Lecture 3: Reinforcement Learning: The Basics

- ECS7002P – Artificial Intelligence in Games
- Raluca D. Gaina – r.d.gaina@qmul.ac.uk
- Office: CS.335

Game AI Group

http://gameai.eecs.qmul.ac.uk

Queen Mary University of London

Outline

- ❑ Heuristic Functions
- ❑ Introduction to Reinforcement Learning
- ❑ Markov Decision Processes

Reinforcement Learning: The Basics

Heuristic Functions

Terminology

Reward

Objective

Value

Loss

Heuristic

Cost

Utility

Evaluation

Fitness

Error

Game States and Goals

Game state = a collection of different **features** describing a moment in the game:

- The game board / level grid / graph
- Positions of objects in the level
- Properties of players (health, speed, inventory items etc.)
- An image of the game area
- ...

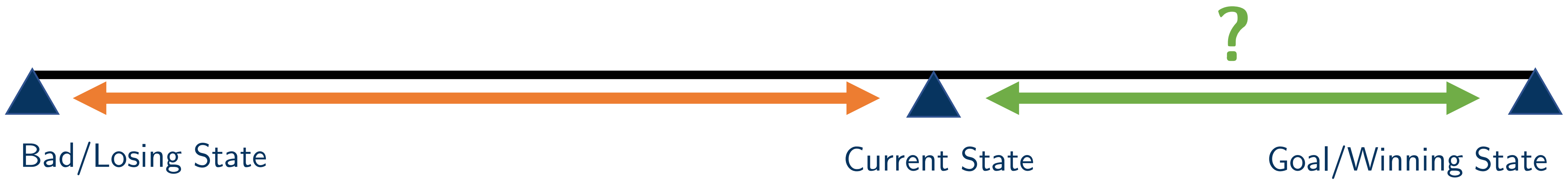
Goal states in games can be **abstract** and described in only some of the features:

- There may be more than 1 winning game board configuration, but they will all have some features in common.
- Heuristic functions analyze the game state features and compare them to the abstract definition of a goal state, e.g.:
    - **No other players left alive in the game** -> Minimize how many are currently alive, ***P***.
    - **Avatar position in the center of the board** -> Minimize the current distance from the avatar position to center, ***D***.
    - **High game score** -> Maximize the current game score, ***Sc***.
    - $\boldsymbol{h(s) = -P - D + Sc}$

Heuristic Function

Purpose: guide the search in the more promising areas. Usually used to rank branches in tree search algorithms and select the best ones, based on current information available.

In general, informs how good a particular state is, in relation to a goal state.



What if we don't know the goal state?

- General video game playing: unknown game played.
- Use generic information (e.g. most games have a score that tells the player how well they're doing).
- Make assumptions (may fail in some cases) or try to **learn** what a goal state is, what features are important, how to get there.
  - E.g. AlphaGo uses Neural Networks to learn a value function used in MCTS.
- No free lunch theorem: there is no single heuristic that is ideal for all possible tasks.

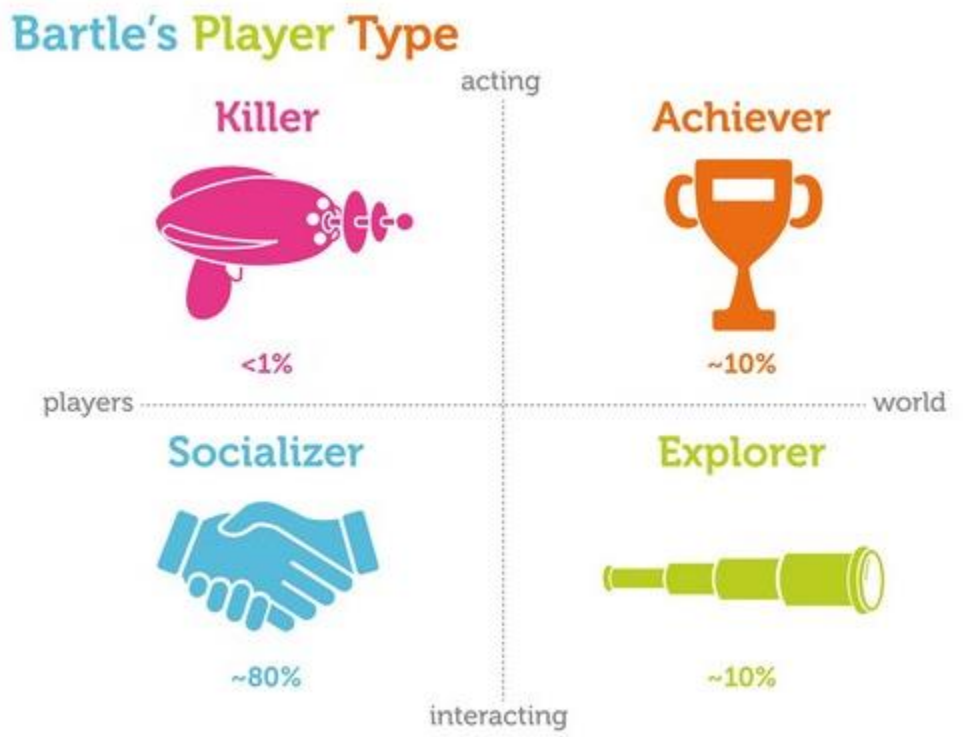
Different Goal States

Game-playing algorithms are great problem solvers, they usually try to win games / solve the problem. But what if our goal is not winning?

- Testing games, find bugs and odd behavior
- Player experience

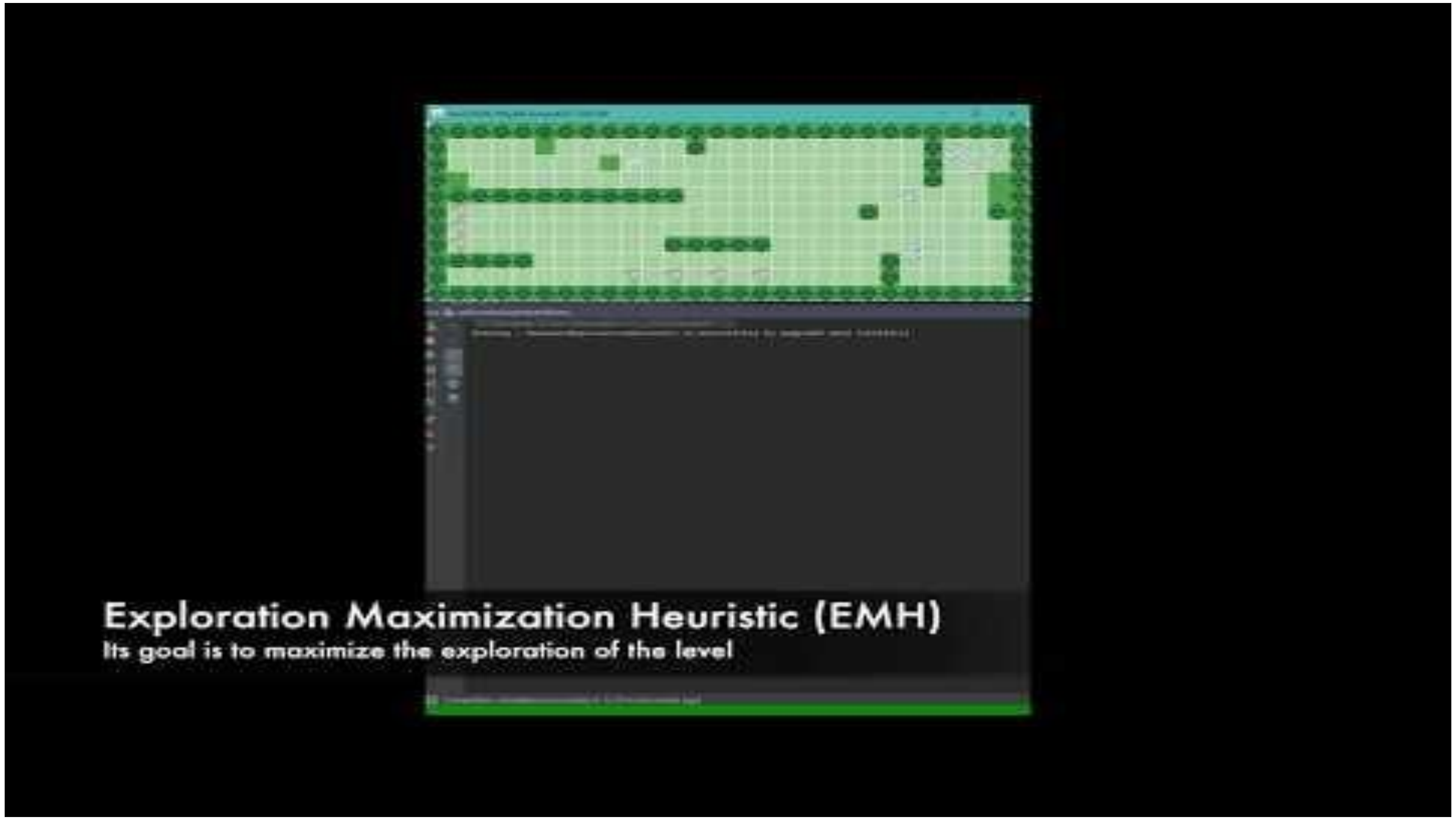
Modelling players, different types:

- Exploring the game world
- Interacting with game objects & other players
- Learning about the world
- Social gamers



Cristina Guerrero-Romero, Simon M Lucas, Diego Perez-Liebana, **Using a Team of General AI Algorithms to Assist Game Design and Testing**, in *Proceedings on the Conference on Computational Intelligence and Games (CIG)* (2018)

Diverse Heuristics


Exploration Maximization Heuristic (EMH)
Its goal is to maximize the exploration of the level

https://www.youtube.com/watch?v=aLgPm9kbfY8

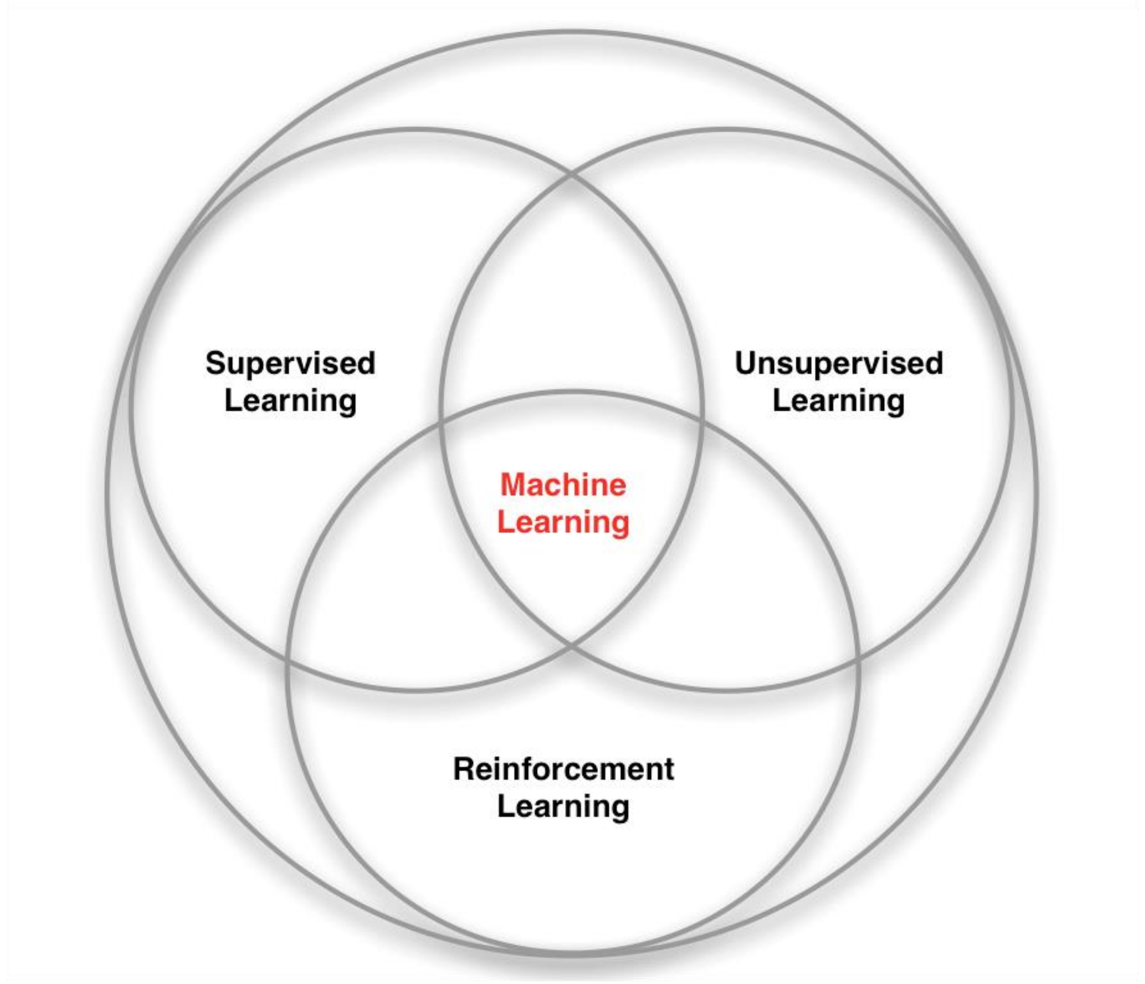
Outline

- ✓ Heuristic Functions
- ❑ Introduction to Reinforcement Learning
- ❑ Markov Decision Processes

Reinforcement Learning: The Basics

Introduction to Reinforcement Learning

Reinforcement Learning in AI

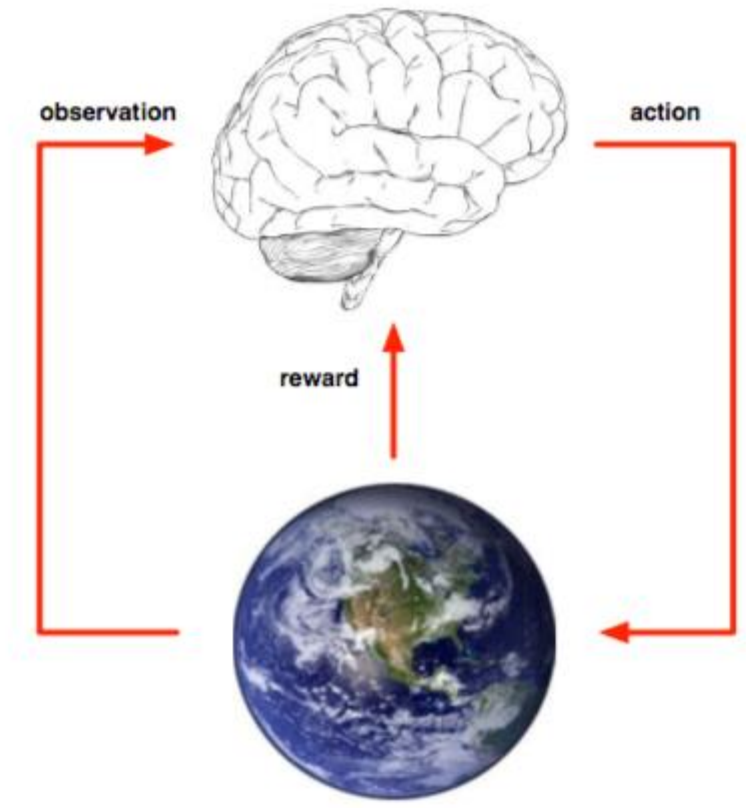


The Reinforcement Learning Problem

- No supervisor, only a reward signal
- Feedback is delayed
- Sequential samples, not iid (time matters)
- Actions influence future observations

- Example: games (but not only!)

idd: independently and identically distributed.



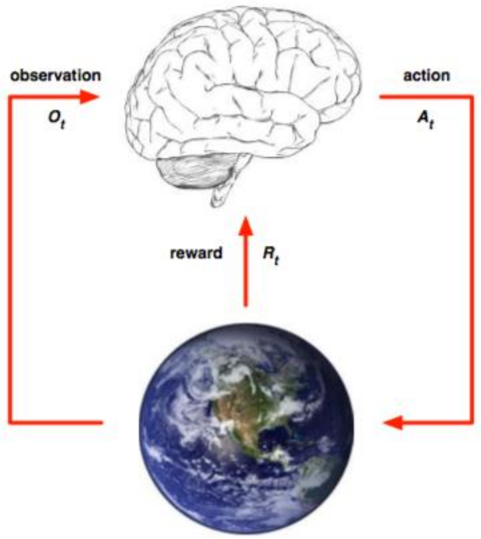
The Reinforcement Learning Problem

- Reward
  - R_t is a scalar feedback signal
  - Indicates how well the agent is doing at step t
  - The agent's job is to maximize the **cumulative** reward: the sum of rewards across an *episode*.

- Sequential decision making
  - Goal: select actions to maximize total future reward
  - Actions may have long term consequences
  - Reward may be delayed
  - Immediate vs long-term reward

The Reinforcement Learning Loop

- At each step t, the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The Environment:
  - Receives action A_t.
  - Emits observation O_{t+1} ($\neq S_{t+1}$).
  - Emits scalar reward R_{t+1}.
- t increments at every step.



If you have a Forward Model, this loop happens in the agent's *brain*. The agent simulates actions in their thinking time because it has a model of the world.

In general, in Reinforcement Learning, we may not have that model. When this is the case, the RL loop happens *outside* the brain. This is, the agent performs actions in the real game/world. Typically, it has to play the game multiple times to learn how to act properly. Let's assume that, in general, we don't have that model.

History

The history is the sequence of observations, actions and rewards:

$$H_t = O_1 R_1 A_1, \dots, O_{t-1} R_{t-1} A_{t-1}$$

In the general case, what happens next depends on the history.

The **state** is the information used to determine what happens next and it is a function of the history:

$$S_t = f(H_t)$$

The Markov State

- A Markov state contains all useful information from the history

Definition (Markov State)

A state S_t is **Markov** if and only if:

$$\mathbb{P}(S_{t+1} \mid S_t) = \mathbb{P}(S_{t+1} \mid S_1, \ldots, S_t)$$

- "The future is independent of the past, given the present" (if S_t is known, H_t can be thrown away)

From Observation to States

Full observability:

- Agent observes the full environment
- This is a Markov Decision Process (**MDP**)
- E.g. Chess

Partial observability:

- Agent observes part of the environment
- This is a Partially Observable Markov Decision Process (**POMDP**)
- E.g. Poker

Components of an RL Agent

Model: p, r

- It is the agent's representation of the environment.
- Predicts what the environment will do next.
- p predicts the next state:

$$p(s'|s,a) = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$$

- r predicts the next (immediate) reward:

$$r(s,a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

Components of an RL Agent

Value function: v

- It is a quantitative measure of how good a state or an action is.
- Predicts the future reward.
- Allows action selection.

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ \dots \mid S_t = s]$$

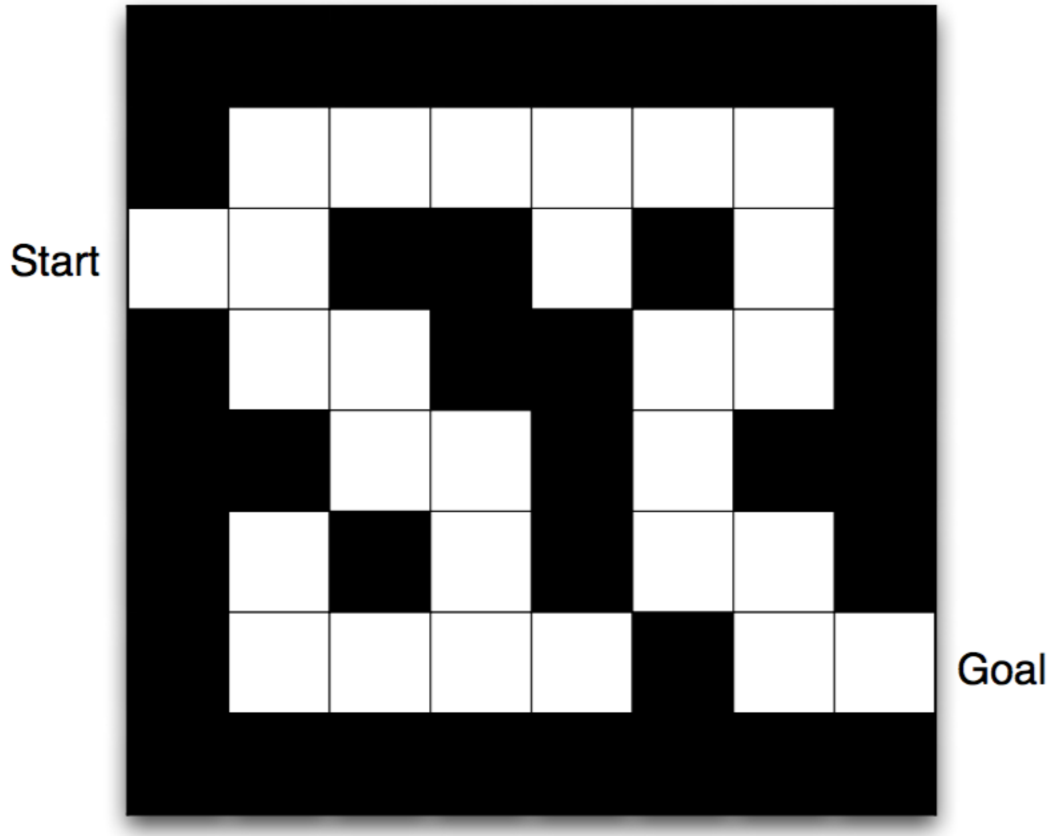
Components of an RL Agent

Policy: π

- It is the agent's behavior.
- A policy is a map from states (S) to actions (A).
- It can be:
    - Deterministic: a $= \pi(s)$
    - Probabilistic: $\pi(a|s) = \mathbb{P}(A_t = a \mid St = s)$

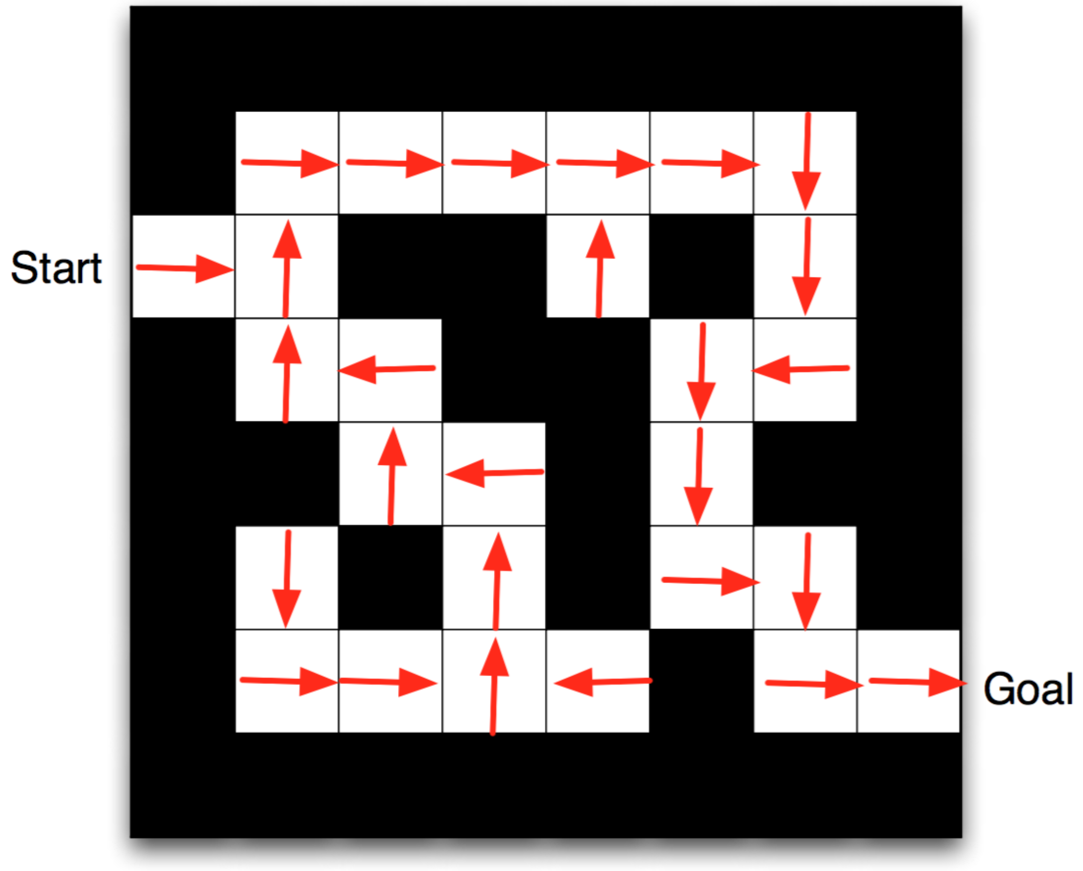
Example: Maze

- **Reward**: -1 per movement step
- **Actions**: N, S, W, E
- **State**: the agent's location
- Game (*episode*) ends when the agent reaches the goal position.



Example: Maze

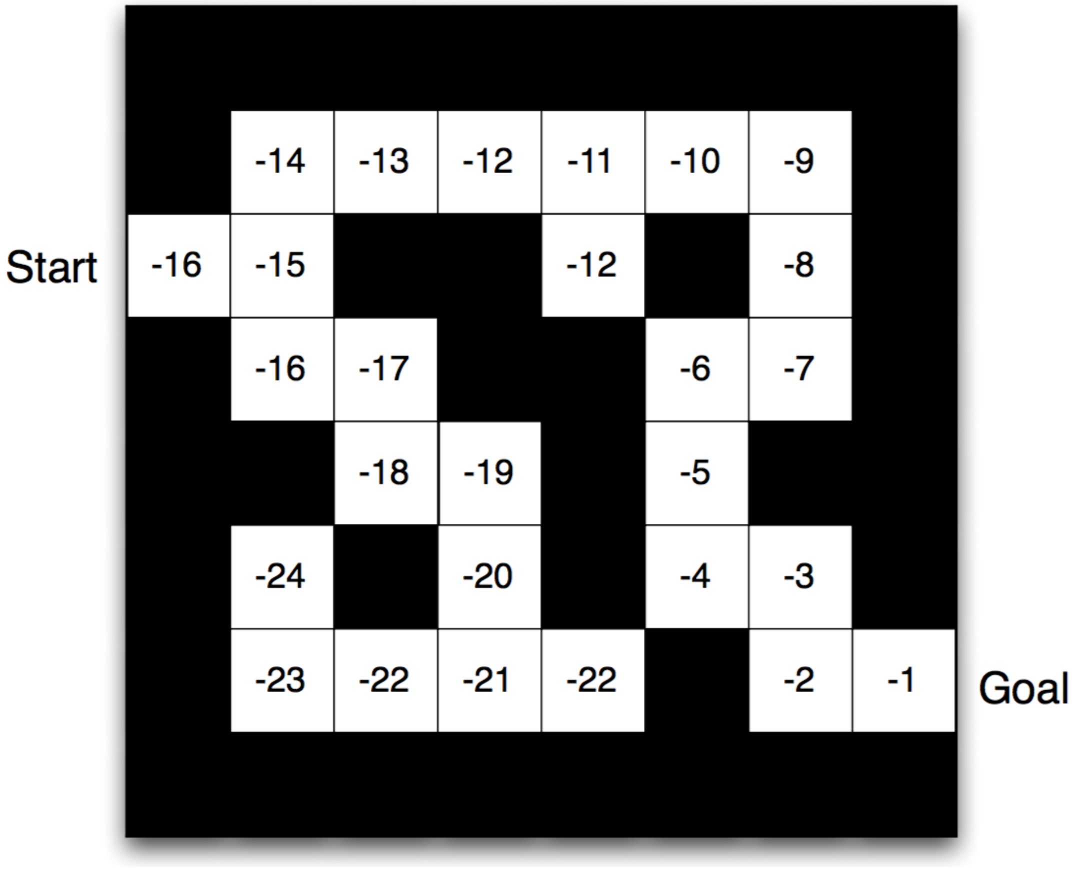
Policy:



- Arrows represent the optimal action from each cell (state s).
- Policy $\pi(s)$ is deterministic, for every state s.

Example: Maze

Value function:



- Numbers are the values $v_\pi(s)$ for each state s.
- For each state (position) in the grid, $v_\pi(s)$ indicates the expected reward from state s, **following** policy π (see previous slide).
- Tells the number of moves until the goal (because of the reward choice).

Categorizing RL

Policy and/or Value function:

RL Category	Policy	Value Function
Value Based	No	Yes
Policy Based	Yes	No
Actor Critic	Yes	Yes

With or without model:

RL Category	Problem	Policy and/or Value Function	Model
Model Based	Planning	Yes	Yes
Model Free	Learning	Yes	No

Outline

- ✓ Heuristic Functions
- ✓ Introduction to Reinforcement Learning
- ❑ Markov Decision Processes

Reinforcement Learning: The Basics

Markov Decision Processes

Markov Decision Process

- A Markov Decision Process (MDP) formally describes an environment for RL.
- The environment is fully observable.
- Almost all RL problems can be formalized as an MDP.

Remember the Markov Property:

Definition (Markov State)

A state S_t is **Markov** if and only if:

$$\mathbb{P}(S_{t+1} \mid S_t) = \mathbb{P}(S_{t+1} \mid S_1, \ldots, S_t)$$

We are going to build up until the MDP definition.

Markov Process (or Markov Chain)

A Markov Process (or Markov Chain) is a memoryless random process (a sequence of random states S_1, S_2, … with the Markov property).

Definition (Markov Process)

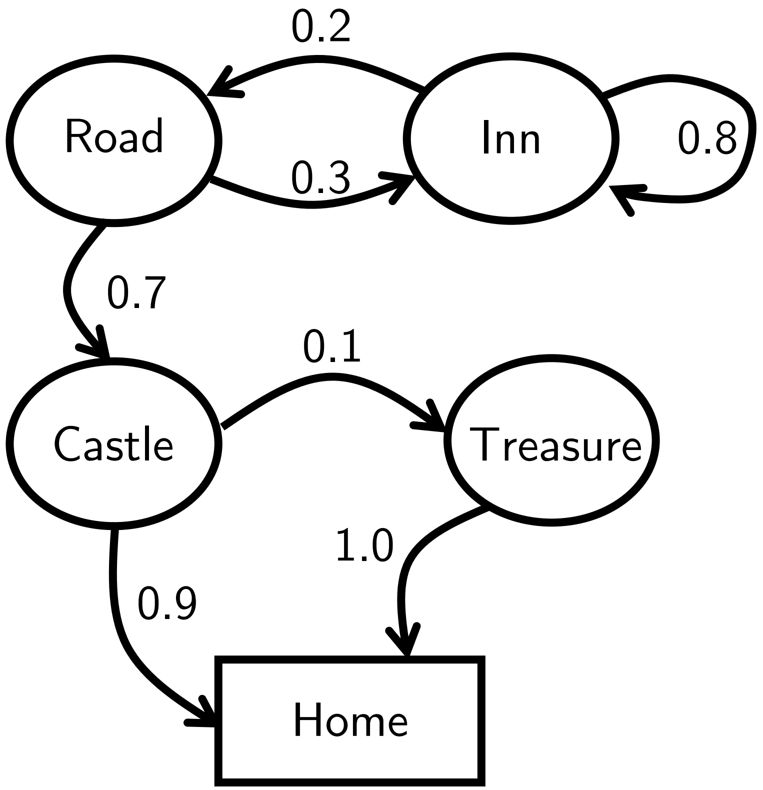
A Markov Process (or Markov Chain) is a tuple $< S, P >$:

- S is a finite set of states.
- P is a state transition probability matrix:

$$P_{ss'} = \mathbb{P}(S_{t+1} = s' | S_t = s)$$

Markov Process, Example: Rogue

Example: Rogue Markov Chain



Probability Transition Matrix $P_{ss'}$:

To (s') / From (s)	Road	Inn	Castle	Treasure	Home
Road (R)		0.3	0.7		
Inn (I)	0.2	0.8			
Castle (C)				0.1	0.9
Treasure (T)					1.0
Home (H)					

Valid Markov Chains (episodes) starting from Road (R) are: *RIIIRCH*, *RCTH*, *RIRCTH*, ...

`http://setosa.io/ev/markov-chains`

Markov Reward Process

A Markov Reward Process is a Markov Chain with rewards.

Definition (Markov Reward Process)

A Markov Reward Process is a tuple $< S, P, R, \gamma >$:

- S is a finite set of states.
- P is a state transition probability matrix:

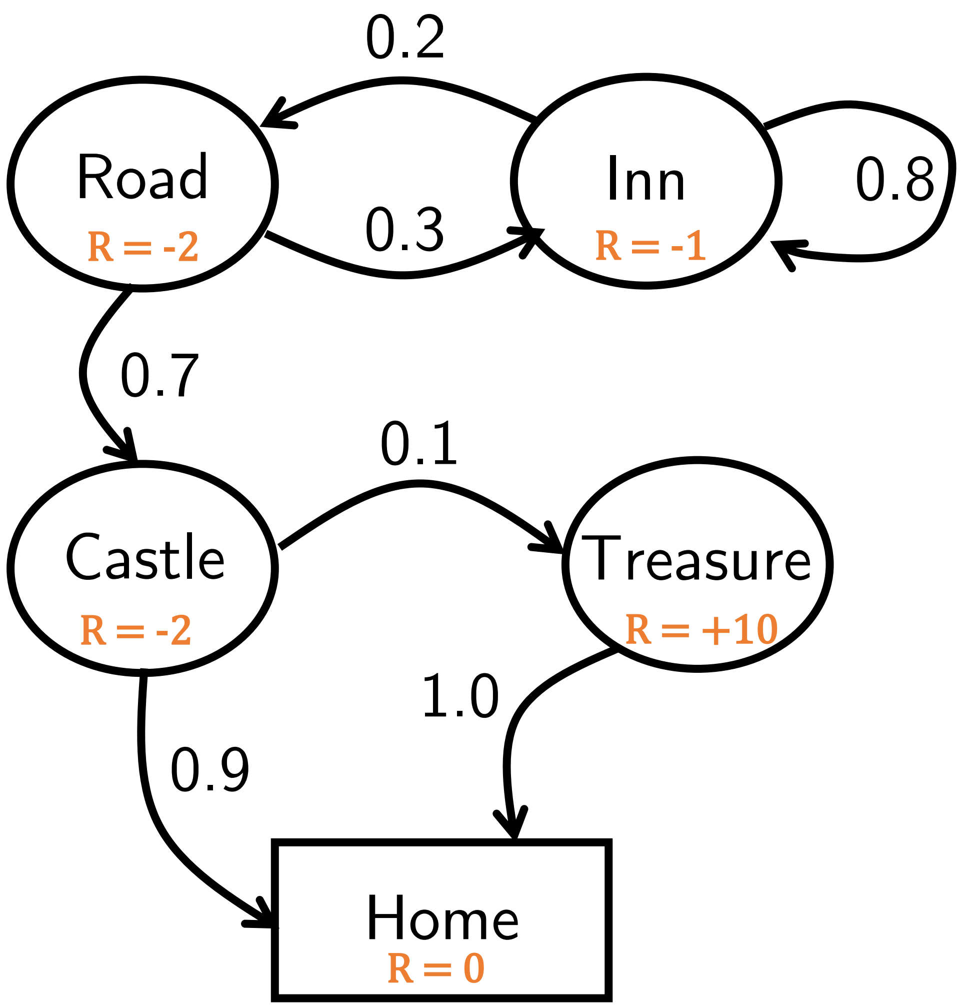
$$P_{ss'} = \mathbb{P}(S_{t+1} = s' \mid S_t = s)$$

- R is a reward function:

$$R_s = \mathbb{E}[R_{t+1} \mid S_t = s]$$

- γ is a discount factor, $\gamma \in [0,1]$.

Markov Reward Process, Example: Rogue



Return

Now that we have rewards, we can define *return*:

Definition (Return)

The return G_t is the total (discounted) reward from time-step t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

The discount $\gamma \in [0,1]$ influences the value of future rewards:

- $\gamma = 0$: “myopic” evaluation, only considering immediate rewards.
- $\gamma = 1$: “far-sighted” evaluation, considering all rewards in the future.

Q? Why discount?

Why Discount?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behavior shows preference for immediate reward
- It is sometimes possible to use *undiscounted* Markov reward processes ($\gamma = 1$), e.g. if all sequences terminate.

Value Function

The value function $v(s)$ gives the long-term value of the state s.

Definition (State-Value Function)

The *state-value* function $v(s)$ of an MRP is the expected return starting from state *s*:

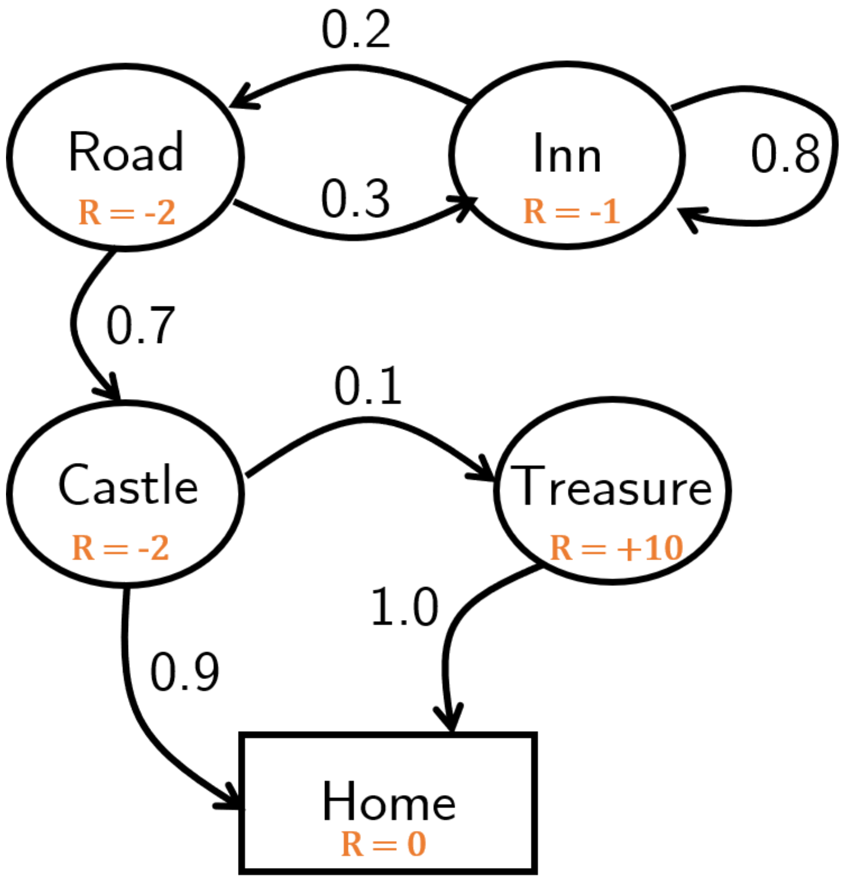
$$v(s) = \mathbb{E}(G_t | S_t = s)$$

It is the expected return (**average** of discounted rewards) an agent would obtain following nature (the rules that govern the environment, *P*).

Value Function in the Rogue Example

What is the value of $v(Road)$ considering only the Markov Chains *RIIIRCH*, *RCTH* and *RIRCTH*, with $\gamma = 0.5$?

The return of each chain would be, using the definition of G_t:
$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$

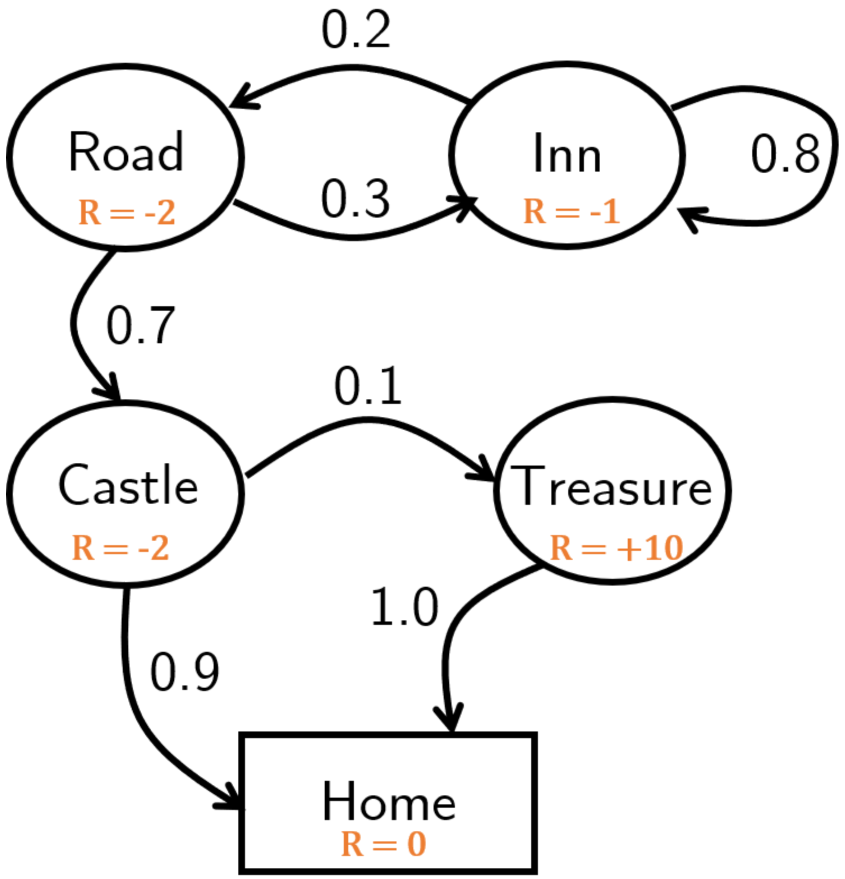


RCTH: $G_1 = -2 - 2 \times (0.5) + 10 \times (0.5)^2 + 0 \times (0.5)^3 = -2 - 2 \times 0.5 + 10 \times 0.25 + 0 = -0.5$

Value Function in the Rogue Example

What is the value of $v(Road)$ considering only the Markov Chains *RIIIRCH*, *RCTH* and *RIRCTH*, with $\gamma = 0.5$?

The return of each chain would be, using the definition of G_t:
$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$

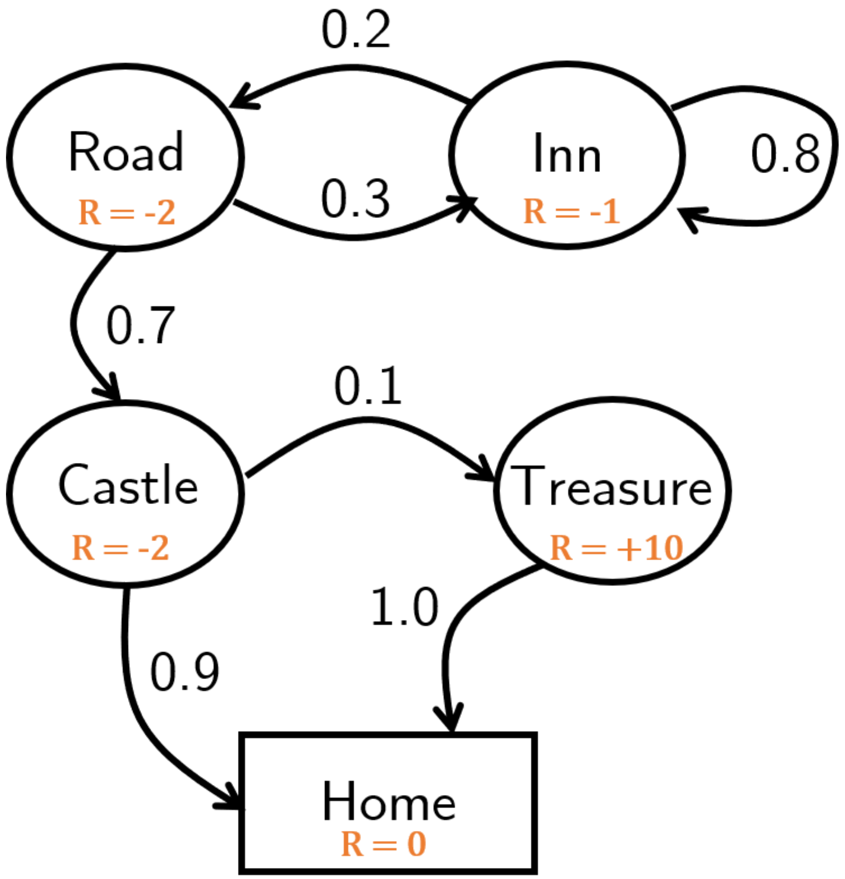


RIRCTH: $G_1 = -2 - 1 \times (0.5) - 2 \times (0.5)^2 - 2 \times (0.5)^3 + 10 \times (0.5)^4 + 0 \times (0.5)^5 = -2 - 1 \times 0.5 - 2 \times 0.25 - 2 \times 0.125 + 10 \times 0.0625 + 0 = -2.625$

Value Function in the Rogue Example

What is the value of $v(Road)$ considering only the Markov Chains *RIIIRCH*, *RCTH* and *RIRCTH*, with $\gamma = 0.5$?

The return of each chain would be, using the definition of G_t:
$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$



RIIIRCH: $G_1 = -2 - 1 \times (0.5) - 1 \times (0.5)^2 - 1 \times (0.5)^3 - 2 \times (0.5)^4 - 2 \times (0.5)^5 + 0 \times (0.5)^6 =$
$-2 \ - 0.5 - 0.25 - 0.125 - 2 \times 0.0625 - 2 \times 0.03125 + 0 = -3.0625$

Value Function in the Rogue Example

RCTH: $G_1 = -2 - 2 \times (0.5) + 10 \times (0.5)^2 + 0 \times (0.5)^3 = -2 \ - 2 \times 0.5 + 10 \times 0.25 + 0 = -0.5$

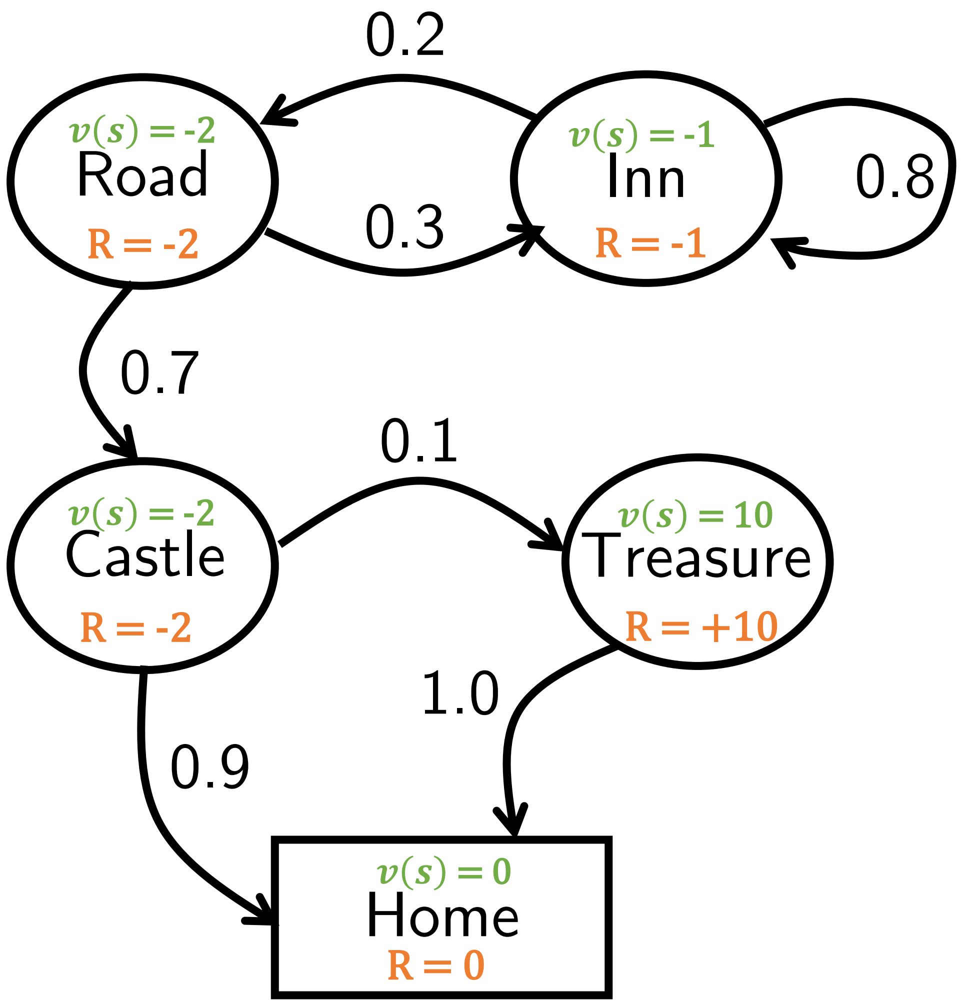
RIRCTH: $G_1 = -2 - 1 \times (0.5) - 2 \times (0.5)^2 - 2 \times (0.5)^3 + 10 \times (0.5)^4 + 0 \times (0.5)^5 = -2 \ - 1 \times 0.5 - 2 \times 0.25 - 2 \times 0.125 + 10 \times 0.0625 + 0 = -2.625$

RIIIRCH: $G_1 = -2 - 1 \times (0.5) - 1 \times (0.5)^2 - 1 \times (0.5)^3 - 2 \times (0.5)^4 - 2 \times (0.5)^5 + 0 \times (0.5)^6 = -2 \ - 0.5 - 0.25 - 0.125 - 2 \times 0.0625 - 2 \times 0.03125 + 0 = -3.0625$

$$v(Road) = \frac{-0.5 + (-2.625) + (-3.0625)}{3} = -2.0625$$

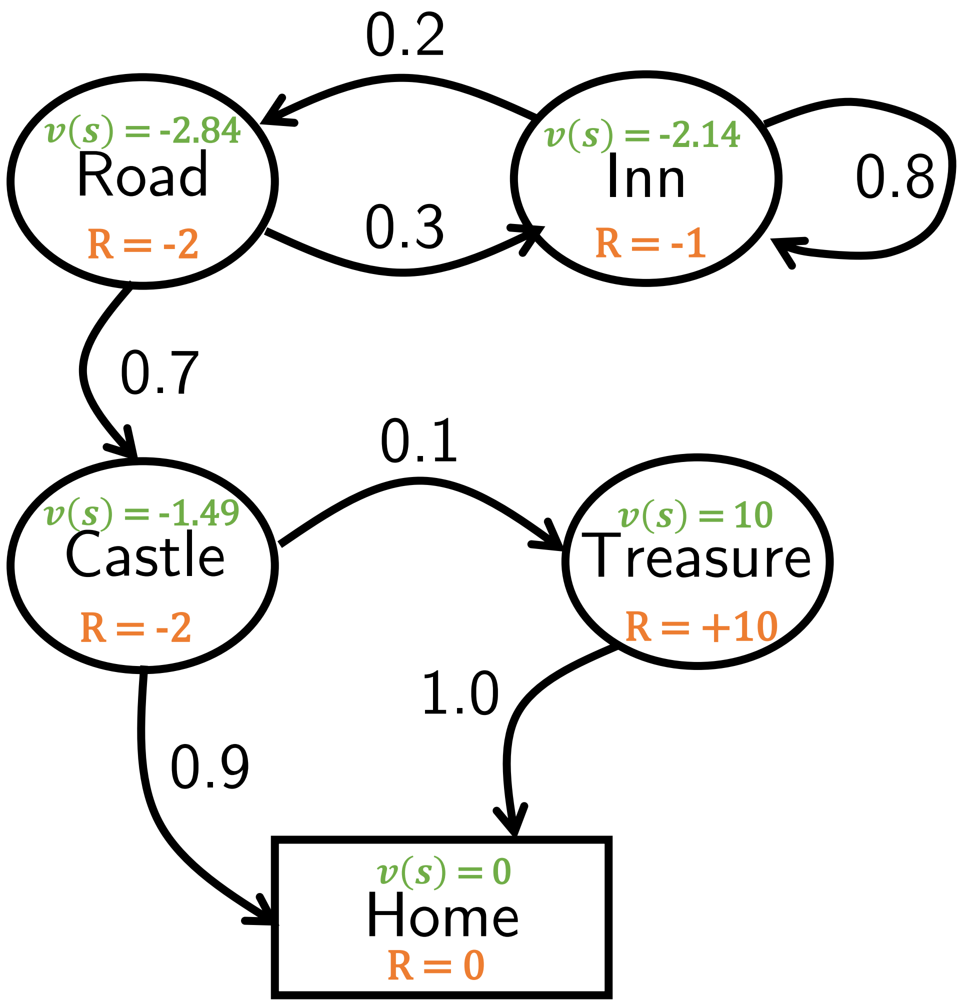
Value Function in the Rogue Example

$v(s)$ with $\gamma = 0.0$, averaged over 10^6 returns:



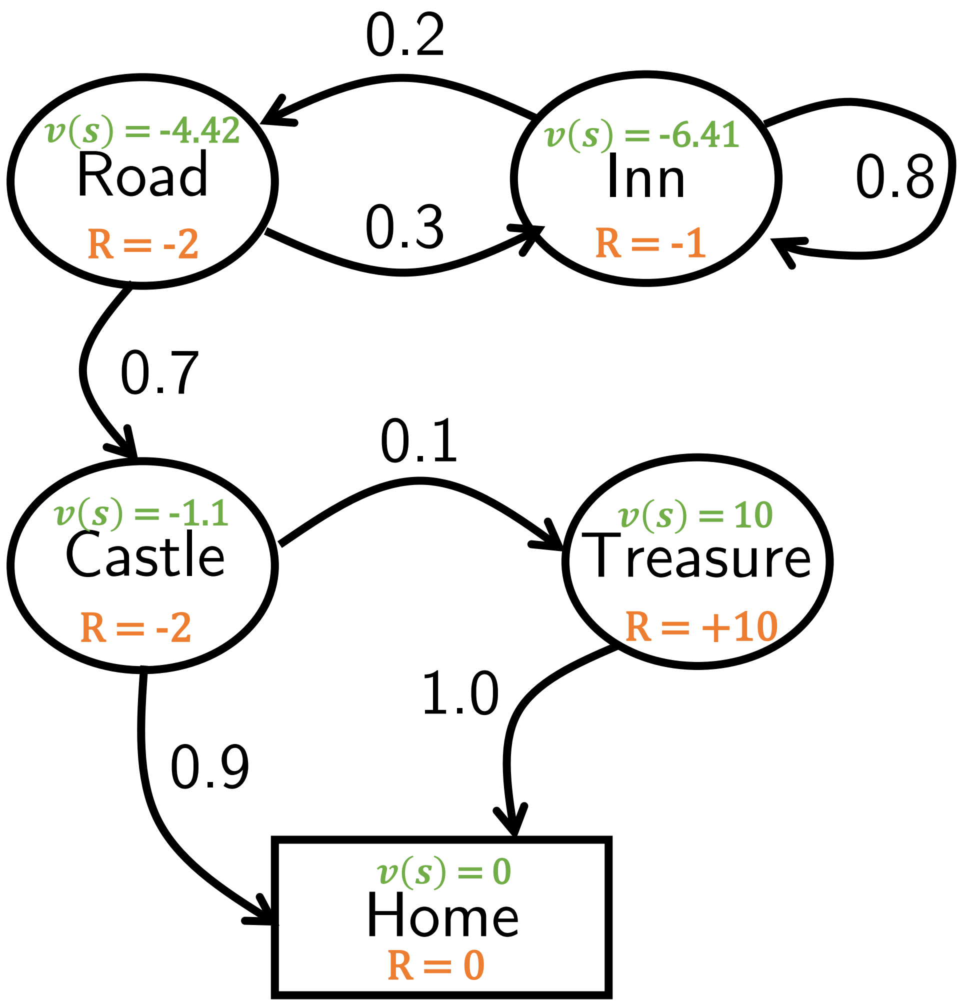
Value Function in the Rogue Example

$v(s)$ with $\gamma = 0.5$, averaged over 10^6 returns:



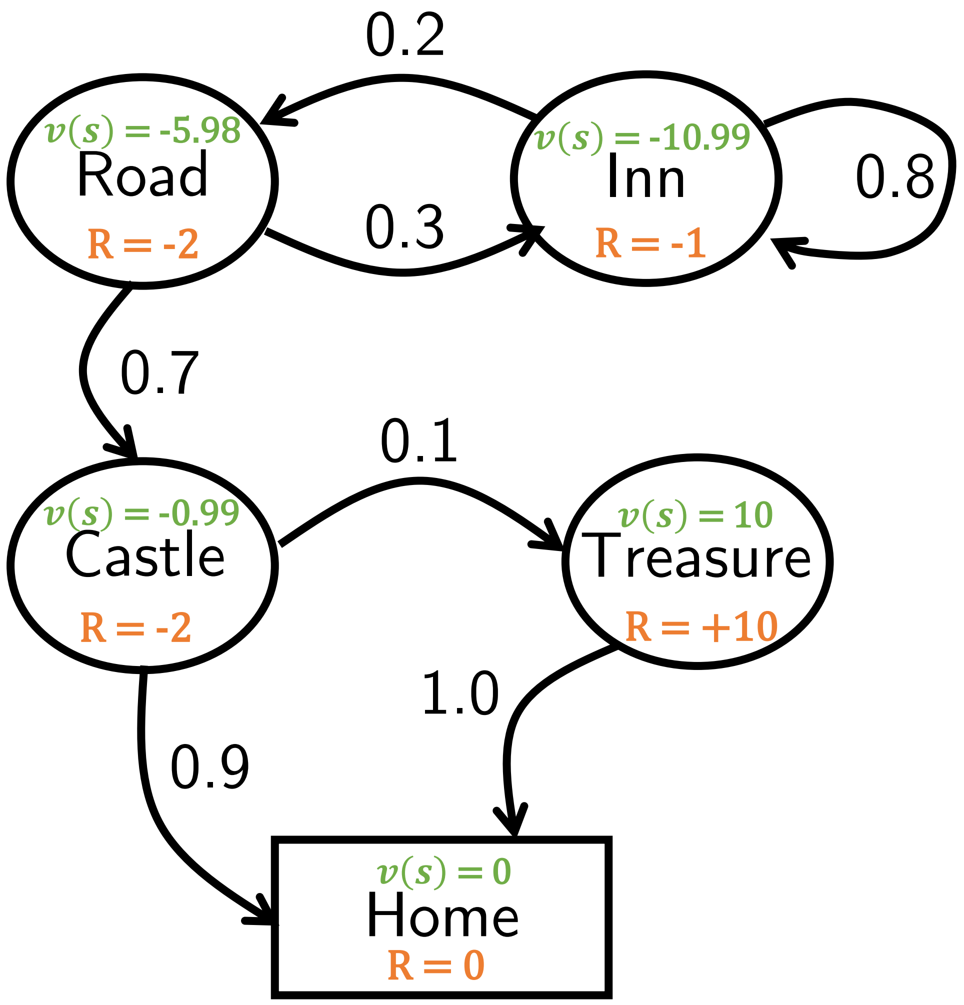
Value Function in the Rogue Example

$v(s)$ with $\gamma = 0.9$, averaged over 10^6 returns:



Value Function in the Rogue Example

$v(s)$ with $\gamma = 1$, averaged over 10^6 returns:



Bellman Equation for MRP

The value function can be decomposed into two parts:

- Immediate reward R_{t+1}
- Discounted value of the successor state $\gamma v(S_{t+1})$

$$\begin{aligned} v(s) &= \mathbb{E}[G_t \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots) \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s] \end{aligned}$$

$$v(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} v(s')$$

Bellman Equation for MRP

$$v(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} v(s')$$

Q? Do we always have to do sampling?

- No, we can solve it analytically with Bellman Equation. Computational complexity is $O(n^3)$ for n states.
- Problem is, the MRP can be large, too large! Iterative methods:
  - Dynamic Programming
  - Temporal Difference Learning
  - Monte Carlo evaluation

Markov Decision Process

A Markov Decision Process is a Markov Reward Process where the agent makes decisions according to a **policy**.

Definition (Markov Decision Process)

A Markov Reward Process is a tuple $< S, A, P, R, \gamma >$:

- S is a finite set of states
- A is a finite set of actions
- P is a state transition probability matrix

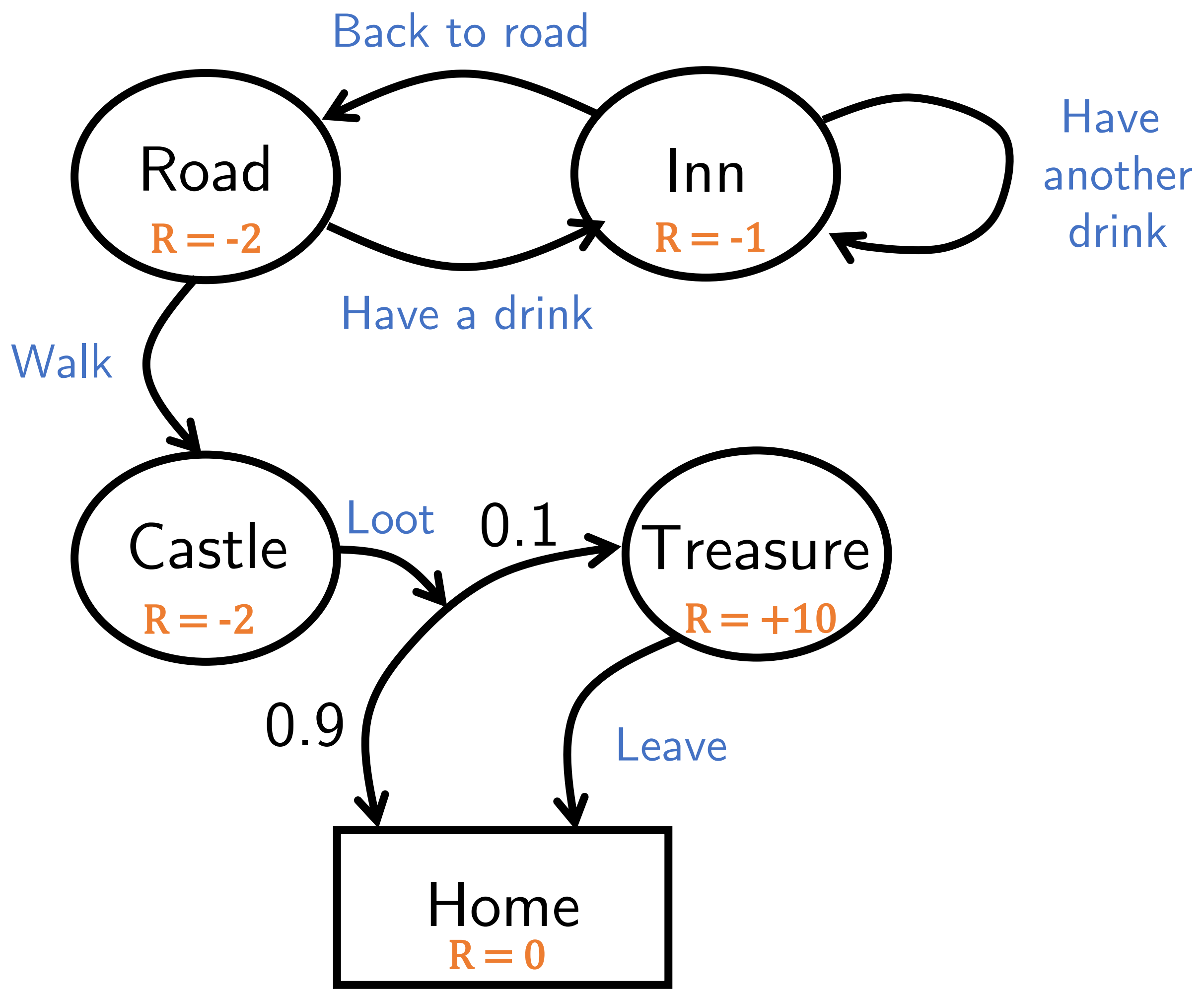
$$P_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

- R is a reward function

$$R_{s}^{a} = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = \mathrm{a}]$$

- γ is a discount factor, $\gamma \in [0,1]$

MDP for the Rogue Example



Note that we have 2 decision makers: agent(via $\pi(a|s)$) and the environment (via $P_{ss'}^{a}$).

Policy and Value Function

Definition (Policy)

A policy is a mapping between states and actions:

$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$$

Definition (State-Value Function)

The state-value function is the expected return starting from the state s and following policy π:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

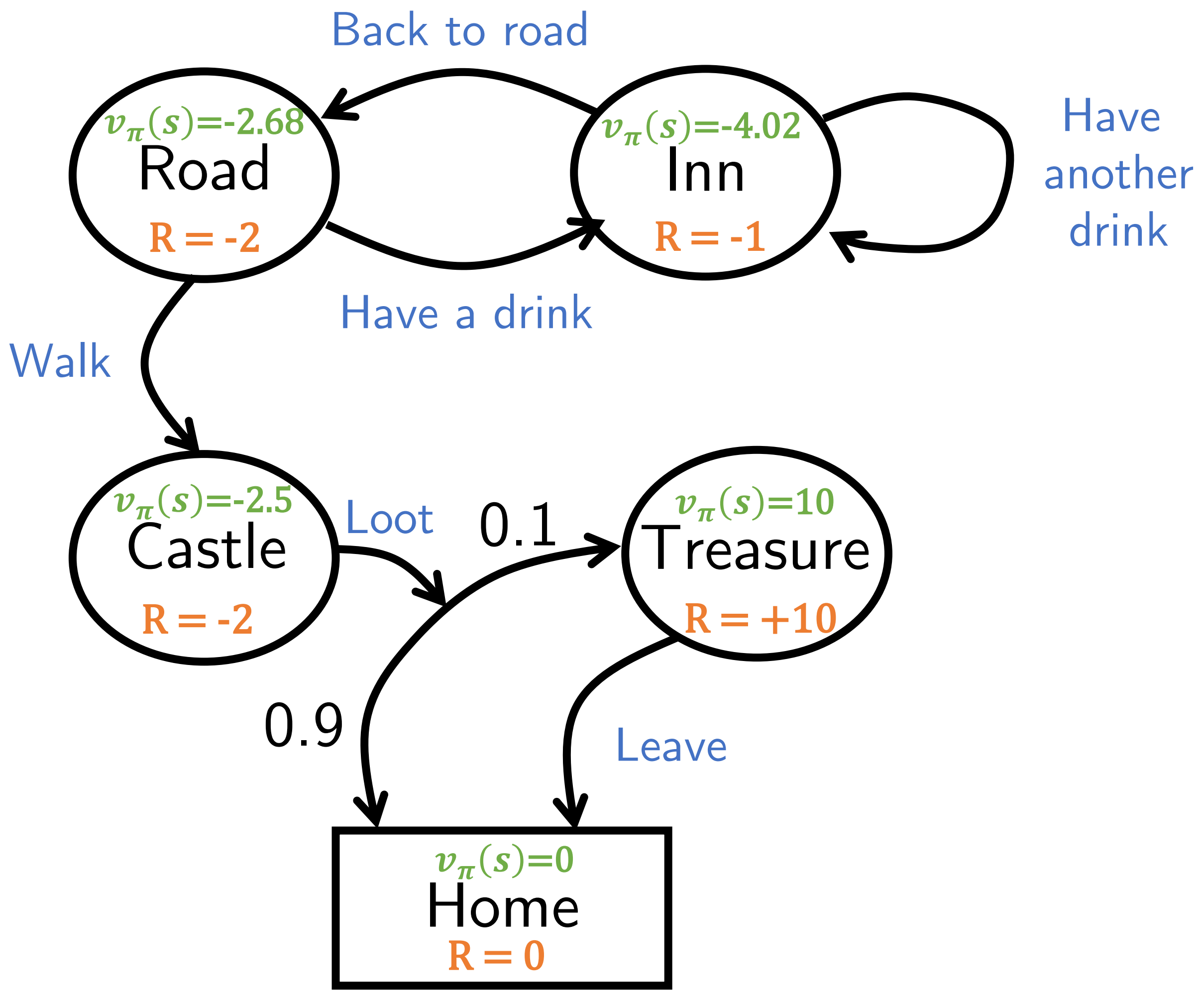
Definition (Action-Value Function)

The action-value function is the expected return starting from the state s, taking action a and then following policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

State-Value Function in the Rogue Example

$v_\pi(s)$ with $\gamma = 0.99$, $\pi(a|s) = 1/\mathrm{N}$ (N = number of available actions), averaged over 10^6 returns:



Bellman Expectation Equation

The state-value function can be decomposed into immediate reward and the discounted reward from the next state:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_{a\in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s'\in S} p(s'|s,a) v_\pi(s') \right) \end{aligned}$$

Similarly, we can do the same with the action-value function:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \\ &= R_s^a + \left(\gamma \sum_{s'\in S} p(s'|s,a) \right) \left(\sum_{a'\in A} \pi(a'|s') q_\pi(s',a') \right) \end{aligned}$$

Q? We know how to estimate the value of a state (and of an action) following a policy. Now what?

Optimal Value Function

Definition (Optimal Value Function)

The **optimal** state-value function $v^*(s)$ is the maximum value over all policies:

$$v^*(s) = \max_{\pi} v_{\pi}(s)$$

The **optimal** action-value function $q^*(s, a)$ is the maximum action-value function over all policies:

$$q^*(s, a) = \max_{\pi} q_{\pi}(s, a)$$

If we find $v^*(s)$, the employed policy π is then optimal. If we know how to act optimally, we have **solved** the problem:

$$v^*(s) = \max_{a} q^*(s, a)$$

Having an optimal policy π^*, we behave optimally if, for every step, we select the action that leads to the state s' with the highest state-value function.

Existence of Optimal Policy

Theorem

For any Markov Decision Process:

- There exists an optimal policy π^* that is at least as good as all other policies:
$$\pi^* \geq \pi \quad \forall \pi \in \Pi$$
- All optimal policies achieve the optimal state-value:
$$v_\pi^*(s) = v^*(s) \quad \forall s \in S$$
- All optimal policies achieve the optimal action-value:
$$q_\pi^*(s,a) = q^*(s,a) \quad \forall s \in S, a \in A$$

Bellman Optimality Equations

Again, there are equivalent Bellman **Optimality** equations:

$$
\begin{aligned}
v^*(s) &= \max_a R_s^a + \mathbb{E}[\gamma v^*(s') \mid S_t = s] \\
&= \max_a R_s^a + \gamma \left(\sum\nolimits_{s'} p(s'|s,a) v^*(s') \right)
\end{aligned}
$$

$$
\begin{aligned}
q^*(s,a) &= R_s^a + \mathbb{E}[\gamma q^*(s',a') \mid S_t = s, A_t = a] \\
&= R_s^a + \gamma \left(\sum\nolimits_{s'} p(s'|s,a) \max_{a'} q^*(s',a') \right)
\end{aligned}
$$

Solving these equations, we solve the MDP and the problem. However, these equations are non-linear. There isn't a closed form solution. How do we solve this?

- Value Iteration
- Policy Iteration
- Q-learning
- Sarsa

Outline

- ✓ Heuristic Functions
- ✓ Introduction to Reinforcement Learning
- ✓ Markov Decision Processes

Acknowledgements

Most of the materials of this lecture are based on:

- Reinforcement Learning: An Introduction, by Andrew Barto and Richard S. Sutton (2017 Edition): `http://incompleteideas.net/book/bookdraft2017nov5.pdf`
- Prof. David Silver's course on Reinforcement Learning: `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html`
- Any other sources cited.