***Introduction***

# REINFORCEMENT LEARNING

***What is Reinforcement Learning?***

- Learning types
  - *Supervised learning*:

    Correct output for each training input is available

  - *Reinforcement learning*:

    Some evaluation of an input is available, but not the exact output

# REINFORCEMENT LEARNING

***What is Reinforcement Learning?***

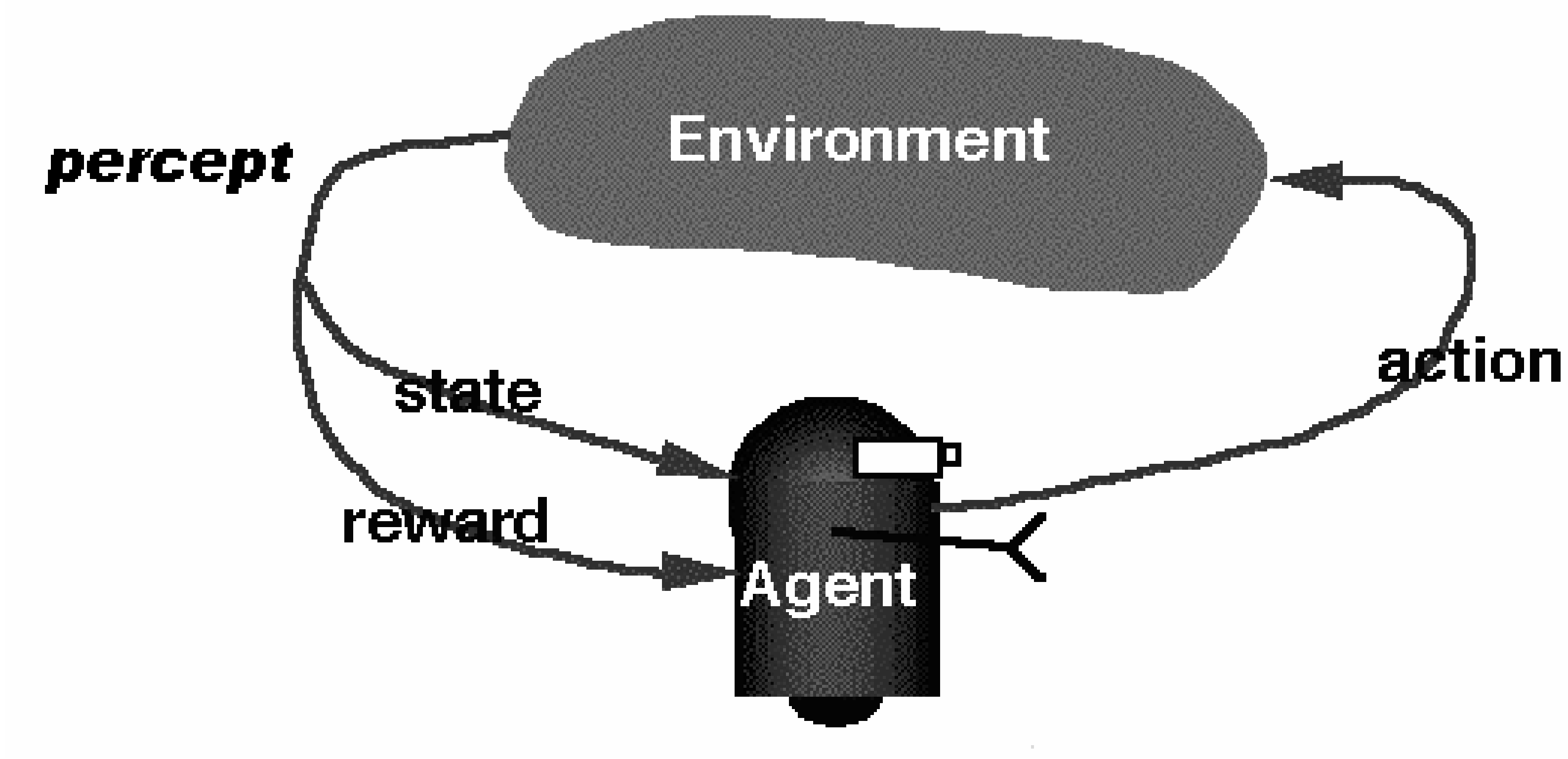
**It is also called *learning with a critic***

**A positive reinforcement is given when the system performs correctly and a negative reinforcement when it performs incorrectly**

**The feedback does not include the *why* or *how* the performance was correct or wrong**

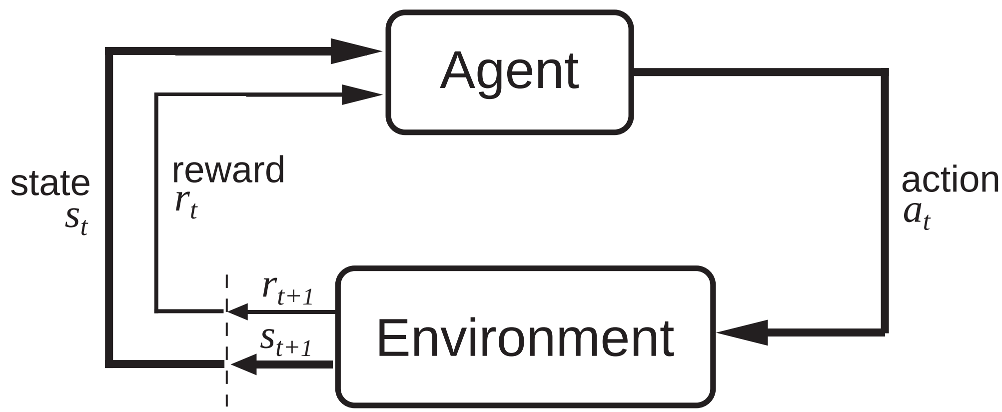
# REINFORCEMENT LEARNING

## *Terminology used in Reinforcement Learning*



# REINFORCEMENT LEARNING

## *Terminology used in Reinforcement Learning*




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

Agent observes state at step $t$: $s_t \in S$

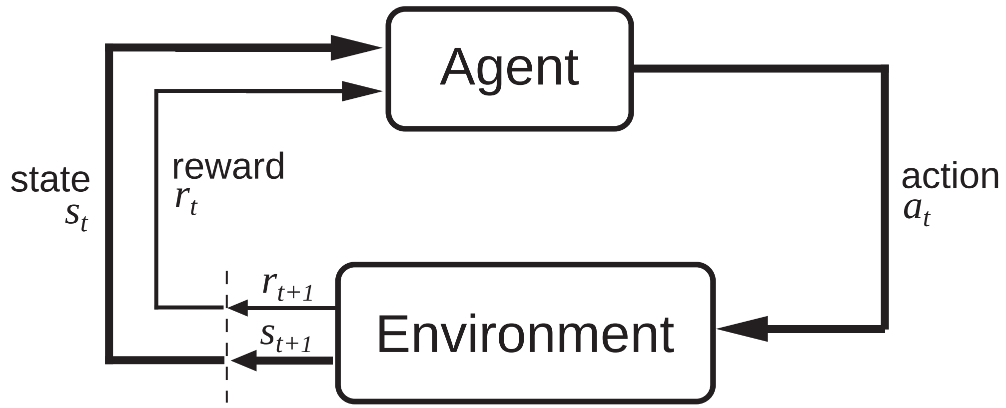
produces action at step $t$: $a_t \in A(s_t)$

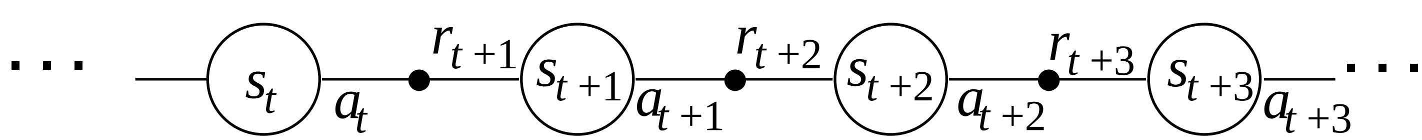
gets resulting reward: $r_{t+1} \in \Re$

and resulting next state: $s_{t+1}$

# REINFORCEMENT LEARNING

## *Terminology used in Reinforcement Learning*

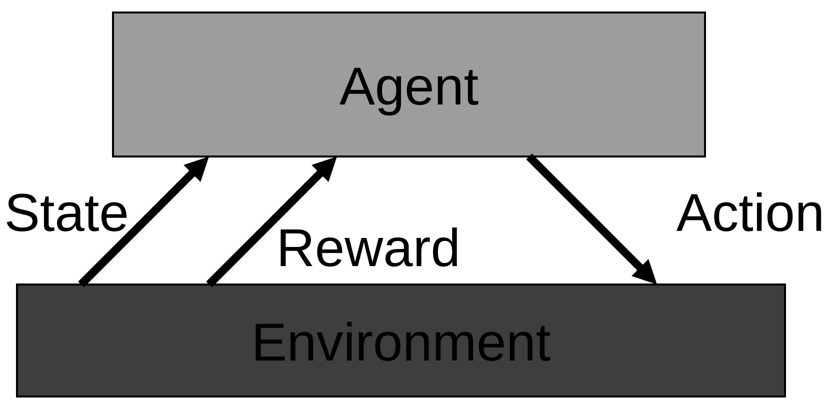


# REINFORCEMENT LEARNING

## *Terminology used in Reinforcement Learning*

Tom Mitchell uses the notation that reward is $r_0$ and not $r_1$ for state-action pair $s_0$-$a_0$




$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r1]{a1} s2 \xrightarrow[r2]{a2} s3$$

# REINFORCEMENT LEARNING

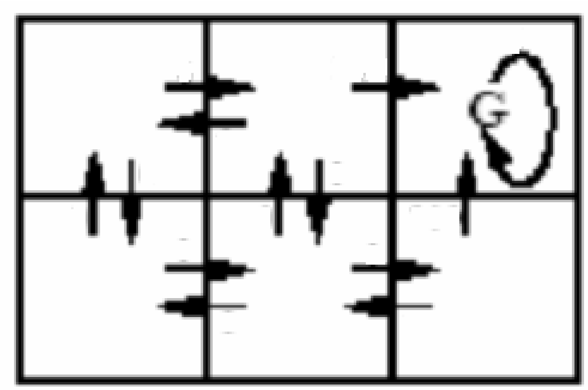
*Terminology used in Reinforcement Learning*

**Example:**

**States: There are 6 states**

**Actions: Going up, down, left & right. In each state only some of these 4 actions are possible**

**Rewards: 100 for actions going into G-state from its neighboring states & 0 for all other actions**



# REINFORCEMENT LEARNING

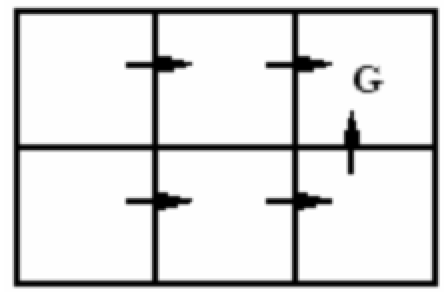
*Learning Problem*

**Learn an optimal action policy $\pi^*$ which maps $S \rightarrow A$**
**such that the agent from any state $s_i$**
**i) reaches its goal**
**ii) does so with least possible cost**

**Note that the only help available during learning is a reward associated with an action which the agent performs when it finds itself in a given state**

# REINFORCEMENT LEARNING

*Utilization after Learning*




**One optimal policy**

**After learning this policy, the agent will utilize it forever**

**From any initial state it will follow the policy and reach the goal state in an optimal way**

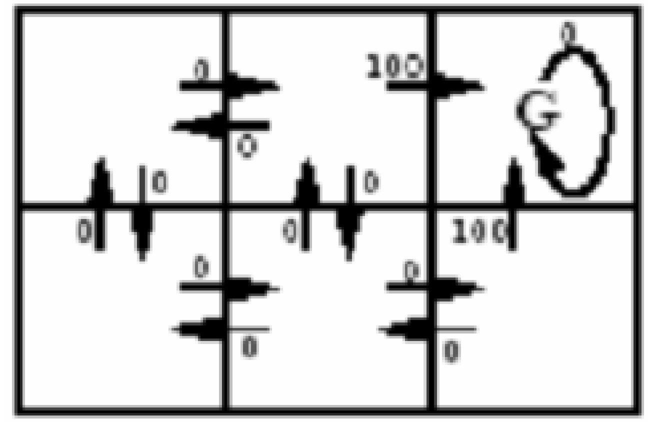
***Learning Problem:***

***How to learn Optimal Action Policy $\pi$*?***

# REINFORCEMENT LEARNING

*Learning Problem*

**If something is known about the proximity of a given state to the goal state, it can be incorporated into the rewards associated with the incoming actions**




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

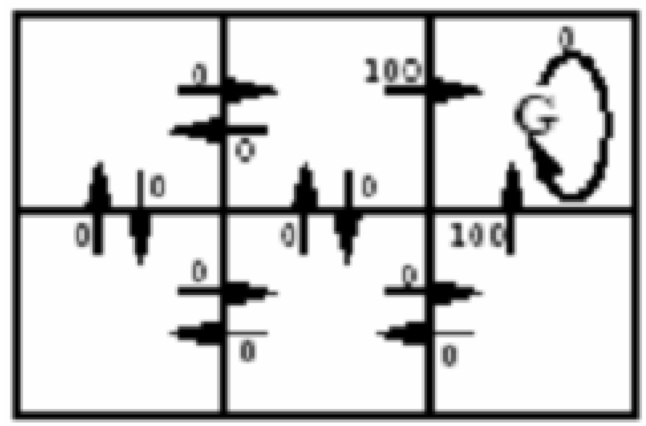
**However, if nothing is known then we have a reward of zero for going into all states, except the goal state**
**For going into the goal state we have a high reward**

# <u>REINFORCEMENT LEARNING</u>

***Learning Problem***

**The policy $\pi^*$ dealing with a state can be the policy which maximizes the sum of individual rewards**

$$r_t + r_{t+1} + r_{t+2} + \ldots$$




**This policy would guarantee to reach the goal from that state, however it does not guarantee reaching goal with minimum steps**

**For this purpose, we can incorporate a discount factor $\gamma$,**

$$r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots + \gamma^n r_n$$

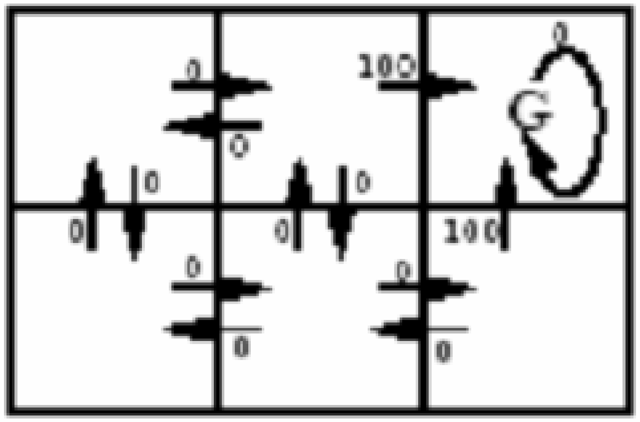
**where $0 \le \gamma \le 1$ and n = goal reaching time**

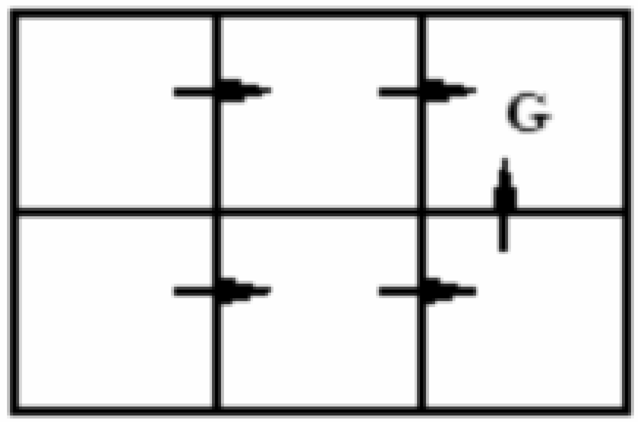
# REINFORCEMENT LEARNING

## *Learning Problem*

**So we search for the policy $\pi^*$ for each state which has actions that maximize**

$$\mathbf{r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots}$$



# REINFORCEMENT LEARNING

## *Learning Problem: Value Function*

**The summation** $\mathbf{r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots}$
**is called value function for a state under policy $\pi$**

$$V^{\pi}(s) \equiv r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots \equiv \sum^{\infty}_{i=0} \gamma^i r_{t+i}$$

**It is the expected return starting at state *s* following $\pi$**
**Note that this equation is a function of the starting state s, which may be any one of the possible states**

**Restated, the task is to learn the optimal policy $\pi^*$**

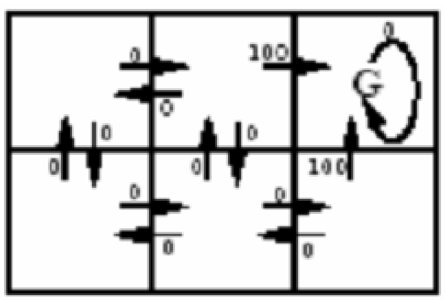
$$\pi^* \equiv \underset{\pi}{\operatorname{argmax}}\, V^{\pi}(s), (\forall s)$$

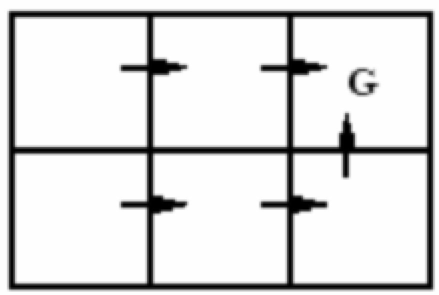
# REINFORCEMENT LEARNING

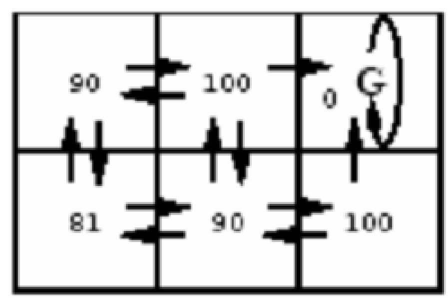
## *Learning Problem: Value Function*

$$V^{\pi}(s) \equiv r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots$$

**For the given rewards, an optimal policy $\pi^*$ and its $V^*(s)$ values (if $\gamma = 0.9$) would be**



# REINFORCEMENT LEARNING

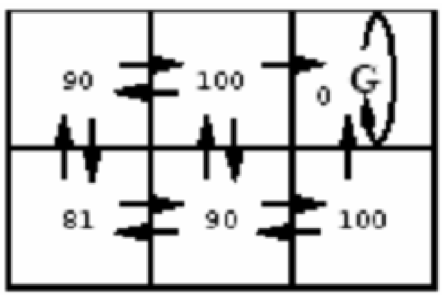
***Learning Problem: Value Function***

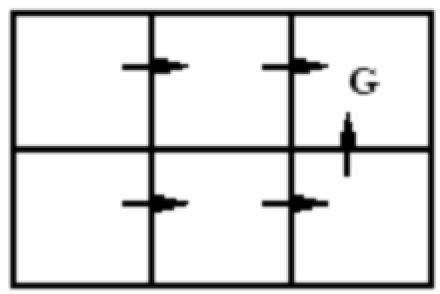
**Now we can attempt to learn $V^*(s)$ values**
**and**
**formulate an optimal policy $\pi^*$ from these values**

# REINFORCEMENT LEARNING

*Learning Problem: Value Function*

**If all the V*(s) values are known, we can formulate an optimal policy π***
**by choosing that action in any state s, which leads to a neighboring state with the highest V value**



### *Learning Problem: Q Function*

**We define a function called Q function**

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

***$Q^\pi(s,a)$*** **The expected return starting at state *s* with action *a* and then following $\pi$**

**Note that Q function is associated with actions whereas V function is associated with states. It can be shown equal to**

$$Q(s_t, a_t) \; = \; r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')$$

# REINFORCEMENT LEARNING

*Learning Problem: Q Function*

**Starting from a state, if we repeatedly choose actions with highest Q values we get the V* value for that state**

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

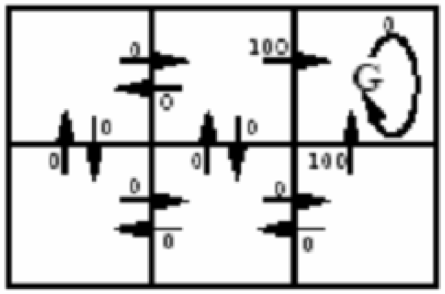
**In other words, the V*(s) value of any state is the maximum of the Q values of the actions possible in s**

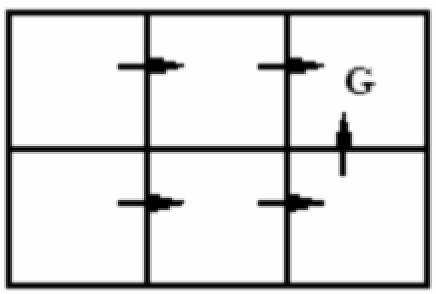
$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \\ &= r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \end{aligned}$$

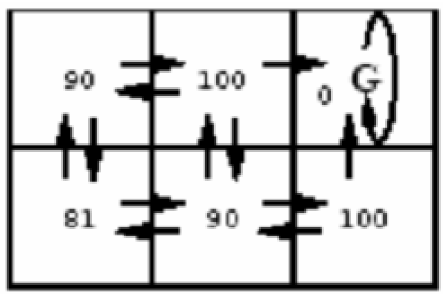
**The optimal policy is** $\pi^*(s) = \operatorname{argmax}_a Q(s, a)$

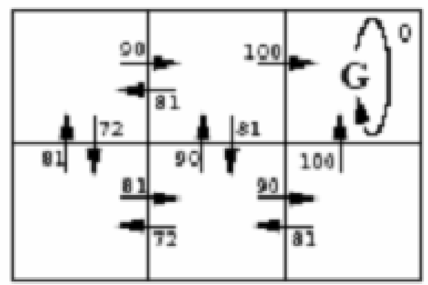
# REINFORCEMENT LEARNING

## *Learning Problem: Q Function*



***Learning Problem: How to learn Q function?***

# REINFORCEMENT LEARNING

*Learning Algorithm for Q*

**Let Q^ denote learner's current approximation to Q.**

**Consider training rule**

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

**where**
***s'*** **is the state resulting from applying action** ***a*** **in state** ***s***

# REINFORCEMENT LEARNING

## *Learning Algorithm for Q*

For each $s, a$ initialize table entry $\hat{Q}(s, a) \leftarrow 0$

Observe current state $s$

Do forever:

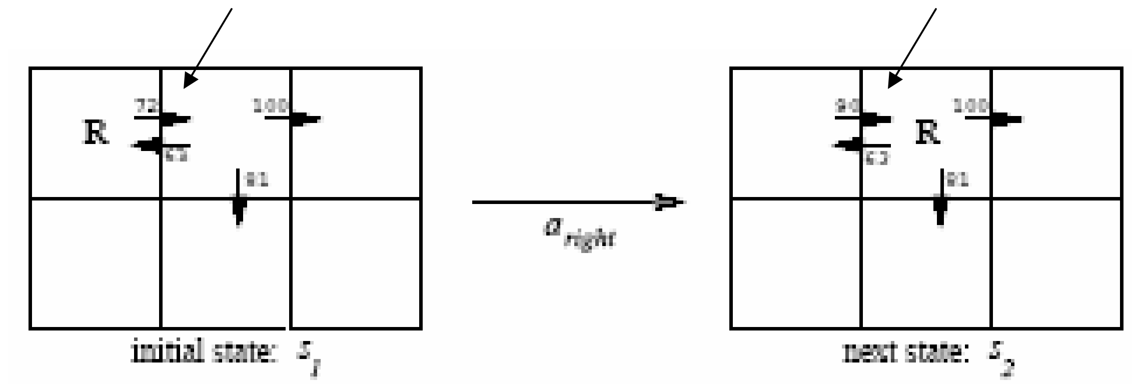
- Select an action $a$ and execute it
- Receive immediate reward $r$
- Observe the new state $s'$
- Update the table entry for $\hat{Q}(s, a)$ as follows:

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

- $s \leftarrow s'$

# REINFORCEMENT LEARNING

## *Learning Algorithm for Q*




$$
\begin{aligned}
\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\
&\leftarrow 0 + 0.9 \ \max\{63, 81, 100\} \\
&\leftarrow 90
\end{aligned}
$$

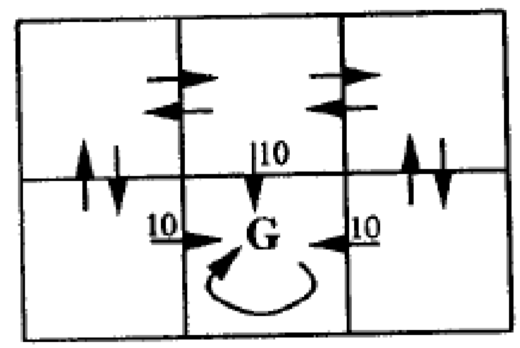
# REINFORCEMENT LEARNING

## *Learning Algorithm for Q*

**In the beginning all Q values are initialized to zero**

Q(state, action) table

| | | Initial |
|---|---|---|
| S11 | (Down) | 0 |
| | (Right) | 0 |
| S12 | (Down) | 0 |
| | (Right) | 0 |
| | (Left) | 0 |
| S13 | (Down) | 0 |
| | (Left) | 0 |
| S21 | (Up) | 0 |
| | (Right) | 0 |
| S22 | (Self) | 0 |
| S23 | (Up) | 0 |
| | (Left) | 0 |



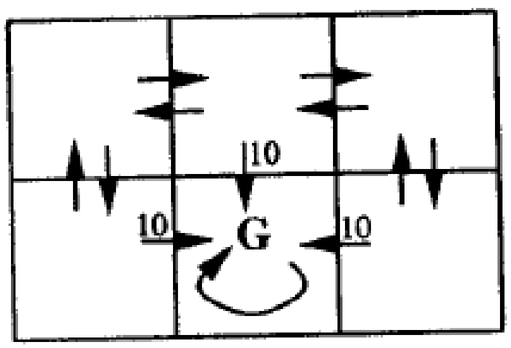
# REINFORCEMENT LEARNING

## *Learning Algorithm for Q*

**As a first training episode assume the agent begins in S21 and then travels clockwise until it reaches the goal state**

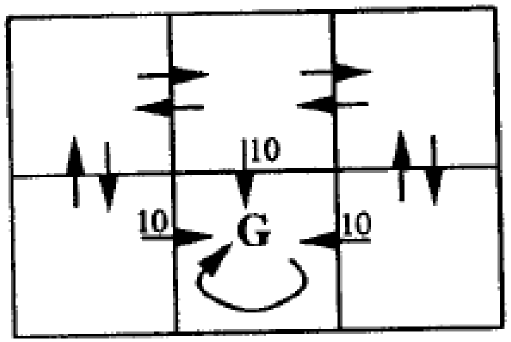
Q(state, action) table

| | | Initial | 1st Episode |
|---|---|---|---|
| S11 | (Down) | 0 | 0 |
| | (Right) | 0 | 0 |
| S12 | (Down) | 0 | 0 |
| | (Right) | 0 | 0 |
| | (Left) | 0 | 0 |
| S13 | (Down) | 0 | 0 |
| | (Left) | 0 | 0 |
| S21 | (Up) | 0 | 0 |
| | (Right) | 0 | 0 |
| S22 | (Self) | 0 | 0 |
| S23 | (Up) | 0 | 0 |
| | (Left) | 0 | 10 |



# REINFORCEMENT LEARNING

*Learning Algorithm for Q*




**After 2nd and 3rd identical training episodes**

Q(state, action) table

| | | Initial | 1st Episode | 2nd | 3rd |
|---|---|---|---|---|---|
| S11 | (Down) | 0 | 0 | 0 | 0 |
| | (Right) | 0 | 0 | 0 | 0 |
| S12 | (Down) | 0 | 0 | 0 | 0 |
| | (Right) | 0 | 0 | 0 | 6.4 |
| | (Left) | 0 | 0 | 0 | 0 |
| S13 | (Down) | 0 | 0 | 8 | 8 |
| | (Left) | 0 | 0 | 0 | 0 |
| S21 | (Up) | 0 | 0 | 0 | 0 |
| | (Right) | 0 | 0 | 0 | 0 |
| S22 | (Self) | 0 | 0 | 0 | 0 |
| S23 | (Up) | 0 | 0 | 0 | 0 |
| | (Left) | 0 | 10 | 10 | 10 |

# REINFORCEMENT LEARNING

*Learning Algorithm for Q*

**Note that if rewards are non-negative (zero or positive), then**

$$(\forall s, a, n) \quad \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)$$

**The Q^ at $n^{th}$ iteration is less than or equal to Q^ at $n+1^{th}$ iteration**

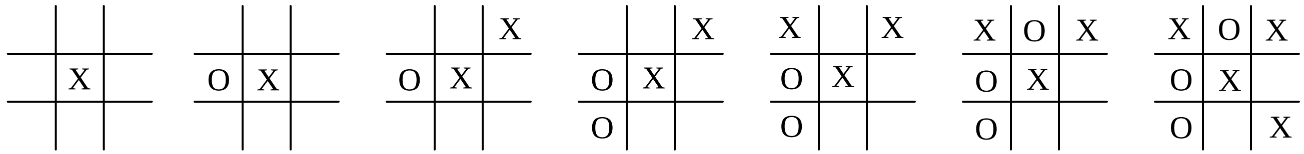
**And also the final Q is greater than or equal to Q^ at $n^{th}$ iteration**

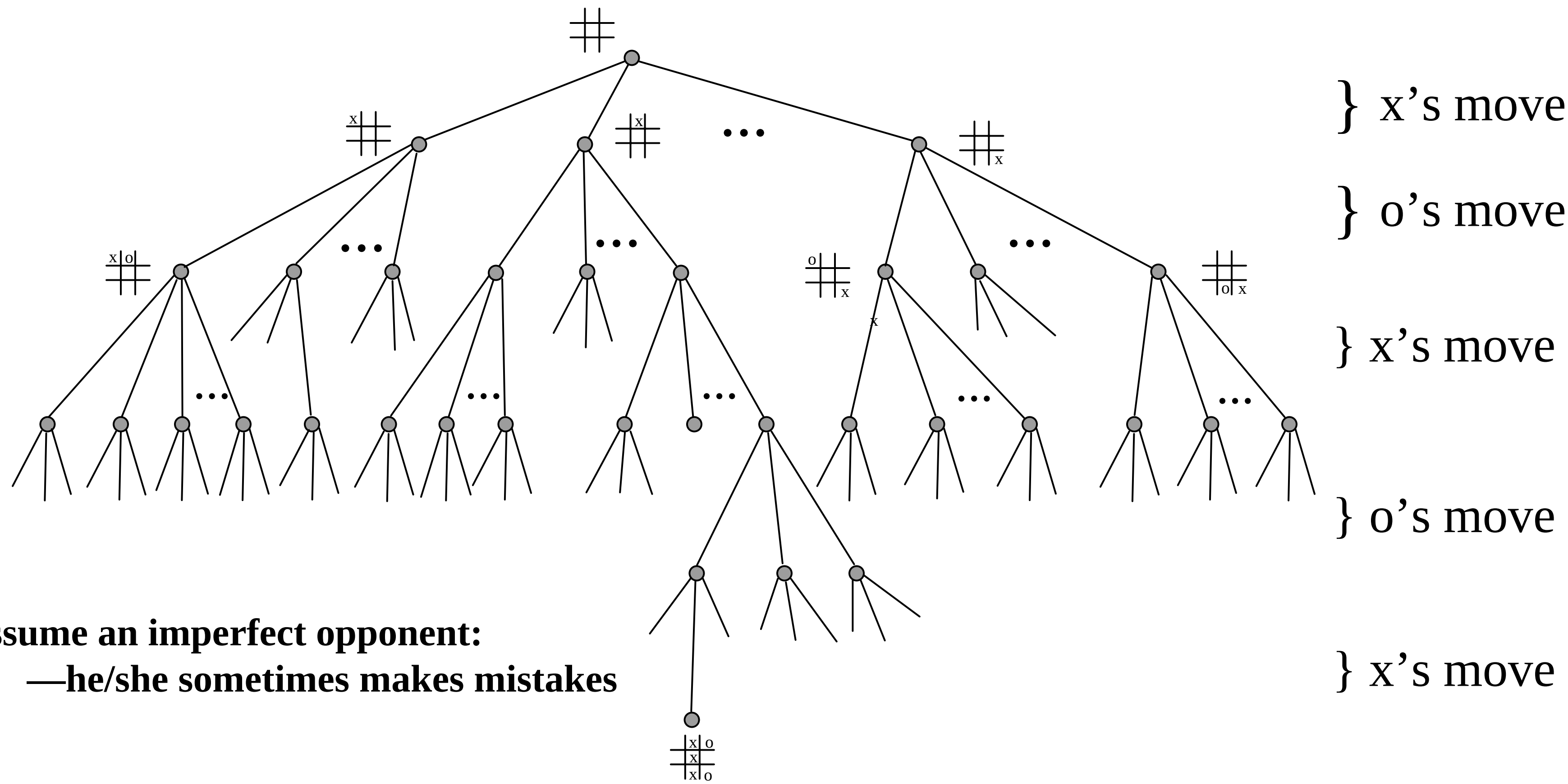
$$(\forall s, a, n) \quad 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$$

**If each <s, a> is visited infinitely often, Q^ converges to Q**

# REINFORCEMENT LEARNING

## *Tic-Tac-Toe Example*

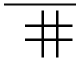
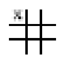
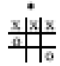
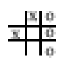
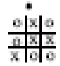




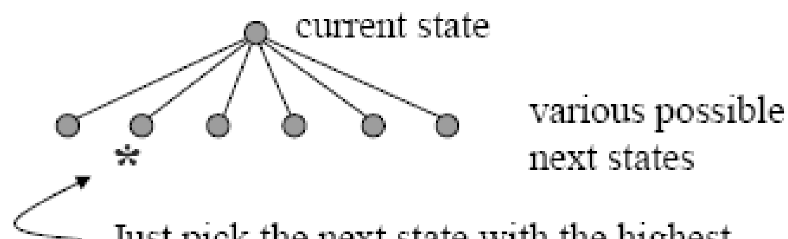

**Assume an imperfect opponent:**
**—he/she sometimes makes mistakes**

# REINFORCEMENT LEARNING

1. Make a table with one entry per state:

| State | $V(s)$ – estimated probability of winning | |
|---|---|---|
|  | .5 | ? |
|  | | |
| ⋮ | ⋮ | |
|  | 1 | win |
| ⋮ | ⋮ | |
|  | 0 | loss |
| ⋮ | ⋮ | |
|  | 0 | draw |

2. Now play lots of games. To pick our moves, look ahead one step:




Just pick the next state with the highest estimated prob. of winning — the largest $V(s)$; a ***greedy*** move.

But 10% of the time pick a move at random; an ***exploratory move***.

# REINFORCEMENT LEARNING

***Points to Note:***

1. **There may be multiple goals**

2. **The rewards may be negative also**

3. **The *rewards* may not only be dependent on current state and action but also on other factors**

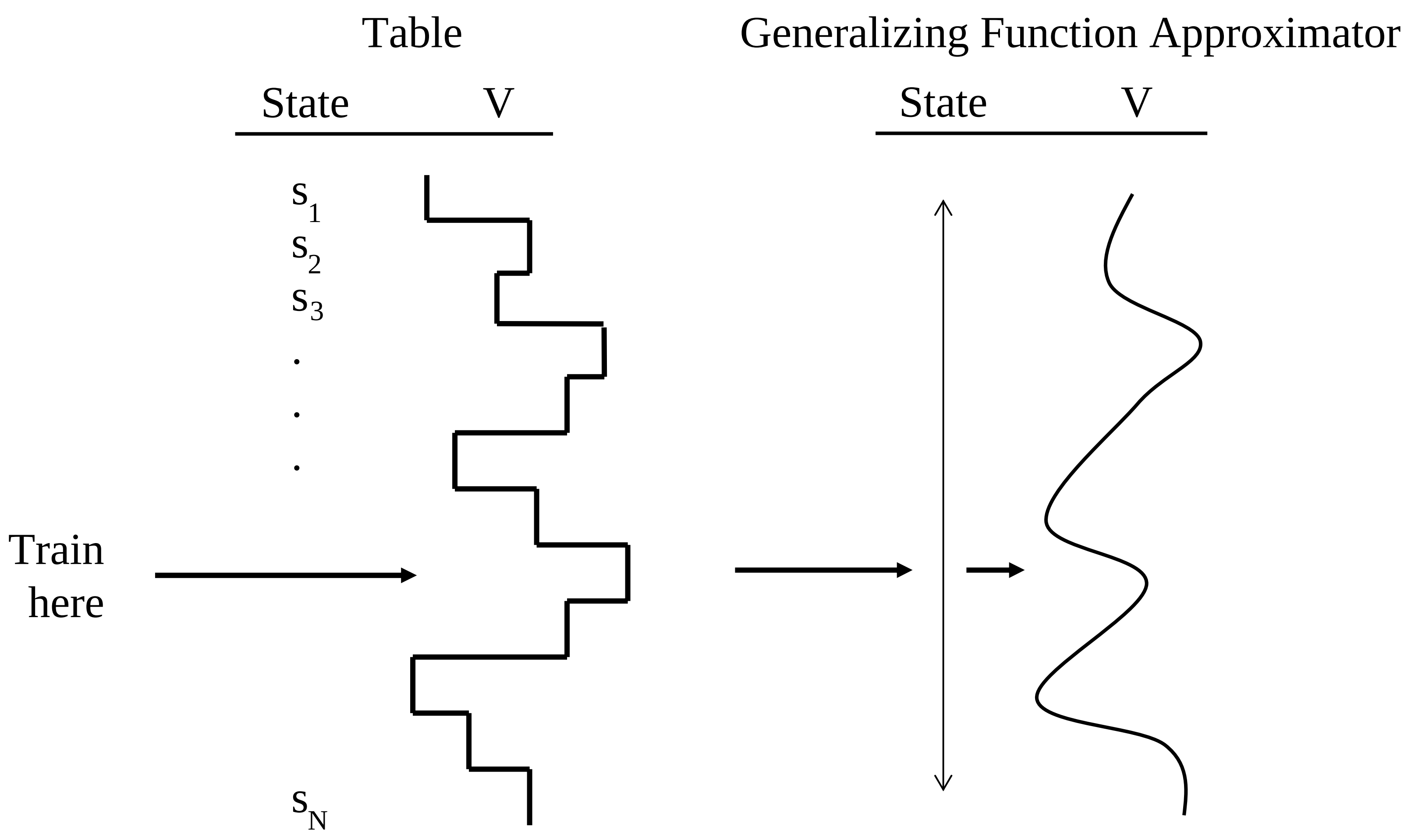
4. **The *next state* may not only be dependent on current state and action but also on other factors**

# REINFORCEMENT LEARNING

*Ongoing Research*

**Replace Q^ table with neural net or some other generalizer**

# REINFORCEMENT LEARNING

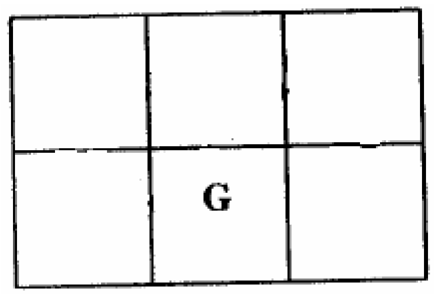


***Summary***

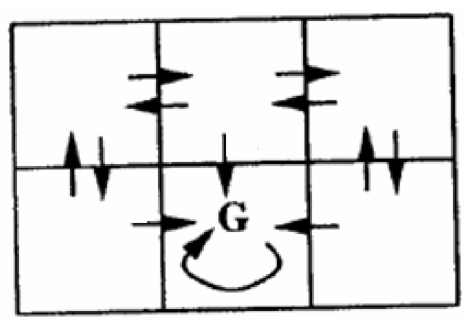
# REINFORCEMENT LEARNING

*Learning Algorithm for Q: Example*

**The states are defined. One of the states is a goal state**



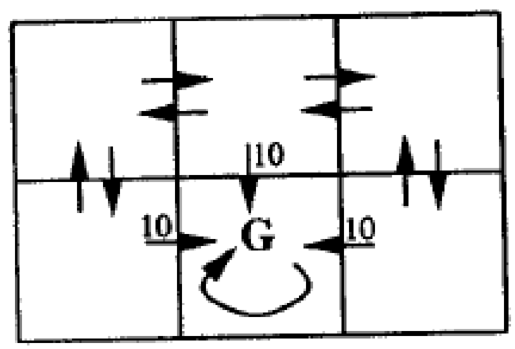

**The possible actions in each state is defined**



# REINFORCEMENT LEARNING

## *Learning Algorithm for Q: Example*

**The rewards associated with each action are defined**



# REINFORCEMENT LEARNING

***Learning Algorithm for Q: Example***

***Learning Task:***

**Learn a set of rules such that the agent from any state si**

**i) reaches its goal**

**ii) does so with least possible cost**

**This set of rules is called an**

**optimal action policy $\pi^*$**

**which maps S → A**

# REINFORCEMENT LEARNING

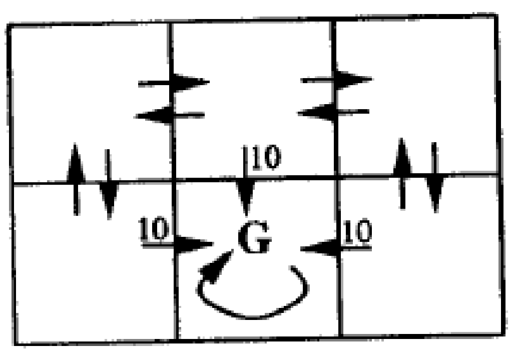
*Learning Algorithm for Q: Example*

**Our aim is to reach the Goal in as few steps as possible, i.e. we want to get the big reward as early as possible**

**In other words, in each state we try to go to another state such that the discounted sum of rewards is maximized**

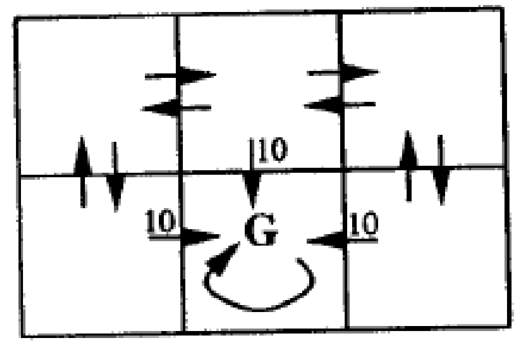
$$r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots$$

**where $0 \le \gamma \le 1$**



# REINFORCEMENT LEARNING

***Learning Algorithm for Q: Example***




**The summation**

$$r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots$$

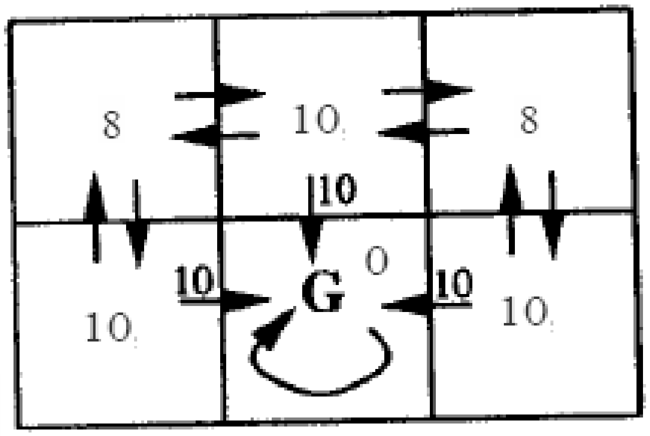
**is called value function for a state under policy $\pi$**

$$V^{\pi}(s) \equiv r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots$$

# REINFORCEMENT LEARNING

## *Learning Algorithm for Q: Example*

**If all the V*(s) values are known, we can formulate an optimal policy π* by choosing that action in any state s, which maximizes**




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

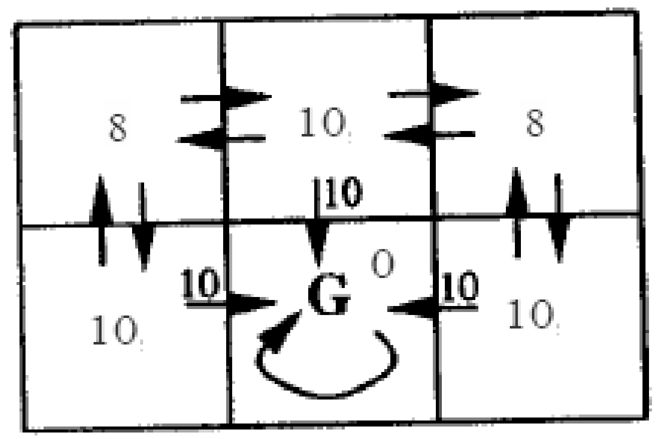
**In other words we go to a neighboring state with the highest V value (+ reward of going there)**

# REINFORCEMENT LEARNING

### *Learning Algorithm for Q: Example*

**The agent would have a look-up table in which the V*(s) values of all the states are present**

**With the help of this table, it can formulate a policy of going to the goal**



# REINFORCEMENT LEARNING

*Learning Algorithm for Q: Example*

**We have another function called Q function defined as**

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

**It can be shown that**

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')$$

**Hence, the optimal policy can be** $\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$

# <u>REINFORCEMENT LEARNING</u>

## Chapter 13 of T. Mitchell