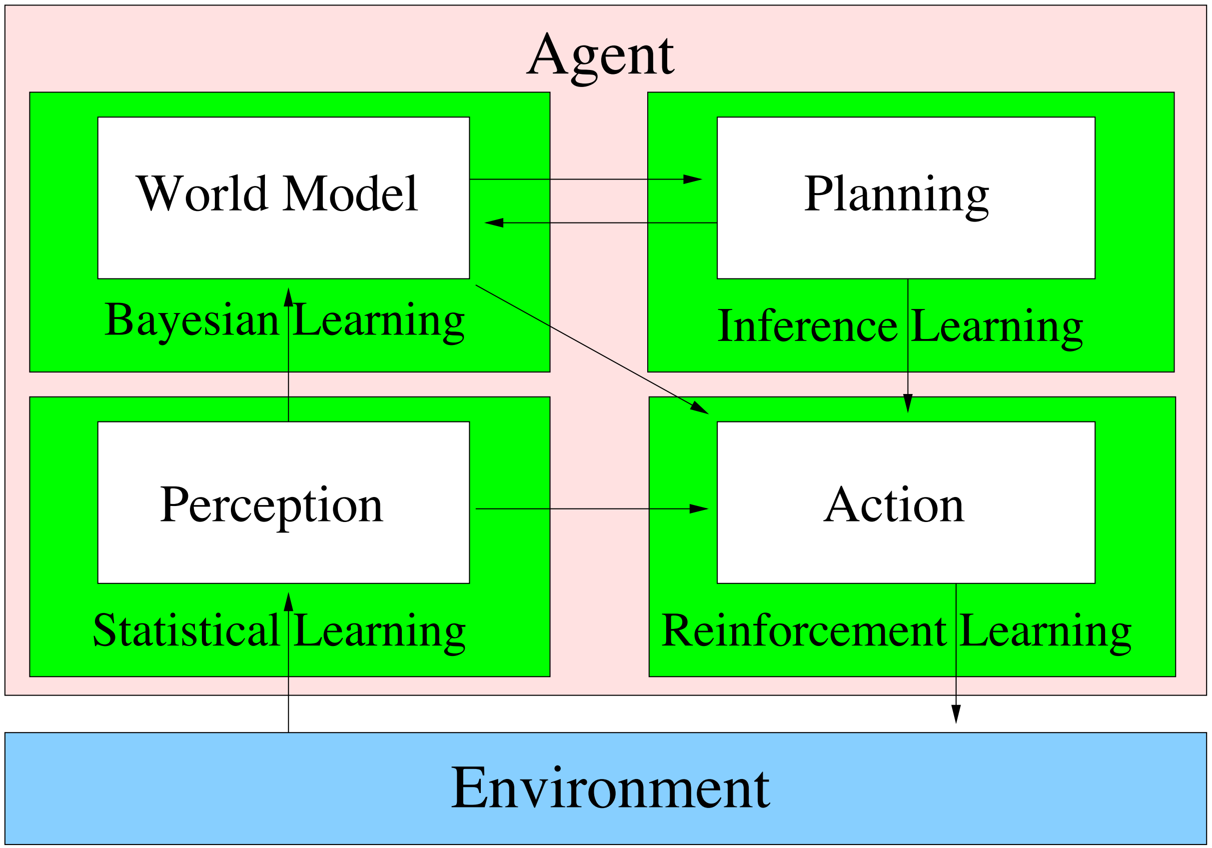
Planning Under Uncertainty
Reinforcement Learning

COMP3411/9814: Artificial Intelligence

Lecture Overview

- Reinforcement Learning vs Supervised Learning
- Boxes
- Exploration vs Exploitation
- Q-Learning

Learning Agent



Types of Learning

- Supervised Learning
  - Agent is given examples of input/output pairs
  - Learns a function from inputs to outputs that agrees with the training examples and generalises to new examples
- Unsupervised Learning
  - Agent is only given inputs
  - Tries to find structure in these inputs
- Reinforcement Learning
  - Training examples presented one at a time
  - Must guess best output based on a reward, tries to maximise (expected) rewards over time

Environment Types

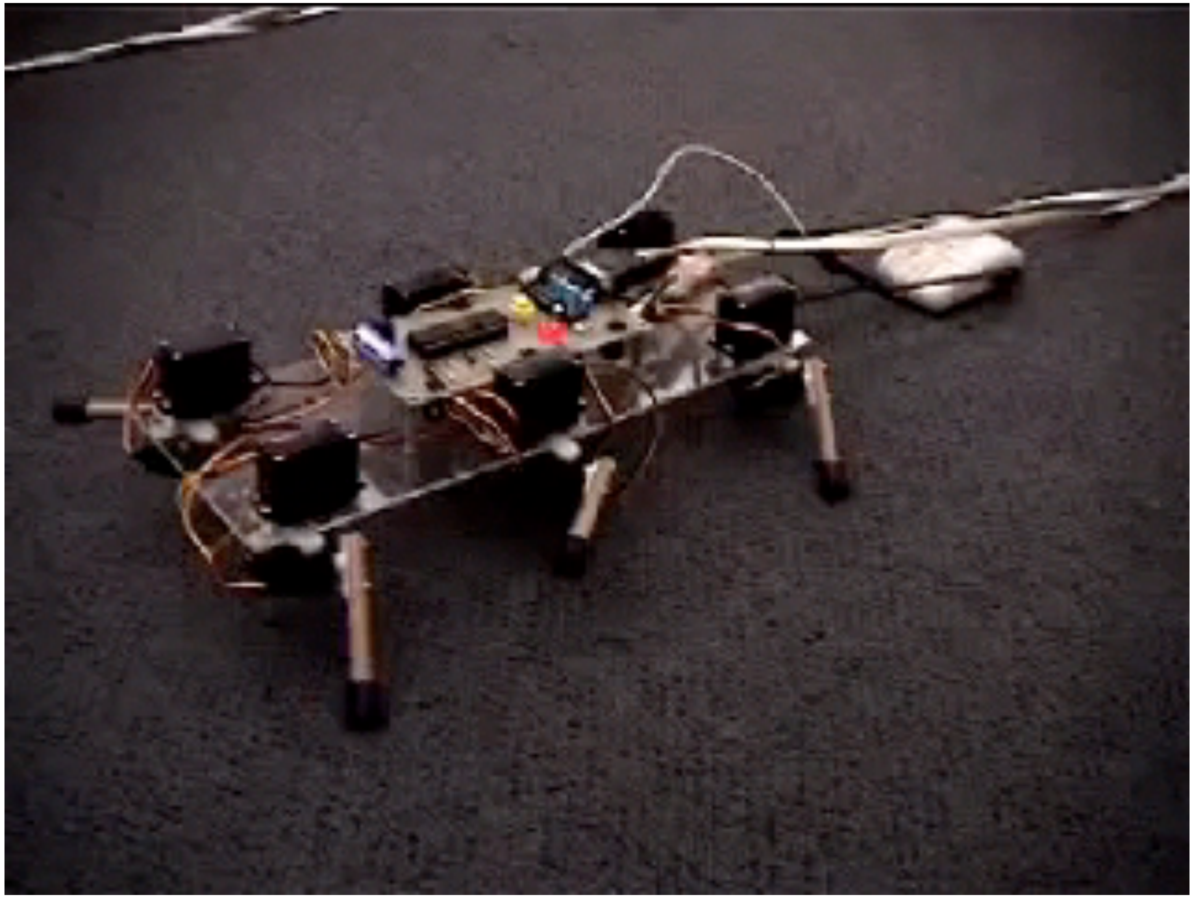
Environments can be:

- passive and deterministic
- passive and stochastic
- active and deterministic (chess)
- active and stochastic (backgammon, robotics)

Reinforcement Learning and Planning

- We start with reinforcement learning because it is also related to planning.
- RL tries to find the best way to act in uncertain and non-deterministic environments.

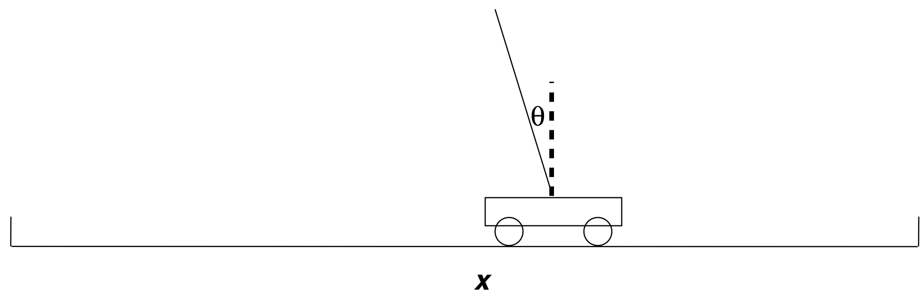
Stumpy - A Simple Learning Robot



Reinforcement Learning

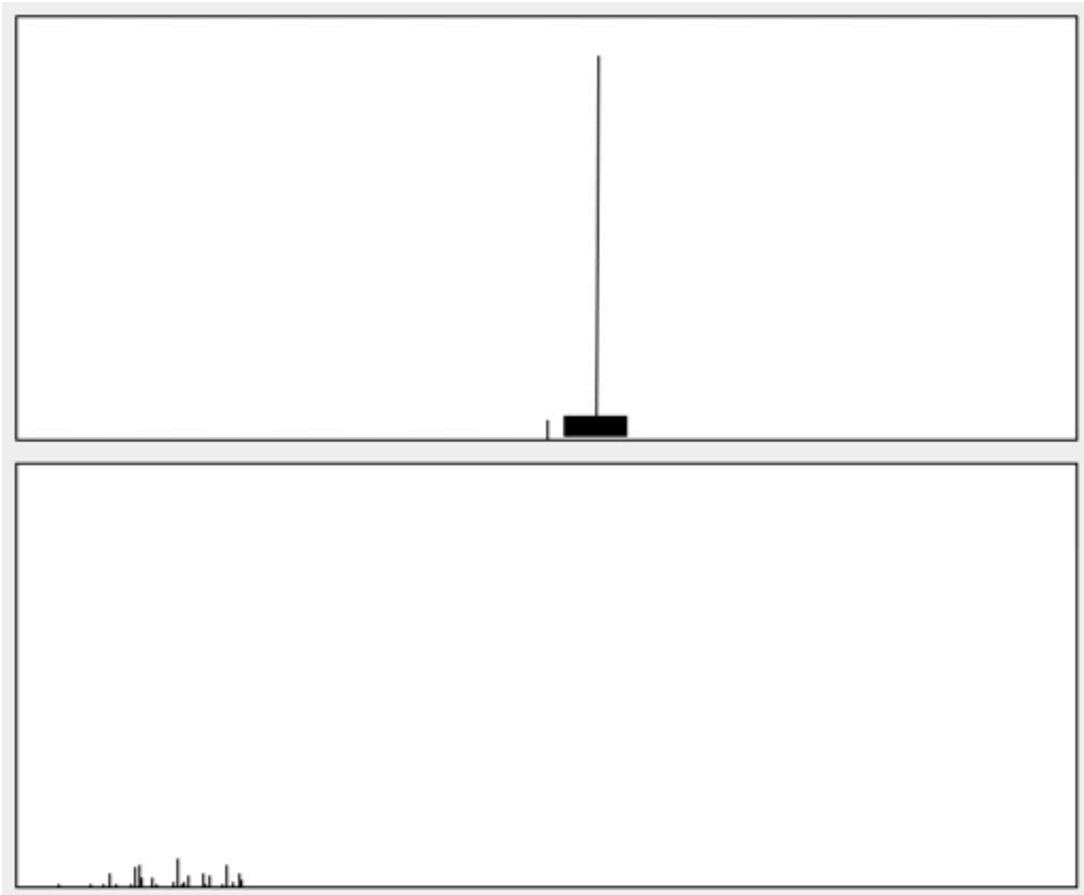
- “Stumpy” receives a *reward* after each action
  - Did it move forward or not?
- After each move, updates its *policy*
- Continues trying to maximise its reward

Pole Balancing



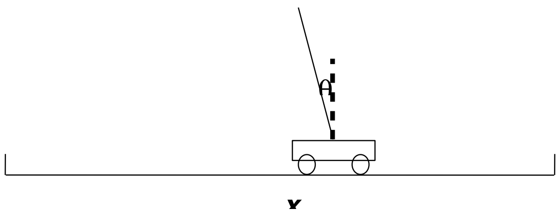
- Pole balancing can be learned the same way except that reward is only received at the end
  - after falling or hitting the end of the track

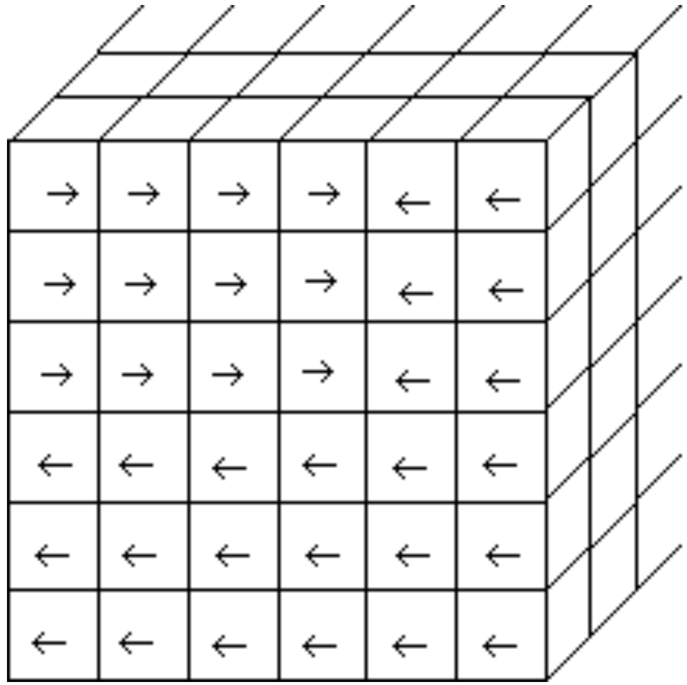
Pole Balancing



Boxes

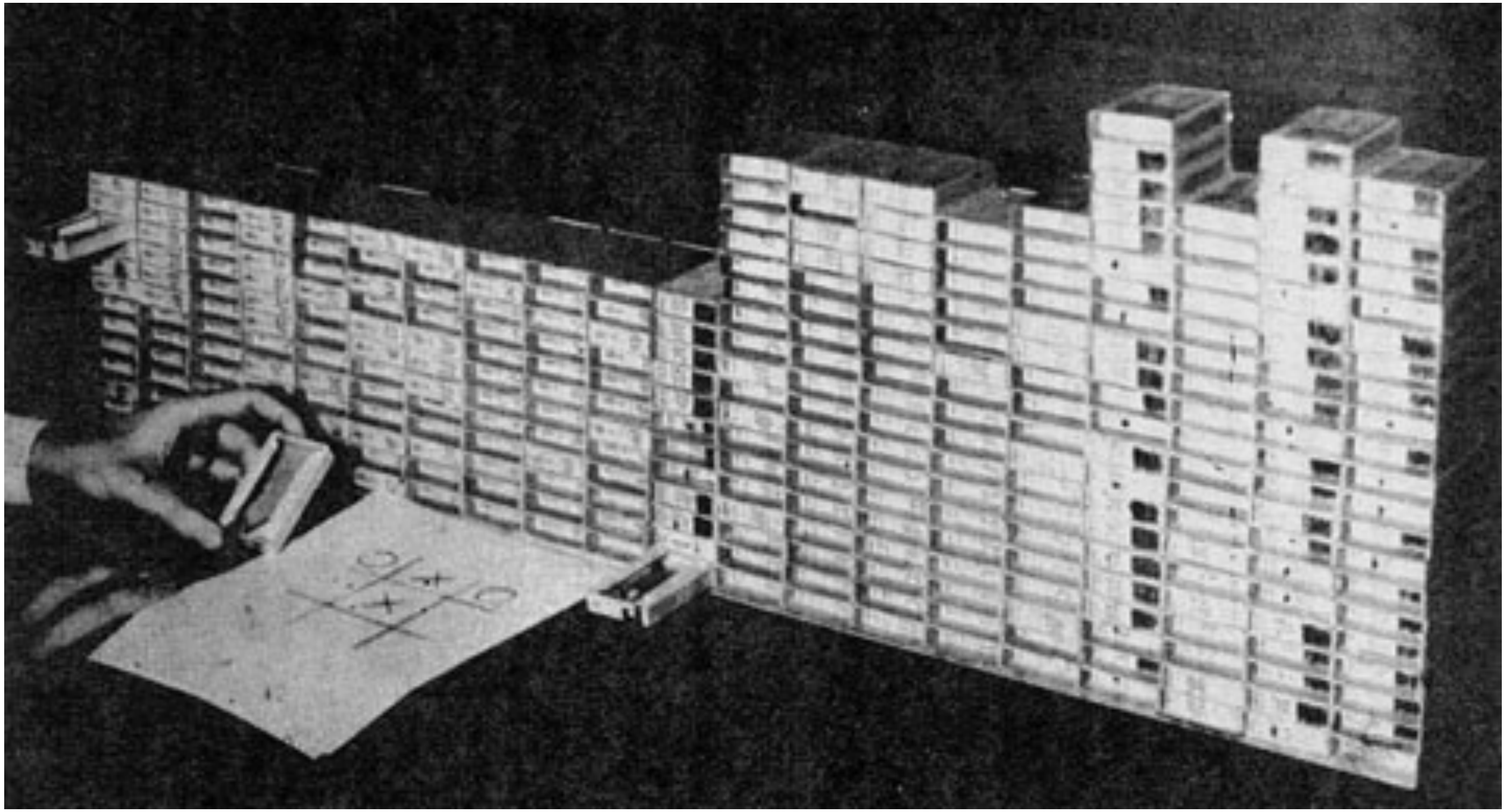
- State variables: $< x, \dot{x}, \theta, \dot{\theta} >$
- State space is discretised
- Each "box" represents a subset of state space
- When system lands in a box, execute action specified
  - left push
  - right push





MENACE

(Machine Educable Noughts and Crosses Engine – D. Michie, 1961)



Simulation

$$x_{t+1} = x_t + \tau \dot{x}_t$$
$$\dot{x}_{t+1} = \dot{x}_t + \tau \ddot{x}_t$$
$$\theta_{t+1} = \theta_t + \tau \dot{\theta}_t$$
$$\dot{\theta}_{t+1} = \dot{\theta}_t + \tau \ddot{\theta}_t$$

$$\ddot{x}_t = \frac{F_t + m_p\ l\ \left[\dot{\theta}_t^2 \sin\theta_t - \ddot{\theta}_t \cos\theta_t\right]}{m_c + m_p}$$

$$\ddot{\theta}_t = \frac{g \sin\theta_t + \cos\theta_t \left[\frac{-F_t - m_p\ l\ \dot{\theta}_t^2 \sin\theta_t}{m_c + m_p}\right]}{l \left[\frac{4}{3} - \frac{m_p \cos^2\theta_t}{m_c + m_p}\right]}$$

$m_c = 1.0$ kg *mass of cart*

$m_p = 1.0$ kg *mass of pole*

$l = 0.5$ m *distance of centre of mass of pole from the pivot*

$g = 9.8$ ms^{-2} *acceleration due to gravity*

$F_t = \pm\ 10$ N *force applied to cart*

$t = 0.02$ s *time interval of simulation*

The BOXES Algorithm

- Each box contains statistics on performance of controller, which are updated after each failure
  - How many times each action has been performed (*usage)*
  - The sum of lengths of time the system has survived after taking a particular action (*LifeTime*)
- Each sum is weighted by a number less than one which places a discount on earlier experience.

Exploration / Exploitation Tradeoff

- Most of the time choose what we think is the “best” action.
- But to learn, must occasionally choose something different from preferred action

Update Rule

if an action has not been tested

choose that action

else if $\dfrac{LeftLife}{LeftUsage^k} > \dfrac{RightLife}{RightUsage^k}$

choose left

else

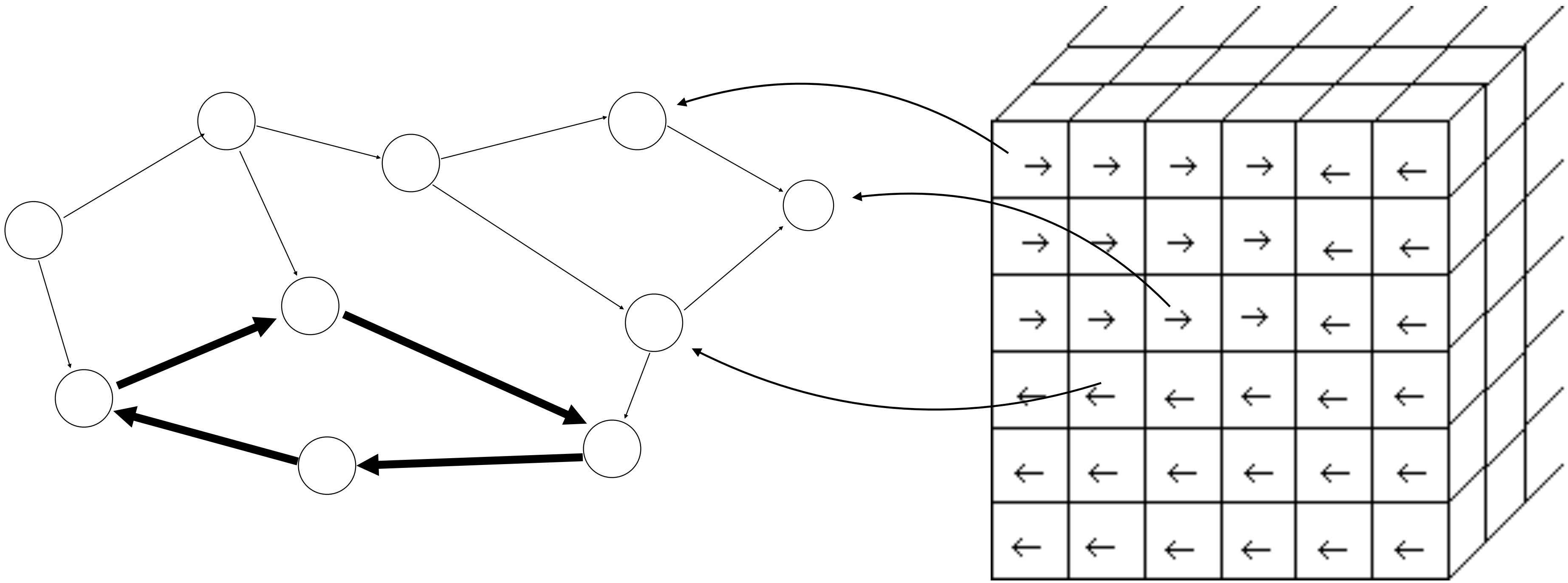
choose right

k is a bias to force exploration
e.g. k = 1.4

Performance

- BOXES is fast
  - Only 75 trials, on average, to reach 10,000 time steps
- But only works for *episodic* problems
  - i.e. has a specific termination
- Doesn’t work for continuous problems like Stumpy

State Transition Graph



States and Actions

- Each node is a *state*
- *Actions* cause transitions from one state to another
- A *policy* is the set of transition rules
    - i.e. which action to apply in a given state
- Agent receives a *reward* after each action
- Actions may be non-deterministic
    - Same action may not always produce same state

Reinforcement Learning Framework

- An agent interacts with its environment.
- There is a set of *states,* S*,* and a set of *actions,* A.
- At each time step t, agent is in state s_t.
- It must choose an action a_t, which changes state to
- $s_{t+1} = \delta(s_t, a_t)$ and receives reward $r(s_t, a_t)$.
  - The world is non-deterministic, i.e. an action may not always take the system to the same state
  - δ, and therefore r, can be multi-valued, with a random element
- Aim is to find an optimal *policy* $\pi : S \to A$ that maximises the cumulative reward.

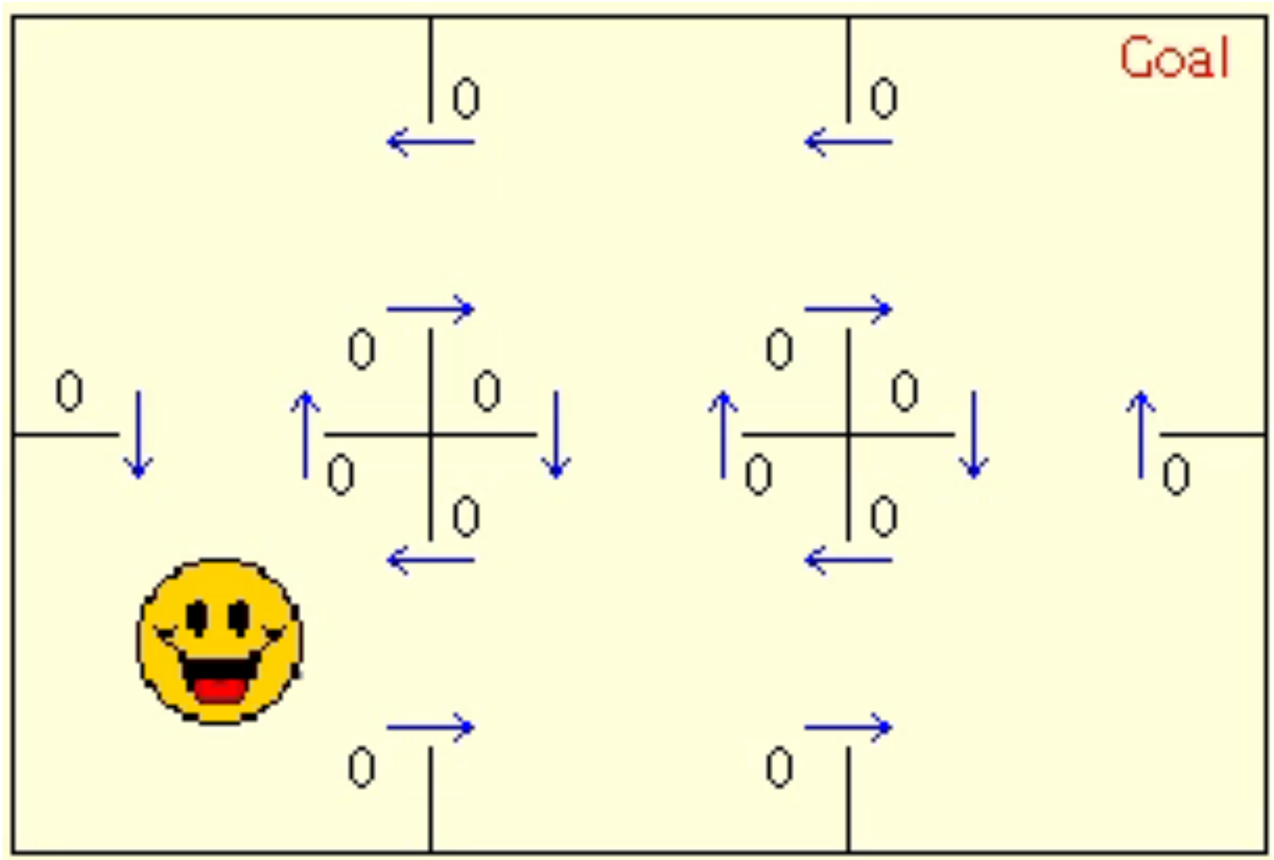
Markov Decision Process (MDP)

- Assume that current state has all the information needed to decide which action to take
- Actions are assumed to have a fixed duration

Learning an MDP

- The agent initially only knows the set of possible states and the set of possible actions.
- The dynamics, $P(s' | a, s)$, and the reward function, $R(s, a)$, are not given to the agent.
- $P(s' | a, s)$ the probability of the agent transitioning into state s' given that the agent is in state s and does action a
- After each action, the agent observes the state it is in and receives a reward.
- Assume that current state has all the information needed to decide which action to take

Grid World Example



Expected Reward

- Try to maximise expected future reward:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$
$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

- $V^{\pi}(s_t)$ is the value of state s_t under policy π

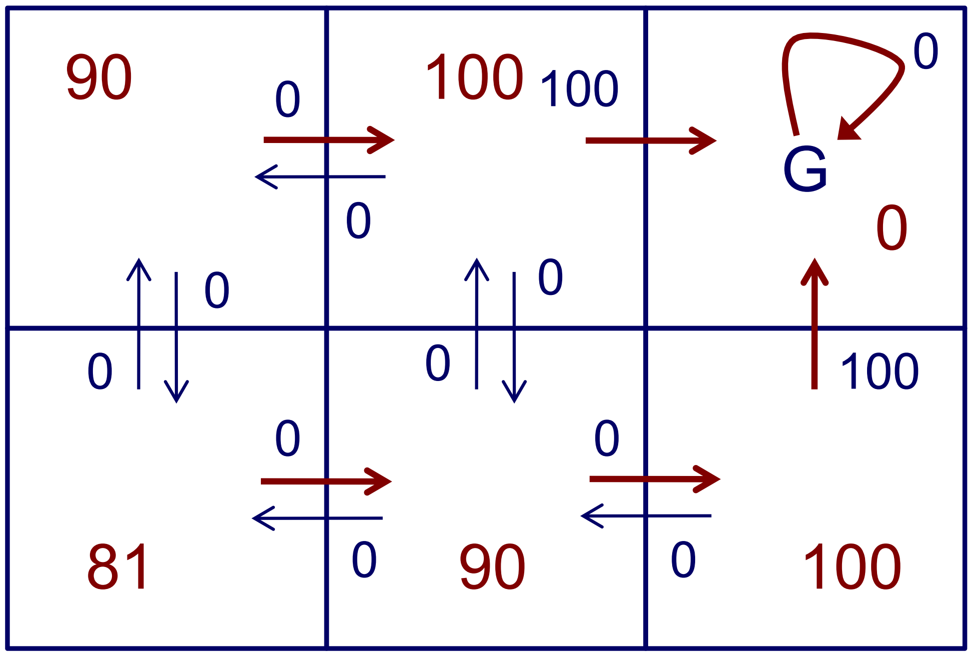
- γ is a discount factor [0..1]

Value Function

- $V^{\pi}(s)$ is the expected value of following policy π in state s
- $V^{*}(s)$ be the maximum discounted reward obtainable from s.
  - i.e. the value of following the optimal policy
- We make the simplification that actions are deterministic, but we don't know which action to take.
  - Other RL algorithms relax this assumption

Value Function

- The red arrows show, π^*, is the optimal policy, with $\gamma = 0.9$
- $V^*(s)$ values shown in red



Q Value

- How to choose an action in a state?

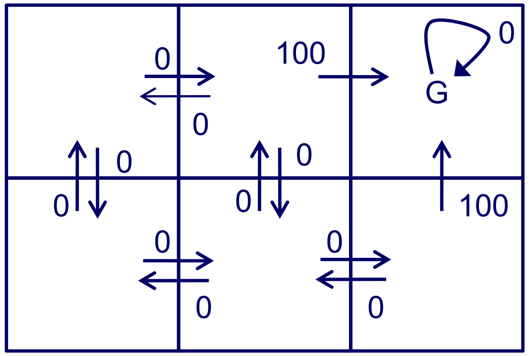
$$Q(s, a) = r(s, a) + \gamma V^*(s')$$

- The Q value for an action, a, in a state, s, is the immediate reward for the action plus the discounted value of following the optimal policy after that action

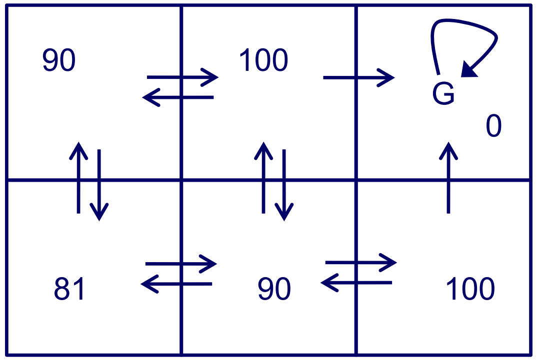
- V^* is value obtained by following the optimal policy

- $s' = \delta(s, a)$ is the succeeding state, assuming the optimal policy

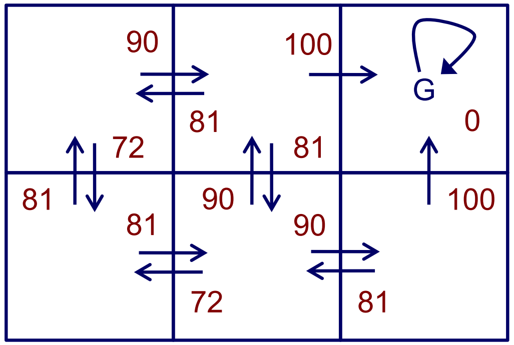
Q values



$r(s, a)$ (immediate reward) values



$V^*(s)$ values



$Q(s, a)$ values

$\gamma = 0.9$

Q Learning

initialise $Q(s,a) = 0$ for all s and a

observe current state s

repeat

select an action a and execute it

observe immediate reward r and next state s'

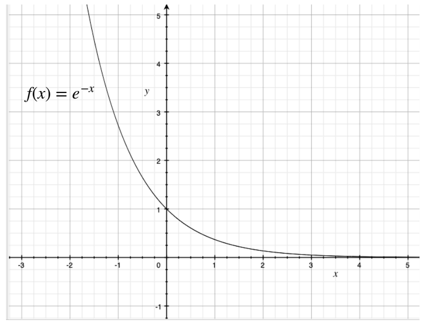
$Q(s,a) \leftarrow r + \max_{a'} Q(s', a')$

$s \leftarrow s'$

Exploration vs Exploitation

- How do you choose an action?
  - Random
  - Pick the current “best” action
  - Combination:
    - most of the time pick the best action
    - occasionally throw in random action
    - Boltzmann exploration:

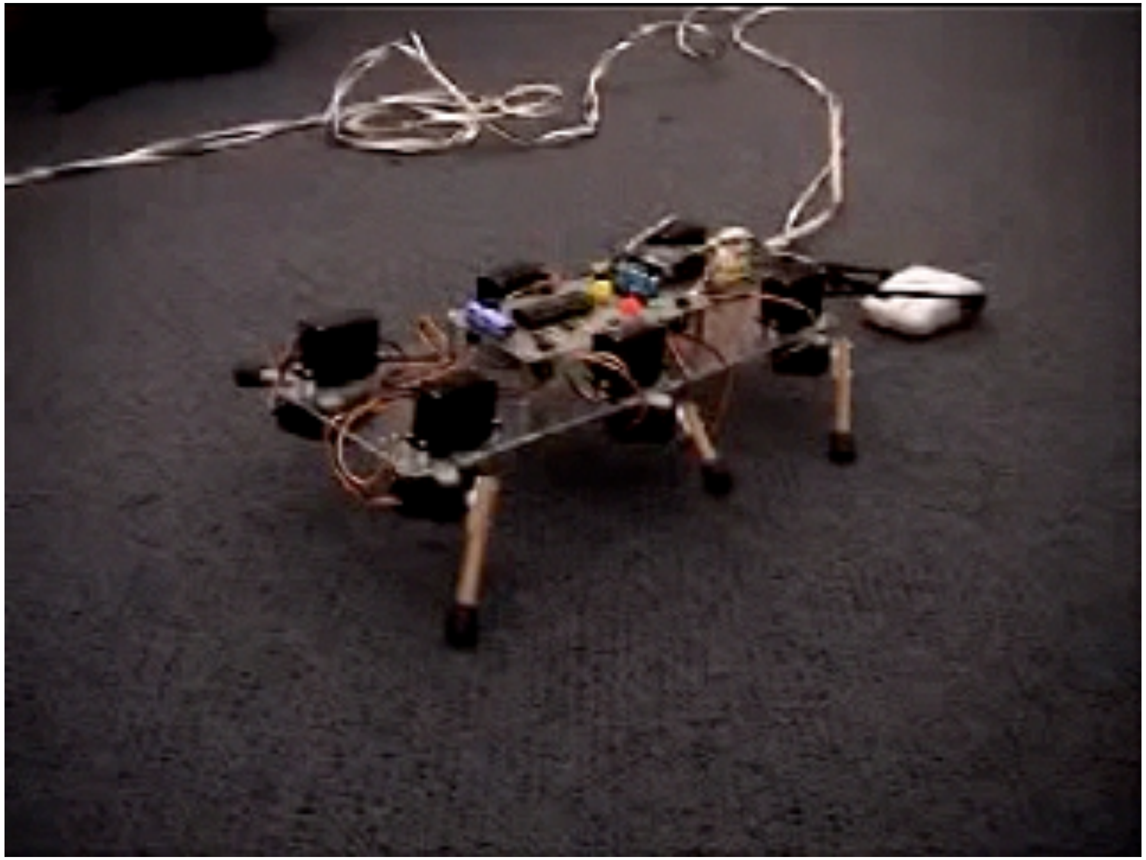
$$\pi(s_t, a) \simeq e^{\frac{-Q_t(s_t, a)}{\tau}}$$



Decrease τ over time

- High τ: exploration
- Low τ: exploitation

Stumpy after 30 minutes



Reinforcement Learning Variants

- There are *many* variations on reinforcement learning to improve search.
- RL is one of the components of alphaZero, which is currently the best Go and Chess player
- Used to learn helicopter aerobatics

Background

- Reinforcement learning is based in earlier work in optimisation: dynamic programming
- Text book: Sutton & Barto