Chapter 4 Machine Learning

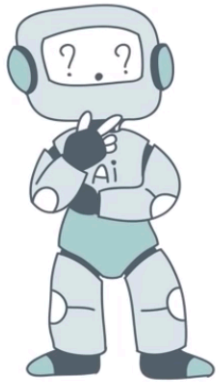
COMP 6721 Introduction of AI

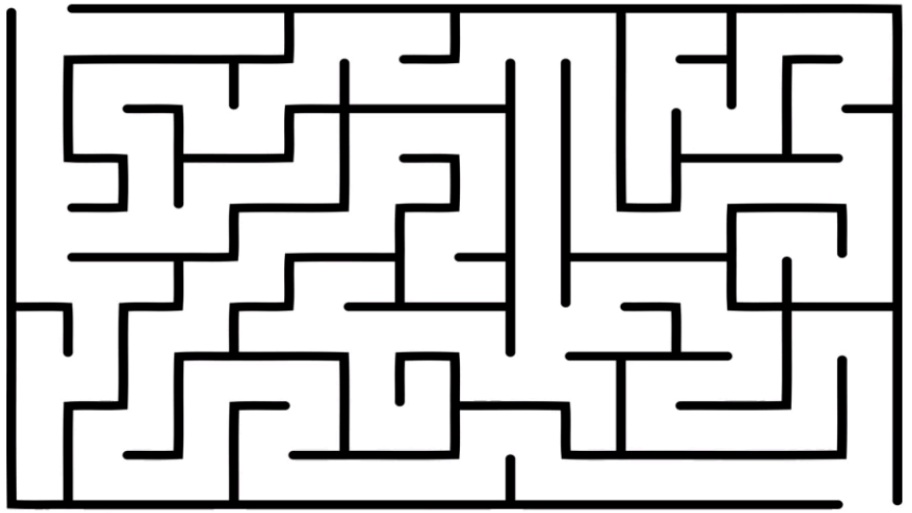
Russell & Norvig – Section 18.1 & 18.2

Reinforcement Learning

- Problems involving an agent interacting with an environment, which provides numeric reward signals.
- Goal: Learn how to take actions in order to maximize reward.

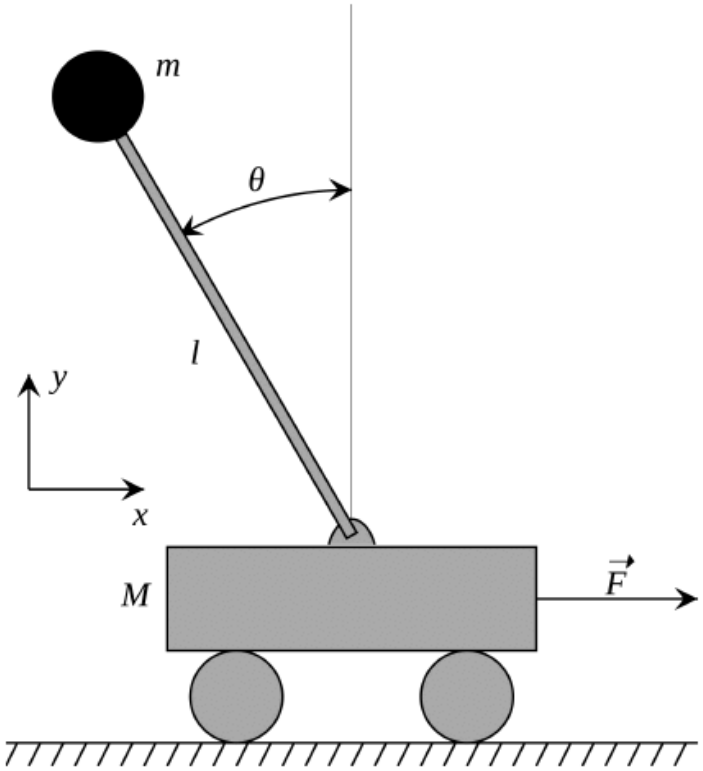
Agent **Environment** **Reward**





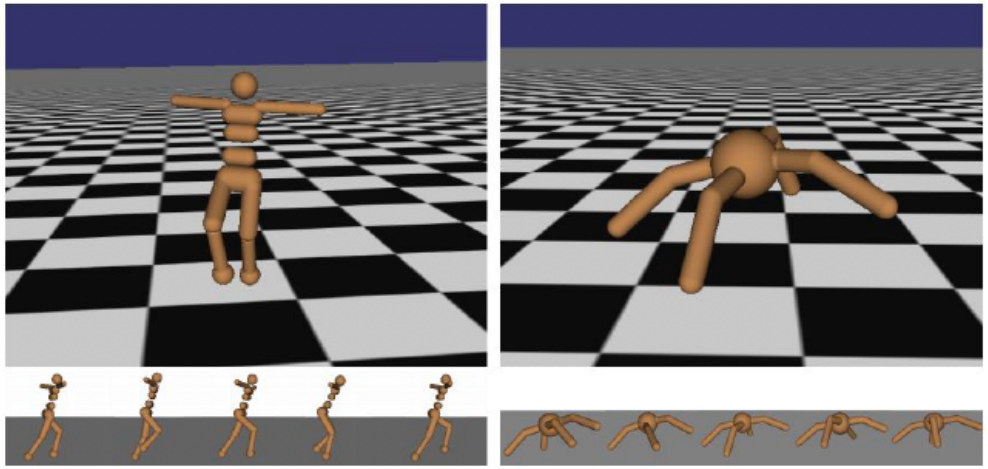
Cart-Pole Problem

- **Objective:** Balance a pole on top of a movable cart
- **State:** angle, angular speed, position, horizontal velocity
- **Action:** horizontal force applied on the cart
- **Reward:** 1 at each time step if the pole is upright



Robot Locomotion

- **Objective:** Make the robot move forward
- **State:** angle and position of the joints
- **Action:** torques applied on joints
- **Reward:** 1 at each time step upright forward movement

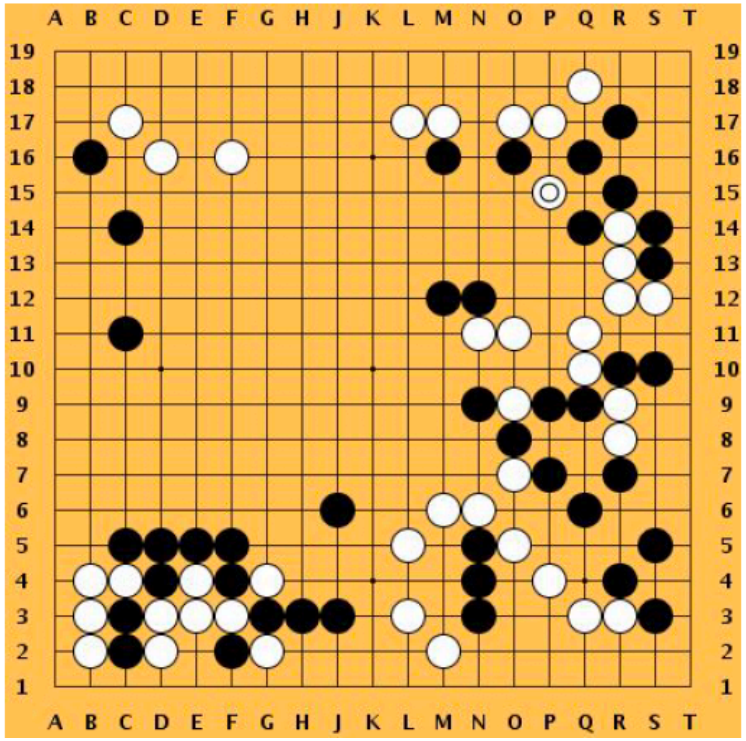


Atari Games

- **Objective:** Complete the game with highest score
- **State:** Raw pixel inputs of the game state
- **Action:** Game controls (Right, Left, Up, Down)
- **Reward:** score increase/decrease at each time step

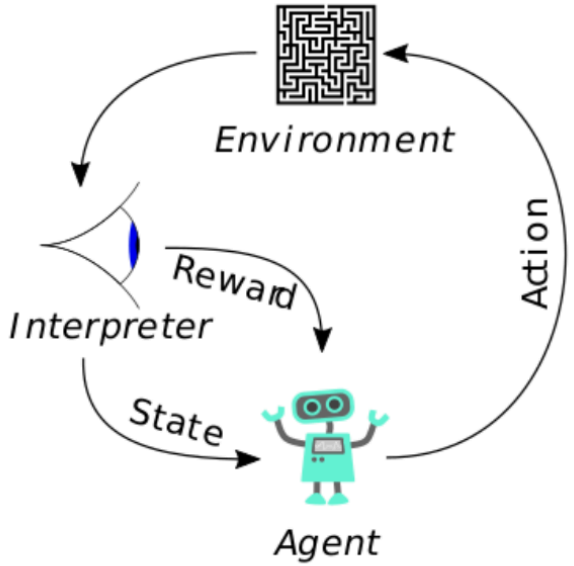
Go

- **Objective:** win the game!
- **State:** Position of all pieces
- **Action:** Where to put the next piece down
- **Reward:** 1 if win at the end of the game, 0 otherwise



Reinforcement Learning

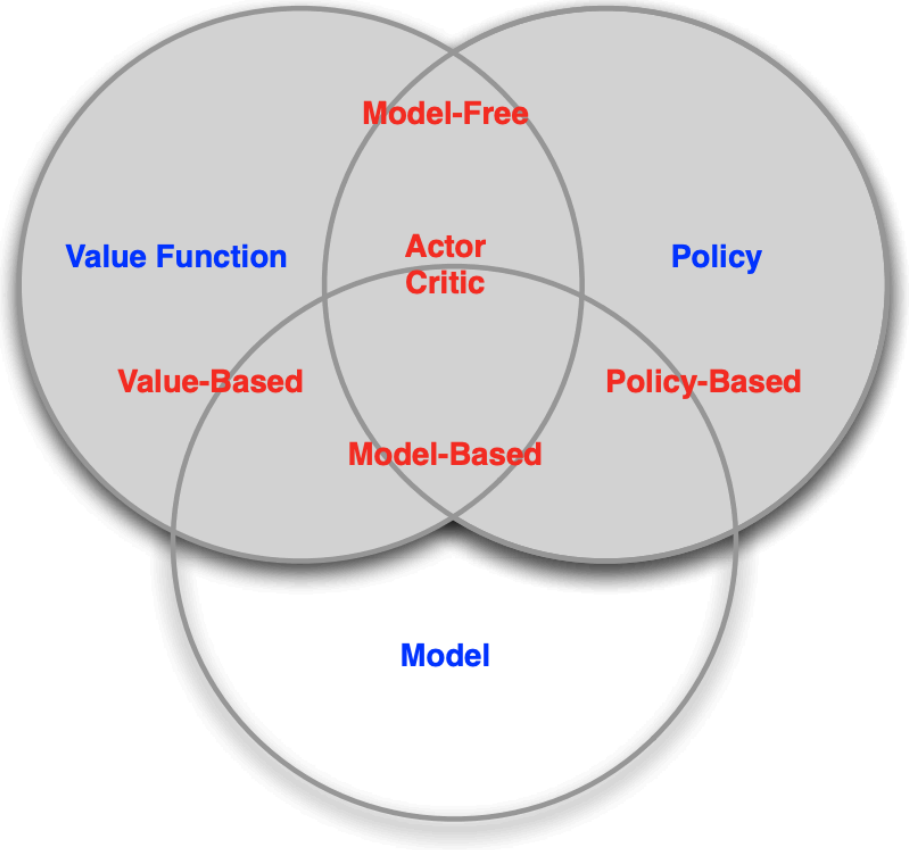
- ***Reinforcement Learning*** is a part of Machine learning where an agent is put in an environment and he learns to behave in this environment by performing certain actions and observing the rewards which it gets from those actions.



The typical RL scenario: an agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent.

Reinforcement Learning

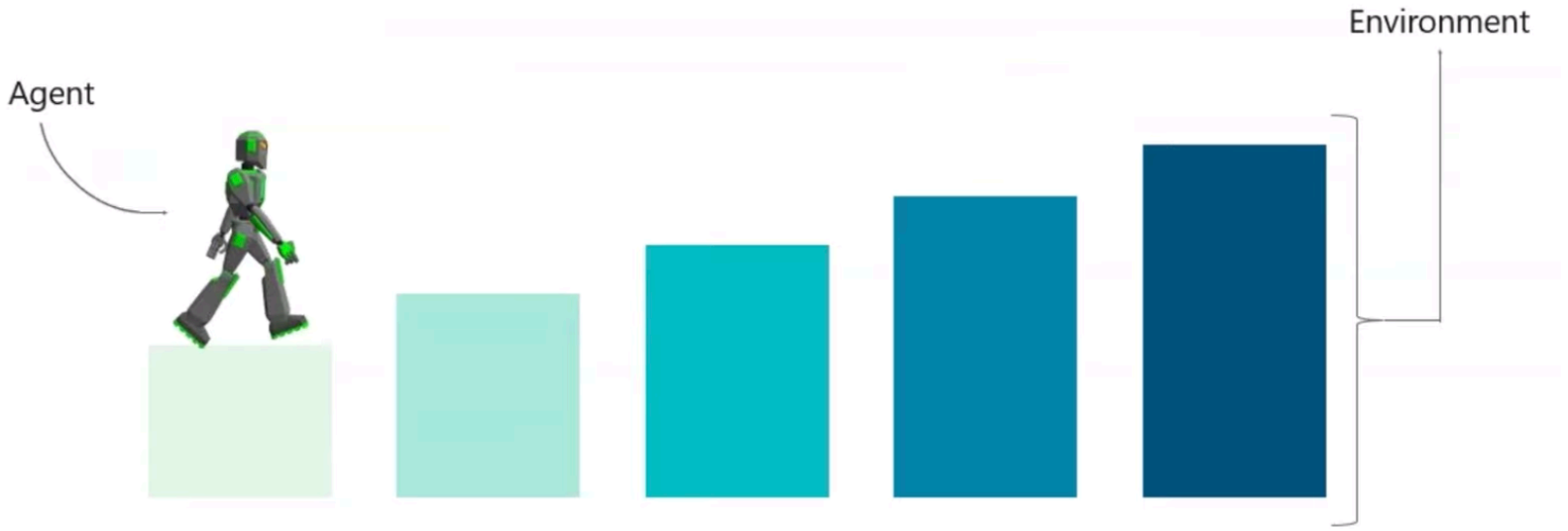
- Definitions:
  - **Agent**: The RL algorithms that learns from trail and error.



Agent Taxonomy

Reinforcement Learning

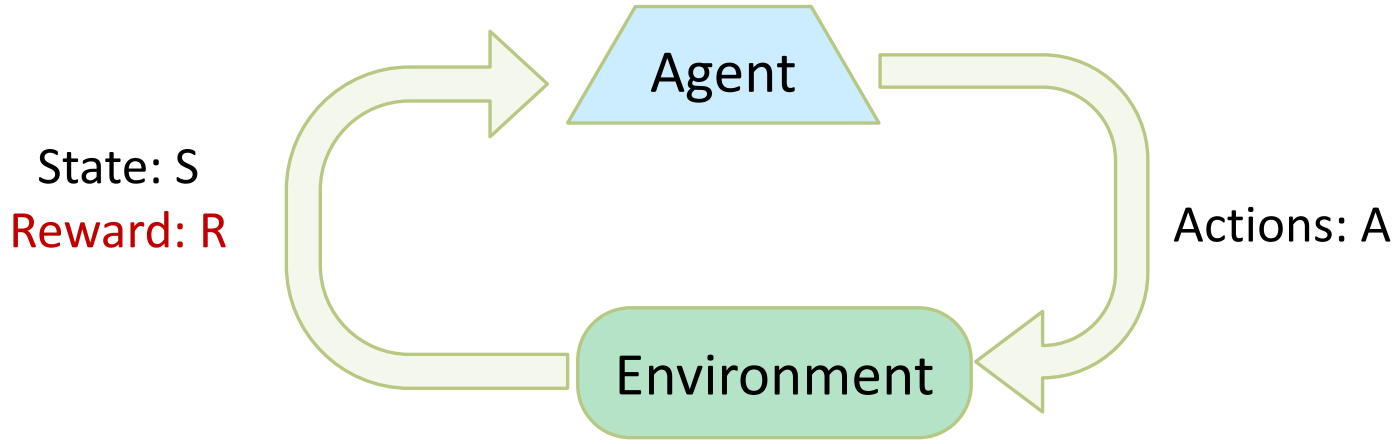
- Definitions:
  - **Environment**: The world through which the agent moves.
  - **Action (A):** All the possible steps that the agent can take
  - **State (S):** Current condition returned by the environment.



Reinforcement Learning

- Definitions:
  - **Reward (R):** An instance return from the environment to appraise the last action.
  - **Policy (π):** The approach that the agent uses to determine the next action based on the current state.
  - **Value (V):** The expected long-term return with discount, as opposed to the short-term reward R.
  - **Action-value (Q):** This is similar to Value, except, it takes an extra parameter, the current action (A).

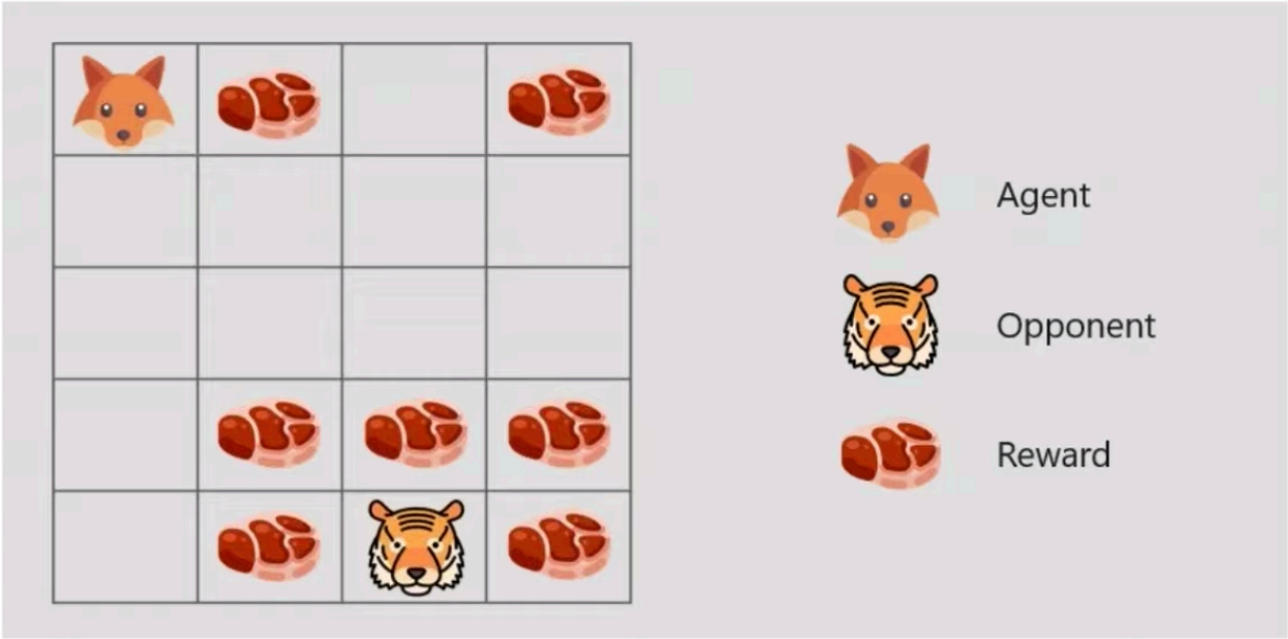
Reinforcement Learning



- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

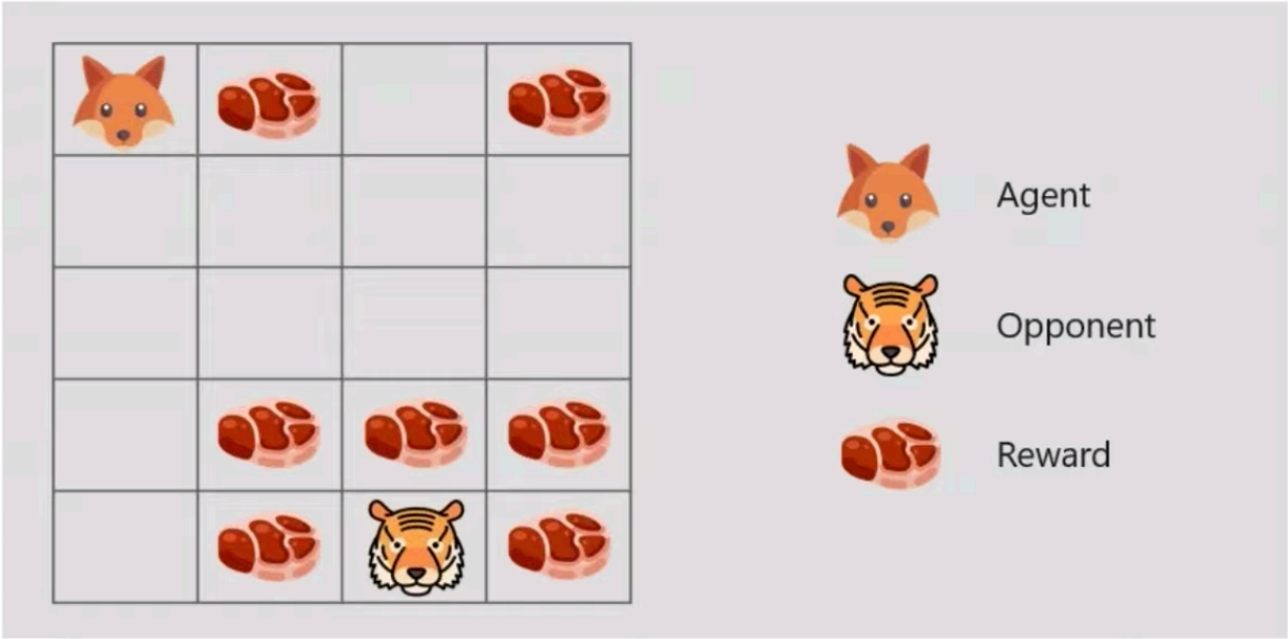
Reward Maximization

- Reward Maximization theory states that, a RL agent must be trained in such a way that, he takes the best action so that the reward is maximum.



Exploration & Exploitation

- ***Exploitation*** is about using the already known exploited information to heighten the rewards.
- ***Exploration*** is about exploring and capturing more information about an environment.

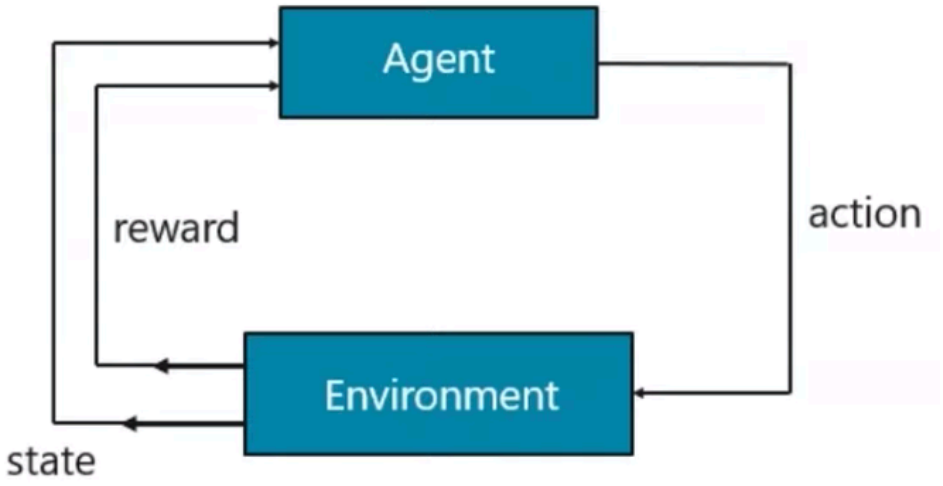


Exploration & Exploitation

Restaurant Selection	
Go to your favourite restaurant	Exploitation
Try a new restaurant	Exploration
Online Banner Advertisements	
Show the most successful advert	Exploitation
Show a different advert	Exploration
Oil Drilling	
Drill at the best known location	Exploitation
Drill at a new location	Exploration
Game Playing	
Play the move you believe is best	Exploitation
Play an experimental move	Exploration

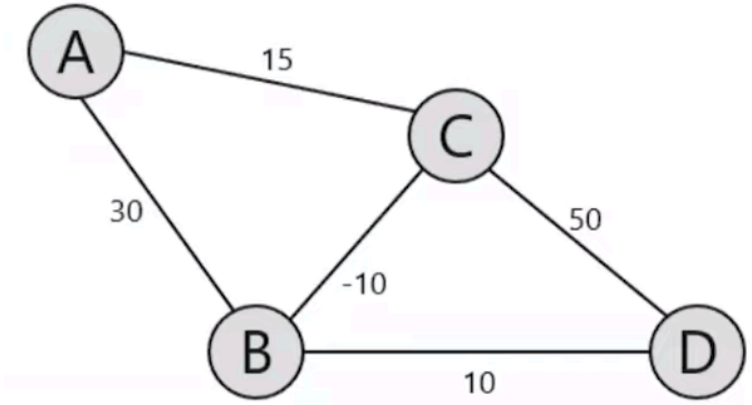
Markov's Decision Process

- The mathematical approach for mapping a solution in reinforcement learning is called ***Markov Decision Process (MDP).***
- The following parameters are used to attain a solution:
  - Set of actions, A
  - Set of states, S
  - Reward, R
  - Policy, π
  - Value, V



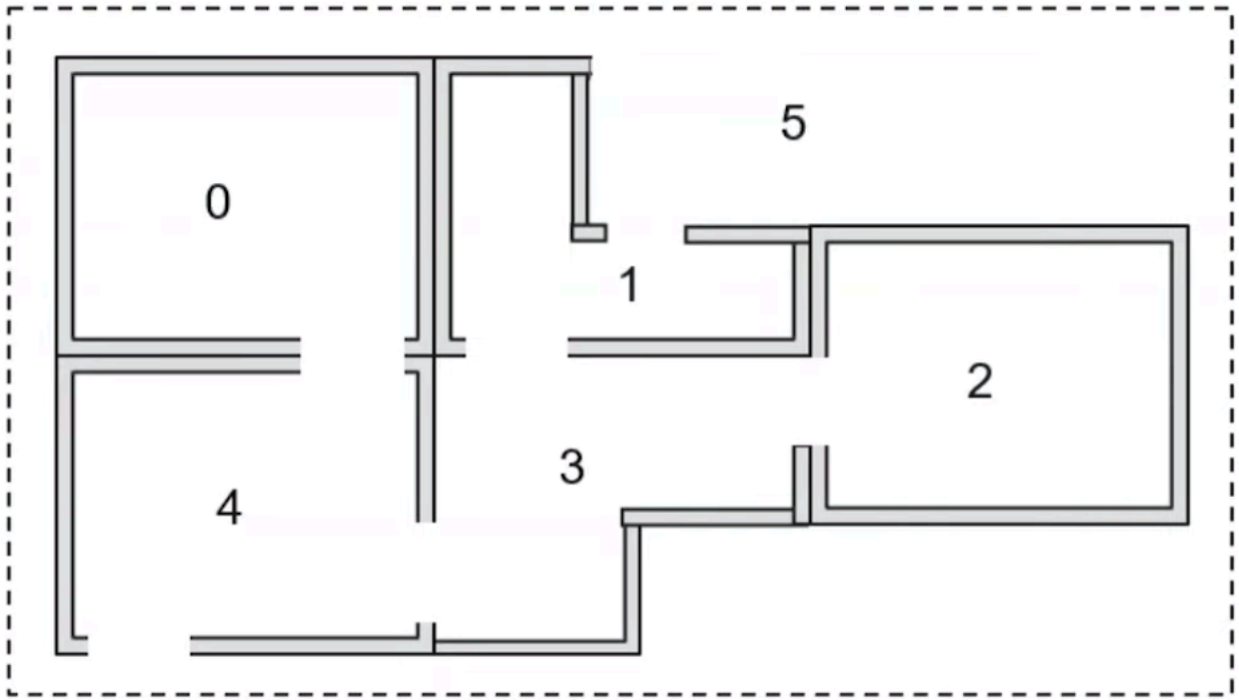
Markov's Decision Process

- Example: Find the shortest path between A and D with minimum possible cost.
- In this problem,
  - Set of states are denoted by nodes, i.e. {A, B, C, D}
  - Action is to traverse from one node to another {A→B, C→D}
  - Reward is the cost represented by each edge
  - Policy is the path taken to reach the destination {A→B→D}



Understanding Q-Learning

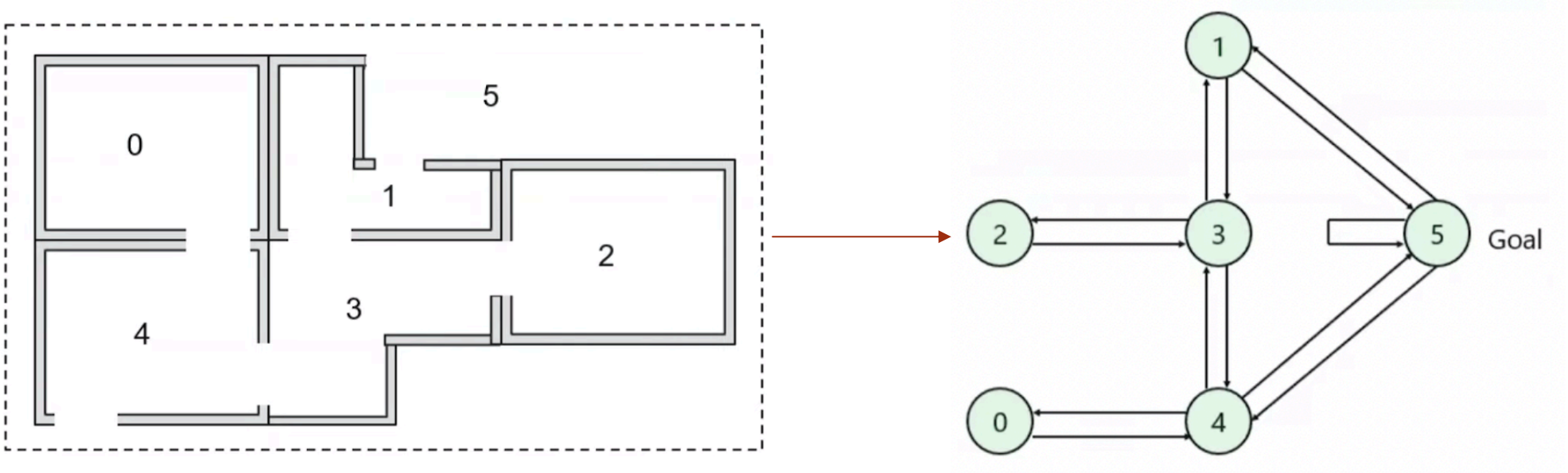
- Place an agent in any one of the rooms (0,1,2,3,4) and the goal is to reach outside the building (5)



- 5 Rooms in a building connected by doors
- Each room is numbered 0 – 4
- The outside of the building can be thought of as one big room (5)
- Doors 1 and 4 lead into the building from room 5 (outside)

Understanding Q-Learning

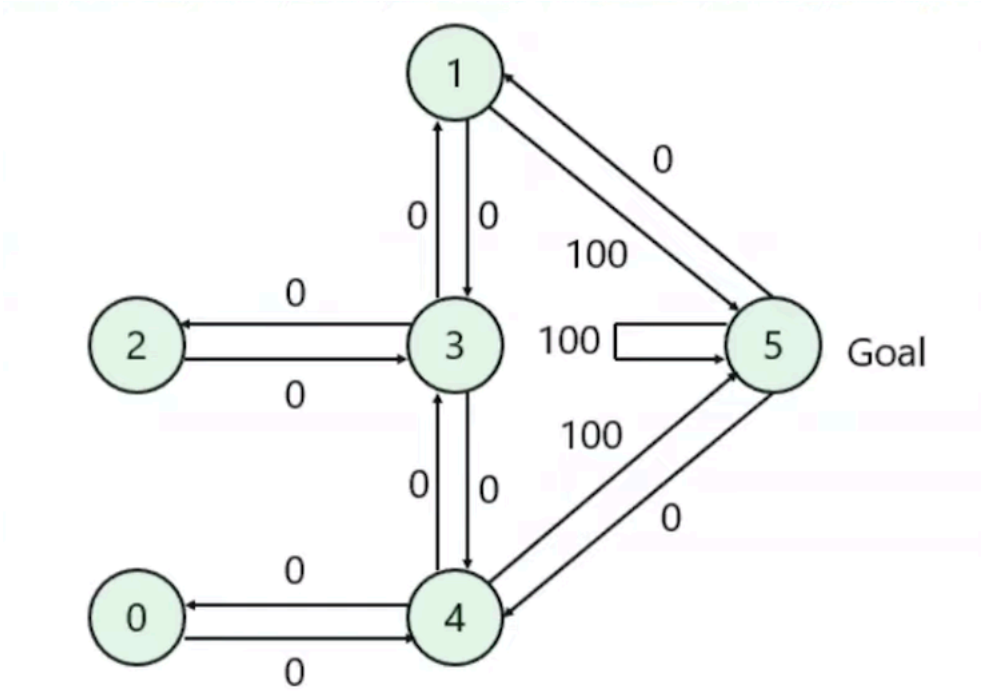
- Let's assume the rooms on a graph, each room as a node, and each door as a link.



Understanding Q-Learning

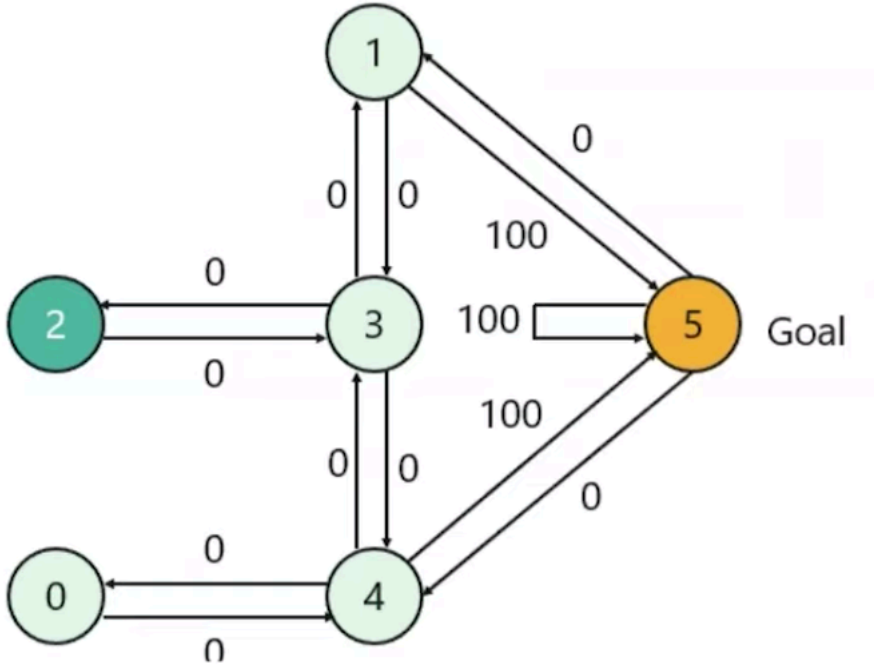
- Next step is to associate a reward value to each door:

- Doors that lead directly to the goal have a reward of 100
- Doors not directly connected to the target room have 0 reward
- Because doors are two-ways, two arrows are assigned to each room
- Each arrow contains an instant reward value



Understanding Q-Learning

- The terminology in Q-learning includes the terms state and action:
  - Room (including room 5) represents a state
  - Agent's movement from one room to another represents an action
  - In the figure, a state is depicted as a node, while "action" is represented by the arrows

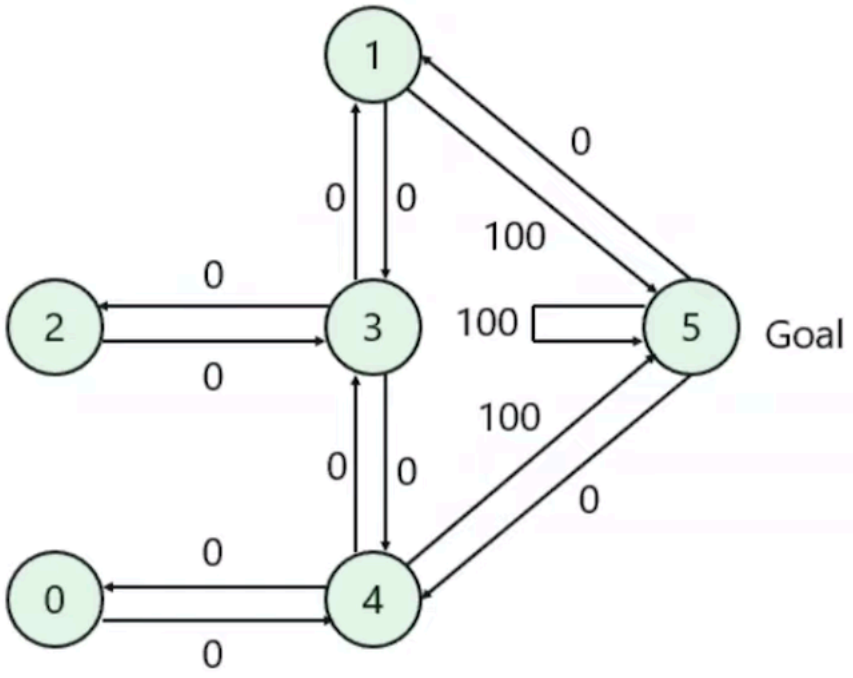


Example: agent traverse from 2 to 5

1. Initial State = state 2
2. State 2 → state 3
3. State 3 → state (2, 1, 4)
4. State 4 → state 5

Understanding Q-Learning

- We can put the state diagram and the instant reward values into a reward table, matrix R.



$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

The -1's in the table represent null values

Understanding Q-Learning

- Add another matrix Q representing the memory of what the agent has learned through experience.
  - The rows of matrix Q represent the current state of the agent
  - Columns represent the possible actions leading to the next state
  - Formula to calculate the Q matrix:

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

- Note: the Gamma parameter has a range of 0 to 1 (0<= Gamma>1)

If Gamma is closer to 0, the agent will tend to consider only immediate rewards.

If Gamma is closer to 1, the agent will consider future rewards with greater weight (Exploration).

Q-Learning Algorithm

- Set the gamma parameter, and environment rewards in matrix R
- Initialize matrix Q to 0
- Select a random initial state
- Set initial state = current state
- Select one among all possible actions for the current state
- Using this possible action, consider going to the next state
- Get maximum Q value for this next state based on all possible actions
- Compute:

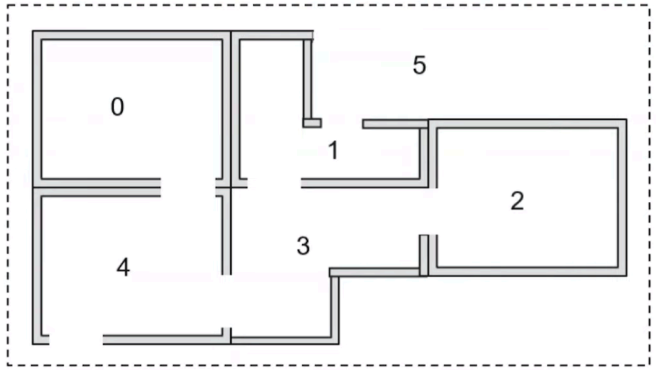
Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

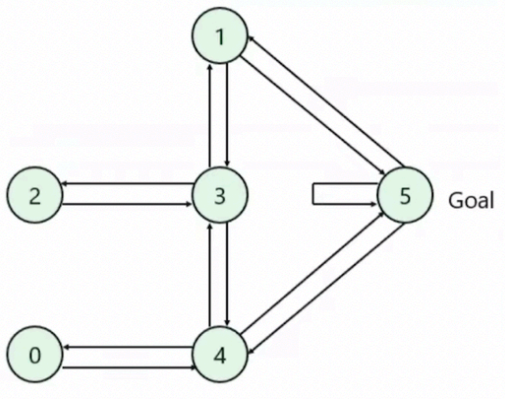
- Repeat above steps until current state = goal state

Q-Learning Example

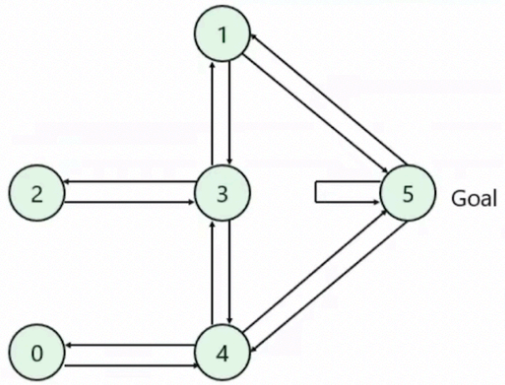
- First step is to set the value of the learning parameter Gamma = 0.8, and the initial state as Room 1.
- Next, initialize matrix Q as a zero matrix:
  - From room 1 you can either go to room 3 or 5, let's select room 5.
  - From room 5, calculate maximum Q value for this next state based on all possible actions:

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]





Q-Learning Example



- Next, initialize matrix Q as a zero matrix:
  - From room 1 you can either go to room 3 or 5, let's select room 5.
  - From room 5, calculate maximum Q value for this next state based on all possible actions:

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

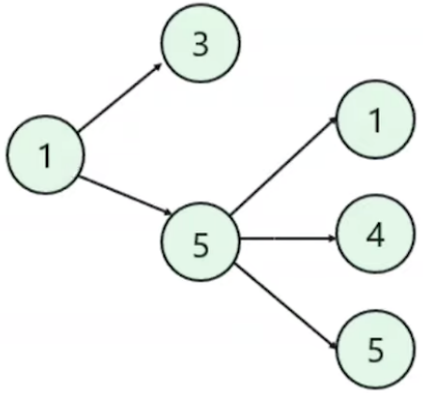
Q(1,5) = R(1,5) + 0.8*Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8*0 = 100

$Q =$

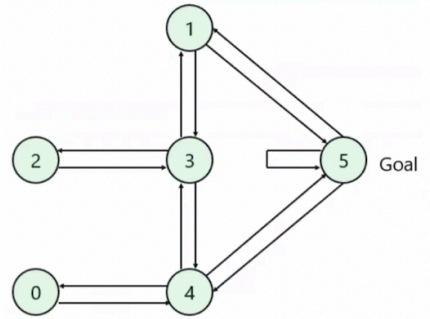
	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100



Q-Learning Example



- For the next episode: we start a randomly chosen initial state, i.e. state 3
  - From room 3 you can either go to room 1,2 or 4, let's select room 1.
  - From room 1, calculate maximum Q value for this next state based on all possible actions:

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

Q(3,1) = R(3,1) + 0.8*Max[Q(1,3), Q(1,5)] = 100 + 0.8*[0, 100] = 80

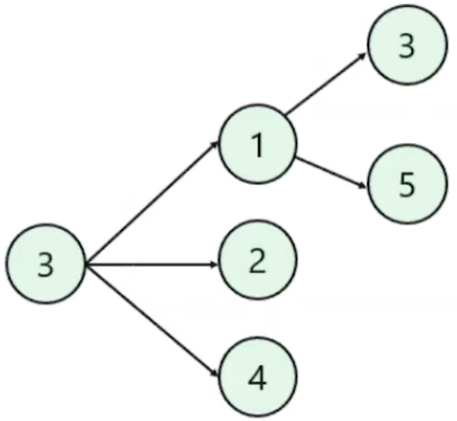
  - The matrix Q gets unpadted

$Q =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	80	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100



Q-Learning Example

- For the next episode, the next state 1 now becomes the current state. We repeat the inner loop of Q learning algorithm because 1 is not the goal state.
  - From room 1 you can either go to room 3 or 5, let's select room 5.
  - From room 5, calculate maximum Q value for this next state based on all possible actions:

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

Q(1,5) = R(1,5) + 0.8*Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8*0 = 100

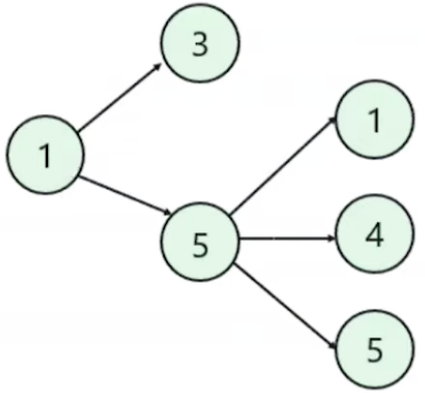
  - The matrix Q remains the same since Q(1,5) is already fed to the agent

$Q =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	80	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100



Example: learning to walk

Initial

A Learning Trial

After Learning [1K Trials]

[Kohl and Stone, ICRA 2004]

Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – initial]

Example: Learning to Walk

Training

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – training]

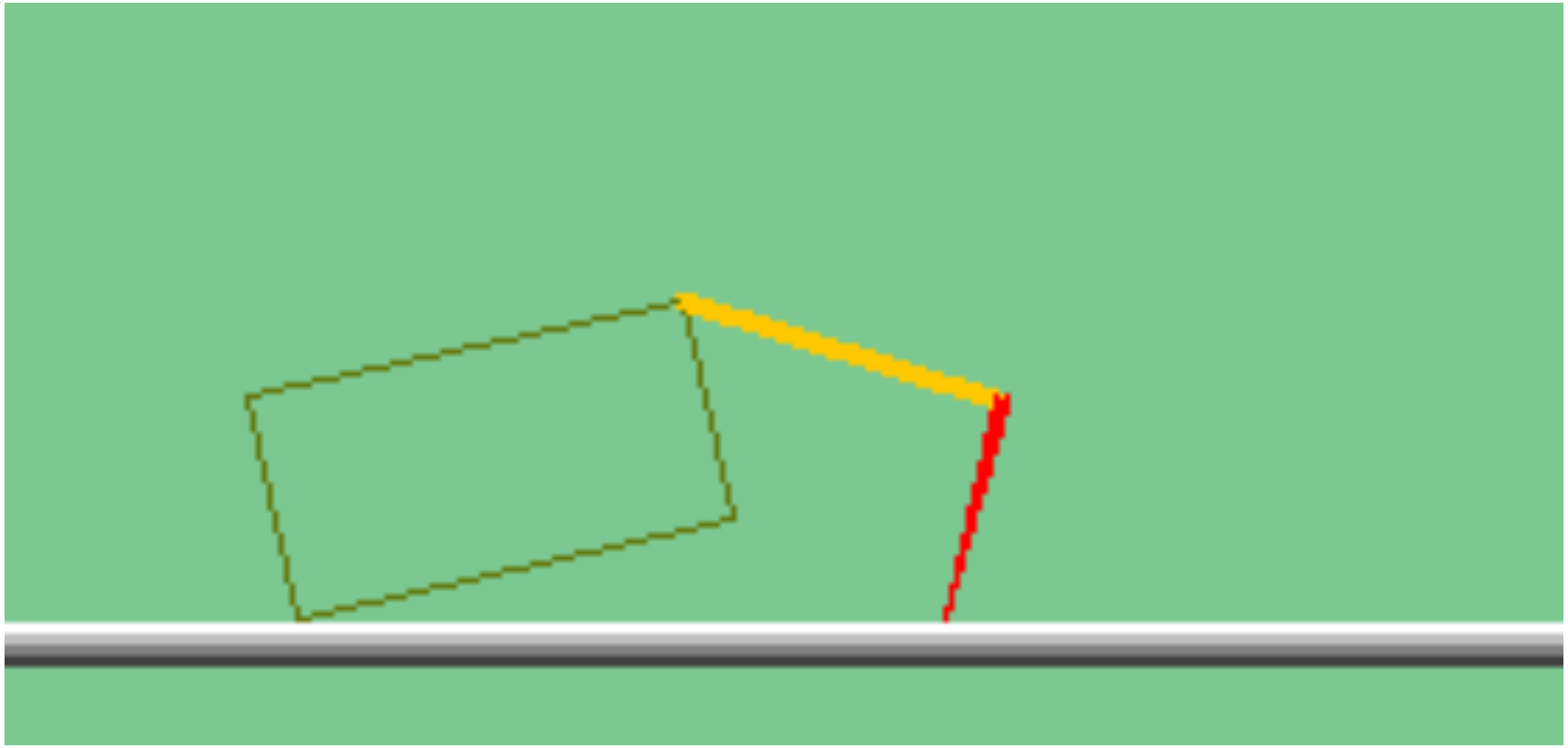
Example: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – finished]

The Crawler!



Video of Demo Crawler Bot

Q-Learning

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)

The End