# Unit-4

# Reinforcement Learning

# Reinforcement Learning

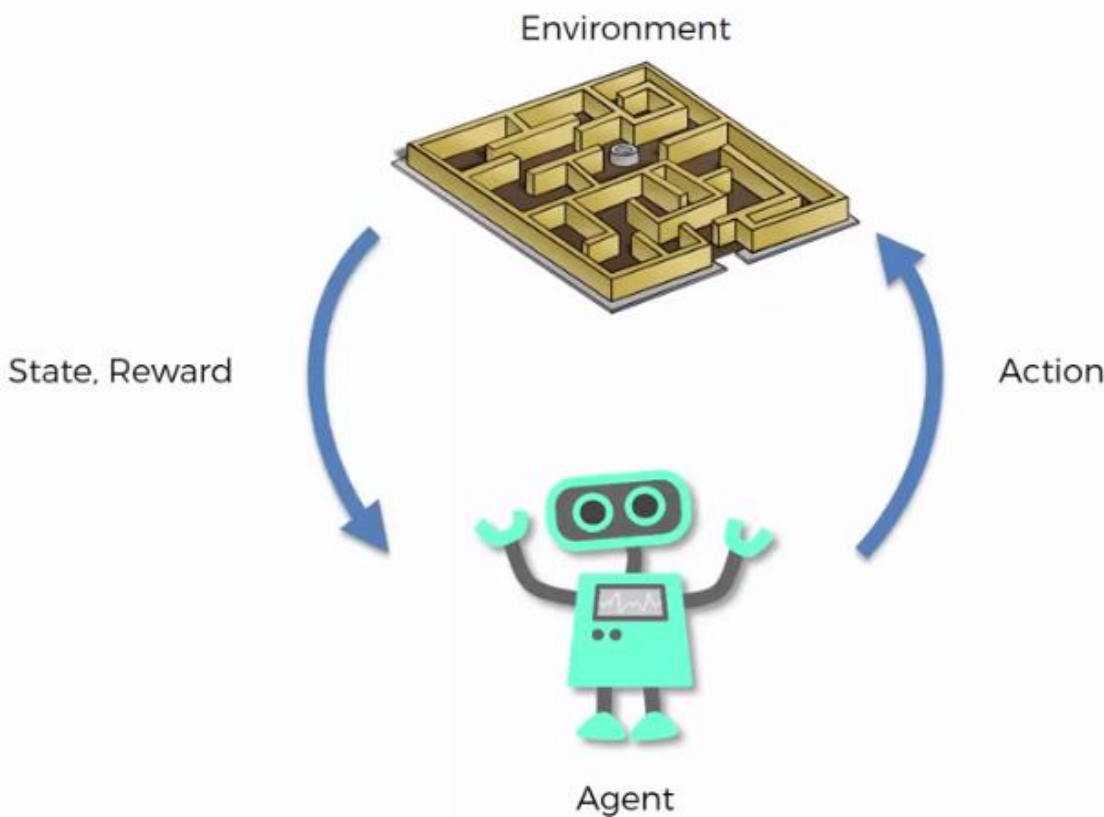
- Science of decision making.
- Learning optimal behavior in environment to maximize the reward.
- Uses trial and error method to learn.
- Uses the algorithm that learns from outcome and decide the action to be taken.
- Algorithm receives feedback after each action.
- Helps in small decision w/o human guidance.

# Reinforcement Learning

- It has 'Environment'- problem set to be solved.
- 'Agent' is an AI algorithm.
- 'Agent' performs some 'action' → leads to its 'State' change.
- Agent will get 'reward' for correct destination and punished for incorrect.
- Agent learns the environment by learning this iterative action-reward process.
- Agent comes to understand which state and action maximizes the reward.

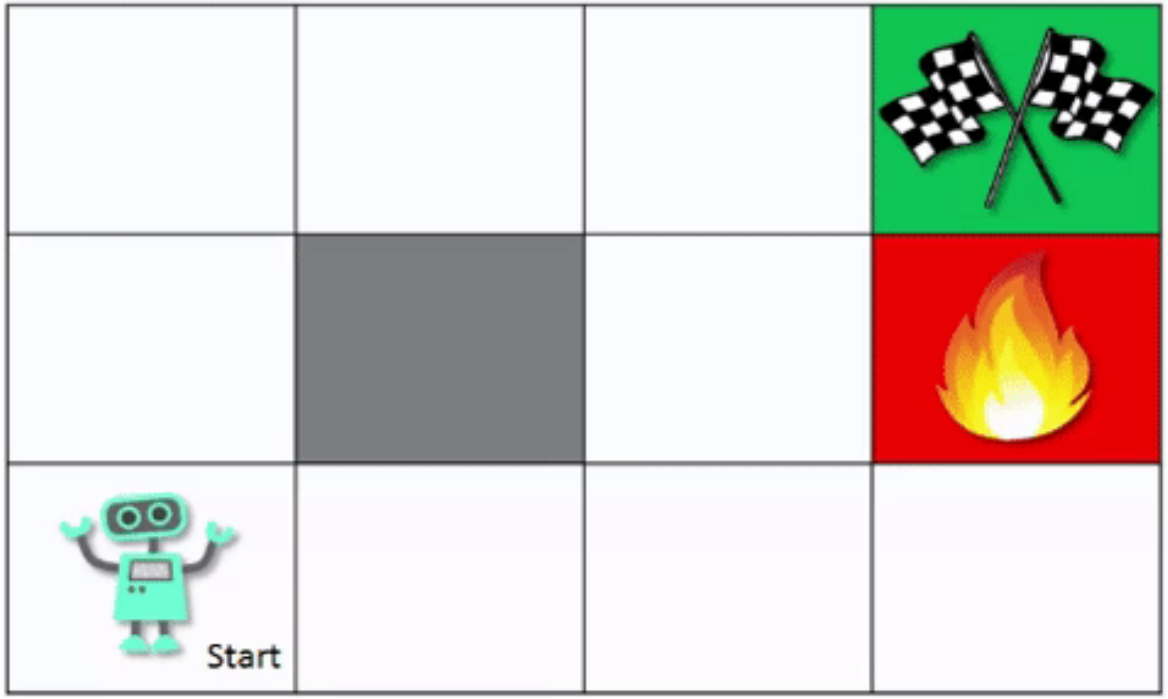
# Reinforcement Learning

- Example:
- Find the best possible path to reach the reward.



# Reinforcement Learning

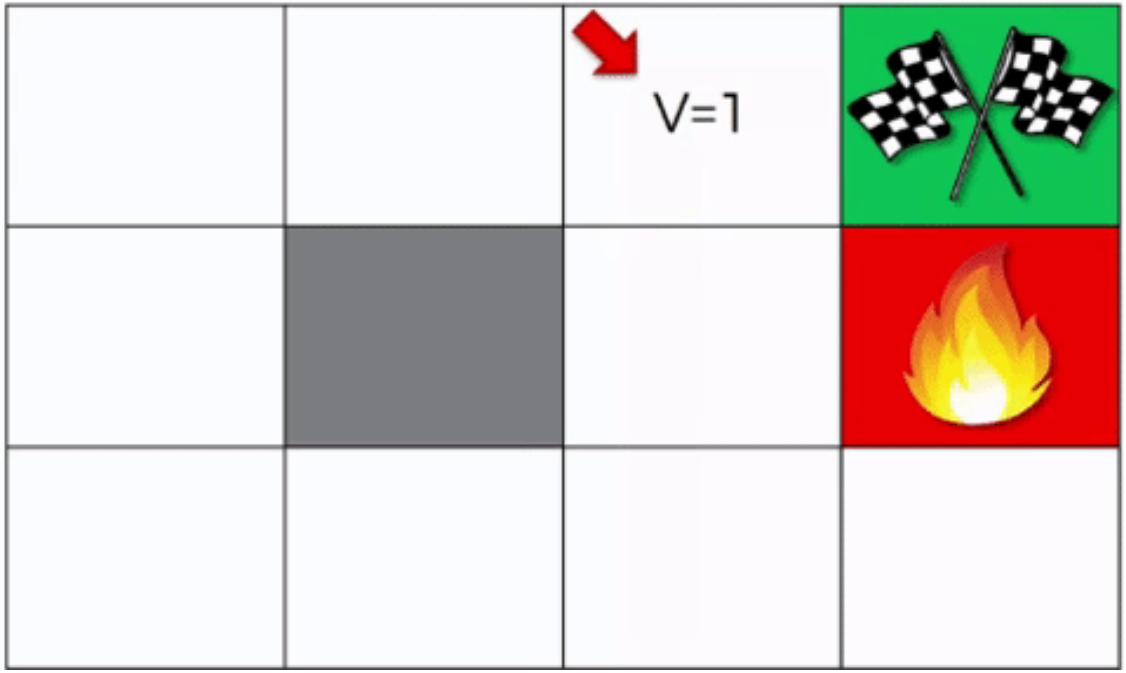
- Example:
- Find the best possible path to reach the reward.




Courtesy: https://towardsdatascience.com/getting-started-with-reinforcement-q-learning-77499b1766b6

# Reinforcement Learning

- Example:
- Find the best possible path to reach the reward.




Courtesy: https://towardsdatascience.com/getting-started-with-reinforcement-q-learning-77499b1766b6

# Reinforcement Learning

- DeepMind's Alphazero learned to play chess by playing against itself,
- Learning from its mistakes, and continuously improving its strategies.

# Terminologies

- Agent: The model that is being trained by the Reinforcement Learning.
- Environment: The training situation that the model must optimize to.
- Action: All possible steps that can be taken by agent.
- State: The current position or condition returned by the model.

# Terminologies: Policy

- It's a rule used by an agent to determine what action to take .
- It defines the learning agent's way of behaving at given time.
- Policy is a mapping from states of environment to action to be taken when in those states.
- $\pi: S \rightarrow a$

# Terminologies: Reward

- It defines the goa min RL problem.
- For each step taken by agent, environment sent it a reward.
- The agent's objective is to maximize the total reward it receives over the long run.

# Terminologies: Value Function

- The value of the state is the total amount of reward an agent can expect to accumulate over the future starting from that state.
- While rewards determine the immediate desirability of state, values indicates the long term desirability of state.

# Terminologies: Model

- Something that mimics the behavior of environment.
- It allows inferences to be made about how the environment will behave.
- Given a state and action, the model might predict the resultant next state and next reward.

# Markov Process

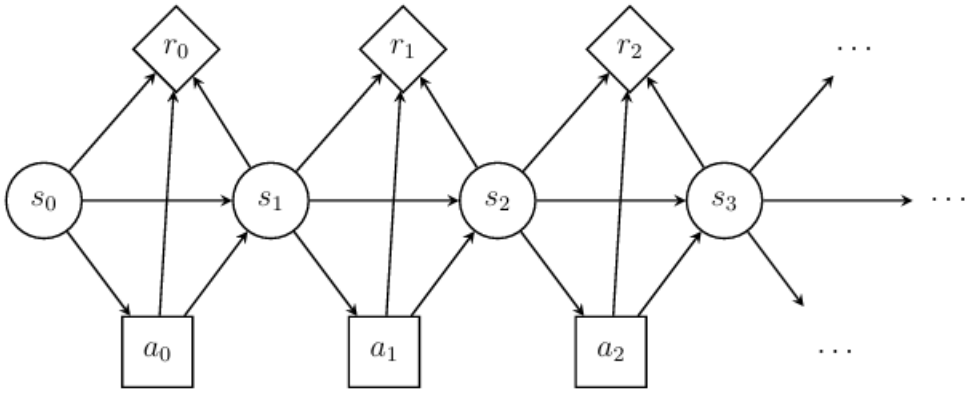
- Process: Sequence of states for an environment and actions taken by agent.
- Discrete Time Process: if process is considered at discrete time step
- Stochastic Process: if the state of environment or action of agent in process are determined randomly.

# Markov Decision Processes

- Markov Decision Policy is a Reinforcement Learning Policy used to map a current state to an action where the agent continuously interacts with the environment to produce new solutions and receive rewards.

# Markov Decision Processes

- we start in the state s0. We can take some action a0 and transition to the state s1. Because of this state transition, we will receive a reward r0.



# Markov Decision Processes

- To define Markov Decision Process, We need following:
  1. Set of States
  2. Set of Actions
  3. Transition Probability p(S'|S, a) (Due to stationary characteristics probability remains constant.)
  4. Reward R(S', s, a)- depends on starting state, action and new state.

# Markov Decision Processes

Rewards: R(S)-the reward of entering state ‘S’

- Three types of rewards:

1. Total Reward: R(S0)+R(S1)+R(S2)+…
   - If sum is infinite we can’t compare two policies and get the best result.
2. Average Reward: $\lim_{n\to\infty}\left(\frac{1}{n}(\text{R(S0)+R(S1)+R(S2)+…})\right)$

   If the total reward is infinite, the average reward become 0.
3. Discounted Reward: R(S0)+ ϒR(S1)+ $\text{ϒ}^2$R(S2)+…

   Where 0<=ϒ<=1 is the discount factor.
   - We prefer getting the reward sooner rather than later.
   - Everyday there is a chance that tomorrow won’t come.
   - The total discounted reward is finite.

# Variations of MDP

- A fully observable MDP
  - Agent knows in which state it is currently in.
- A partially observable MDP(POMDP)
  - Combined MDP and hidden markov model
  - Agent doesn’t know in which state it is currently in, but it can get some noisy signal of state.

# A Grid World

- Example:
  - A robot is in a 3 4 grid world. What should the robot do to maximize its rewards?

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | Start | | | |
| 2 | | X | | -1 |
| 3 | | | | +1 |

  - Let sij be the position in row I and column j.
  - S11 is the initial state.
  - There is a wall at S22.
  - S24 and S34 are goal states. So goals are observable(Use MDP).
  - The robot escapes the world at either goal state.
  - The value in each goal state cell is its corresponding reward.

# A Grid World

- Example:
  - There are four actions: up, down, left, right. Every action is possible in every state.
  - The transition model P(S' | s,a)
    - The action achieves its intended effect with probability 0.8.
    - Action leads to 90-degree left turn with probability 0.1.
    - Action leads to 90-degree right turn with probability 0.1.
    - If robot bumps into a wall, it remains in same state.
  - The reward Function R(S) is the reward of entering state S.
    - R(S24)=-1
    - R(S34)=+1
    - Otherwise R(S)=-0.04 (the cost for exploring the world)

# A Grid World

- Understanding transition model:
  - The robot is currently in S14 and tries to move to our right. What is the probability that the robot stays in S14?

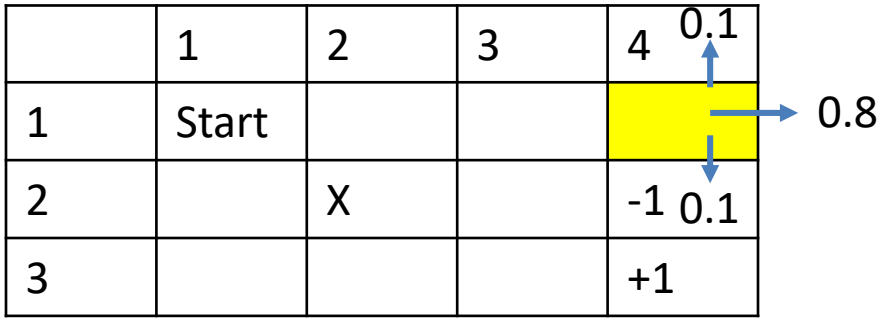
    A. 0.1

    B. 0.2

    C. 0.8

    D. 0.9

    E. 1.0




  - The intended direction is to our right.
  - With probability 0.8, the robot will go to our right, but it will bump into the right wall and stay in the same state.
  - With probability 0.1, the robot will go up, but it will bump into the top wall and stay in the same state.
  - With probability 0.1, the robot will go down and move to s24.
  - Therefore, the robot will stay in s14 with probability 08+01=09.
  - The correct answer is(D)

# A Grid World

- Fixed sequence of action:
  - MDP is sufficient to tell the agent fixed sequence of action.
  - If the environment is deterministic, an optimal solution to the grid world problem is the fixed action sequence: down, down, right, right, and right.
    A. True
    B. False
    C. Don't Know

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | Start | | | |
| 2 | | X | | -1 |
| 3 | | | | +1 |

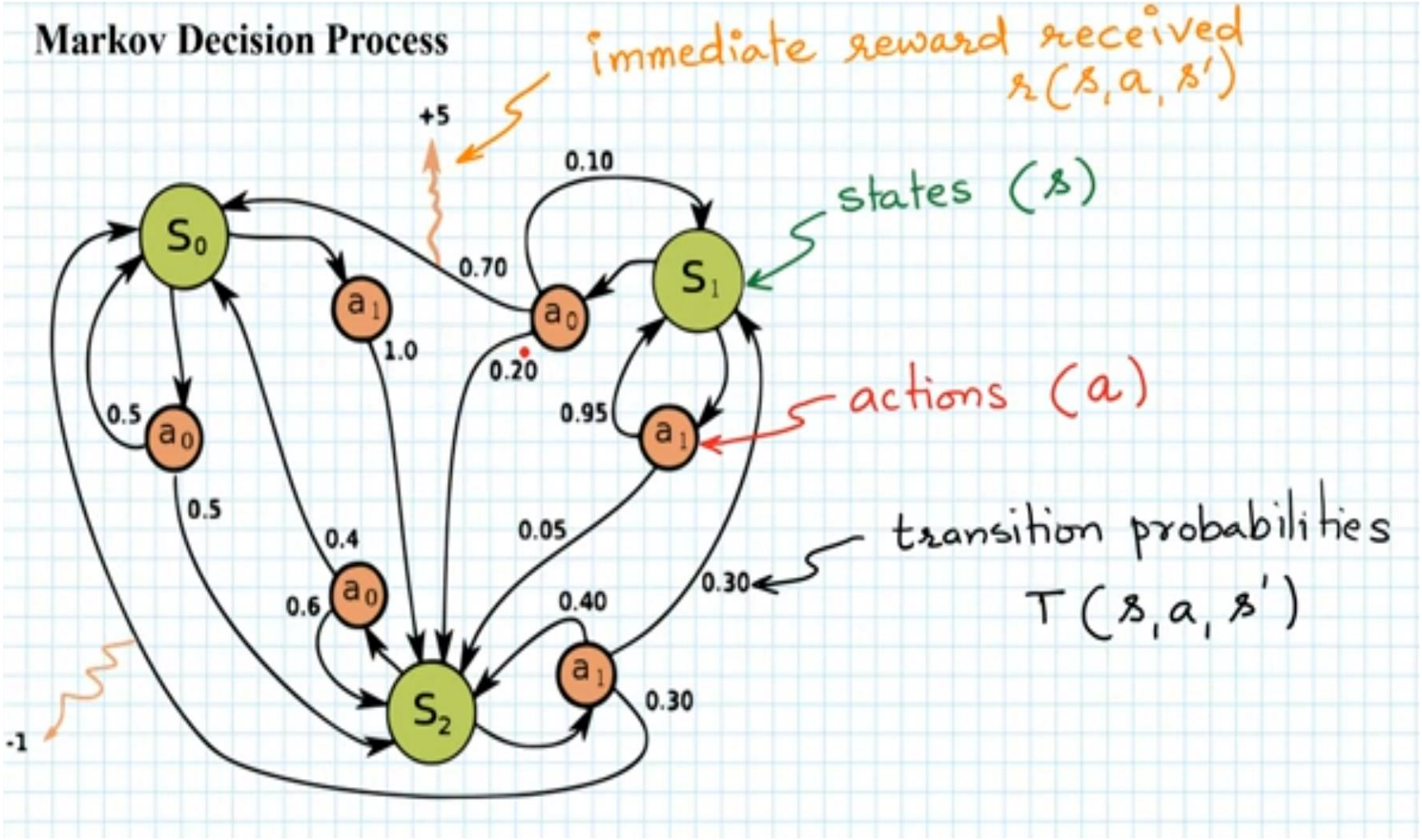
  - In a deterministic world, a fixed action sequence is indeed a possible optimal solution since we know exactly where our actions will take us. In fact, there are several fixed action sequences which are also optimal.
  - For example, we could begin by going right instead of down.
  - The correct answer is (A)

# Markov Decision Process

- STATES: S
- MODEL: T(S, a, S') $\rightarrow$ P(S'|S,a)
- ACTIONS: A(S), A
- REWARDS:R(S), R(S, a) or R(S', a, S')
- Policy: $\pi(S) \rightarrow a$ (Solution to MDP) that will maximize the reward.
- $\pi^*(S)$ (Optimal Policy)

- Policy tells the robot what action needs to be taken in each state.

# Markov Decision Process



# Bellman Equation

Bellman equation writes value of a decision problem for a given state in terms of immediate reward from the action taken in that state and value of state resulting from the action.

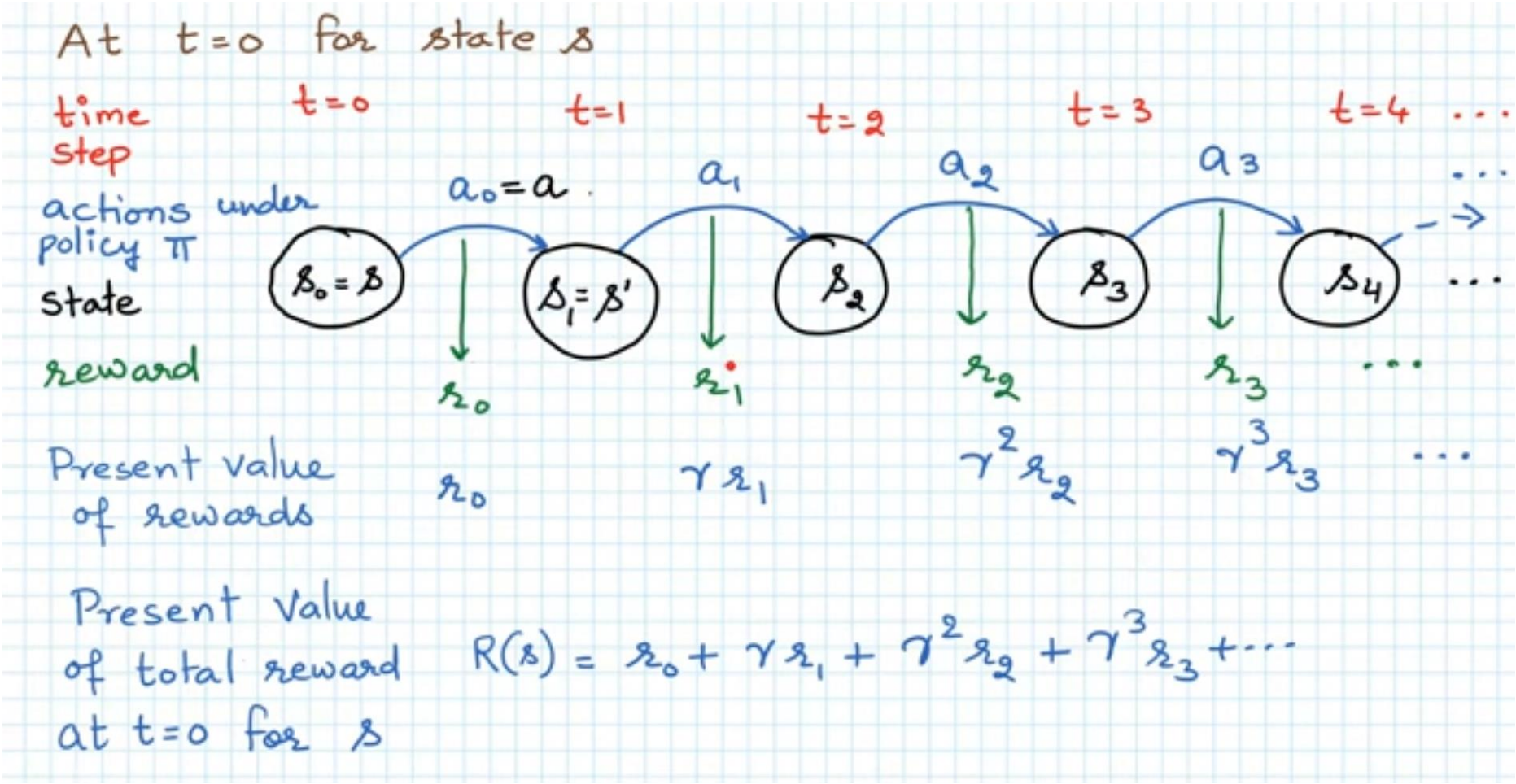
$$V_\pi(s) = r(s,a) + \gamma V_\pi(s') \quad \text{— Bellman Equation}$$

$s$ = current state; $a$ = action taken in state $s$ according to policy $\pi$; $s'$ = state resulting from action $a$ in $s$.
$r(s,a)$ = immediate reward received due to action $a$ in $s$.

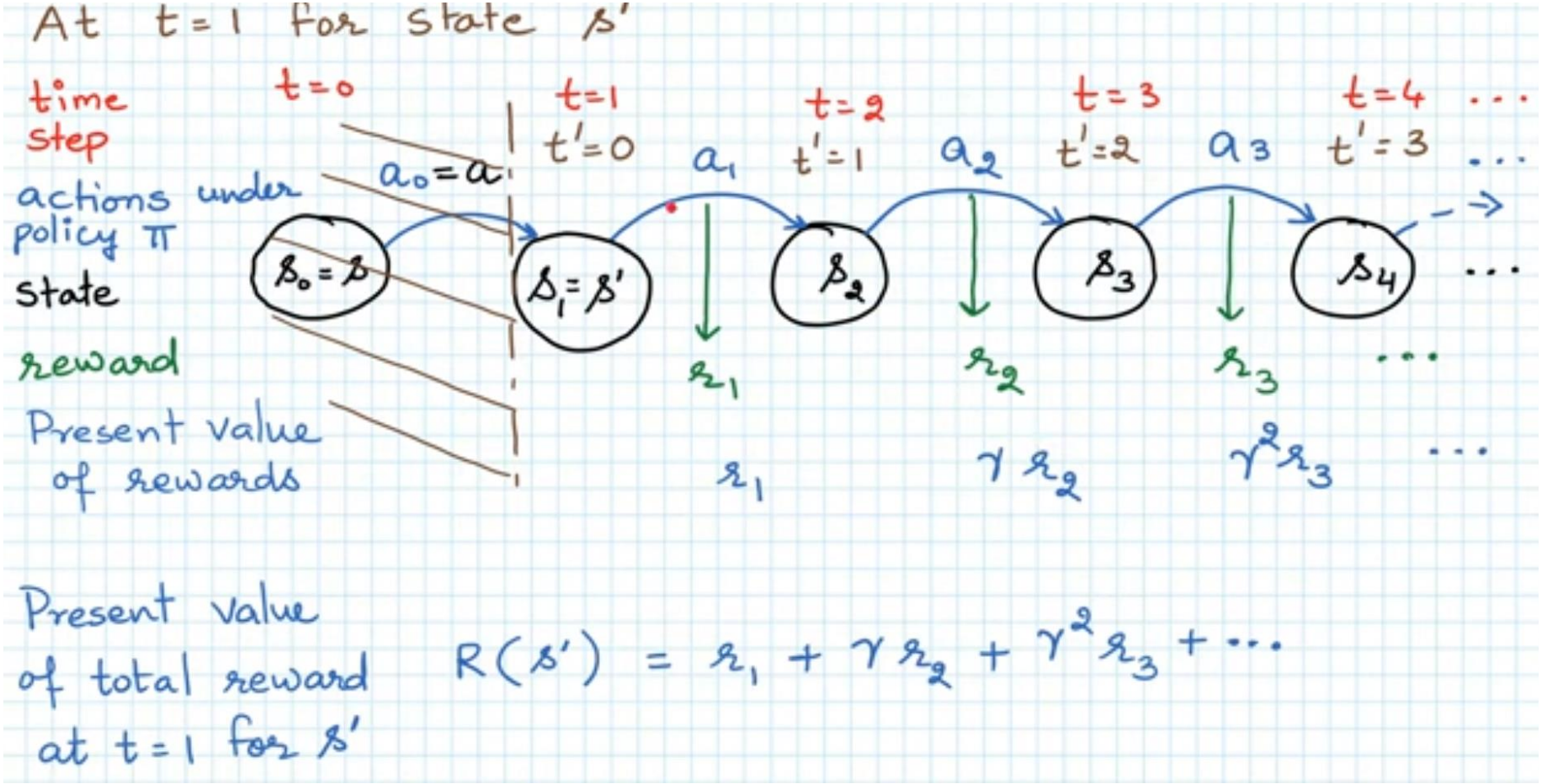
Bellman equation can be solved by backward induction to find optimal policy, e.g., Travelling Sales Man problem.

Exact solution is feasible only for finite Markov Decision Processes with small number of states

# Bellman Equation



# Bellman Equation



# Bellman Equation

$$R(s) = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots$$

$$R(s') = r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots$$

$$R(s) = r_0 + \gamma \{ r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots \}$$

$$R(s) = r_0 + \gamma R(s') \quad \text{—①}$$

$$\Rightarrow \boxed{V_\pi(s) = r(s, a) + \gamma V_\pi(s')}$$

Bellman Equation

# Bellman Principal of Optimality

The principle states that an optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

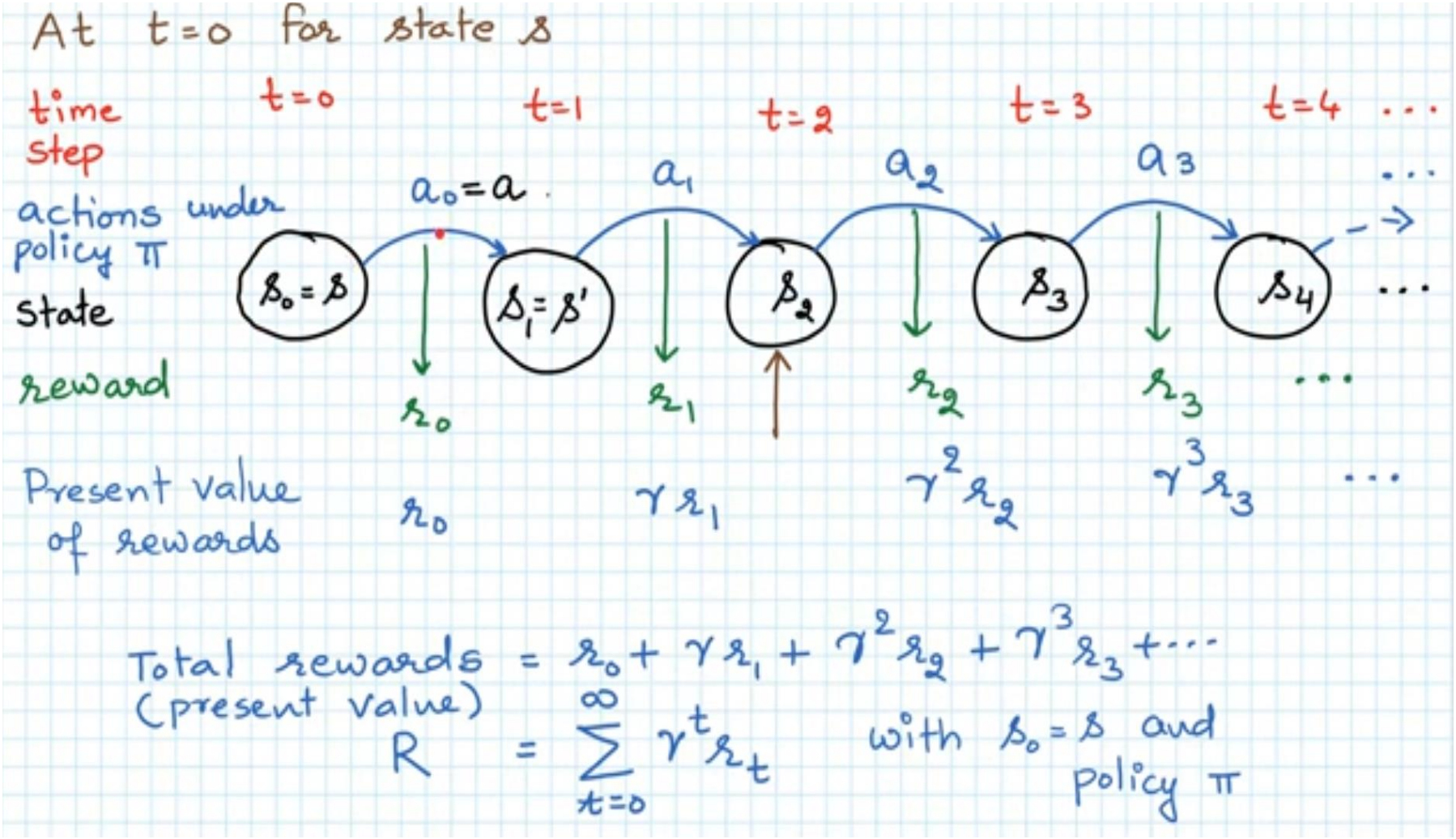
Dynamic Programming method breaks down a multi-step decision problem into smaller (recursive) subproblems using Bellman's Principle of Optimality.

Such problem is said to have optimal substructure, i.e., optimal policy at any state is independent of decisions taken at previous states. This allows us to separate initial decisions from the future decisions and optimize future decisions.

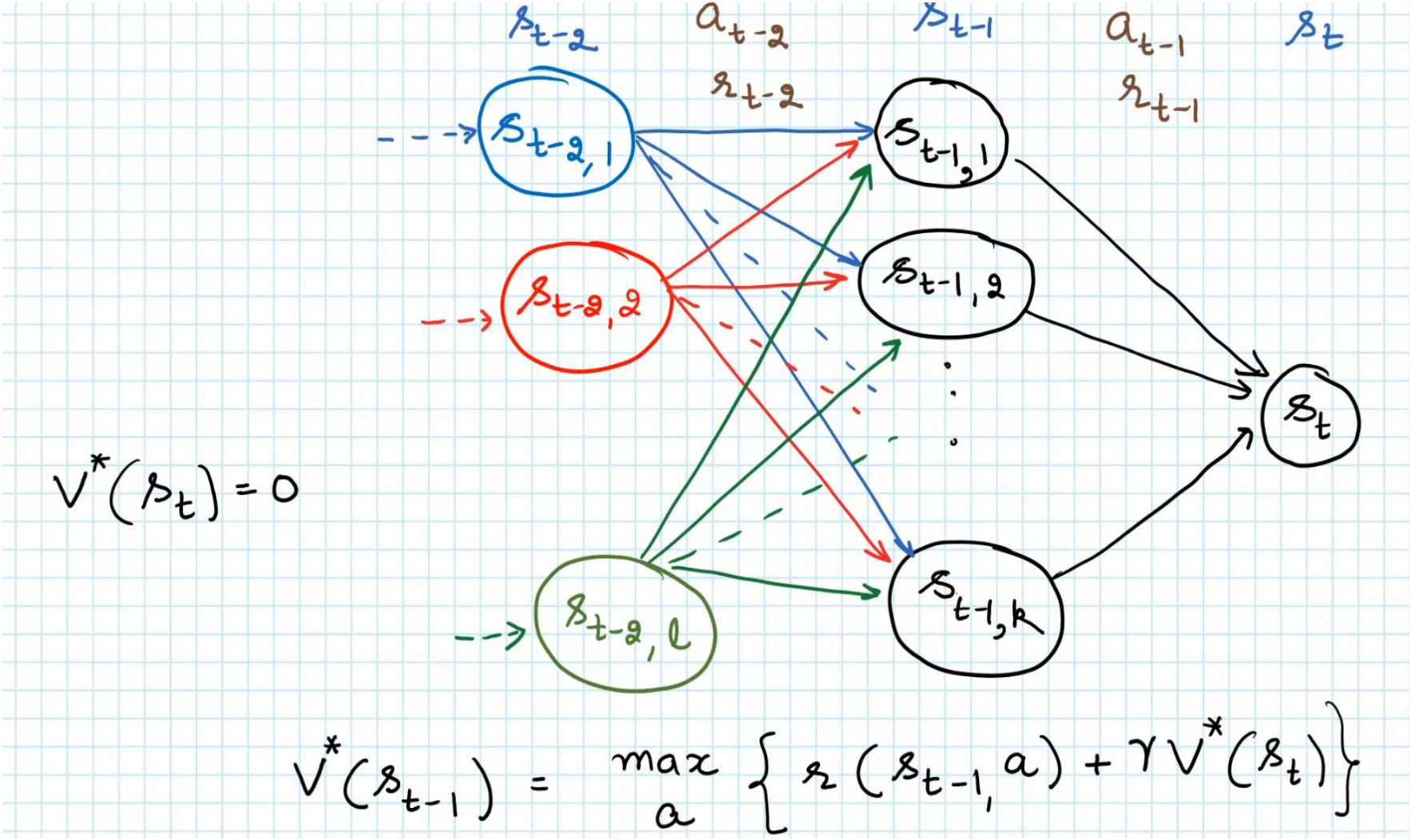
$$V_\pi(s) = r(s, a) + \gamma V_\pi(s')$$

Bellman Equation

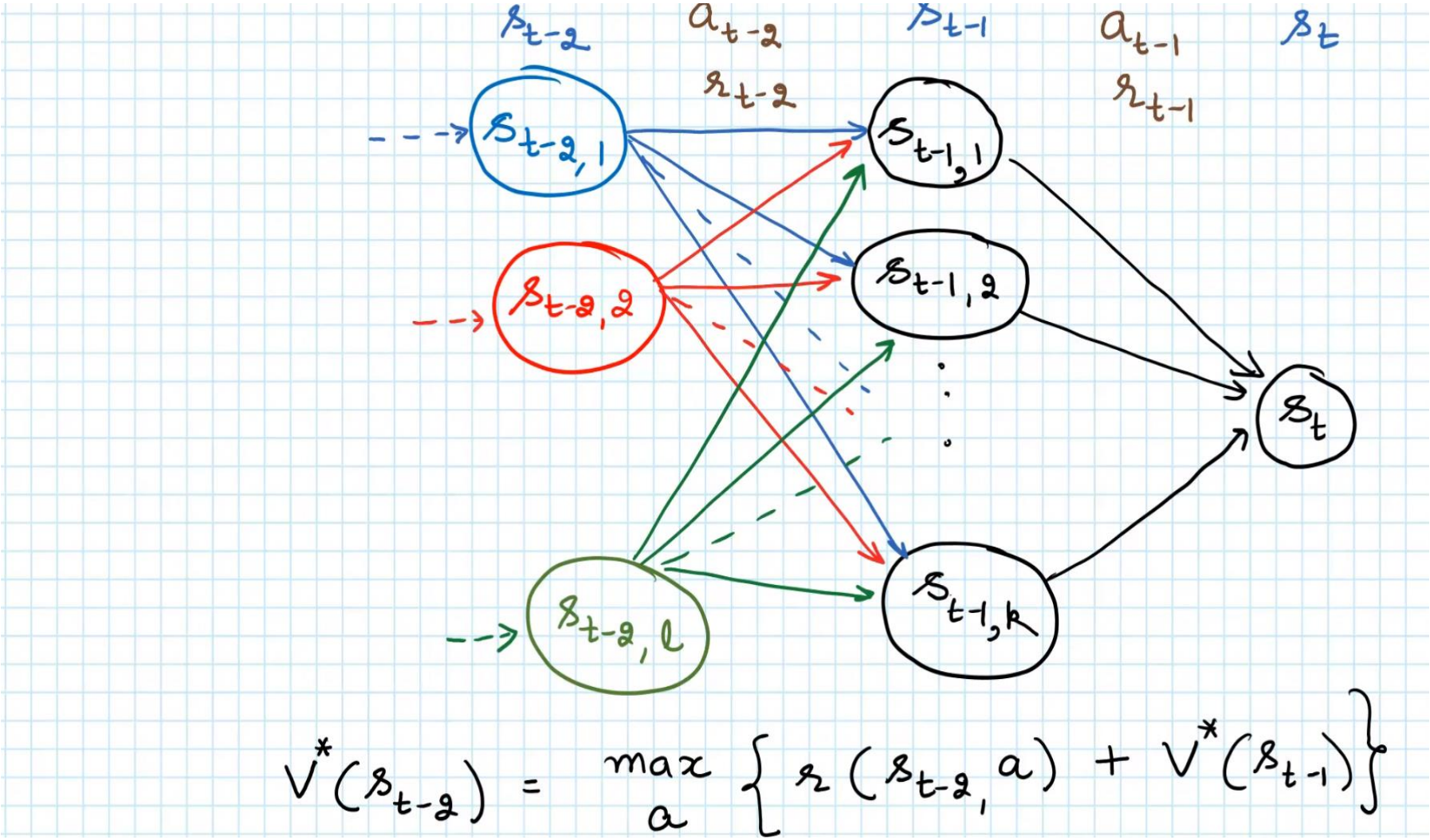
# Bellman Principal of Optimality



# Bellman Principal of Optimality



# Bellman Principal of Optimality



# Approximate Optimal solution for RL Games

Finding optimal policy using Bellman optimality principle (backward induction algorithm) is computationally hard. So, several approximate algorithms based on iterative schemes to find optimal policy have come up (i) Model based algorithms (ii) Model free algorithms

**Model Based Reinforcement Learning**

In model-based reinforcement learning algorithm, the environment is modelled as a Markov Decision Process (MDS) with following elements:

* A set of states (individual state is denoted by $s$)
* A set of actions available in each state (individual action is denoted by $a$ )
* Transition probability function from current state ($s$) to next state ($s'$) under action $a$

  $T(s, a, s')$
* Reward function: immediate reward received on transition from current state ($s$) to next state ($s'$) under action $a$ = $r(s, a, s')$

# Model based Reinforcement Learning

- There are two common approaches to find the optimal policy using recursive relation of Bellman equation.
  - Value Iteration
  - Policy Iteration

# Value Iteration

- Optimal policy(i.e. optimal action at state) is obtained by selecting action that maximize optimal state value function for given state.
- Optimal state value function is obtained using iterative algorithm so it’s called value iteration.

# Value Iteration

**Value Iteration**

1. Set $V_0(s)$ for all states $s$
2. Iterate following until $V_i(s)$ converge to $V^*(s)$ for all states $s$

$$V_{i+1}(s) = \max_a \sum_{s' \in S} T(s,a,s')\{r(s,a,s') + \gamma V_i(s')\}$$

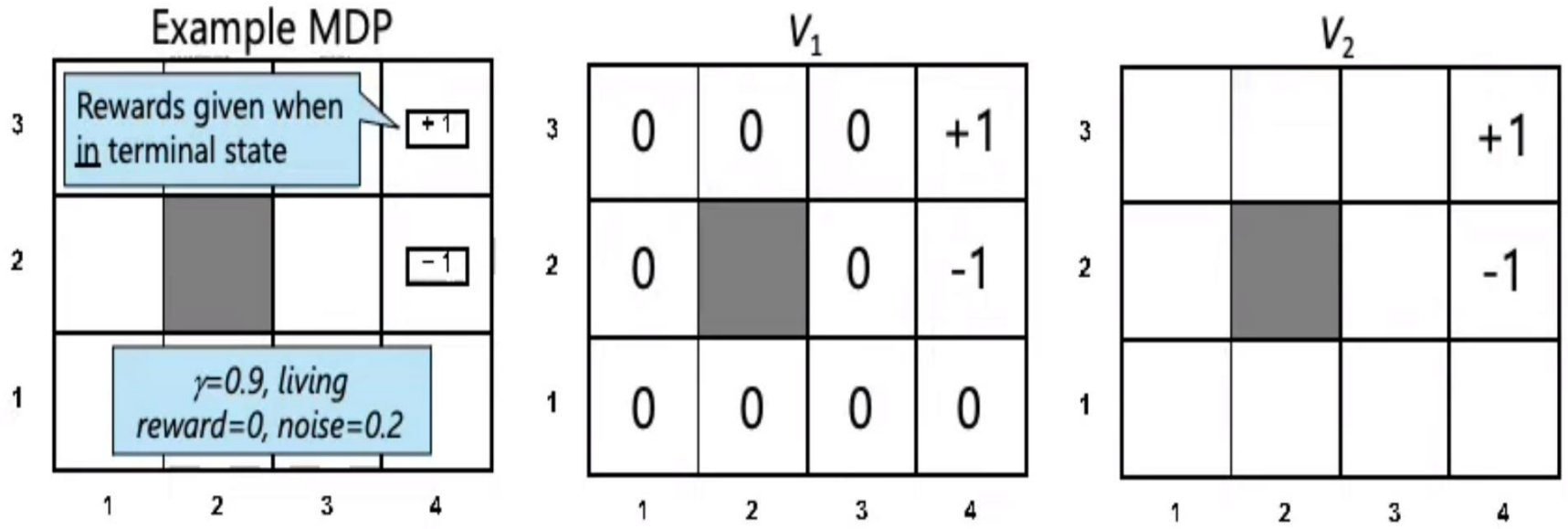
**Optimal Policy**

$$\pi^*(s) = \operatorname{argmax}_a \sum_{s' \in S} T(s,a,s')\{r(s,a,s') + \gamma V^*(s')\}$$

# Value Iteration

## Example: Value Iteration

Bellman Update Rule: $V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[r(s,a,s') + \gamma V_i(s')]$




$V_2(\langle 3,4 \rangle) \leftarrow 1$

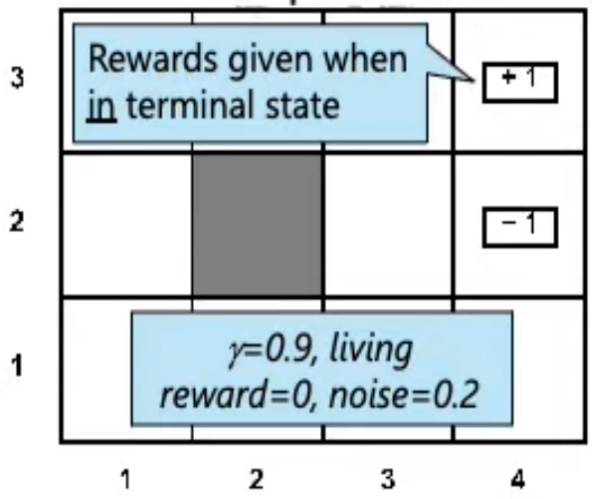
$V_2(\langle 2,4 \rangle) \leftarrow -1$

# Value Iteration

## Example: Value Iteration

Bellman Update Rule: $V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$

Example MDP




$V_1$

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | +1 |
| 0 | | 0 | -1 |
| 0 | 0 | 0 | 0 |

$V_2$

| | | | |
|---|---|---|---|
| | | ? | +1 |
| | | | -1 |
| | | | |

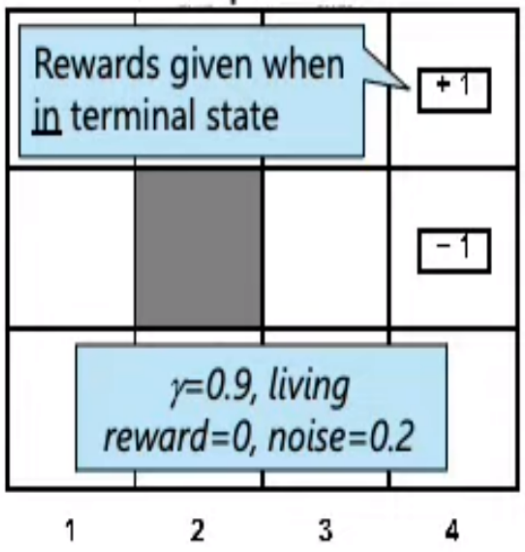
$V_2(\langle 3,3 \rangle) \leftarrow \sum_{s' \in S} P(s'|\langle 3,3 \rangle, right)[r(\langle 3,3 \rangle, right, s') + 0.9V_i(s')]$

$\leftarrow 0.8[0 + 0.9 \times 1] + 0.1[0 + 0.9 \times 0] + 0.1[0 + 0.9 \times 0] = 0.72$

# Value Iteration

Bellman Update Rule: $$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$$

Example MDP




$V_2$

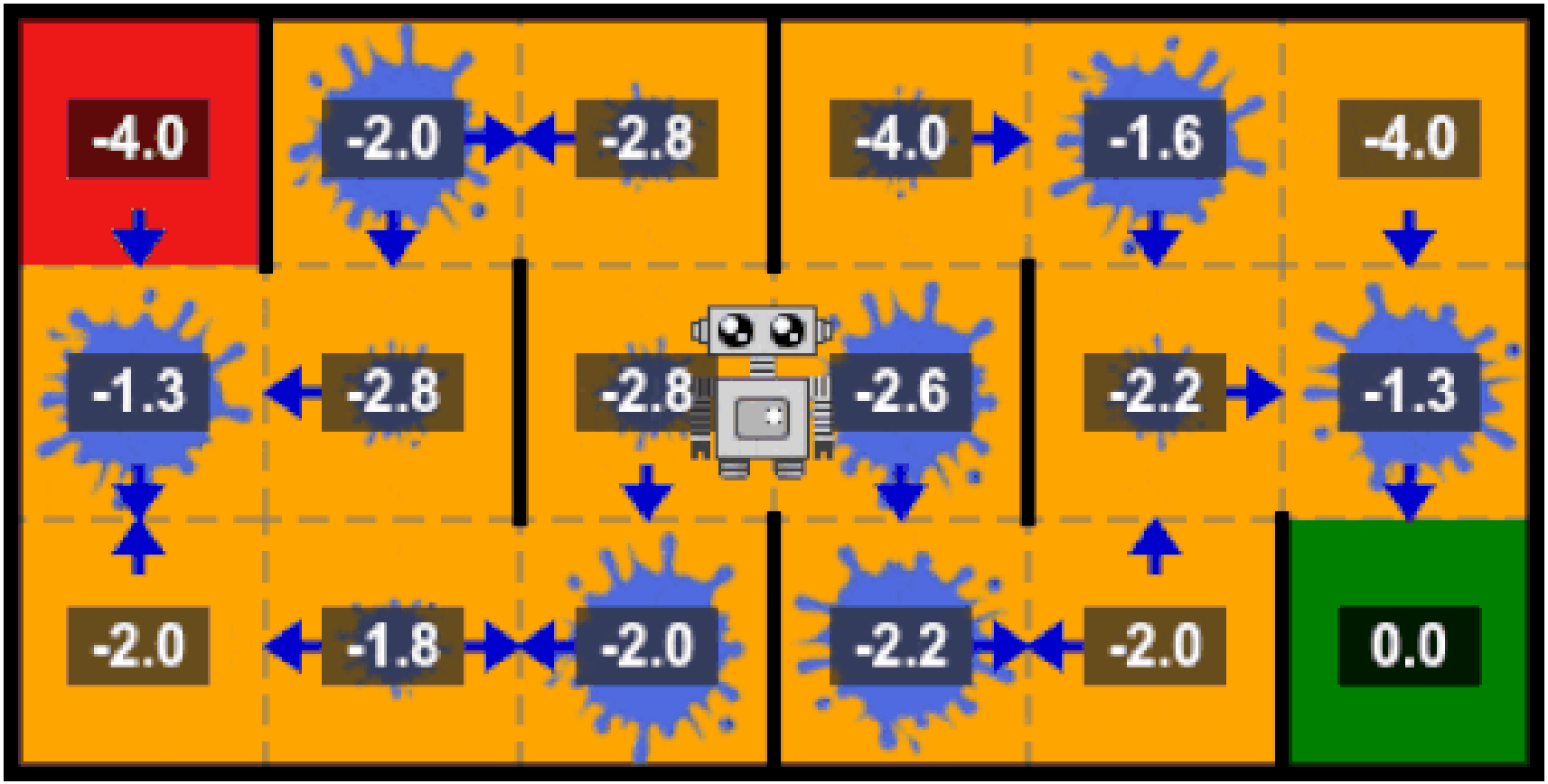
| | | | | |
|---|---|---|---|---|
| 3 | 0 | 0 | 0.72 | +1 |
| 2 | 0 | | 0 | -1 |
| 1 | 0 | 0 | 0 | 0 |
| | 1 | 2 | 3 | 4 |

$V_3$

| | | | | |
|---|---|---|---|---|
| 3 | 0 | 0.52 | 0.78 | +1 |
| 2 | 0 | | 0.43 | -1 |
| 1 | 0 | 0 | 0 | 0 |
| | 1 | 2 | 3 | 4 |

- Information propagates outward from terminal states
- Eventually all states have correct value estimates

# Value Iteration



# Value Iteration

**Value Iteration:**

- Start with $V_0(s) = 0$
- Iterate until convergence: $V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$

- Can prove that value iteration converges to the optimal value function

# Policy Iteration

- Optimal policy(i.e. optimal action at state) is obtained by finding better estimates of optimal policy functions iteratively.
- There are 2 steps in this method.
  - Policy evaluation: evaluate the state value function for current policy.
  - Policy Improvement: policy function is improved by selecting actions that maximize the state value function for each state.

# Policy Iteration

**2. Policy Iteration:** In this method, optimal policy is obtained by finding better estimates of optimal policy function iteratively. That's why this method is called policy iteration. There are two steps in this method

(i) Policy evaluation: in this step we evaluate the state value function for current policy

(ii) Policy improvement: in this step the policy function is improved by selecting the action that maximizes the state value for each state.

$$\pi_0 \xrightarrow{E} V_0(s) \xrightarrow{I} \pi_1 \xrightarrow{E} V_1(s) \xrightarrow{I} \pi_2 \xrightarrow{E} V_2(s) \cdots \xrightarrow{I} \pi^* \xrightarrow{E} V^*(s)$$

# Policy Iteration

Q: What happens if you do only one iteration?

Repeat steps until convergence:

1. **Policy evaluation:** keep current policy $\pi$ fixed, find value function $V^{\pi_k}(\cdot)$
   - Iterate simplified Bellman update until values converge:

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} P(s'|s, \pi_k(s))[R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')]$$

Chooses actions according to $\pi$

2. **Policy improvement:** find the best action for $V^{\pi_k}(\cdot)$ via one-step lookahead

$$\pi_{k+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^{\pi_k}(s')]$$

Policy iteration is optimal too!

- Faster than value iteration in terms of number of (outer) loops, but remember that step 1 has an inner loop too.

# Policy Iteration

**Policy Iteration**

1. Initialize the optimal policy $\pi_0$ randomly
2. Iterate following until $\pi_i$ converge to $\pi^*$

Policy Evaluation:

$$V(s) = \sum_{s' \epsilon S} T(s, \pi(s), s') \left[ r(s, \pi(s), s') + \gamma V(s') \right] \quad \text{— (1)} \quad \text{for all states } s$$

Policy Improvement:

$$\pi_{i+1}(s) = \underset{a}{argmax} \sum_{s' \epsilon S} T(s, a, s') [ r(s, a, s') + \gamma V^*(s') ]$$

Eq (1) can be solved in two ways (i) as a system of linear equations in V($s$) for all states $s$. (ii) iteratively, iterative solution here will be much easier then in value iteration as the policy $\pi_i$ at step $i$ is known.

# Value Iteration vs Policy Iteration

| Aspect | Value Iteration | Policy Iteration |
| --- | --- | --- |
| **Methodology** | Iteratively updates value functions until convergence | Alternates between policy evaluation and improvement |
| **Goal** | Converges to optimal value function | Converges to the optimal policy |
| **Execution** | Directly computes value functions | Evaluate and improve policies sequentially |
| **Complexity** | Typically simpler to implement and understand | Involves more steps and computations |
| **Convergence** | May converge faster in some scenarios | Generally converges slower but yields better policies |

# Q-Learning

- The training information available to the learner is the sequence of immediate rewards r(si,ai) for i = 0, 1,2, . . . .
- Given this kind of training information, it is easier to learn a numerical evaluation function defined over states and actions, then implement the optimal policy in terms of this evaluation function.

# Q-Learning

- One obvious choice is V*.
- The agent should prefer state s1 over state s2 whenever V*(s1) > V*(s2), because the cumulative future reward will be greater from s1.
- The optimal action in state s is the action a that maximizes the sum of the immediate reward r(s, a) plus the value V* of the immediate successor state, discounted by γ.

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}[r(s,a) + \gamma V^*(\delta(s,a))] \qquad \text{....equ (1)}$$

# Q-Learning

- The Q-Function:
- The value of Evaluation function Q(s, a) is the reward received immediately upon executing action a from state s, plus the value (discounted by γ ) of following the optimal policy thereafter.

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a)) \qquad \text{....equ (2)}$$

# Q-Learning

- Rewrite Equation (1) in terms of Q(s, a) as

$$\pi^*(s) = \operatorname{argmax} Q(s, a) \qquad \text{....equ (3)}$$

- Equation (3) makes clear, it need only consider each available action a in its current state s and choose the action that maximizes Q(s, a).

# Q-Learning Algorithm

- Learning the Q function corresponds to learning the optimal policy.
- The key problem is finding a reliable way to estimate training values for Q, given only a sequence of immediate rewards r spread out over time.
- This can be accomplished through iterative approximation

$$V^*(s) = \max_{a'} Q(s, a')$$

# Q-Learning Algorithm

- Rewriting Equation

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a')$$

$Q$ learning algorithm

For each $s, a$ initialize the table entry $\hat{Q}(s, a)$ to zero.
Observe the current state $s$
Do forever:

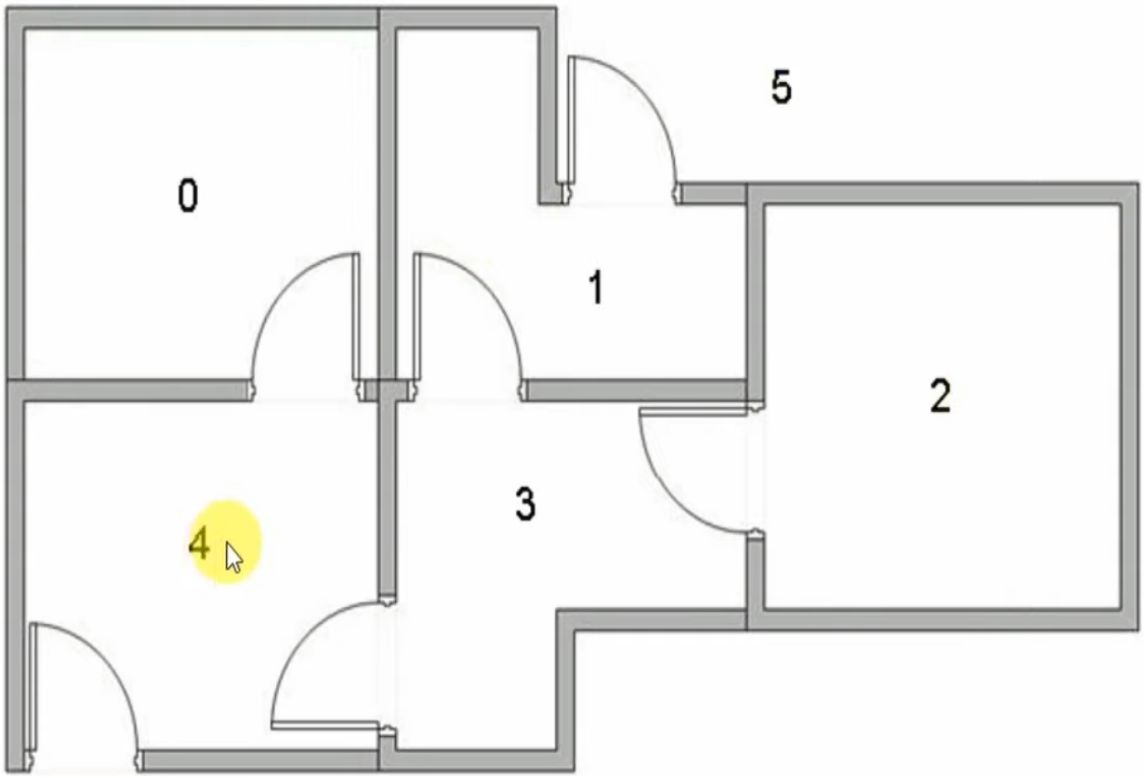
- Select an action $a$ and execute it
- Receive immediate reward $r$
- Observe the new state $s'$
- Update the table entry for $\hat{Q}(s, a)$ as follows:

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

- $s \leftarrow s'$

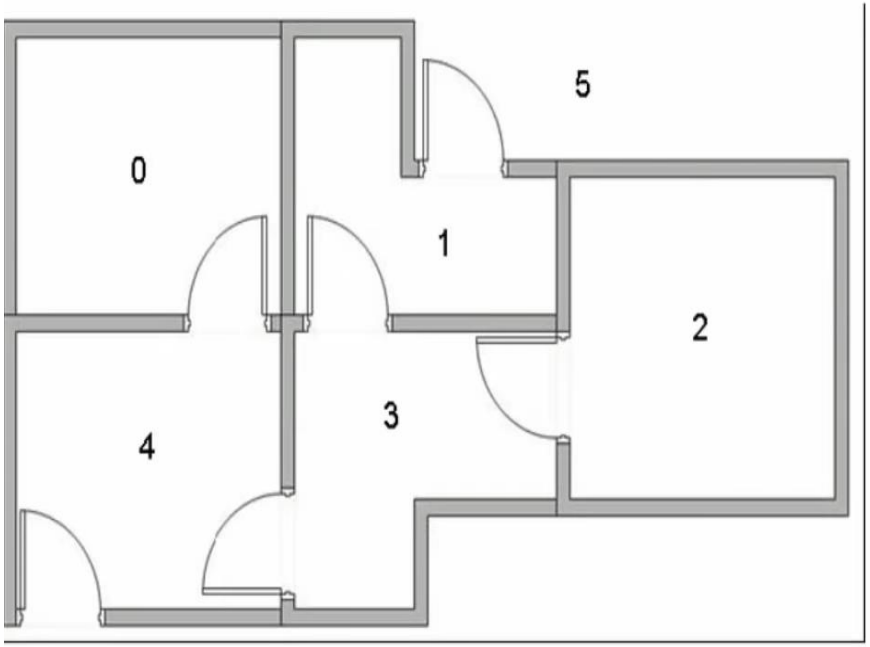
# Q-Learning Examples

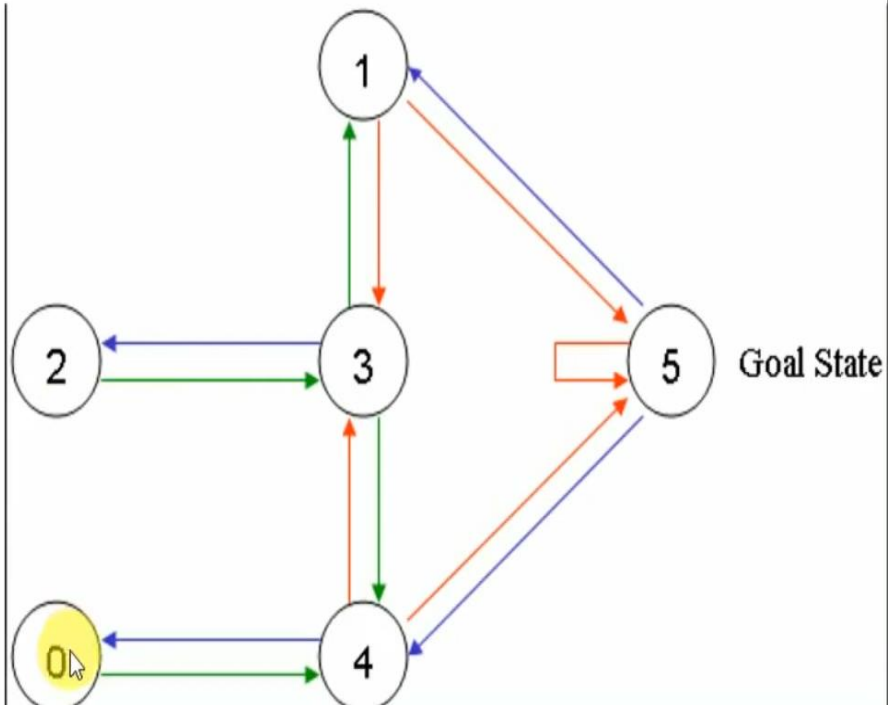
Suppose we have 5 rooms in a building connected by doors as shown in the figure below. We'll number each room 0 through 4. The outside of the building can be thought of as one big room (5). Notice that doors 1 and 4 lead into the building from room 5 (outside).



# Q-Learning Examples

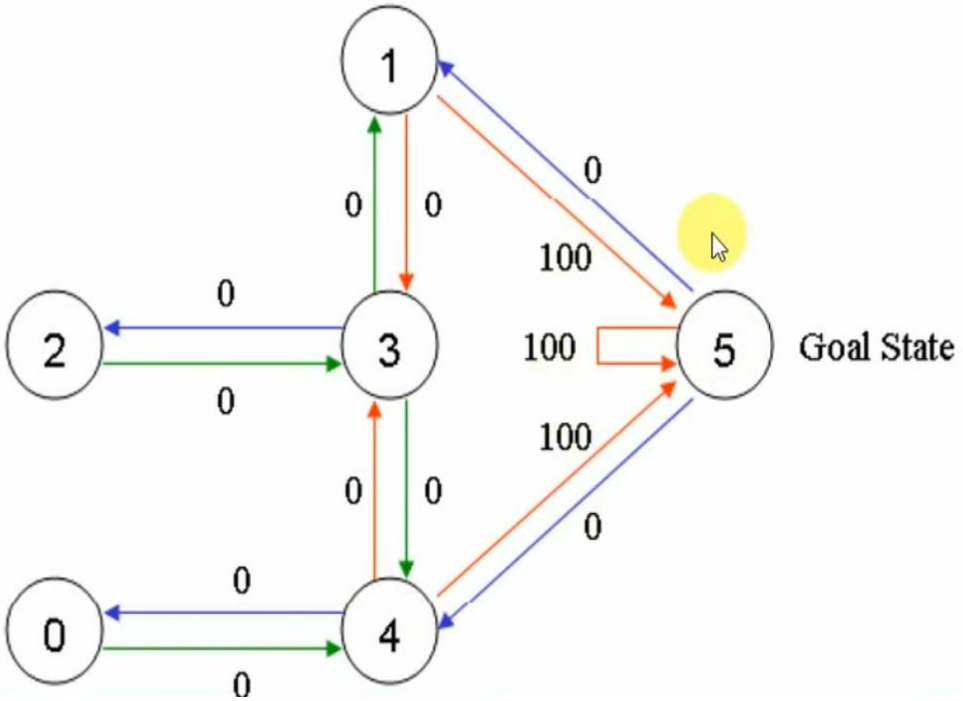
- We can represent the rooms on a graph, each room as a node, and each door as a link.



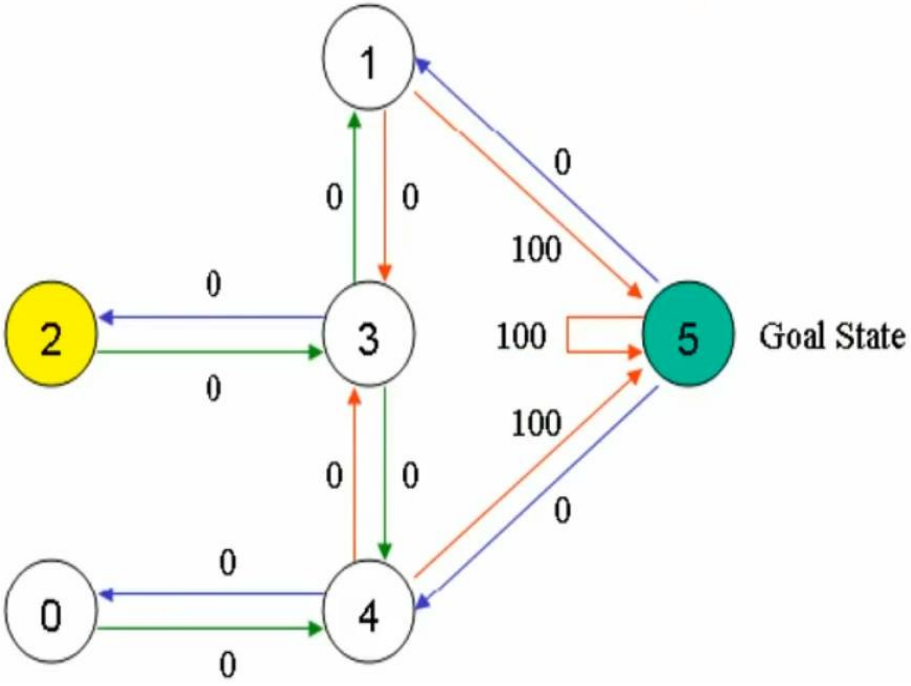
# Q-Learning Examples

- The goal room is number 5
- The doors that lead immediately to the goal have an instant reward of 100. Other doors not directly connected to the target room have zero reward.
- Each arrow contains an instant reward value, as shown below:



# Q-Learning Examples

- We can put the state diagram and the instant reward values into the following reward table, "matrix R". The -1's in the table represent null values (i.e.; where there isn't a link between nodes). For example, State 0 cannot go to State 1.




$$R = \begin{array}{c} \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

# Q-Learning Examples

- Learning rate = 0.8 and the initial state as Room 1.
- Initialize matrix Q as a zero matrix:

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

# Q-Learning Examples

- Now let's imagine what would happen if our agent were in state 5 (next state).
- Look at the sixth row of the reward matrix R (i.e. state 5).
- It has 3 possible actions: go to state 1, 4 or 5.
- 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

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} & & & \text{Action} & & \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

# Q-Learning Examples

- The next state, 5, now becomes the current state.
- Because 5 is the goal state, we've finished one episode.

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

# Q-Learning Examples

- For the next episode, we randomly choose the initial state – say 3 (can go to 1, 2 & 4)

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} & & & \text{Action} & & \end{array} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{cccccc} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{array}\right] \end{array}$$

- Now we imagine that we are in state 1 (next state).

# Q-Learning Examples

- Now we imagine that we are in state 1 (next state).
- Look at the second row of reward matrix R (i.e. state 1).
- It has 2 possible actions: go to state 3 or state 5.
- Then, we compute the Q value:
- 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)] = 0 + 0.8 * Max(0, 100) = 80

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} & & & \text{Action} & & \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

# Q-Learning Examples

- If our agent learns more through further episodes, it will finally reach convergence values in matrix Q like:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

- Tracing the best sequences of states is as simple as following the links with the highest values at each state.