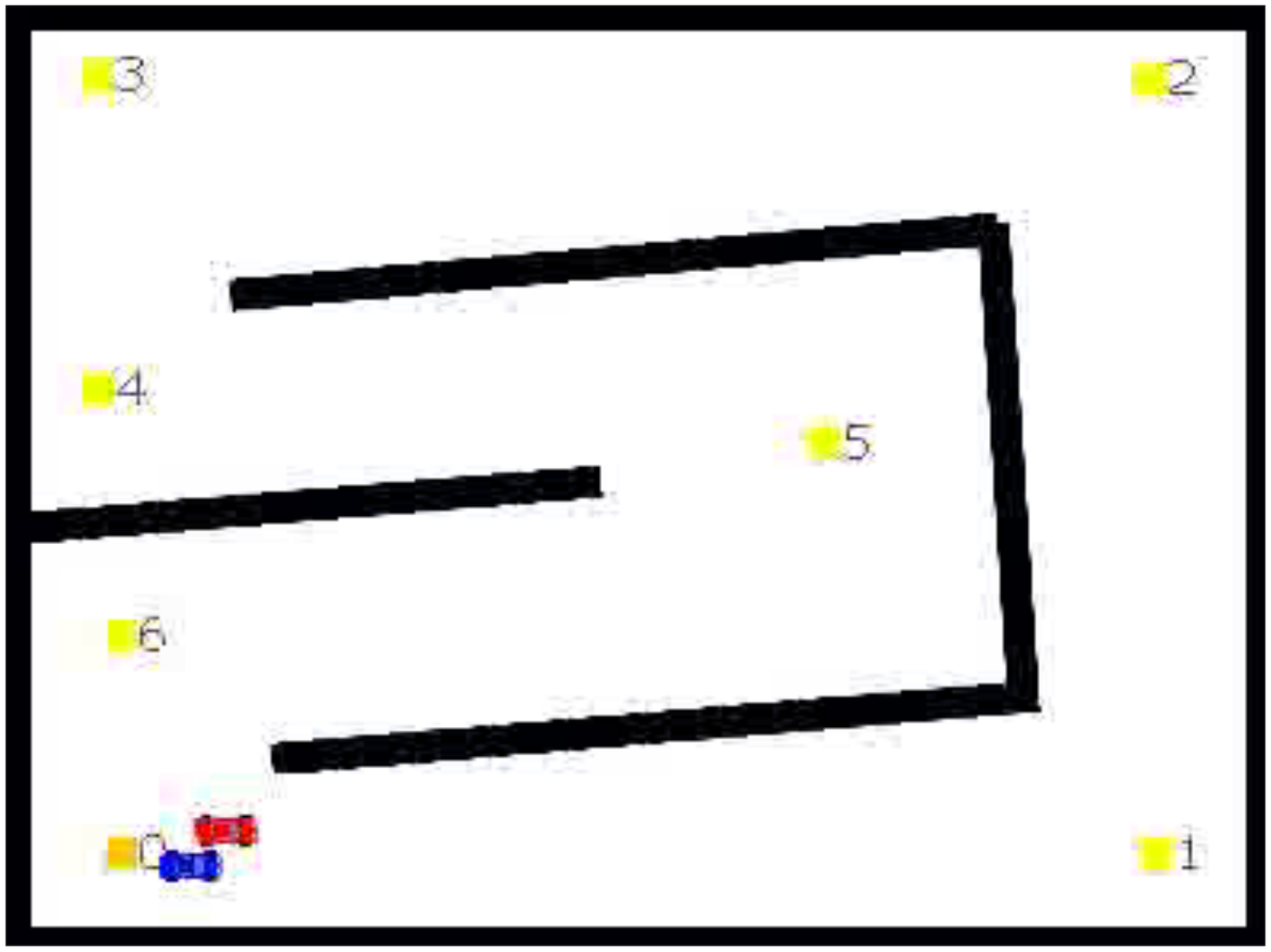
Video: absolute fitness



Video: 50/50 fitness



Video: relative fitness



Is neuroevolution a good way of doing RL?

- In theory, it should be less efficient
- In practice, it is often more robust
- Plus, you can do multiobjective evolution, quality-diversity illumination, etc.