NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

# SC3000/CZ3005 Artificial Intelligence

## Markov Decision Process

Prof Bo AN

*Research area*: artificial intelligence, computational game theory, reinforcement learning, optimization
www.ntu.edu.sg/home/boan
*Email*: boan@ntu.edu.sg
*Office*: N4-02b-55

# Lesson Outline

- Introduction
- Markov Decision Process
- Two methods for solving MDP
  - Value iteration
  - Policy iteration

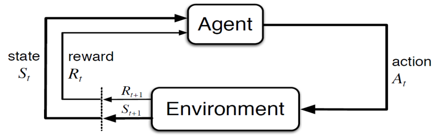


# Introduction

- We consider a framework for decision making under uncertainty
- Markov decision processes (MDPs) and their extensions provide an extremely general way to think about how we can act optimally under uncertainty
- For many medium-sized problems, we can use the techniques from this lecture to compute an optimal decision policy
- For large-scale problems, approximate techniques are often needed (more on these in later lectures), but the paradigm often forms the basis for these approximate methods

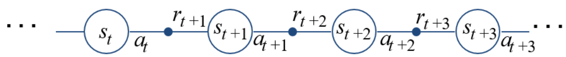


# The Agent-Environment Interface




Agent and environment interact at discrete time steps: $t = 0,1,2, \ldots$
Agent:

1. observes state at step $t$: $s_t \in S$
2. Produces action at step $t$: $a_t \in A(s_t)$
3. Gets resulting reward: $r_{t+1}$ and the next state: $s_{t+1} \in S$






We know the

MDP vs RL

↑ model (knows tran function)

↓ we don't know the model

# Making Complex Decisions

- Make a sequence of decisions
  - Agent's utility depends on a sequence of decisions
  - Sequential Decision Making
- Markov Property
  - Transition properties depend only on the current state, not on previous history (how that state was reached)
  - Markov Decision Processes

deterministic

· Go. w some assumptions -

stochastic



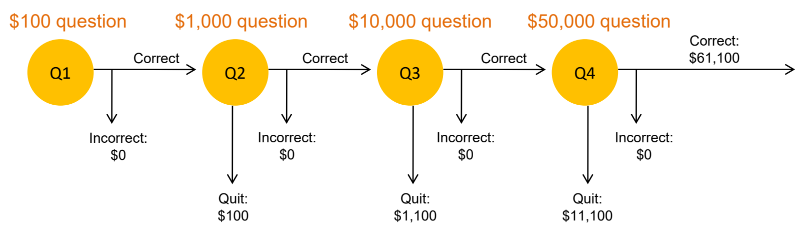
# Markov Decision Processes

- Formulate the agent-environment interaction as an MDP
- Components:
  - *Markov* **States** $s$, beginning with initial state $s_0$
  - **Actions** $a$
    - Each state $s$ has actions $A(s)$ available from it
  - **Transition model** $P(s' \mid s, a)$ take action a at state s ⇒ prob that reach state s'
    - *assumption*: the probability of going to $s'$ from $s$ depends only on $s$ and $a$ and not on any other past actions or states
  - **Reward function** $R(s)$, or r(s)
- **Policy** $\pi(s)$: the action that an agent takes in any given state
  - The "solution" to an MDP ↓ Goal: Find policy ⇒ How to make the decision.



# Game Show

- A series of questions with increasing level of difficulty and increasing payoff
- Decision: at each step, take your earnings and quit, or go for the next question
  - If you answer wrong, you lose everything




Reason from backwards: Backward induction

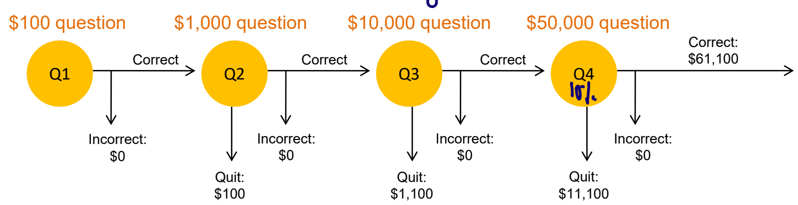
similar to min-max search.



# Game Show

- Consider $50,000 question
  - Probability of guessing correctly: 1/10
  - Quit or go for the question?
- What is the expected payoff for continuing?

  0.1 * 61,100 + 0.9 * 0 = 6,110 6110 << 11100 ∴ After 4 you should not try.
- What is the optimal decision?

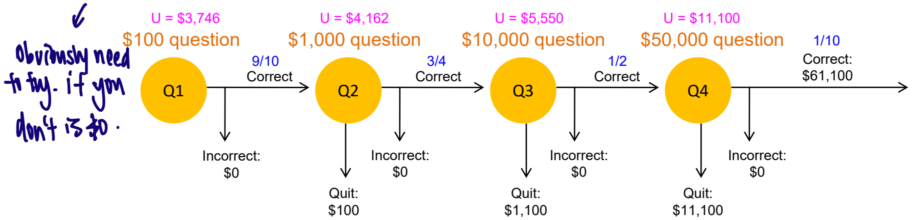


# Game Show

- What should we do in Q3?
  - Payoff for quitting: $1,100
  - Payoff for continuing: 0.5 * $11,100 = $5,550
- What about Q2?
  - $100 for quitting vs. $4,162 for continuing
- What about Q1?

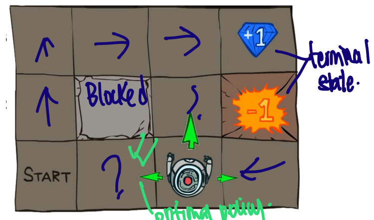
5550 > 1100
∴ you should try.
after Q3

75% x 5550



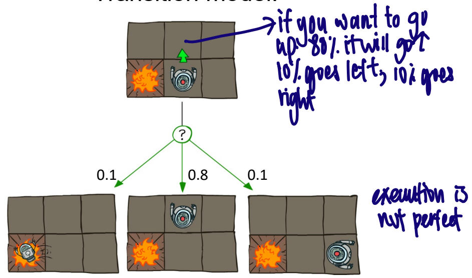



# Grid World




R(s) = -0.04 for every non-terminal state

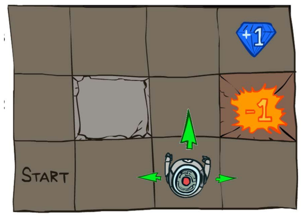
Transition model:

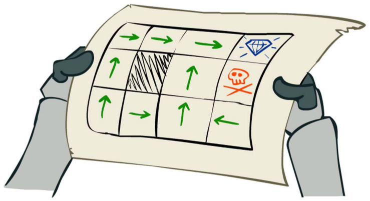





Source: P. Abbeel and D. Klein

# Goal: Policy

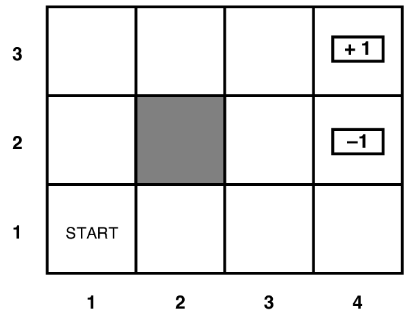


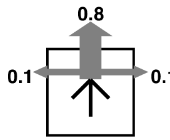


Source: P. Abbeel and D. Klein

# Grid World

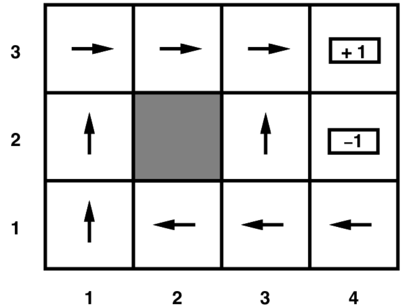



Transition model:




R(s) = -0.04 for every non-terminal state

# Grid World




Optimal policy when R(s) = -0.04 for every non-terminal state



# Atari Video Games


Execute actions accumulated rewards

to get the maximum



# Solving MDPs

- MDP components:
  - **States** $s$
  - **Actions** $a$
  - **Transition model** $P(s' \mid s, a)$ ⇒ after taking some action ⇒ prob of reaching another state
  - **Reward function** $R(s)$
- The solution:
  - **Policy** $\pi(s)$ mapping from states to actions
  - How to find the optimal policy?

max total ave reward



# Maximizing Accumulated Rewards

→ continue until terminal state.

- The optimal policy should maximise the accumulated *rewards* *over* given a trajectories like $\tau = < S_1, A_1, R_1, \ldots, S_T, A_T, R_T >$ under some policies:

if we treat γ=1, we are just summing up total reward.

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

↪ what we care about is the total reward.

- How to define the accumulated rewards of a state sequence?
  - Discounted sum of rewards of individual states
  - Problem: infinite state sequences
  - If finite, LP can be applied

-γ ~ 1 usually i.e. 0.99. 0.98

Value Func

## Accumulated Rewards

- Normally, we would define the *accumulated rewards* of trajectories as the discounted sum of the rewards
- **Problem:** infinite time horizon
- **Solution:** *discount* the individual rewards by a factor $\gamma$ between 0 and 1:

Why we use discounted value
- money total is worth more compared to tmr.

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots$$
$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k} \leq \frac{R_{max}}{1-\gamma} \quad (0 < \gamma < 1)$$

- Sooner rewards count more than later rewards
- Makes sure the total *accumulated rewards* stays bounded
- Helps algorithms converge

If there's no discount goes to ∞ computer cannot handle.



## Value Function

$V^{\pi}(s)$ = expected utility starting in s & acting according to π

$V^{\pi *}(s)$ = expected utility starting in s & acting optimally

- The "true" value of a state, denoted $V(s)$, is the expected sum of discounted rewards if the agent executes an *optimal* policy starting in state $s$

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

- Similarly, we define the action-value of a state-action pair as

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

- The relationship between Q and V

$$V^{\pi}(s) = \sum_{a \in A} Q^{\pi}(s,a) \underbrace{\pi(a|s)}_{\text{prob we will take the action a in state s}}$$

→ policy



Action value function

$Q^{\pi}(s,a)$ - expected utility taking a in s & the following π

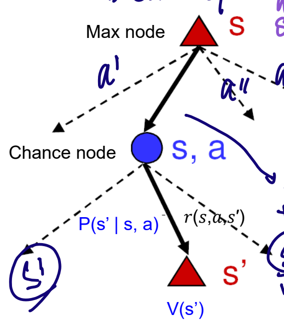
$Q(s,a_1)=1$ $Q(s,a_2)=10$ $Q(s,a_3)=8$

$\pi'(s)$ ⇒ Returns max $Q(s,a)$ ⇒ Returns $a_2$

$V(s)$ ⇒ Returns highest value ⇒ 10.

## Finding the Value Function of States

☆ Bellman eqn.




- What is the expected value of taking action $a$ in state $s$?

If you take action a & state s, this is what you get

prob that you will reach diff s'

$$Q(s,a) = \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma * V(s')]$$

prob that you reach diff s'

→ after you reach s', you get immediate reward → plus future.

- How do we choose the optimal action?

$$\pi^*(s) = \text{argmax}_{a \in A} \underbrace{\sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V(s')]}_{Q(s,a)}$$

- What is the recursive expression for $V(s)$ in terms of the utilities of its successor states?

$$V(s) = \max_{a \in A} \underbrace{\sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V(s')]}_{Q(s,a)}$$

optimal value at this state s

you choose the best action that max Q function

you can take many action ∴ you take action that max

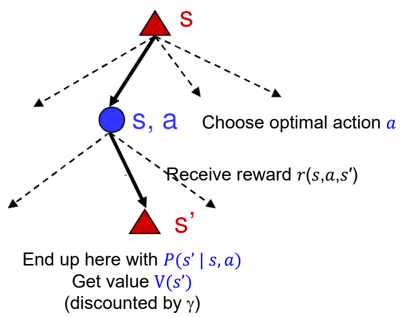


## The Bellman Equation

- Recursive relationship between the accumulated rewards of successive states:

$$V(s) = \max_{a \in A} \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V(s')]$$

- For $N$ states, we get $N$ equations in $N$ unknowns
  - Solving them solves the MDP
  - We could try to solve them through expectimax search, but that would run into trouble with infinite sequences
  - Instead, we solve them algebraically
  - Two methods: **value iteration** and **policy iteration**



# Method 1: Value Iteration

- Start out with every $V_0(s) = 0$
- Iterate until convergence
  - During the $i$th iteration, update the value of each state according to this rule:

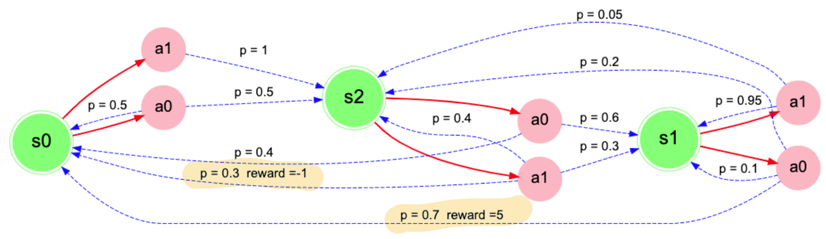
$$V_{i+1}(s) = \max_a \sum_{s' \in S} P(s'|s,a)\,[R(s,a,s') + \gamma V_i(s')]$$

- In the limit of infinitely many iterations, guaranteed to find the correct values
  - In practice, don't need an infinite number of iterations…



# Value Iteration: Example

- A simple example to show how VI works
  - State, action, reward (non-zero) and transition probability are shown in the figure
  - We use this example as an MDP and solve it using VI and PI




These structure can be easily implemented by `dict` of Python or `HashMap` of Java



# Value Iteration: Example (cont'd)

- Given the states and actions are finite, we can use matrices to represent the value function $V(s)$
- Pseudo-code of VI

1. Initialize $V_0(s)$, for all $s$
2. For $i = 0, 1, 2 \ldots$
3. $V_{i+1}(S) = \max_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_i(s')]$ for all state $s$



# Value Iteration: Example (cont'd)

- How VI works in an iteration? Given iteration $i = 0$

For all state find

$$V_{i+1}(S) = \max_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_i(s')]$$

We can instead calculate Q(s, a) values for each s and $a$ and get the best V(s)

$$Q_{i+1}(s,a) = \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_i(s')]$$

Finally

$$V_{i+1}(S) = \max_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_i(s')] = \max_a Q_{i+1}(s,a)$$

## Value Iteration: Example (cont'd)

- We use the previous example and solve it using VI
- Construct a Q table, Q(s, a)
  - 3 rows (3 states) and 2 columns (2 actions)
  - $V_0 = [0, 0, 0]$ # 3 states

$$Q_{i+1}(s,a) = \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_i(s')]$$

$Q_0$

| | $a_0$ | $a_1$ |
|---|---|---|
| $s_0$ | 0 | 0 |
| $s_1$ | 0 | 0 |
| $s_2$ | 0 | 0 |

$V_0$

| |
|---|
| 0 |
| 0 |
| 0 |

The initial Q and V table

$Q_1$

| | $a_0$ | $a_1$ |
|---|---|---|
| $s_0$ | 0 | 0 |
| $s_1$ | 3.5 | 0 |
| $s_2$ | 0 | -0.3 |

$V_1$

| |
|---|
| 0 |
| 3.5 |
| 0 |

$$Q_1(s_1, a_0) = P(s_0|s_1,a_0)[r(s_1,a_0,s_0) + \gamma V_0(s_0)] + P(s_1|s_1,a_0)\,[r(s_1,a_0,s_1) + \gamma V_0(s_1)] + P(s_2|s_1,a_0)\,[r(s_1,a_0,s_2) + \gamma V_0(s_2)]$$

$$Q_1(s_1, a_0) = 0.7 * [5 + 0] + 0.1 * [0 + 0] + 0.2 * [0 + 0] = 3.5$$

Repeat this process until it converges



## Value Iteration: Example (cont'd)

- Iterating the process (code available on later slides)
  - Discount factor 0.9

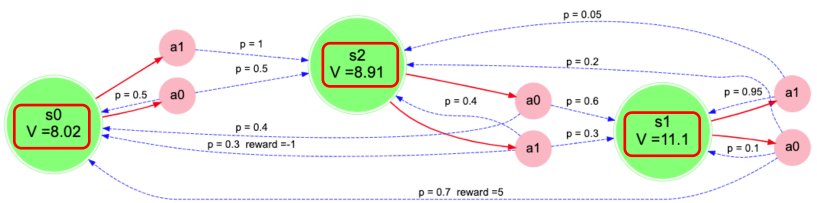
```
iter   0  |  V(s0) = 0.000   V(s1) = 0.000   V(s2) = 0.000

iter   1  |  V(s0) = 0.000   V(s1) = 3.500   V(s2) = 0.000

iter   2  |  V(s0) = 0.000   V(s1) = 3.815   V(s2) = 1.890

iter   3  |  V(s0) = 1.701   V(s1) = 4.184   V(s2) = 2.060

 . . .  . . .

iter  63  |  V(s0) = 8.020   V(s1) = 11.160   V(s2) = 8.912

iter  64  |  V(s0) = 8.021   V(s1) = 11.161   V(s2) = 8.913

iter  65  |  V(s0) = 8.022   V(s1) = 11.162   V(s2) = 8.915
```

Very good, the values converge now



## Value Iteration: Example (cont'd)

- Put the optimal values on the graph






## Value Iteration: Example (cont'd)

during this process, we don't calc the process
- just apply Bellman eqn
- until conv & the use argmax to find the policy

- Use $V^*$ to find optimal policy
  - aka optimal actions in each state

$$\pi^*(s) = \underset{a}{\text{argmax}} \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_i(s')] = \underset{a}{\text{argmax}}\, Q_i(s,a)$$

$\pi^*$

| | $a^*$ |
|---|---|
| $s_0$ | $a_1$ |
| $s_1$ | $a_0$ |
| $s_2$ | $a_0$ |

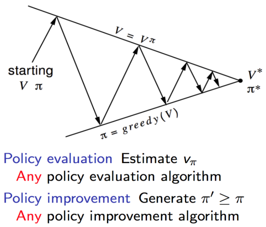
Done! We get the optimal policy of the example

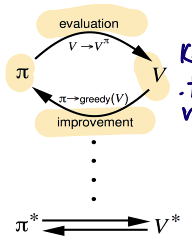
Python Code on Colab: https://colab.research.google.com/drive/1DnYIr3QJxpfs_rR_jAAUrqHZMjvsjGSx?usp=sharing

# Method 2: Policy Iteration

- Start with some initial policy $\pi_0$ and alternate between the following steps:
  - **Policy evaluation:** calculate $V^{\pi_i}(s)$ for every state *s, like VI*
  - **Policy improvement:** calculate a new policy $\pi_{i+1}$ based on the updated utilities

$$\pi_{i+1}(s) = \operatorname{argmax}_{a\in A} \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V^{\pi_i}(s')]$$




Keep repeating this until value converges.



# Policy Iteration: Main Steps

- Unlike VI, policy iteration has to maintain a policy - chosen actions from all states - and estimate $V^{\pi_i}$ based on this policy.
  - Iterate this process until convergence (like VI)
- Steps of PI
  1. Initialization
  2. Policy Evaluation (calculating the $V$)
  3. Policy Improvement (calculate the policy $\pi$)



# Policy Iteration: Detailed Steps

1. Initialization
   a. Initialize $V(s)$ and $\pi(s)$ for all state $s$

2. Policy Evaluation (calculate the $V$)
   a. Repeat
   b. $\Delta \leftarrow 0$
   c. For each state $s$:
   d. $v \leftarrow V(s)$
   e. $V(s) \leftarrow \sum_{s'} p(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V(s')]$
   f. $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   g. until $\Delta < \omega$ (a small positive number)

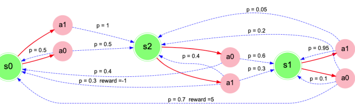
3. Policy Improvement (calculate the new policy $\pi$)
   a. $policystable \leftarrow True$
   b. For each state $s$:
   c. $b \leftarrow \pi(s)$
   d. $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s'} p(s'|s,a)[R(s,a,s') + \gamma V(s')]$
   e. If $b \neq \pi(s)$, then $policystable \leftarrow False$
   f. If $policystable \leftarrow True$, then stop; else go to step 2;



# Policy Iteration: Policy Evaluation

- Use the example used in VI
- Start iteration $i = 0$, initialize random $\pi$ , $\gamma = 0.99$, and $V(s) = 0$ for all state $s$

$$V^{\pi_i}(s) = \sum_{s'} P(s,\pi_i(s),s')[R(s,\pi_i(s),s') + \gamma V^{\pi_i}(s')]$$



$$\begin{aligned}
V(s_0) \leftarrow\ & P(s_0,\pi(s_0),s_0)[R(s_0,\pi(s_0),s_0) + \gamma V(s_0)] + \\
& P(s_0,\pi(s_0),s_1)[R(s_0,\pi(s_0),s_1) + \gamma V(s_1)] + \\
& P(s_0,\pi(s_0),s_2)[R(s_0,\pi(s_0),s_2) + \gamma V(s_2)]
\end{aligned}$$

$$\begin{aligned}
V(s_1) \leftarrow\ & P(s_1,\pi(s_1),s_0)[R(s_1,\pi(s_1),s_0) + \gamma V(s_0)] + \\
& P(s_1,\pi(s_1),s_1)[R(s_1,\pi(s_1),s_1) + \gamma V(s_1)] + \\
& P(s_1,\pi(s_1),s_2)[R(s_1,\pi(s_1),s_2) + \gamma V(s_2)]
\end{aligned}$$

$$\begin{aligned}
V(s_2) \leftarrow\ & P(s_2,\pi(s_2),s_0)[R(s_2,\pi(s_2),s_0) + \gamma V(s_0)] + \\
& P(s_2,\pi(s_2),s_1)[R(s_2,\pi(s_2),s_1) + \gamma V(s_1)] + \\
& P(s_2,\pi(s_2),s_2)[R(s_2,\pi(s_2),s_2) + \gamma V(s_2)]
\end{aligned}$$

## Policy Iteration: Policy Evaluation (cont'd)

- Use the example used in VI
- Start iteration $i = 0$, $\gamma = 0.99$, initialize random $\pi = [a_1, a_0, a_1]$ and $V(s) = 0$ for states $s_0$, $s_1$ and $s_2$

Reward = 0

transition to $s_0, s_1$ is 0 & $s_2 = 1$

$V(s_0) \leftarrow 0.0[0.0 + \gamma V(s_0)] + 0.0[0.0 + \gamma V(s_1)] + 1.0[0.0 + \gamma V(s_2)]$

10% goes $s_0$ with reward = 5

$V(s_1) \leftarrow 0.7[5.0 + \gamma V(s_0)] + 0.1[0.0 + \gamma V(s_1)] + 0.2[0.0 + \gamma V(s_2)]$

$V(s_2) \leftarrow 0.3[-1 + \gamma V(s_0)] + 0.3[0.0 + \gamma V(s_1)] + 0.4[0.0 + \gamma V(s_2)]$

- Values are calculated asynchronously
- The converged values are used for policy improvement

Reward of reaching state

prob of going to state by taking $a_t$

10% stays in $s_1$

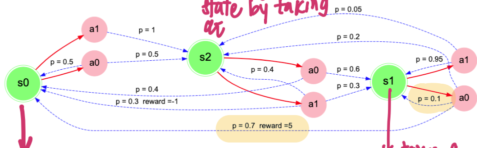
According to initial policy.

$V(s_0) = 0$

$V(s_1) = 0.7 * 5.0 = 3.5$

$V(s_2) = 0.3 * (-1) + 0.3 * (0.99 * 3.5) = 0.7395$

... loop this process as instructed in step 2 ...




takes $a_1$ goes to $s_2$ only

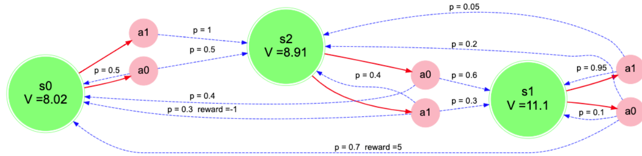
takes $a_0$



## Policy Iteration: Example (cont'd)

- The optimal Q values are
  - Nearly the same as that of VI (as shown in the figure below)

| | $V^*$ |
|---|---|
| $s_0$ | 8.03 |
| $s_1$ | 11.2 |
| $s_2$ | 8.9 |




  - We can easily calculate the optimal policy. Can you try it?

$\pi^*$

| | $a^*$ |
|---|---|
| $s_0$ | $a_1$ |
| $s_1$ | $a_0$ |
| $s_2$ | $a_0$ |

argmax



## Further Reading [AAAI'18: http://www.ntu.edu.sg/home/boan/papers/AAAI18_Malmo.pdf]

We Won 2017 Microsoft Collaborative AI Challenge

- Collaborative AI
  - *How can AI agents learn to recognise someone's intent (that is, what they are trying to achieve)?*
  - *How can AI agents learn what behaviours are helpful when working toward a common goal?*
  - *How can they coordinate or communicate with another agent to agree on a shared strategy for problem-solving?*



## Further Reading [AAAI'18: http://www.ntu.edu.sg/home/boan/papers/AAAI18_Malmo.pdf]

- Microsoft Malmo Collaborative AI Challenge
  - *Collaborative mini-game, based on an extension "stag hunt"*
  - *Uncertainty of pig movement*
  - *Unknown type of the other agent*
  - *Detection noise (frequency 25%)*
- Our team HogRider won the challenge (out of more than 80 teams from 26 countries)
  - *learning + game theoretic reasoning + sequential decision making + optimisation*