Ayudantía 11

Reinforcement Learning

Daniel Florea

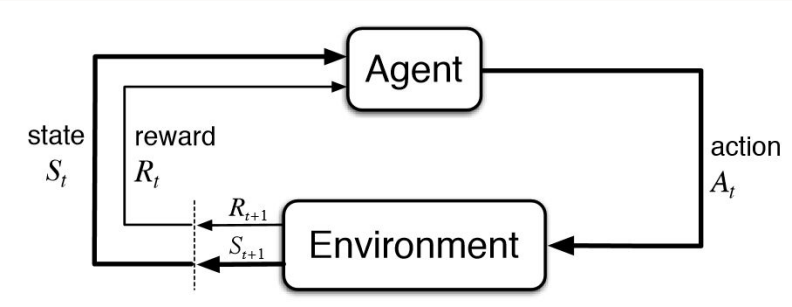
17 de noviembre 2023

¿En qué consiste?

"Consiste en el desarrollo de una política de comportamiento para un agente, que le permite el mapeo desde situaciones hacia acciones para maximizar una señal de recompensa."

Se centra en la interacción de un agente y su entorno

El agente ejecuta acciones dentro de un determinado entorno y este le responde a las acciones mediante una nuevo estado y una recompensa asociada a la calidad de esa acción.



https://miro.medium.com/v2/1*7cuAqjO97x1H_sBIeAVVZg.png

Markov Decision Process

- Trabajaremos los problemas tratandolos como un **Markov Decision Process**.

  Un proceso de decisión de Markov supone la hipótesis Markoviana, la cual afirma que un estado siguiente del entorno solamente depende del estado anterior en el tiempo y la acción tomada.

- Un MDP está compuesto por:

  1. Un set finito de estados : $s_1, ..., s_N$
  2. Un set de acciones : $a_1, ..., a_A$
  3. Un set de recompensas : $r_{1,1}, ..., r_{1,A}, ..., r_{N,1}, ..., r_{N,A}$
  4. Un set de probabilidades de transición entre estados:

$$P_{ij}^{k} = P(s_j | s_i, a_k)$$

Markov Decision Process

SI es un MDP

Sacar una carta de un mazo durante una partida de blackjack

- La probabilidad de sacar una carta depende de todas las cartas extraídas anteriormente
- Las cartas se encuentran dadas vueltas, son todas visibles en un estado s_t

NO es un MDP

Sacar una carta de un mazo durante una partida de poker

- La probabilidad de sacar una carta depende de todas las cartas extraídas anteriormente
- Las cartas extraídas anteriormente no son visibles en un estado s_t

Política de comportamiento (π)

- El objetivo del agente es encontrar una política de comportamiento que le indique qué acción tomar en un instante determinado dado un estado del mundo s_t

$$\pi(s_t) = a_t$$

- Diremos que una política π es óptima (π^*) cuando maximiza la recompensa del agente a lo largo del tiempo, la llamada *value function*

$$V^*(s) = \max_{\pi} \; E\{\sum_{t=0}^{\infty} \gamma^t r_t^{\pi}\}$$

Value function

- La *value function* consiste en una estimación del retorno esperado que un agente puede obtener desde un cierto estado si sigue una política π

$$V(s) = E\{\sum_{t=0}^{\infty} \gamma^t r_t^\pi\}$$

Función de valor del agente al iniciar en el estado *s* y seguir la política π

Tasa de descuento para recompensas más lejanas en el tiempo (entre 0 y 1)

Recompensa del agente en el instante *t* al seguir la política π

Value function

Una forma de reescribir la ecuación anterior a nivel de acción es la siguiente:

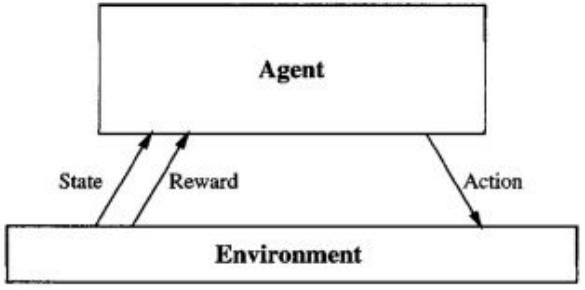
$$V^*(s) = \max_a \{R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') V^*(s')\}$$

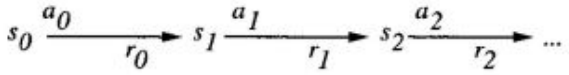
Recompensa del estado actual

Recompensa futura esperada

Con $P(s,a,s')$ la probabilidad de pasar desde un estado s a un estado s' al tomar la acción a

Esta es conocida como la **Ecuación de Bellman**





Goal: Learn to choose actions that maximize

$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$, where $0 \leq \gamma < 1$

Desarrollo de una política π

- Una política óptima sería aquella que escoge siempre las acciones que maximizan el retorno esperado desde un cierto estado.

  En otras palabras, escoge la siguiente acción mediante:

$$V^*(s) = arg\max_a \{R(s,a) + \gamma \sum_{s' \in S} P(s,a,s')V^*(s')\}$$

¿Cómo estimamos el retorno esperado sin recorrer todo el problema recursivamente?

Value iteration

Buscamos la función de valor óptima resolviendo la ecuación de Bellman

Value iteration

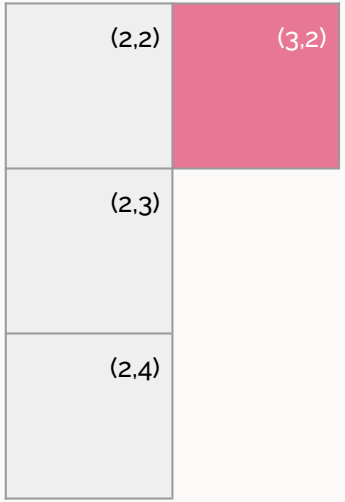
```
Initialize V(s) arbitrarily
loop until policy good enough
      loop for s ∈ S
            loop for a ∈ A
                  Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
            end loop
            V̂(s) := max_a Q(s,a)
      end loop
end loop
return{V̂(s)}
```

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
      loop for s ∈ S
            loop for a ∈ A
                  Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
            end loop
            V̂(s) := max_a Q(s,a)
      end loop
end loop
return{V̂(s)}
```

Value iteration

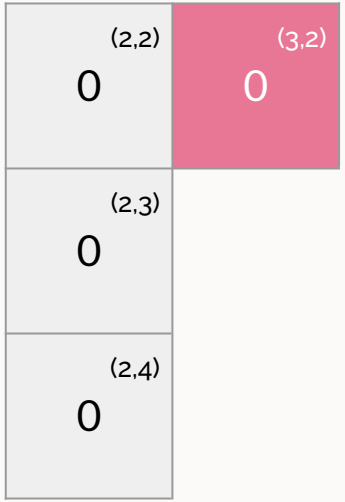
```
Initialize V(s) arbitrarily
loop until policy good enough
      loop for s ∈ S
            loop for a ∈ A
                  Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
            end loop
            V̂(s) := max_a Q(s,a)
      end loop
end loop
return{V̂(s)}
```



$$\gamma = 0.9$$

Value iteration

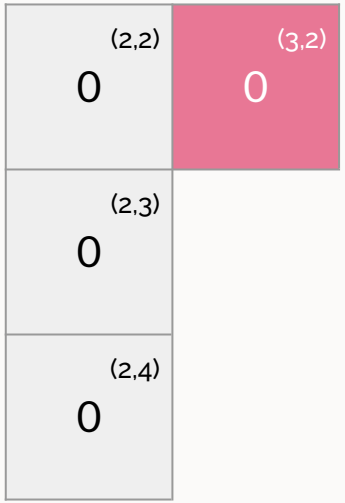
```
Initialize V(s) arbitrarily
loop until policy good enough
      loop for s ∈ S
            loop for a ∈ A
                  Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
            end loop
            V̂(s) := max_a Q(s,a)
      end loop
end loop
return{V̂(s)}
```



$\gamma = 0.9$

Value iteration

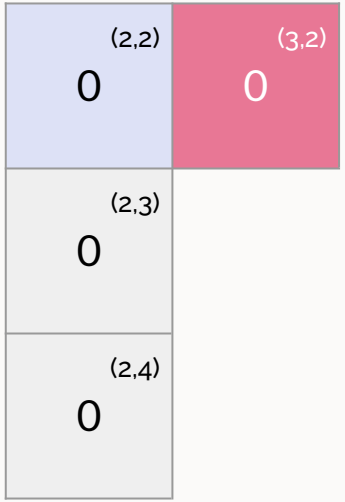
```
Initialize V(s) arbitrarily
loop until policy good enough
      loop for s ∈ S
            loop for a ∈ A
                  Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
            end loop
            V̂(s) := max_a Q(s,a)
      end loop
end loop
return{V̂(s)}
```



$\gamma = 0.9$

Value iteration

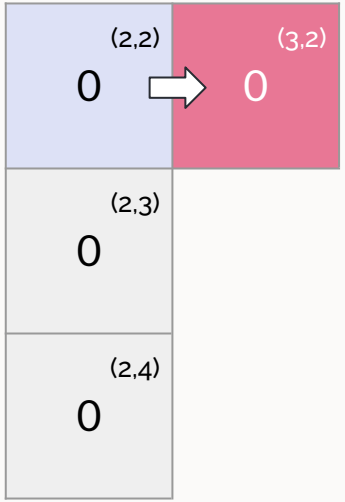
```
Initialize V(s) arbitrarily
loop until policy good enough
    loop for s ∈ S
        loop for a ∈ A
            Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
        end loop
        V̂(s) := max_a Q(s,a)
    end loop
end loop
return{V̂(s)}
```



$\gamma = 0.9$

Value iteration

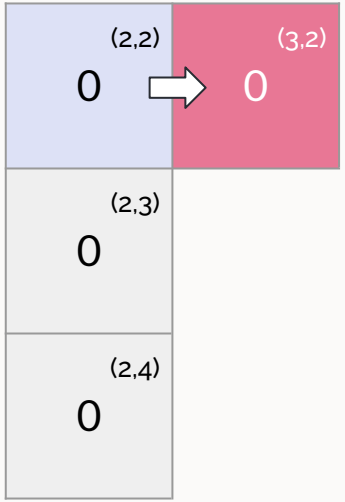
```
Initialize V(s) arbitrarily
loop until policy good enough
     loop for s ∈ S
          loop for a ∈ A
               Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
          end loop
          V̂(s) := max_a Q(s,a)
     end loop
end loop
return{V̂(s)}
```



$\gamma = 0.9$

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
     loop for s ∈ S
          loop for a ∈ A
               Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
          end loop
          V̂(s) := max_a Q(s,a)
     end loop
end loop
return{V̂(s)}
```

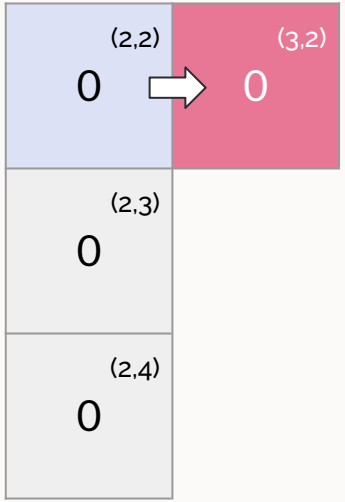


$\gamma = 0.9$

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
     loop for s ∈ S
          loop for a ∈ A
               Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
```

$$Q([2,2],r) = R([2,2],r) + \gamma \cdot T([2,2],r,[3,2]) \cdot V_0([3,2])$$

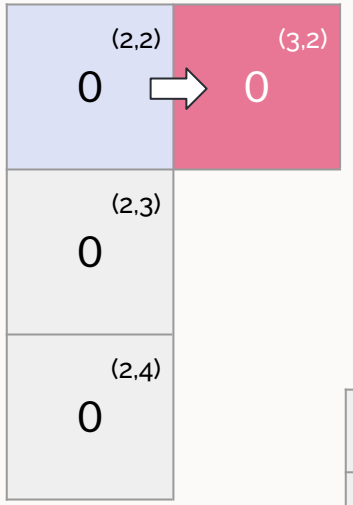


$$\gamma = 0.9$$

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
      loop for s ∈ S
            loop for a ∈ A
                  Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
```

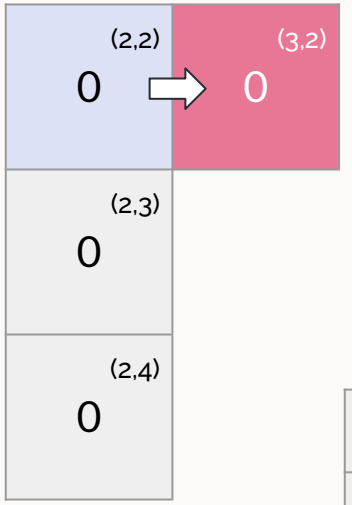
$$
\begin{aligned}
Q([2,2],r) &= R([2,2],r) + \gamma \cdot T([2,2],r,[3,2]) \cdot V_0([3,2]) \\
&= 1 + 0{,}9 \cdot 0{,}25 \cdot 0 \\
&= 1
\end{aligned}
$$



Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
      loop for s ∈ S
            loop for a ∈ A
                  Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
            end loop
```

$$Q([2,2],r) = 1$$
$$Q([2,2],l) = 0$$
$$Q([2,2],u) = 0$$
$$Q([2,2],d) = 0$$



$\gamma = 0.9$

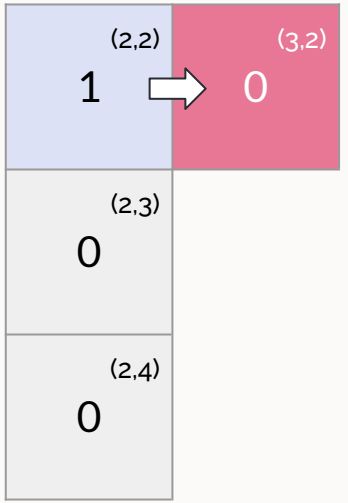
t = 0

0	0
0	
0	

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
      loop for s ∈ S
            loop for a ∈ A
                  Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
            end loop
            V̂(s) := max_a Q(s,a)
```

$$Q([2,2],r) = 1$$
$$Q([2,2],l) = 0$$
$$Q([2,2],u) = 0$$
$$Q([2,2],d) = 0$$



$$\gamma = 0.9$$

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
    loop for s ∈ S
        loop for a ∈ A
            Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
        end loop
        V̂(s) := max_a Q(s,a)
```

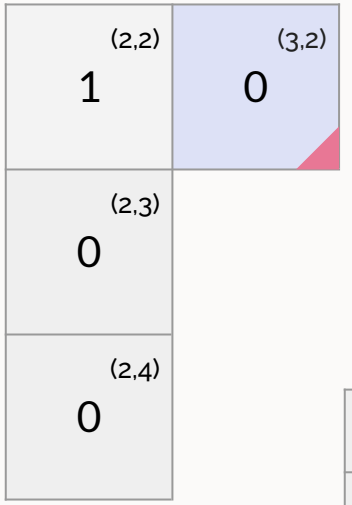
(2,2) 1	(3,2) 0
(2,3) 0	
(2,4) 0	

$$\gamma = 0.9$$

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
    loop for s ∈ S
        loop for a ∈ A
            Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
        end loop
        V̂(s) := max_a Q(s,a)
```

$$Q([3,2],r) = 1$$
$$Q([3,2],l) = 0$$
$$Q([3,2],u) = 1$$
$$Q([3,2],d) = 1$$



$\gamma = 0.9$

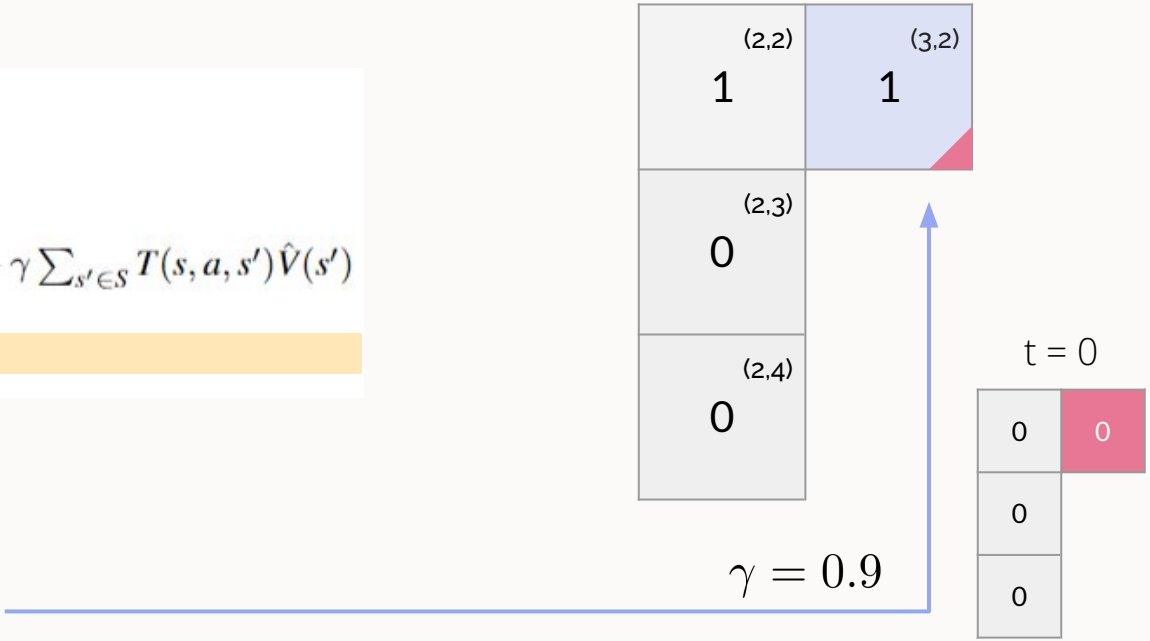
t = 0

0	0
0	
0	

Value iteration

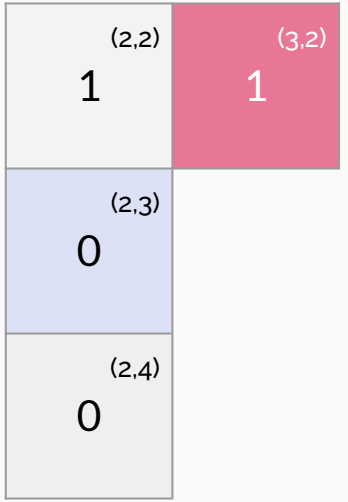
```
Initialize V(s) arbitrarily
loop until policy good enough
      loop for s ∈ S
            loop for a ∈ A
                  Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
            end loop
            V̂(s) := max_a Q(s,a)
```

$$Q([3,2],r) = 1$$
$$Q([3,2],l) = 0$$
$$Q([3,2],u) = 1$$
$$Q([3,2],d) = 1$$



Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
    loop for s ∈ S
        loop for a ∈ A
            Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
        end loop
        V̂(s) := max_a Q(s,a)
```

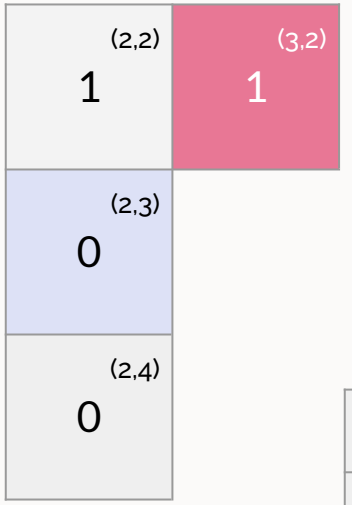


$\gamma = 0.9$

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
    loop for s ∈ S
        loop for a ∈ A
            Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
        end loop
        V̂(s) := max_a Q(s,a)
```

$$Q([2,3],r) = 0$$
$$Q([2,3],l) = 0$$
$$Q([2,3],u) = 0$$
$$Q([2,3],d) = 0$$



$\gamma = 0.9$

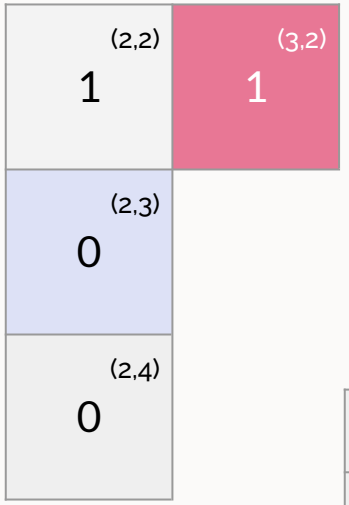
t = 0

0	0
0	
0	

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
      loop for s ∈ S
            loop for a ∈ A
                  Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
            end loop
            V̂(s) := max_a Q(s,a)
```

$$Q([2,3],r) = 0$$
$$Q([2,3],l) = 0$$
$$Q([2,3],u) = 0$$
$$Q([2,3],d) = 0$$



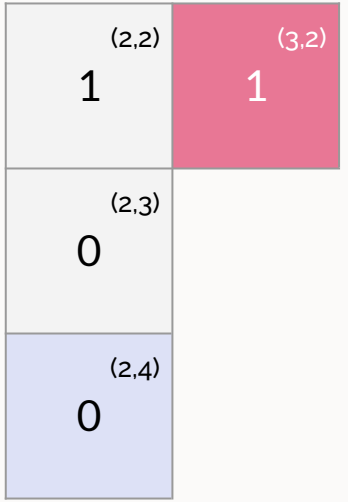
$\gamma = 0.9$

t = 0

0	0
0	
0	

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
    loop for s ∈ S
        loop for a ∈ A
            Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
        end loop
        V̂(s) := max_a Q(s,a)
```

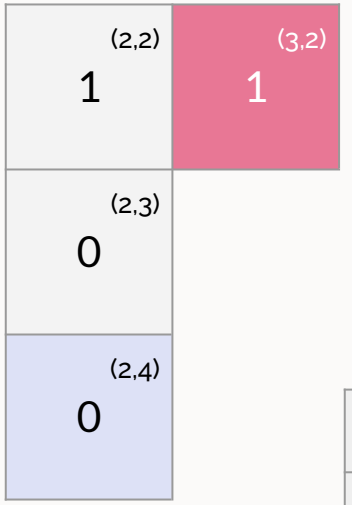


$\gamma = 0.9$

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
    loop for s ∈ S
        loop for a ∈ A
            Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
        end loop
        V̂(s) := max_a Q(s,a)
```

$$Q([2,4],r) = 0$$
$$Q([2,4],l) = 0$$
$$Q([2,4],u) = 0$$
$$Q([2,4],d) = 0$$



$\gamma = 0.9$

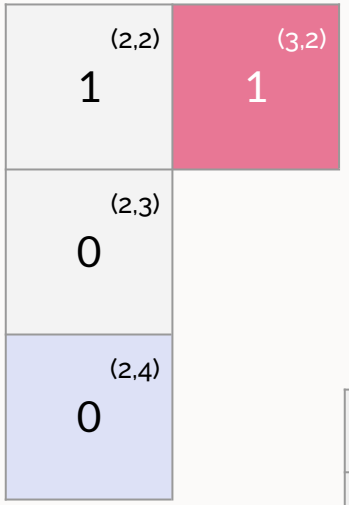
t = 0

0	0
0	
0	

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
      loop for s ∈ S
            loop for a ∈ A
                  Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
            end loop
            V̂(s) := max_a Q(s,a)
```

$$Q([2,4],r) = 0$$
$$Q([2,4],l) = 0$$
$$Q([2,4],u) = 0$$
$$Q([2,4],d) = 0$$



$\gamma = 0.9$

t = 0

0	0
0	
0	

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
      loop for s ∈ S
            loop for a ∈ A
                  Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
            end loop
            V̂(s) := max_a Q(s,a)
      end loop
end loop
return{V̂(s)}
```

t = 1

(2,2) 1	(3,2) 1
(2,3) 0	
(2,4) 0	

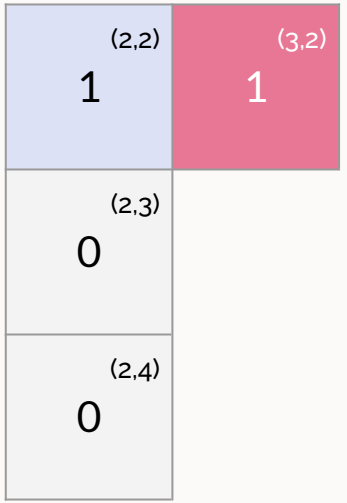
$\gamma = 0.9$

t = 0

0	0
0	
0	

Value iteration

```
Initialize V(s) arbitrarily
loop until policy good enough
    loop for s ∈ S
        loop for a ∈ A
            Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s') V̂(s')
        end loop
        V̂(s) := max_a Q(s,a)
    end loop
end loop
return{V̂(s)}
```

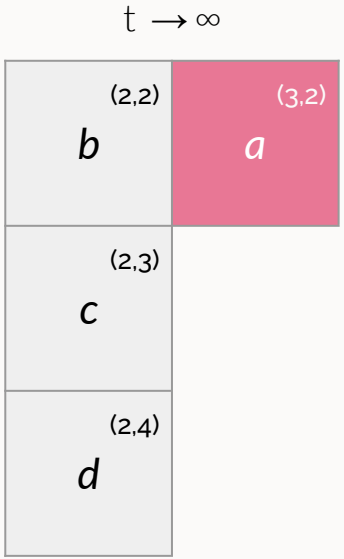


$$\gamma = 0.9$$

Value iteration

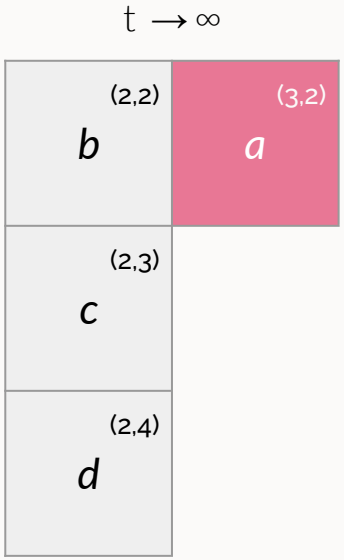
(...)

Value iteration



¿ Qué relación existirá entre *a*, *b*, *c* y *d*?

Value iteration

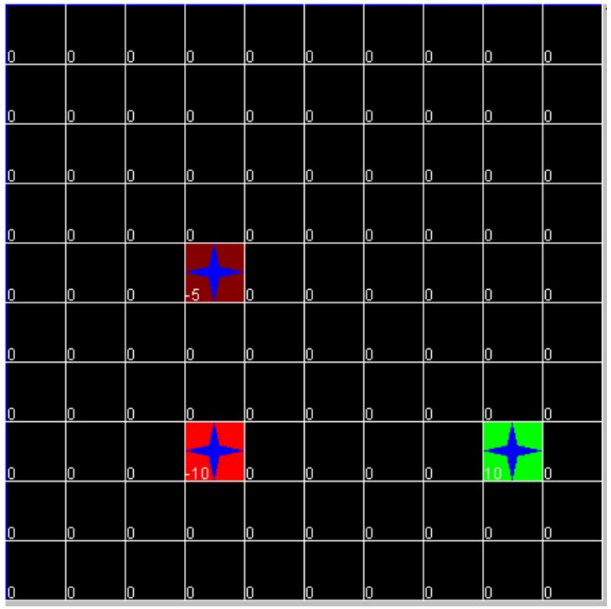


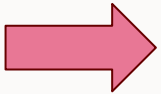
¿ Qué relación existirá entre *a*, *b*, *c* y *d*?

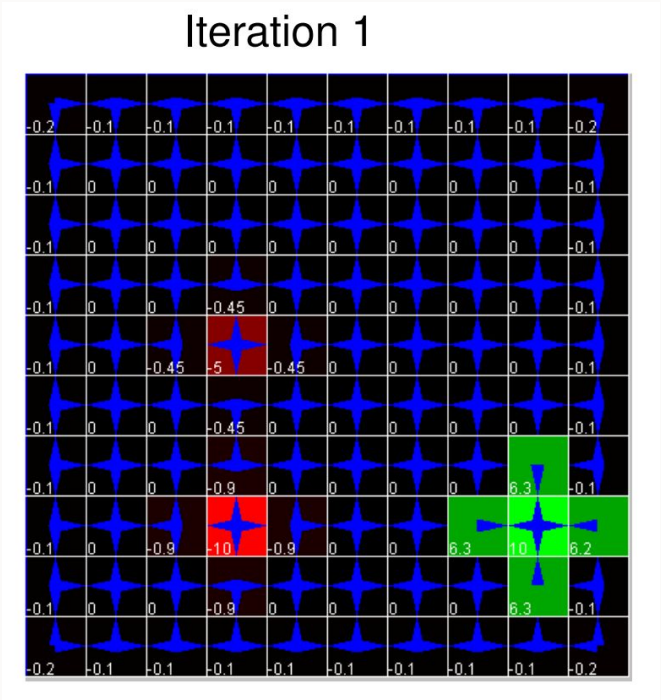
Para este caso en particular

$$a > b > c > d$$

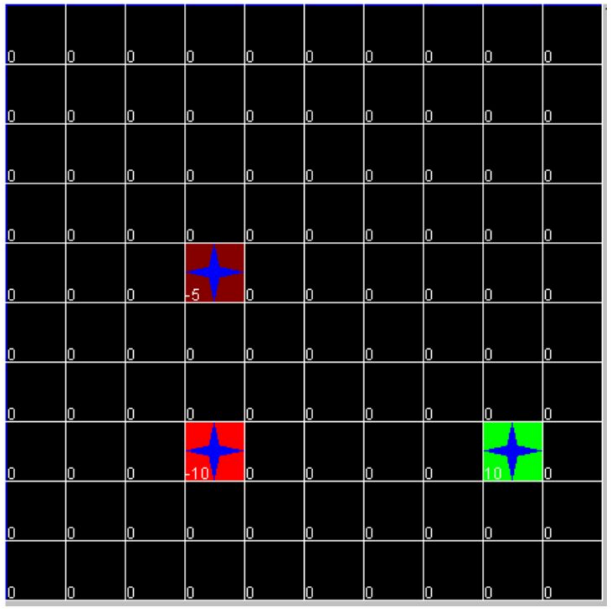
Value iteration



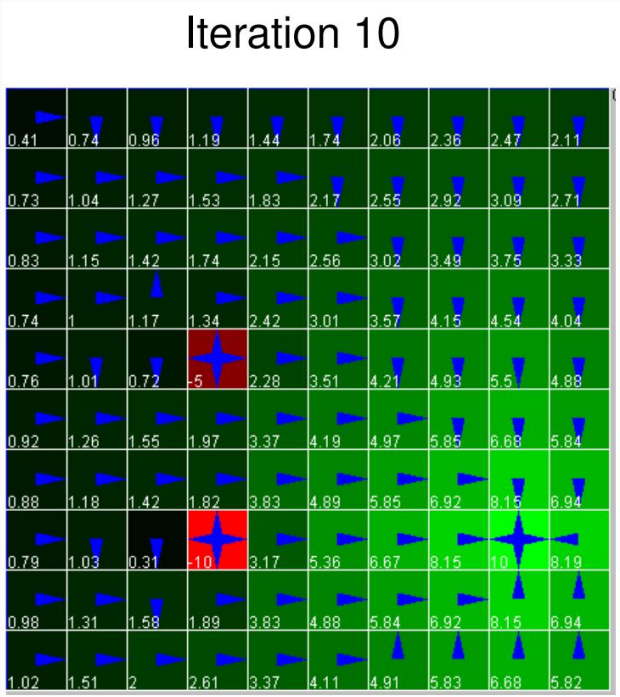




Value iteration



(...)



Value iteration

En resumen:

- Value iteration busca actualizar la *value function* de cada estado, **utilizando la aproximación del instante de tiempo anterior**.
- La acción a ser ejecutada es definida como aquella que me otorgará el **mayor retorno esperado** en el futuro, como vimos anteriormente:

$$arg\max_{a} \{R(s,a) + \gamma \sum_{s' \in S} P(s,a,s')V^*(s')\}$$

- La ejecución del algoritmo se acaba cuando los valores de cada estado convergen en un valor fijo.
- El algoritmo nos garantiza **optimalidad** en la política encontrada :)

Policy iteration

Actualizamos la política de comportamiento en cada iteración

Policy iteration

Choose an arbitray policy π'

Loop

$\pi := \pi'$

Compute value function of policy π:

#solve linear equations

$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$

Improve the policy at each state

$\pi'(s) := \arg\max_a \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s') \right)$

until $\pi = \pi'$

Policy iteration

```
Choose an arbitray policy π'
Loop
        π := π'
        Compute value function of policy π:
                #solve linear equations
                V_π(s) := R(s, π(s)) + γ Σ_{s'∈S} T(s, π(s), s') V_π(s')
        Improve the policy at each state
                π'(s) := arg max_a (R(s, a) + γ Σ_{s'∈S} T(s, a, s') V_π(s'))
until π = π'
```

(2,2)	(3,2)
(2,3)	
(2,4)	

Policy iteration

Choose an arbitray policy π'

Loop

$\pi := \pi'$

Compute value function of policy π:

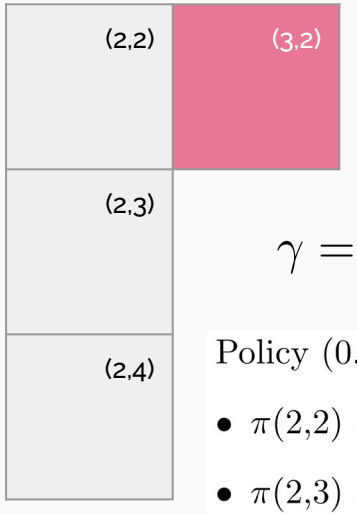
#solve linear equations

$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$

Improve the policy at each state

$\pi'(s) := \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s') \right)$

until $\pi = \pi'$



$\gamma = 0.9$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

Choose an arbitray policy π'

Loop

$\pi := \pi'$

Compute value function of policy π:

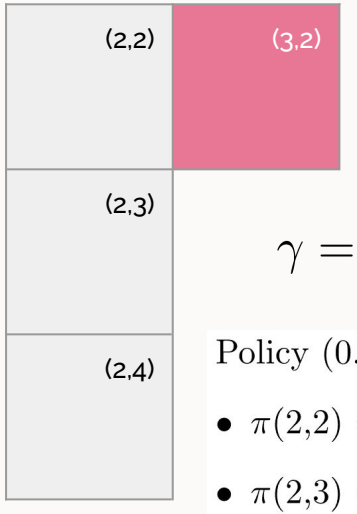
#solve linear equations

$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$

Improve the policy at each state

$\pi'(s) := \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s') \right)$

until $\pi = \pi'$



$\gamma = 0.9$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

Choose an arbitray policy π'

Loop

$\pi := \pi'$

Compute value function of policy π:

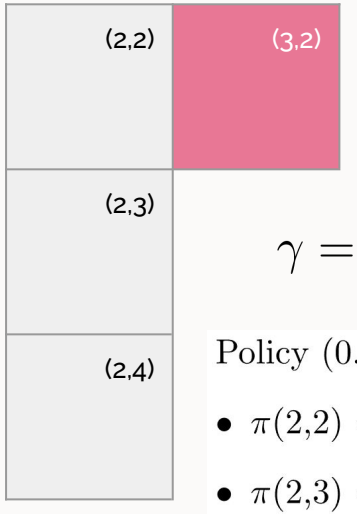
#solve linear equations

$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$

Improve the policy at each state

$\pi'(s) := \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s') \right)$

until $\pi = \pi'$



$\gamma = 0.9$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

Choose an arbitray policy π'

Loop

$\pi := \pi'$

Compute value function of policy π:

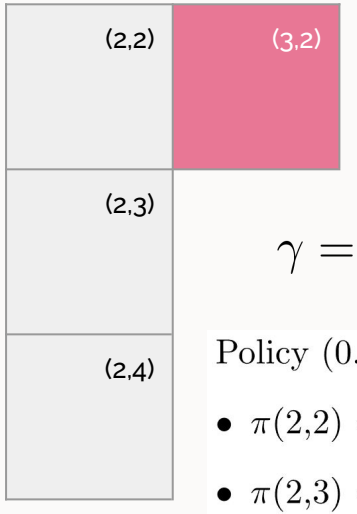
#solve linear equations

$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$

Improve the policy at each state

$\pi'(s) := \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s') \right)$

until $\pi = \pi'$



$\gamma = 0.9$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

Choose an arbitray policy π'

Loop

$\pi := \pi'$

Compute value function of policy π:

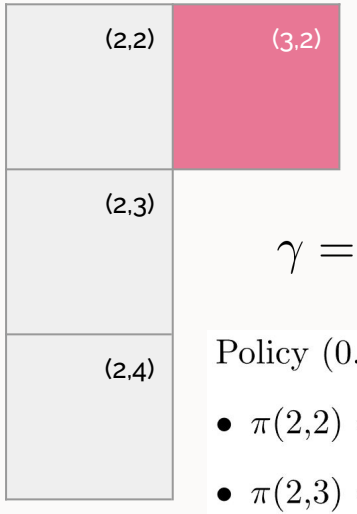
#solve linear equations

$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$

Improve the policy at each state

$\pi'(s) := \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s') \right)$

until $\pi = \pi'$



$\gamma = 0.9$

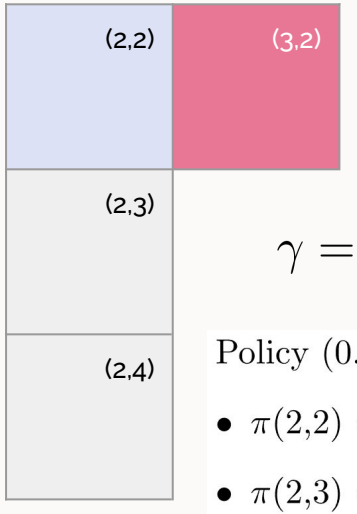
Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

$$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$$

$$\begin{aligned} V([2,2]) = R([2,2], \pi) + \gamma(\\ & T([2,2], \pi, [3,2]) \cdot V([3,2]) + \\ & T([2,2], \pi, [2,2]) \cdot V([2,2]) + \\ & T([2,2], \pi, [2,3]) \cdot V([2,3]) \\ &) \end{aligned}$$



$\gamma = 0.9$

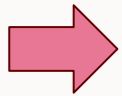
Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

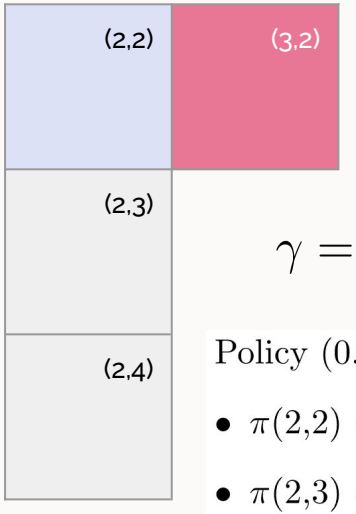
Policy iteration

$$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$$

$$V([2,2]) = R([2,2], \pi) + \gamma(\\ T([2,2], \pi, [3,2]) \cdot V([3,2]) + \\ T([2,2], \pi, [2,2]) \cdot V([2,2]) + \\ T([2,2], \pi, [2,3]) \cdot V([2,3]) \\ )$$



$$V([2,2]) = 1 + \gamma(\\ 0{,}7 \cdot V([3,2]) + \\ 0{,}2 \cdot V([2,2]) + \\ 0{,}1 \cdot V([2,3]) \\ )$$



$\gamma = 0.9$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

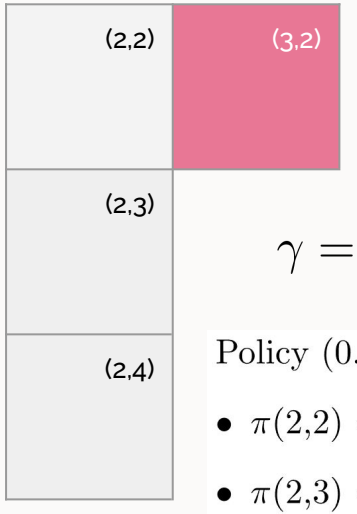
$$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$$

$$V([2,2]) = 1 + \gamma(0{,}7 \cdot V([3,2]) + 0{,}2 \cdot V([2,2]) + 0{,}1 \cdot V([2,3]))$$

$$V([3,2]) = 1 + \gamma(1 \cdot ([3,2]))$$

$$V([2,3]) = 0 + \gamma(0{,}8 \cdot V([2,3]) + 0{,}1 \cdot V([2,2]) + 0{,}1 \cdot V([2,4]))$$

$$V([2,4]) = 0 + \gamma(0{,}9 \cdot V([2,4]) + 0{,}1 \cdot V([2,3]))$$



$\gamma = 0.9$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

$$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$$

$$V([2,2]) = 1 + \gamma(0{,}7 \cdot V([3,2]) + 0{,}2 \cdot V([2,2]) + 0{,}1 \cdot V([2,3]))$$

$$V([2,3]) = 0 + \gamma(0{,}8 \cdot V([2,3]) + 0{,}1 \cdot V([2,2]) + 0{,}1 \cdot V([2,4]))$$

$$V([2,4]) = 0 + \gamma(0{,}9 \cdot V([2,4]) + 0{,}1 \cdot V([2,3]))$$

$$V([3,2]) = 1 + \gamma(1 \cdot ([3,2]))$$

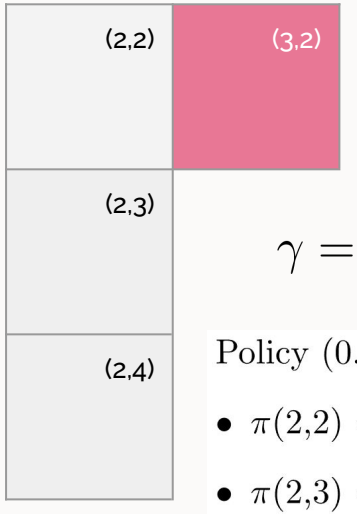
solve

$$a = 1 + 0.9\,(0.2\,a + 0.1\,b + 0.7\,d)$$

$$b = 0.9\,(0.1\,a + 0.8\,b + 0.1\,c)$$

$$c = 0.9\,(0.1\,b + 0.9\,c)$$

$$d = 1 + 0.9\,d$$



$$\gamma = 0.9$$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

$$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$$

$$V([2,2]) = 9{,}289$$
$$V([2,3]) = 3{,}522$$
$$V([2,4]) = 1{,}668$$
$$V([3,2]) = 10$$

(2,2) 9.289	(3,2) 10
(2,3) 3.522	
(2,4) 1.668	

$\gamma = 0.9$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

Choose an arbitray policy π'

Loop

$\pi := \pi'$

Compute value function of policy π:

#solve linear equations

$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$

Improve the policy at each state

$\pi'(s) := \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s') \right)$

until $\pi = \pi'$

(2,2) 9.289	(3,2) 10
(2,3) 3.522	
(2,4) 1.668	

$\gamma = 0.9$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

Choose an arbitray policy π'

Loop

$\pi := \pi'$

Compute value function of policy π:

#solve linear equations

$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$

Improve the policy at each state

$\pi'(s) := \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s') \right)$

until $\pi = \pi'$

(2,2) 9.289	(3,2) 10
(2,3) 3.522	
(2,4) 1.668	

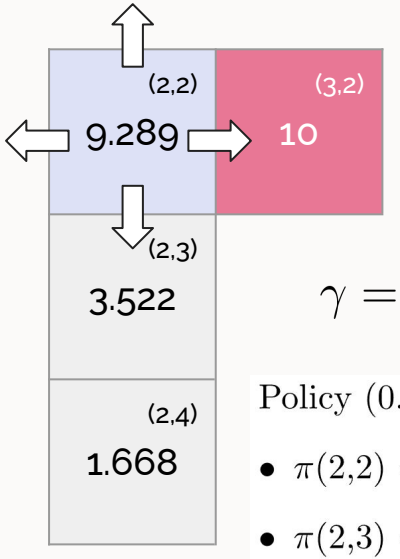
$\gamma = 0.9$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

$$\arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s')\right)$$



$\gamma = 0.9$

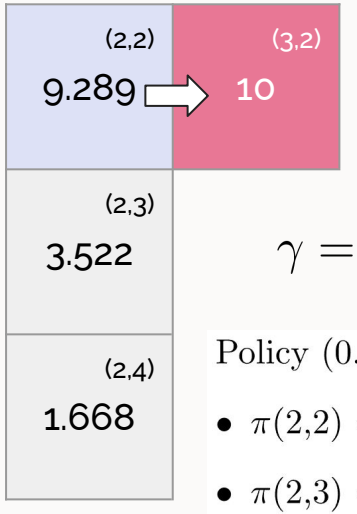
Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

$$\arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s')\right)$$

$$right \rightarrow 1 + 0{,}9 \cdot (0{,}7 \cdot 10 + 0{,}2 \cdot 9{,}289 + 0{,}1 \cdot 3{,}522) = 9{,}289$$
$$left \rightarrow 0 + 0{,}9 \cdot (0{,}8 \cdot 9{,}289 + 0{,}1 \cdot 10 + 0{,}1 \cdot 3{,}522) = 7{,}90506$$
$$up \rightarrow 0 + 0{,}9 \cdot (0{,}8 \cdot 9{,}289 + 0{,}1 \cdot 10 + 0{,}1 \cdot 3{,}522) = 7{,}90506$$
$$down \rightarrow 0 + 0{,}9 \cdot (0{,}7 \cdot 3{,}522 + 0{,}2 \cdot 9{,}289 + 0{,}1 \cdot 10) = 4{,}79088$$



$\gamma = 0.9$

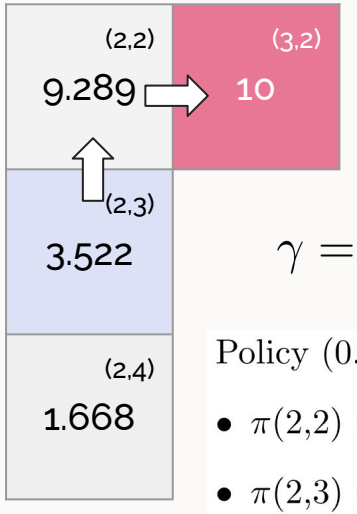
Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

$$\arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s')\right)$$

$$right \rightarrow 0 + 0{,}9 \cdot (0{,}8 \cdot 3{,}522 + 0{,}1 \cdot 9{,}289 + 0{,}1 \cdot 1{,}668) = 3{,}52197$$
$$left \rightarrow 0 + 0{,}9 \cdot (0{,}8 \cdot 3{,}522 + 0{,}1 \cdot 9{,}289 + 0{,}1 \cdot 1{,}668) = 3{,}52197$$
$$up \rightarrow 1 + 0{,}9 \cdot (0{,}7 \cdot 9{,}289 + 0{,}2 \cdot 3{,}522 + 0{,}1 \cdot 1{,}668) = 7{,}63615$$
$$down \rightarrow 0 + 0{,}9 \cdot (0{,}7 \cdot 1{,}668 + 0{,}2 \cdot 3{,}522 + 0{,}1 \cdot 9{,}289) = 2{,}52081$$



$\gamma = 0.9$

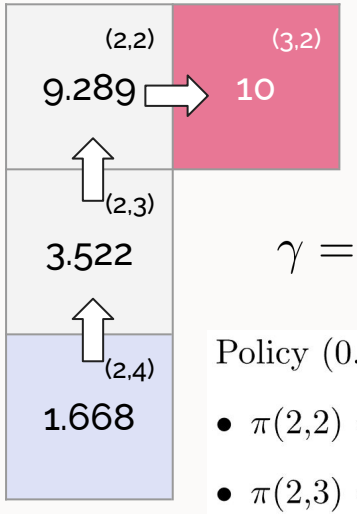
Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

$$\arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s')\right)$$

$$\begin{aligned}
right &\rightarrow 0 + 0{,}9 \cdot (0{,}9 \cdot 1{,}668 + 0{,}1 \cdot 3{,}522) = 1{,}66806 \\
left &\rightarrow 0 + 0{,}9 \cdot (0{,}9 \cdot 1{,}668 + 0{,}1 \cdot 3{,}522) = 1{,}66806 \\
up &\rightarrow 1 + 0{,}9 \cdot (0{,}7 \cdot 3{,}522 + 0{,}3 \cdot 1{,}668) = 3{,}66922 \\
down &\rightarrow 0 + 0{,}9 \cdot (0{,}9 \cdot 1{,}668 + 0{,}1 \cdot 3{,}522) = 1{,}66806
\end{aligned}$$



$\gamma = 0.9$

Policy (0.3 noise):

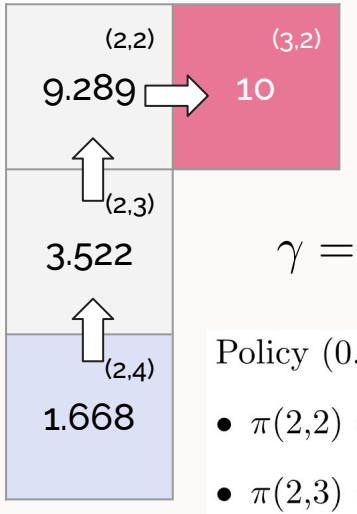
- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

t = 1

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Up}$
- $\pi(2,4) = \text{Up}$
- $\pi(3,2) = \text{Stay}$

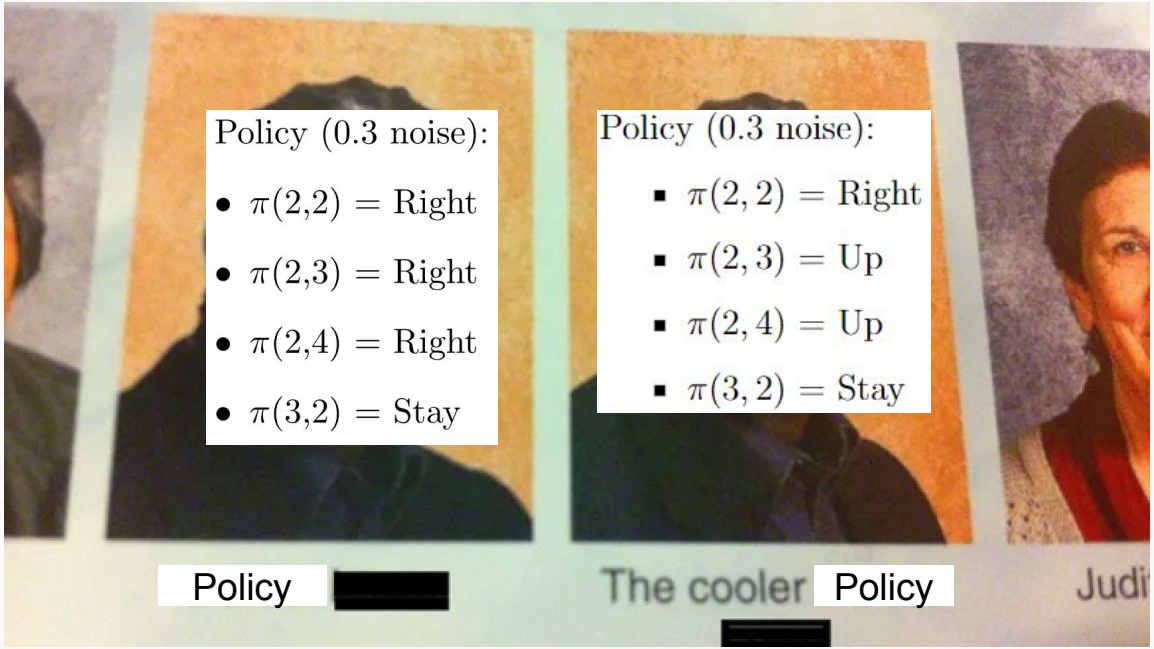


$\gamma = 0.9$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Right}$
- $\pi(2,4) = \text{Right}$
- $\pi(3,2) = \text{Stay}$

Policy iteration



Policy iteration

Choose an arbitray policy π'

Loop

$\pi := \pi'$

Compute value function of policy π:

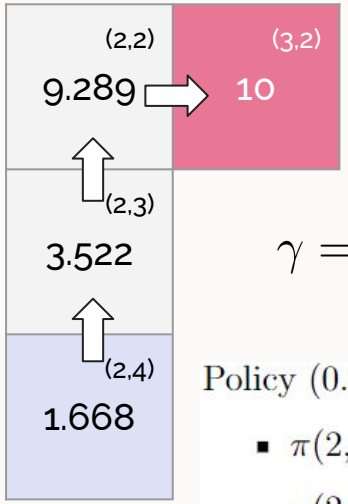
#solve linear equations

$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$

Improve the policy at each state

$\pi'(s) := \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s') \right)$

until $\pi = \pi'$



$\gamma = 0.9$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Up}$
- $\pi(2,4) = \text{Up}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

Choose an arbitray policy π'

Loop

$\pi := \pi'$

Compute value function of policy π:

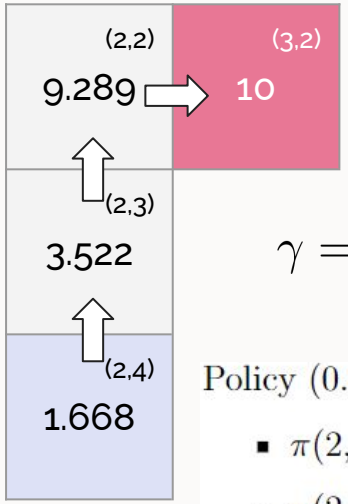
#solve linear equations

$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$

Improve the policy at each state

$\pi'(s) := \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s') \right)$

until $\pi = \pi'$



$\gamma = 0.9$

Policy (0.3 noise):

- $\pi(2,2) = \text{Right}$
- $\pi(2,3) = \text{Up}$
- $\pi(2,4) = \text{Up}$
- $\pi(3,2) = \text{Stay}$

Policy iteration

(...)

Policy iteration

En resumen:

- Policy iteration busca actualizar la *política de comportamiento* de cada estado, **utilizando la aproximación del instante de tiempo anterior**.
- La acción a ser ejecutada es definida como aquella que me otorgará el **mayor retorno esperado** en el futuro, como vimos anteriormente:

$$arg\max_{a}(R(s,\pi(s)) + \gamma \sum_{s' \in S} T(s,\pi(s),s')V_{\pi}(s'))$$

- La ejecución del algoritmo se acaba cuando los valores de cada estado convergen en un valor fijo y la política nueva es idéntica a la anterior.
- El algoritmo nos garantiza **optimalidad** en la política encontrada :)

Q-Learning

Estimamos la calidad de cada par estado-acción en base a experiencia

Q-Learning

Initialize $Q(s, a)$ arbitrarily

Repeat (for each episode):

  Initialize s

  Repeat (for each step of episode):

    Choose a from s using policy derived from Q

    Take action a, observe r, s'

    Update

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

    $s \leftarrow s'$;

  Until s is terminal

Q-Learning

	Izquierda	Derecha
Q-Value	0	0

Q-Learning

	Izquierda	Derecha
Q-Value	-5	0

-5

Q-Learning

	Izquierda	Derecha
Q-Value	-5	0

?

Q-Learning

	Izquierda	Derecha
Q-Value	-5	+5

+5

Q-Learning

	Izquierda	Derecha
Q-Value	-5	+5

Q-Learning

Los principales componentes de Q-Learning son:

- Una **tabla de estados** (Q-Table) **Q**: En ella almacenamos la calidad estimada de cada par estado-acción

	Izquierda	Derecha
Q-Value	0	0

Q-Learning

Los principales componentes de Q-Learning son:

- Una **tabla de estados** (Q-Table) **Q**: En ella almacenamos la calidad estimada de cada par estado-acción

	Izquierda	Derecha
Q-Value	0	0

- Una **tasa de exploración ϵ**: Qué tanto queremos que el agente explore/explote la política actual

Con probabilidad **ϵ** elegimos una acción random, si no, elegimos la que tenga mejor calidad asociada en Q para el estado actual s_t

Q-Learning

Los principales componentes de Q-Learning son:

- Una **tabla de estados** (Q-Table) **Q**: En ella almacenamos la calidad estimada de cada par estado-acción

	Izquierda	Derecha
Q-Value	0	0

- Una **tasa de exploración ϵ**: Qué tanto queremos que el agente explore/explote la política actual

> Con probabilidad **ϵ** elegimos una acción random, si no, elegimos la que tenga mejor calidad asociada en Q para el estado actual s_t

- Una **tasa de aprendizaje α**: Qué tanto consideramos nueva información a nuestro conocimiento

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmax Q[x_{i+1}])$$

Q-Learning

Los principales componentes de Q-Learning son:

- Una **tabla de estados** (Q-Table) **Q**: En ella almacenamos la calidad estimada de cada par estado-acción

	Izquierda	Derecha
Q-Value	0	0

- Una **tasa de exploración ϵ**: Qué tanto queremos que el agente explore/explote la política actual

> Con probabilidad **ϵ** elegimos una acción random, si no, elegimos la que tenga mejor calidad asociada en Q para el estado actual s_t

- Una **tasa de aprendizaje α**: Qué tanto consideramos nueva información a nuestro conocimiento

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmaxQ[x_{i+1}])$$

- Un conjunto de **estados del entorno S** = [s_1, s_2, ..., s_n]

Q-Learning

Los principales componentes de Q-Learning son:

- Una **tabla de estados** (Q-Table) **Q**: En ella almacenamos la calidad estimada de cada par estado-acción

	Izquierda	Derecha
Q-Value	0	0

- Una **tasa de exploración ϵ**: Qué tanto queremos que el agente explore/explote la política actual

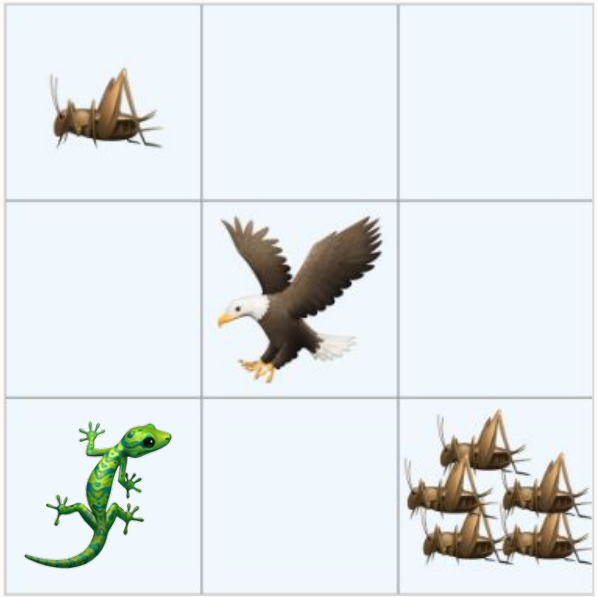
> Con probabilidad **ϵ** elegimos una acción random, si no, elegimos la que tenga mejor calidad asociada en Q para el estado actual s_t

- Una **tasa de aprendizaje α**: Qué tanto consideramos nueva información a nuestro conocimiento

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmaxQ[x_{i+1}])$$

- Un conjunto de **estados del entorno S** = $[s_1, s_2, ..., s_n]$
- Un conjunto de **acciones** entre cuales elegir **A** = $[a_1, a_2, ..., a_k]$

Q-Learning: Un ejemplo práctico



Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

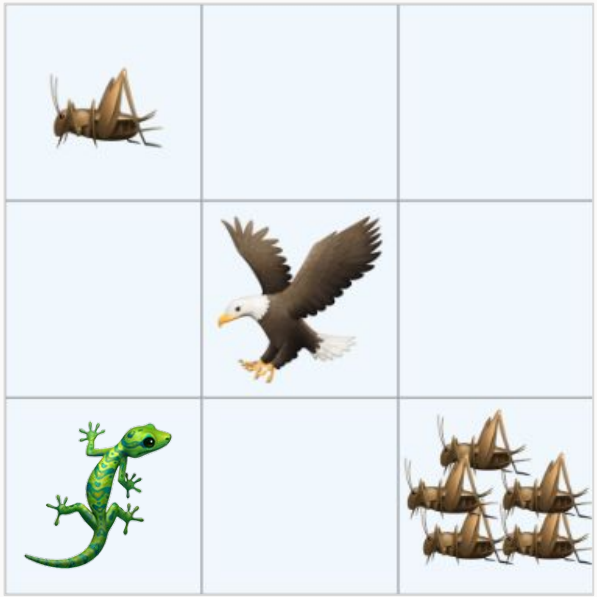
$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$

El juego acaba si el lagarto cae en la casilla del pájaro o bien come los 5 grillos.

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$

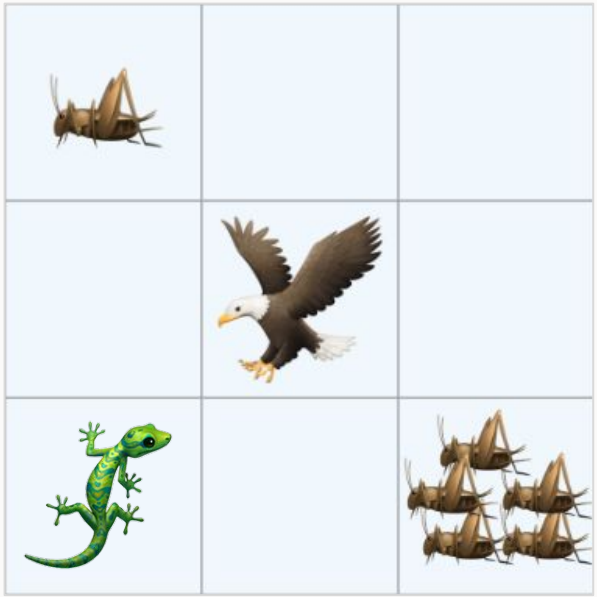


S_t	Arriba	Abajo	Izq.	Der.
(0,0)	-1	0	-1	0
(0,1)	-1	0	0	0
(0,2)	-1	0	0	-1
(1,0)	0	0	-1	0
(1,1)	0	0	0	0
(1,2)	0	0	0	-1
(2,0)	0	-1	-1	0
(2,1)	0	-1	0	0
(2,2)	0	-1	0	-1

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Con probabilidad ϵ = 0.3 vamos a elegir un movimiento aleatorio.

 = 0.4

0.3 < 0.4, por lo que explotamos

Buscamos el movimiento de mejor valor Q:

S_t	Arriba	Abajo	Izq.	Der.
(2,0)	0	-1	-1	0

Como son idénticos, tomamos uno aleatorio

A_0 = Derecha

Q-Learning:

$R_1 = -1$

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$

Actualizamos Q

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmax Q[x_{i+1}])$$

Q[(2,0), der] = (1 - 0.5) · Q[(2,0), der] + 0.5 · (-1 + 0.9 · argmax Q[(2,1), :])

Q[(2,0), der] = 0.5 · 0 + 0.5 · (-1 + 0.9 · 0)

Q[(2,0), der] = 0.5 · -1

Q[(2,0), der] = -0.5

Q-Table anterior

S_t	↑	↓	←	→
(2,0)	0	-1	-1	0
(2,1)	0	-1	0	0

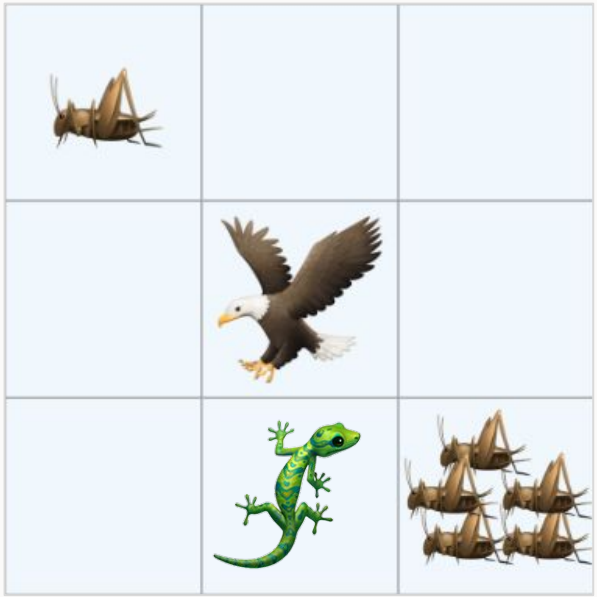
Q-Table actualizada

S_t	↑	↓	←	→
(2,0)	0	-1	-1	-0.5
(2,1)	0	-1	0	0

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$

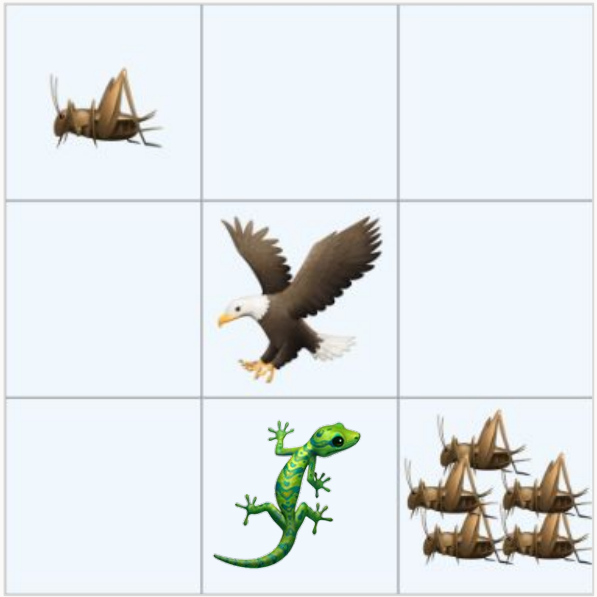


S_t	Arriba	Abajo	Izq.	Der.
(0,0)	-1	0	-1	0
(0,1)	-1	0	0	0
(0,2)	-1	0	0	-1
(1,0)	0	0	-1	0
(1,1)	0	0	0	0
(1,2)	0	0	0	-1
(2,0)	0	-1	-1	-0.5
(2,1)	0	-1	0	0
(2,2)	0	-1	0	-1

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Con probabilidad ϵ = 0.3 vamos a elegir un movimiento aleatorio.

 = 0.7

0.3 < 0.7, por lo que explotamos

Buscamos el movimiento de mejor valor Q:

S_t	Arriba	Abajo	Izq.	Der.
(2,1)	0	-1	0	0

Como son idénticos, tomamos uno aleatorio

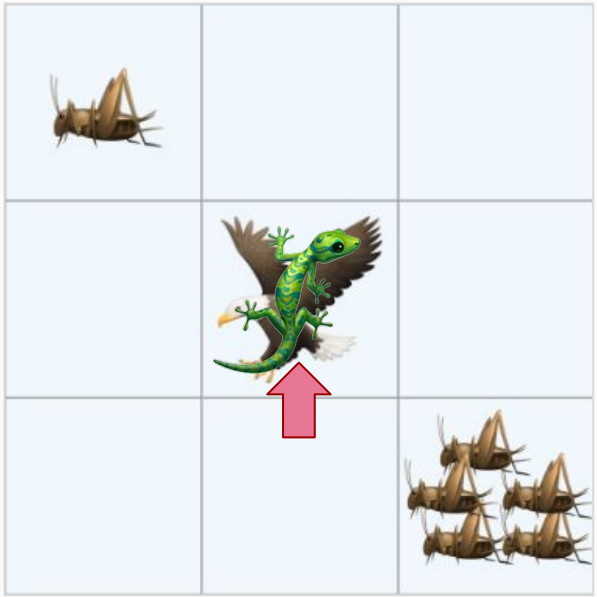
A_1 = Arriba

Q-Learning:

$R_2 = -10$

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Actualizamos Q

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmax Q[x_{i+1}])$$

Q[(2,1), arr] = (1 - 0.5) · Q[(2,1), arr] + 0.5 · (-10 + 0.9 · argmax Q[(1,1), :])

Q[(2,1), arr] = 0.5 · 0 + 0.5 · (-10 + 0.9 · 0)

Q[(2,1), arr] = 0.5 · -10

Q[(2,1), arr] = -5

Q-Table anterior

S_t	↑	↓	←	→
(1,1)	0	0	0	0
(2,1)	0	-1	0	0

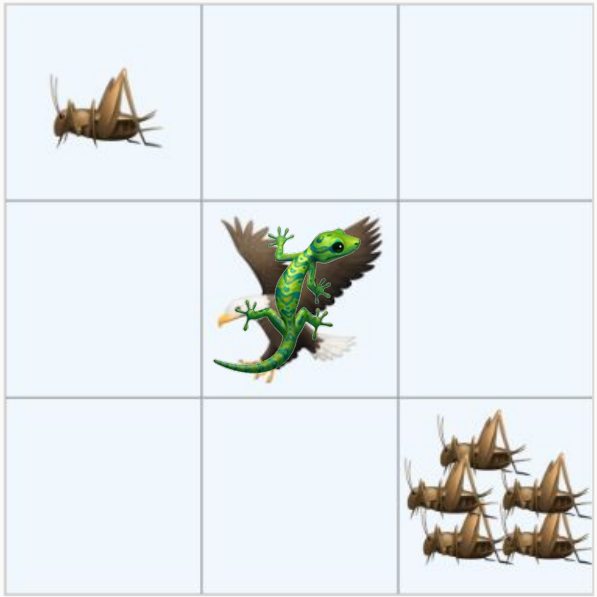
Q-Table actualizada

S_t	↑	↓	←	→
(1,1)	0	0	0	0
(2,1)	-5	-1	0	0

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$

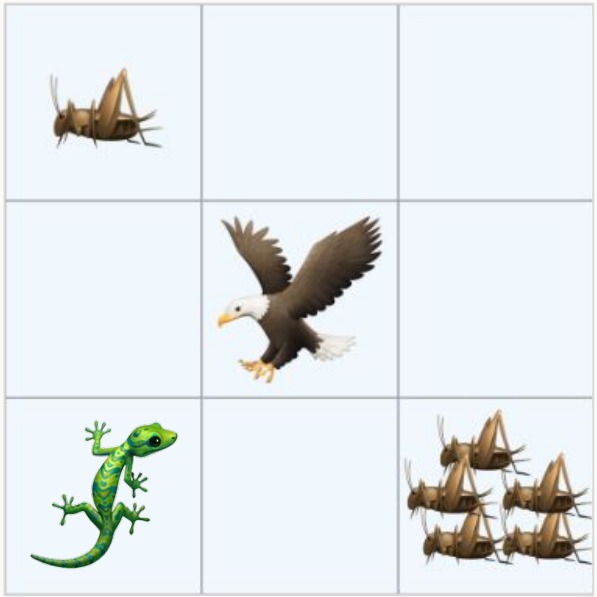


S_t	Arriba	Abajo	Izq.	Der.
(0,0)	-1	0	-1	0
(0,1)	-1	0	0	0
(0,2)	-1	0	0	-1
(1,0)	0	0	-1	0
(1,1)	0	0	0	0
(1,2)	0	0	0	-1
(2,0)	0	-1	-1	-0.5
(2,1)	-5	-1	0	0
(2,2)	0	-1	0	-1

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$

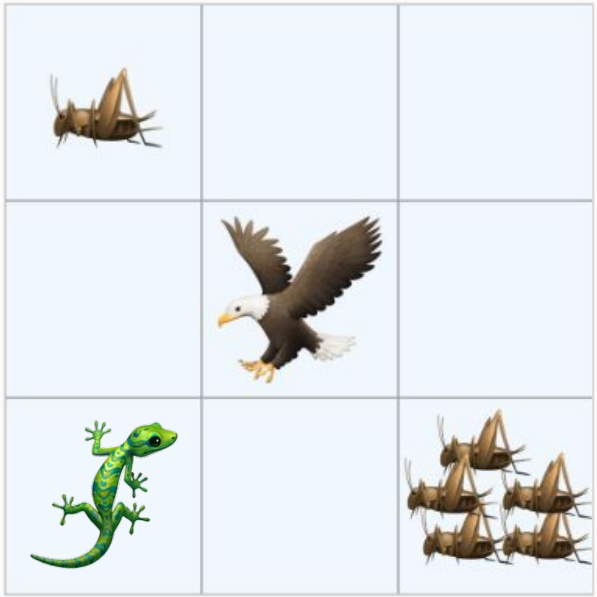


S_t	Arriba	Abajo	Izq.	Der.
(0,0)	-1	0	-1	0
(0,1)	-1	0	0	0
(0,2)	-1	0	0	-1
(1,0)	0	0	-1	0
(1,1)	0	0	0	0
(1,2)	0	0	0	-1
(2,0)	0	-1	-1	-0.5
(2,1)	-5	-1	0	0
(2,2)	0	-1	0	-1

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Con probabilidad ϵ = 0.3 vamos a elegir un movimiento aleatorio.

 = 0.9

0.3 < 0.9, por lo que explotamos

Buscamos el movimiento de mejor valor Q:

S_t	Arriba	Abajo	Izq.	Der.
(2,0)	0	-1	-1	-0.5

En este caso, corresponde a arriba (no podemos ir abajo o izq)

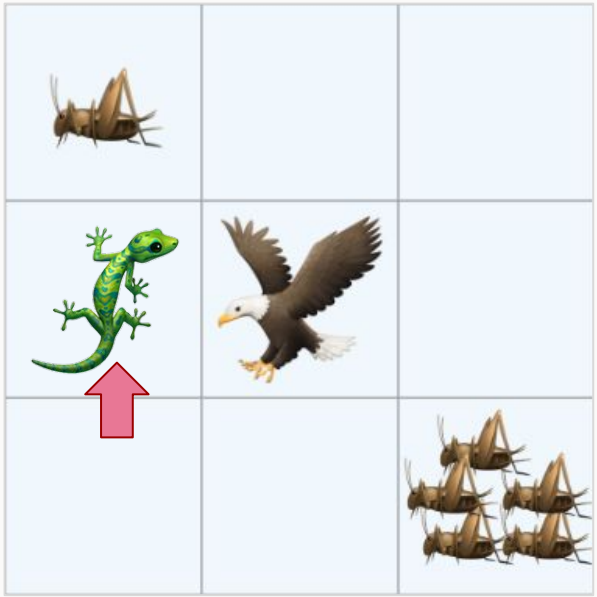
A_0 = Arriba

Q-Learning:

$R_1 = -1$

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Actualizamos Q

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmax Q[x_{i+1}])$$

Q[(2,0), arr] = (1 - 0.5) · Q[(2,0), arr] + 0.5 · (-1 + 0.9 · argmax Q[(1,0), :])

Q[(2,0), arr] = 0.5 · 0 + 0.5 · (-1 + 0.9 · 0)

Q[(2,0), arr] = 0.5 · -1

Q[(2,0), arr] = -0.5

Q-Table anterior

S_t	↑	↓	←	→
(2,0)	0	-1	-1	-0.5
(1,0)	0	-1	0	0

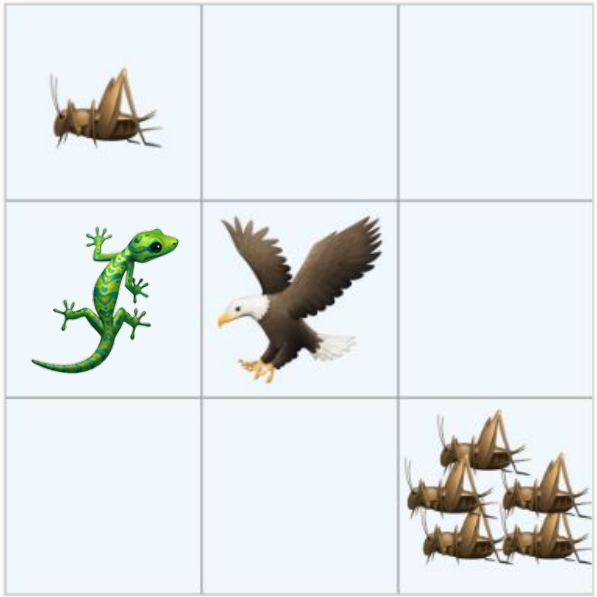
Q-Table actualizada

S_t	↑	↓	←	→
(2,0)	-0.5	-1	-1	-0.5
(1,0)	0	-1	0	0

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Con probabilidad $\epsilon = 0.3$ vamos a elegir un movimiento aleatorio.

 = 0.2

0.3 > 0.2, por lo que exploramos

Elegimos una acción aleatoria (y válida)

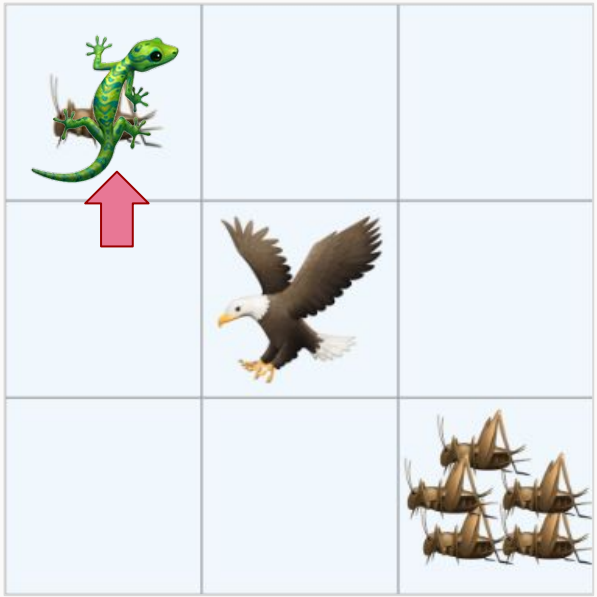
A_1 = Arriba

Q-Learning:

$R_2 = 1$

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Actualizamos Q

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmax Q[x_{i+1}])$$

Q[(1,0), arr] = (1 - 0.5) · Q[(1,0), arr] + 0.5 · (1 + 0.9 · argmax Q[(0,0), :])

Q[(1,0), arr] = 0.5 · 0 + 0.5 · (1 + 0.9 · 0)

Q[(1,0), arr] = 0.5 · 1

Q[(1,0), arr] = 0.5

Q-Table anterior

S_t	↑	↓	←	→
(0,0)	-1	0	-1	0
(1,0)	0	0	-1	0

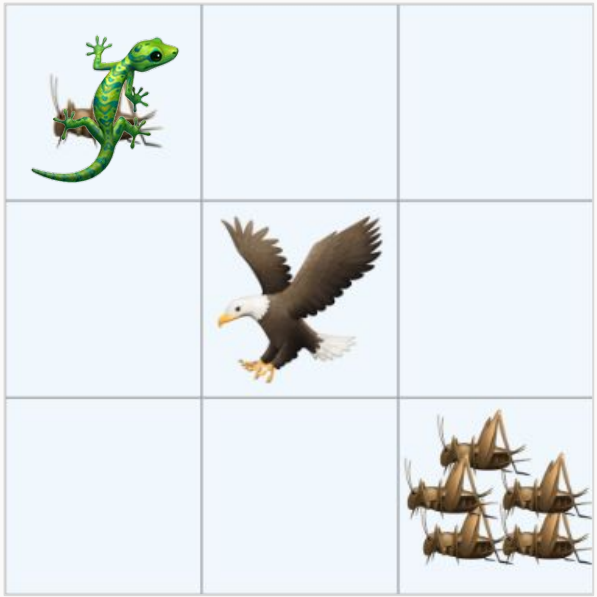
Q-Table actualizada

S_t	↑	↓	←	→
(0,0)	-1	0	-1	0
(1,0)	0.5	0	-1	0

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Con probabilidad ϵ = 0.3 vamos a elegir un movimiento aleatorio.

 = 0.5

0.3 < 0.5, por lo que explotamos

Buscamos el movimiento de mejor valor Q:

S_t	Arriba	Abajo	Izq.	Der.
(0,0)	-1	0	-1	0

Como son idénticos, tomamos uno aleatorio

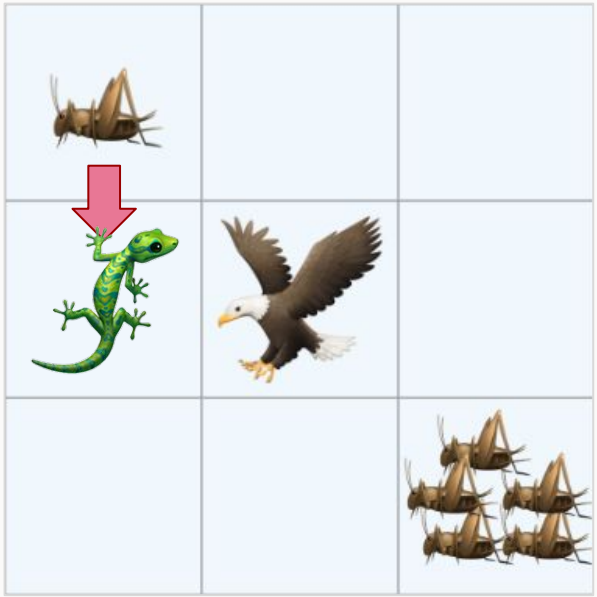
A_2 = Abajo

Q-Learning:

$R_3 = -1$

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Actualizamos Q

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmax Q[x_{i+1}])$$

Q[(0,0), ab] = (1 - 0.5) · Q[(0,0), ab] + 0.5 · (-1 + 0.9 · argmax Q[(1,0), :])

Q[(0,0), ab] = 0.5 · 0 + 0.5 · (-1 + 0.9 · 0.5)

Q[(0,0), ab] = 0.5 · (-1 + 0.45)

Q[(0,0), ab] = -0.275

Q-Table anterior

S_t	↑	↓	←	→
(0,0)	-1	0	-1	0
(1,0)	0.5	0	-1	0

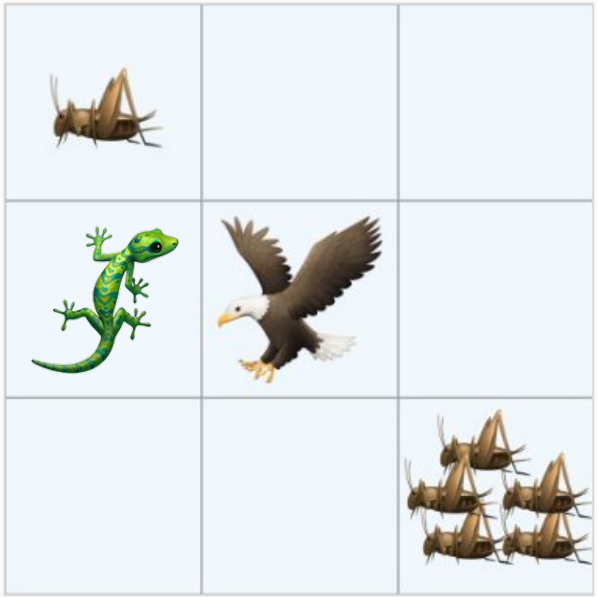
Q-Table actualizada

S_t	↑	↓	←	→
(0,0)	-1	-0.275	-1	0
(1,0)	0.5	0	-1	0

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Con probabilidad ϵ = 0.3 vamos a elegir un movimiento aleatorio.

 = 0.1

0.3 > 0.1, por lo que exploramos

Elegimos una acción aleatoria (y válida)

A_3 = Derecha

Q-Learning:

$R_4 = -10$

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$

Actualizamos Q

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmax Q[x_{i+1}])$$

Q[(1,0), der] = (1 - 0.5) · Q[(1,0), der] + 0.5 · (-10 + 0.9 · argmax Q[(1,1), :])

Q[(1,0), der] = 0.5 · 0 + 0.5 · (-10 + 0.9 · 0)

Q[(1,0), der] = 0.5 · -10

Q[(1,0), der] = -5

Q-Table anterior

S_t	↑	↓	←	→
(1,0)	0.5	0	-1	0
(1,1)	0	0	0	0

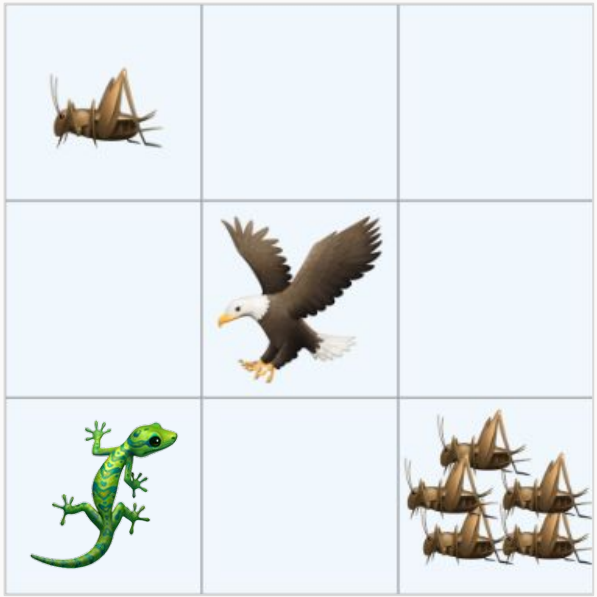
Q-Table actualizada

S_t	↑	↓	←	→
(1,0)	0.5	0	-1	-5
(1,1)	0	0	0	0

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$

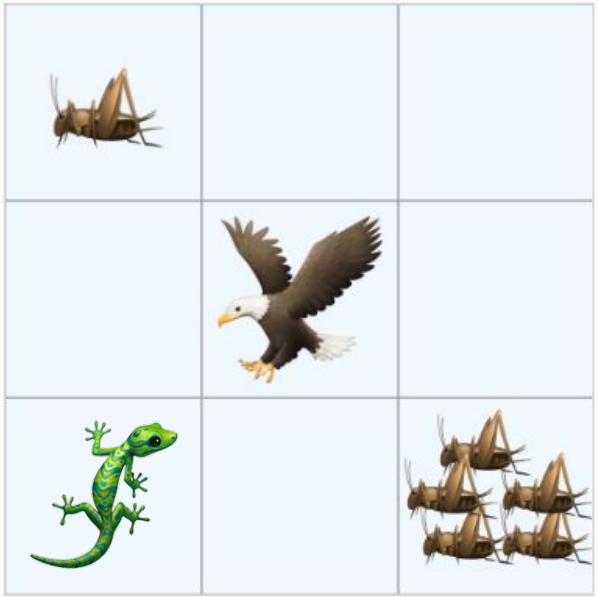


S_t	Arriba	Abajo	Izq.	Der.
(0,0)	-1	-0.275	-1	0
(0,1)	-1	0	0	0
(0,2)	-1	0	0	-1
(1,0)	0.5	0	-1	-5
(1,1)	0	0	0	0
(1,2)	0	0	0	-1
(2,0)	-0.5	-1	-1	-0.5
(2,1)	-5	-1	0	0
(2,2)	0	-1	0	-1

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Con probabilidad ϵ = 0.3 vamos a elegir un movimiento aleatorio.

 = 0.7

0.3 < 0.7, por lo que explotamos

Buscamos el movimiento de mejor valor Q:

S_t	Arriba	Abajo	Izq.	Der.
(2,0)	-0.5	-1	-1	-0.5

Como son idénticos, tomamos una aleatoria

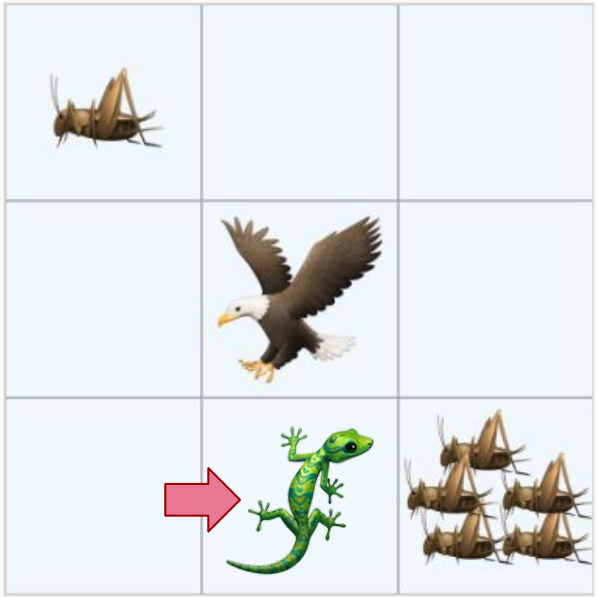
A_0 = Derecha

Q-Learning:

$R_1 = -1$

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Actualizamos Q

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmax Q[x_{i+1}])$$

Q[(2,0), der] = (1 - 0.5) · Q[(2,0), der] + 0.5 · (-1 + 0.9 · argmax Q[(2,1), :])

Q[(2,0), der] = 0.5 · -0.5 + 0.5 · (-1 + 0.9 · 0)

Q[(2,0), der] = -0.25 + 0.5 · -1

Q[(2,0), der] = -0.75

Q-Table anterior

S_t	↑	↓	←	→
(2,0)	-0.5	-1	-1	-0.5
(2,1)	-5	-1	0	0

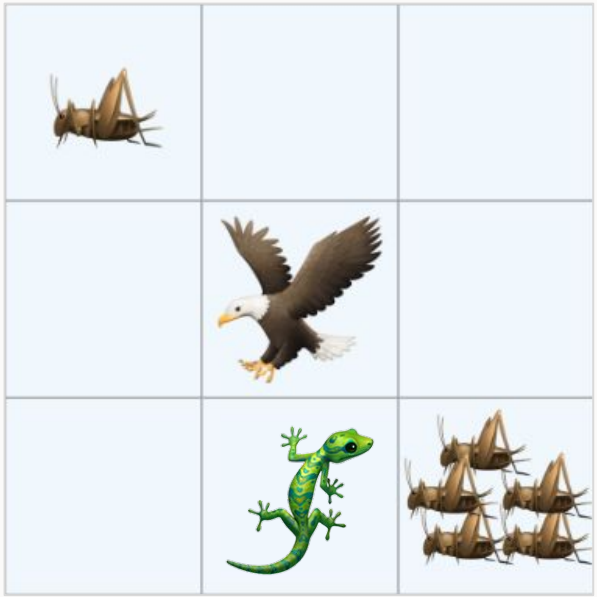
Q-Table actualizada

S_t	↑	↓	←	→
(2,0)	-0.5	-1	-1	-0.75
(2,1)	-5	-1	0	0

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Con probabilidad $\epsilon = 0.3$ vamos a elegir un movimiento aleatorio.

 = 0.4

0.3 < 0.4, por lo que explotamos

Buscamos el movimiento de mejor valor Q:

S_t	Arriba	Abajo	Izq.	Der.
(2,1)	-5	-1	0	0

Como son idénticos, tomamos una aleatoria

A_1 = Derecha

Q-Learning:

$R_2 = 10$

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$

Actualizamos Q

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmax Q[x_{i+1}])$$

Q[(2,1), der] = (1 - 0.5) · Q[(2,1), der] + 0.5 · (10 + 0.9 · argmax Q[(2,2), :])

Q[(2,1), der] = 0.5 · 0 + 0.5 · (10 + 0.9 · 0)

Q[(2,1), der] = 0.5 · 10

Q[(2,1), der] = 5

Q-Table anterior

S_t	↑	↓	←	→
(2,1)	-5	-1	0	0
(2,1)	0	-1	0	-1

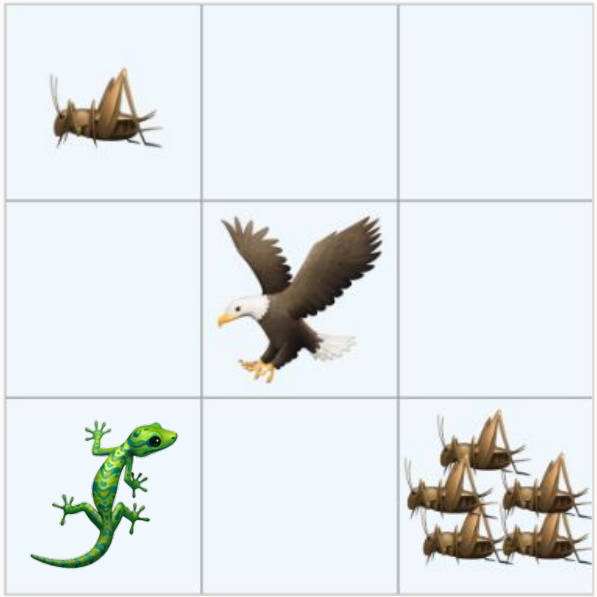
Q-Table actualizada

S_t	↑	↓	←	→
(2,1)	-5	-1	0	5
(2,1)	0	-1	0	-1

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$

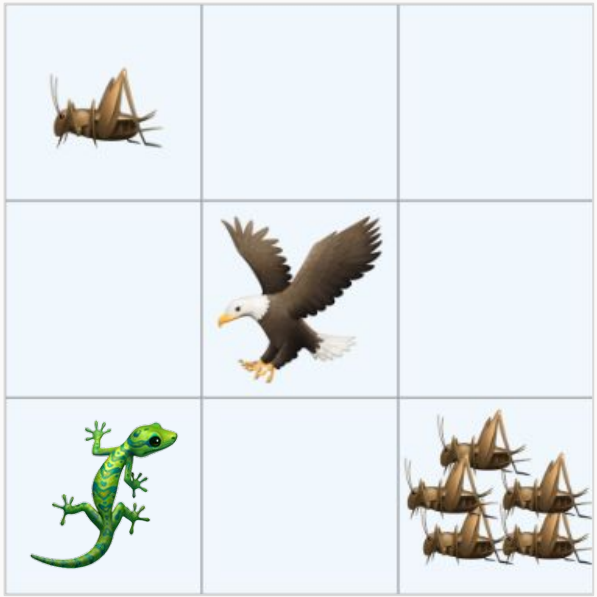


S_t	Arriba	Abajo	Izq.	Der.
(0,0)	-1	-0.275	-1	0
(0,1)	-1	0	0	0
(0,2)	-1	0	0	-1
(1,0)	0.5	0	-1	-5
(1,1)	0	0	0	0
(1,2)	0	0	0	-1
(2,0)	-0.5	-1	-1	-0.75
(2,1)	-5	-1	0	5
(2,2)	0	-1	0	-1

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Con probabilidad ϵ = 0.3 vamos a elegir un movimiento aleatorio.

 = 0.2

0.3 > 0.2, por lo que exploramos

Elegimos una acción aleatoria (y válida)

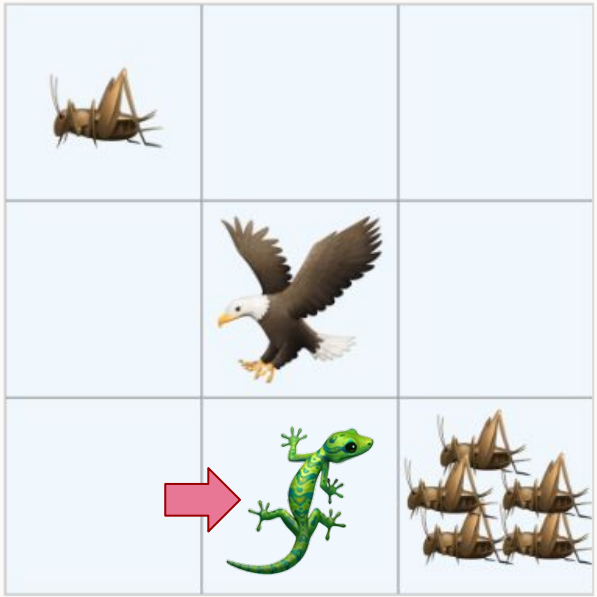
A_0 = Derecha

Q-Learning:

$R_1 = -1$

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Actualizamos Q

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmax Q[x_{i+1}])$$

Q[(2,0), der] = (1 - 0.5) · Q[(2,0), der] + 0.5 · (-1 + 0.9 · argmax Q[(2,1), :])

Q[(2,0), der] = 0.5 · -0.75 + 0.5 · (-1 + 0.9 · 5)

Q[(2,0), der] = -0.375 + 0.5 · 3.5

Q[(2,0), der] = 1.375

Q-Table anterior

S_t	↑	↓	←	→
(2,0)	-0.5	-1	-1	-0.75
(2,1)	-5	-1	0	5

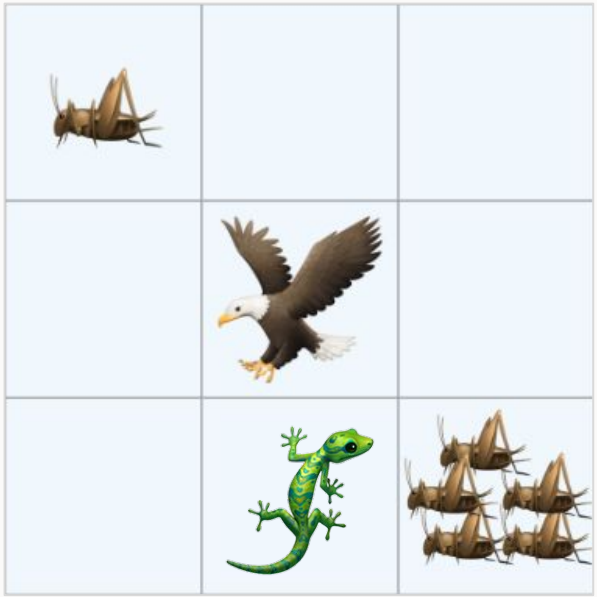
Q-Table actualizada

S_t	↑	↓	←	→
(2,0)	-0.5	-1	-1	1.375
(2,1)	-5	-1	0	5

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



Con probabilidad ϵ = 0.3 vamos a elegir un movimiento aleatorio.

 = 0.9

0.3 < 0.9, por lo que explotamos

Buscamos el movimiento de mejor valor Q:

S_t	Arriba	Abajo	Izq.	Der.
(2,1)	-5	-1	0	5

En este caso, corresponde a derecha

A_2 = Derecha

Q-Learning:

$R_2 = 10$

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$

Actualizamos Q

$$Q[x_t, a_t] = (1 - \alpha) \cdot Q[x_t, a_t] + \alpha \cdot (R_t + \gamma \cdot argmax Q[x_{i+1}])$$

Q[(2,1), der] = (1 - 0.5) · Q[(2,1), der] + 0.5 · (10 + 0.9 · argmax Q[(2,2), :])

Q[(2,1), der] = 0.5 · 5 + 0.5 · (10 + 0.9 · 0)

Q[(2,1), der] = 2.5 + 0.5 · 10

Q[(2,1), der] = 7.5

Q-Table anterior

S_t	↑	↓	←	→
(2,1)	-5	-1	0	5
(2,1)	0	-1	0	-1

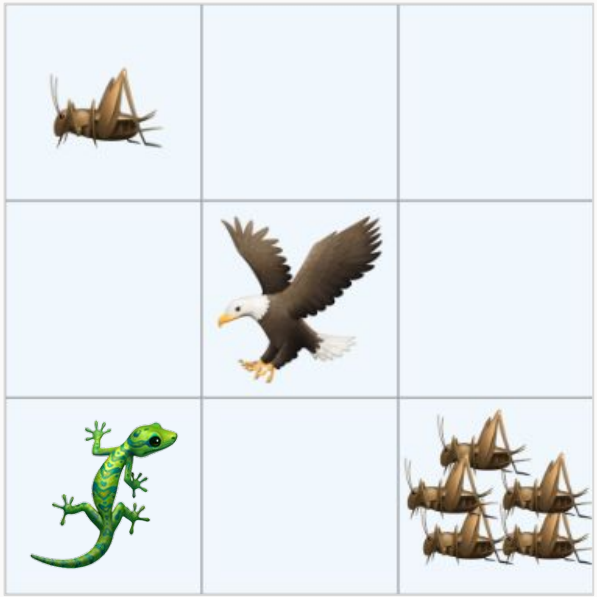
Q-Table actualizada

S_t	↑	↓	←	→
(2,1)	-5	-1	0	7.5
(2,1)	0	-1	0	-1

Q-Learning:

Estado	Recompensa
1 Grillo	+1
5 Grillos	+10
Pájaro	-10
Vacío	-1

$$\epsilon = 0.3$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$



S_t	Arriba	Abajo	Izq.	Der.
(0,0)	-1	-0.275	-1	0
(0,1)	-1	0	0	0
(0,2)	-1	0	0	-1
(1,0)	0.5	0	-1	-5
(1,1)	0	0	0	0
(1,2)	0	0	0	-1
(2,0)	-0.5	-1	-1	1.375
(2,1)	-5	-1	0	7.5
(2,2)	0	-1	0	-1

Q-Learning:

Grafiquemos la política actual

S_t	Arriba	Abajo	Izq.	Der.
(0,0)	-1	-0.275	-1	0
(0,1)	-1	0	0	0
(0,2)	-1	0	0	-1
(1,0)	0.5	0	-1	-5
(1,1)	0	0	0	0
(1,2)	0	0	0	-1
(2,0)	-0.5	-1	-1	1.375
(2,1)	-5	-1	0	7.5
(2,2)	0	-1	0	-1

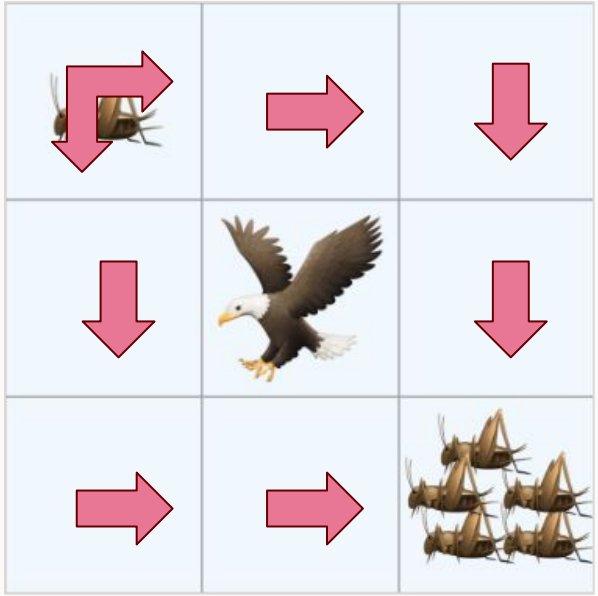
Q-Learning:

(...)

Un millón de iteraciones más tarde

Q-Learning:

Grafiquemos la política tras 1 millón de simulaciones:



S_t	Arriba	Abajo	Izq.	Der.
(0,0)	-1	4.58	-1	4.58
(0,1)	-1	-10	5.122	6.2
(0,2)	-1	8	4.58	-1
(1,0)	5.122	6.2	-1	-10
(1,1)	0	0	0	0
(1,2)	6.2	10	-10	-1
(2,0)	4.58	-1	-1	8
(2,1)	-10	-1	6.2	10
(2,2)	0	-1	0	-1

Q-Learning

En resumen:

- Q-Learning busca continuamente asignar y **actualizar** un valor de *calidad* a cada par estado-acción.
- La acción a ser ejecutada es definida como aquella que tiene **mayor calidad asociada** en la tabla al estado actual, dado por:

$$a_t = arg\max(Q[s_t, :])$$

- El algoritmo puede acabar tras un número *k* de partidas jugadas o bien bajo un criterio de convergencia de los valores en la Q-Table
- El algoritmo nos garantiza **optimalidad** en la política encontrada *(pero ojo, solo en $t \to \infty$)*
- Normalmente solemos variar el parámetro $\in$ a lo largo de la ejecución, partiendo desde un valor alto para **promover la exploración temprana**, hacia uno bajo, **promoviendo la explotación tardía**

Ayudantía 11

Reinforcement Learning

Daniel Florea

17 de noviembre 2023