

Reinforcement Learning

(II)



1

**Elementos de Reinforcement Learning:
concepto de estado**

2

Markov Decision Process

3

State space y action space

4

Self-driving cab problem

5

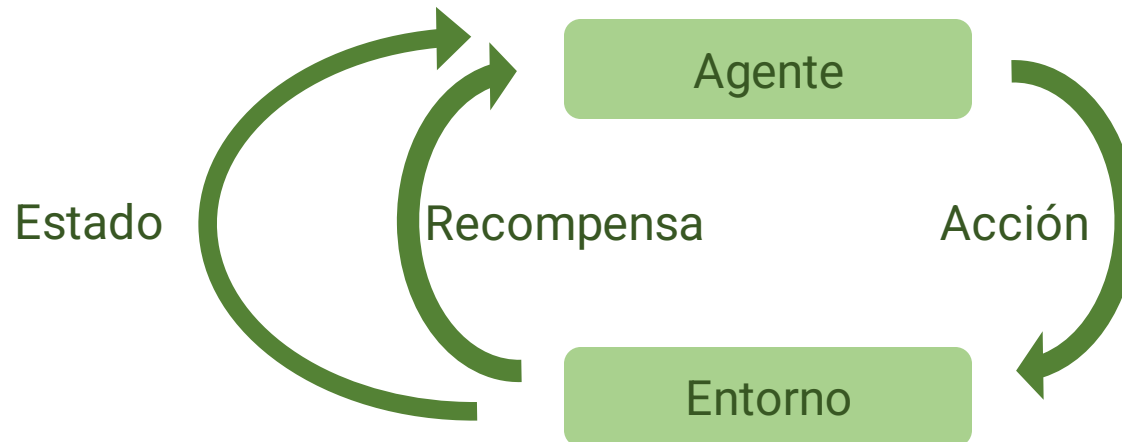
Q-learning

1. Elementos de Reinforcement Learning: concepto de estado

Concepto de estado

El aprendizaje por refuerzo tiene cuatro (+1!) elementos esenciales:

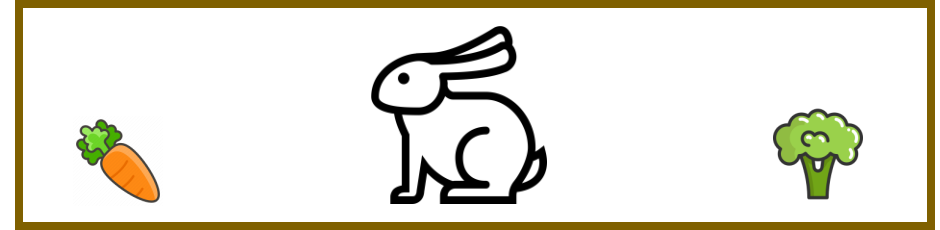
1. **Agente:** El elemento inteligente que toma decisiones
2. **Entorno:** El mundo, real o virtual, en el que el agente realiza acciones
3. **Acción:** Un movimiento realizado por el agente
4. **Recompensa:** La valoración de una acción, que puede ser positiva o negativa
5. **Estado:** Situación/posición en la que se encuentra actualmente el agente



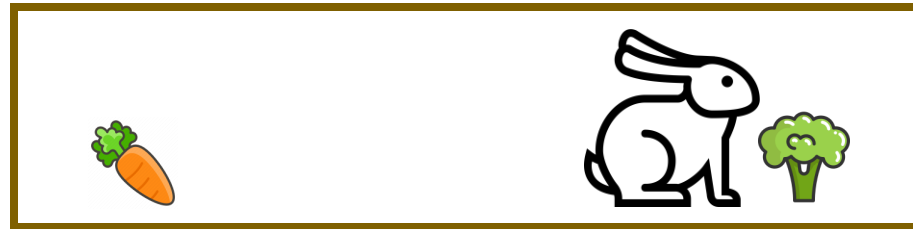
Concepto de estado



Estado 1

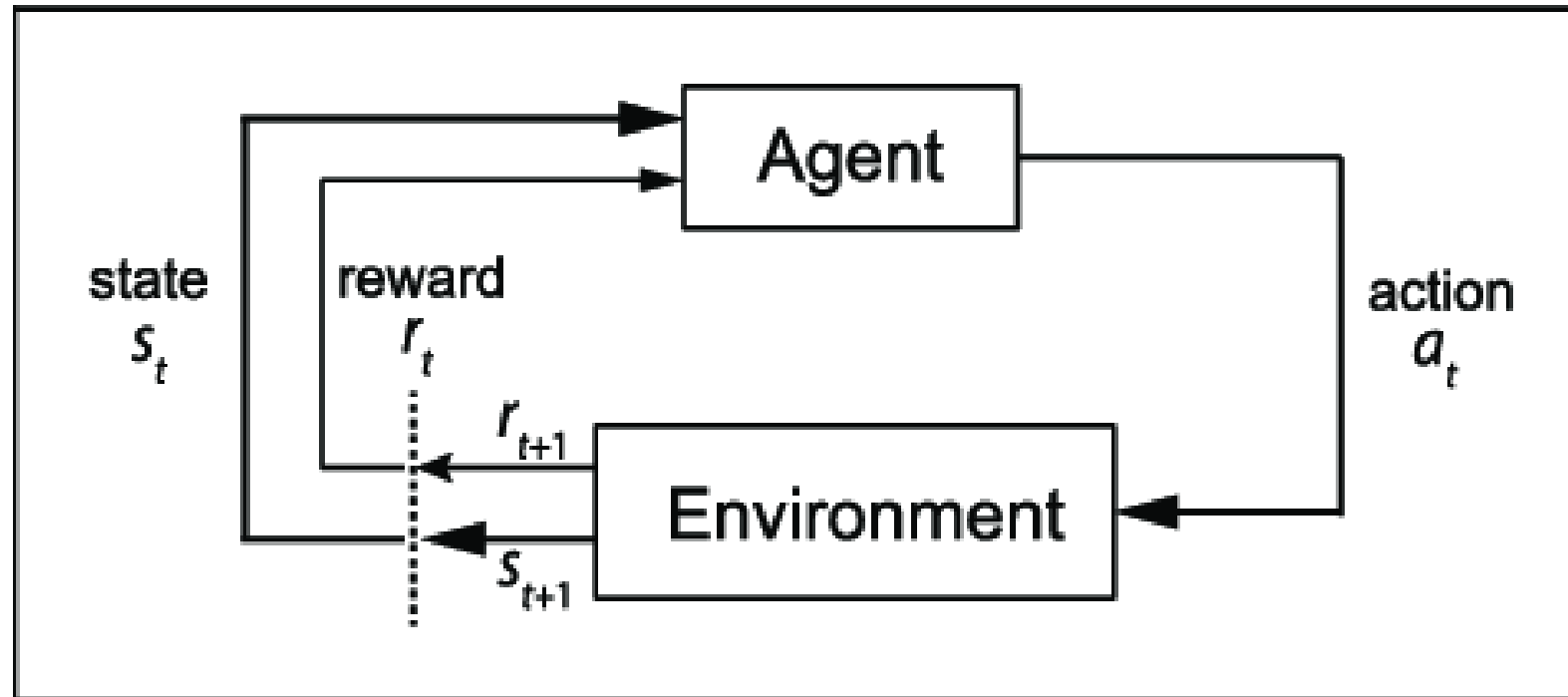


Estado 2



Estado 3

Concepto de estado



2. Markov Decision Process

Markov Decision Process

El problema del **k-armed bandit** que analizamos anteriormente presenta muchas preguntas interesantes.

Sin embargo, no incluye muchos aspectos de los problemas del mundo real, no cambia el estado.

Markov Decision Process

En los problemas reales, hay dos aspectos importantes que destacar:

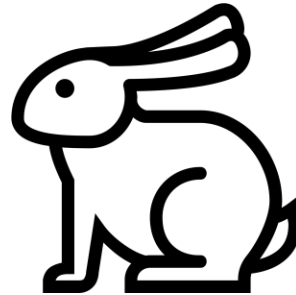
1. Diferentes situaciones exigen diferentes respuestas
2. Las acciones que elegimos ahora afectan la cantidad de recompensa que podemos obtener en el futuro

El **proceso de decisión de Markov** (MDP) captura estos dos aspectos

Markov Decision Process - Ejemplo del conejo

Nuestro conejo está en un campo en busca de comida y se encuentra en una situación en la que hay una zanahoria a su izquierda y un brócoli a su derecha.

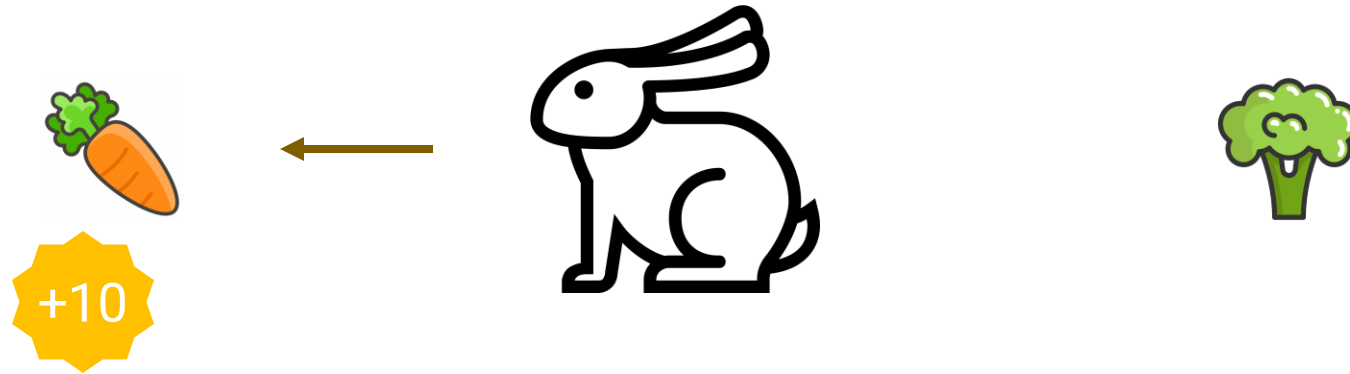
El conejo prefiere la zanahoria, así que se mueve a la izquierda.



Markov Decision Process - Ejemplo del conejo

Nuestro conejo está en un campo en busca de comida y se encuentra en una situación en la que hay una zanahoria a su izquierda y un brócoli a su derecha.

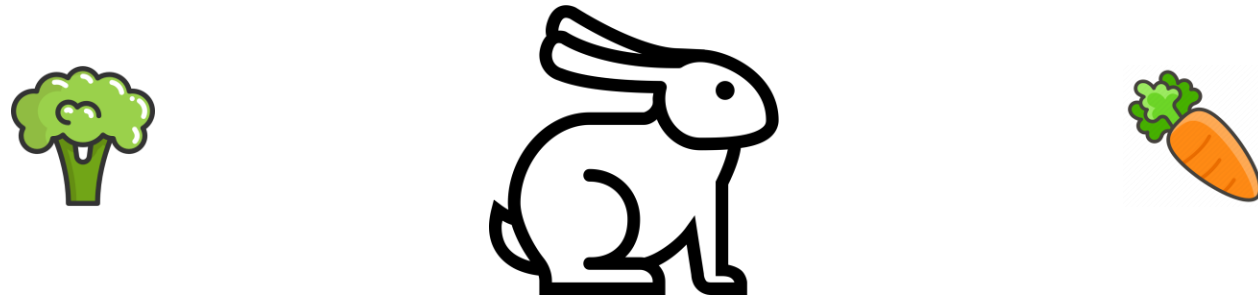
El conejo prefiere la zanahoria, así que se mueve a la izquierda.



Markov Decision Process - Ejemplo del conejo

¿Pero qué pasa si la situación cambia? ¿Y si ahora la zanahoria está a la derecha?

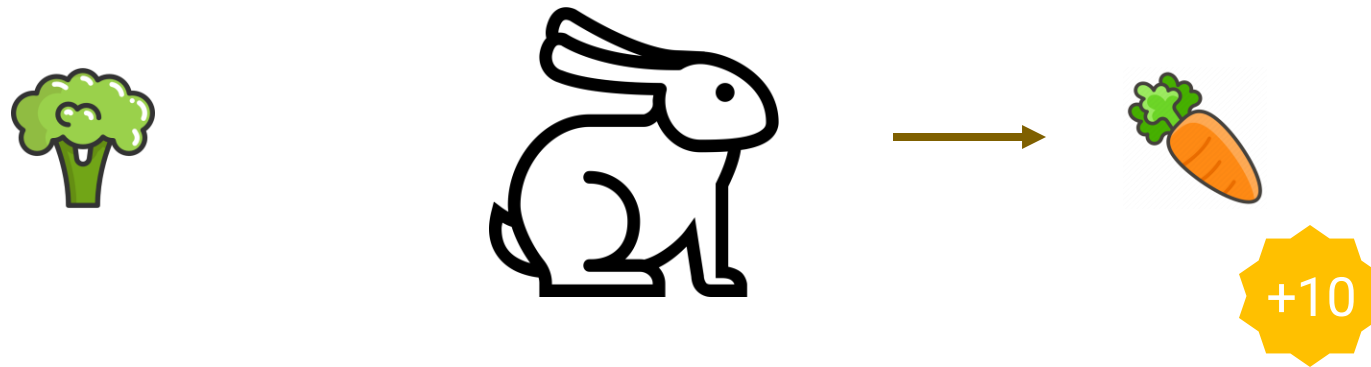
Aquí, el conejo claramente preferiría ir a la derecha en lugar de a la izquierda.



Markov Decision Process - Ejemplo del conejo

¿Pero qué pasa si la situación cambia? ¿Y si ahora la zanahoria está a la derecha?

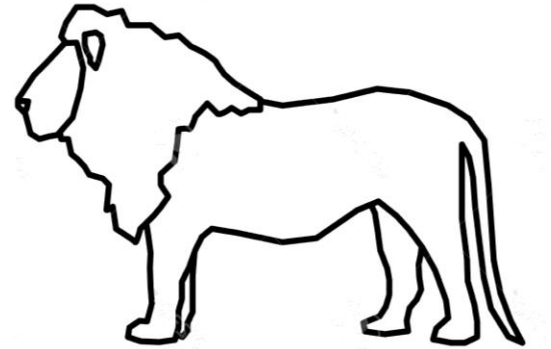
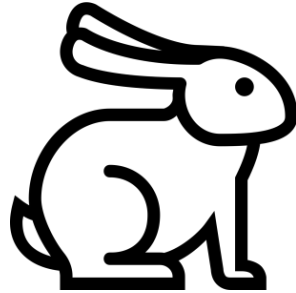
Aquí, el conejo claramente preferiría ir a la derecha en lugar de a la izquierda.



1. Diferentes situaciones exigen diferentes respuestas

Markov Decision Process - Ejemplo del conejo

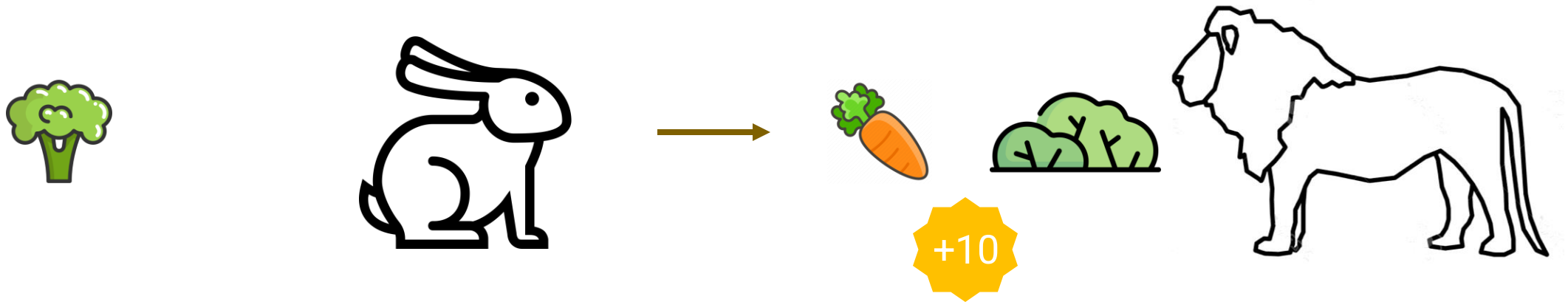
En este otro estado, el conejo se mueve a la derecha, donde está la zanahoria.



Markov Decision Process - Ejemplo del conejo

Sin embargo, ir a la derecha también afectará la próxima situación en la que se encuentre el conejo.

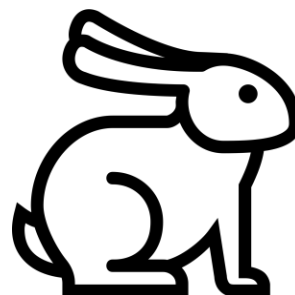
A la derecha de la zanahoria hay un león.



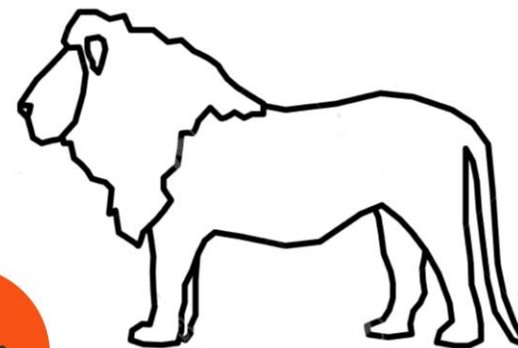
Markov Decision Process - Ejemplo del conejo

El conejo se come la zanahoria, pero ahora se encuentra cara a cara con el león.

En este sentido, si tenemos en cuenta el impacto a largo plazo de nuestras acciones, el conejo debería ir a la izquierda y conformarse con el brócoli para tener una mejor oportunidad de escapar.



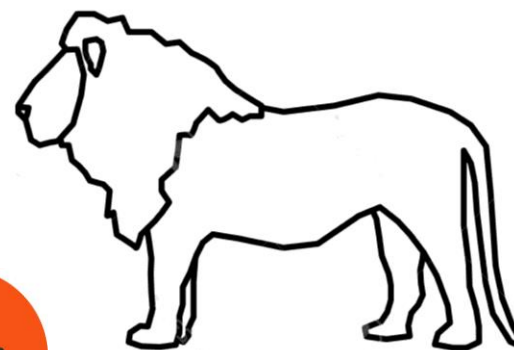
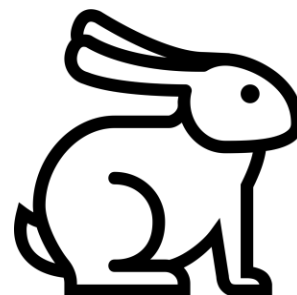
-100000



Markov Decision Process - Ejemplo del conejo

El conejo se come la zanahoria, pero ahora se encuentra cara a cara con el león.

En este sentido, si tenemos en cuenta el impacto a largo plazo de nuestras acciones, el conejo debería ir a la izquierda y conformarse con el brócoli para tener una mejor oportunidad de escapar.



-100000

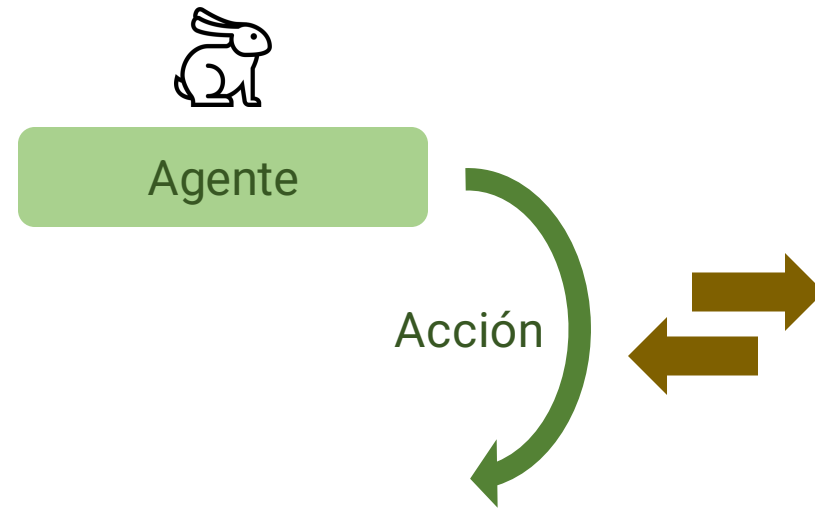
2. Las acciones que elegimos ahora afectan la cantidad de recompensa futura

Markov Decision Process - Ejemplo del conejo

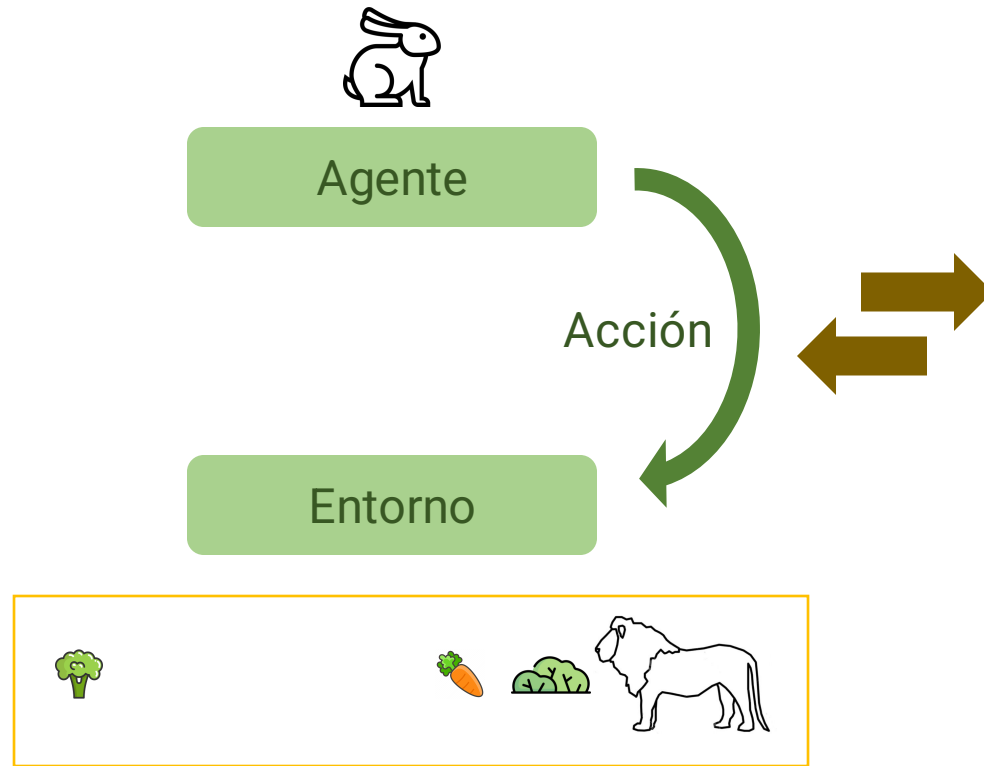


Agente

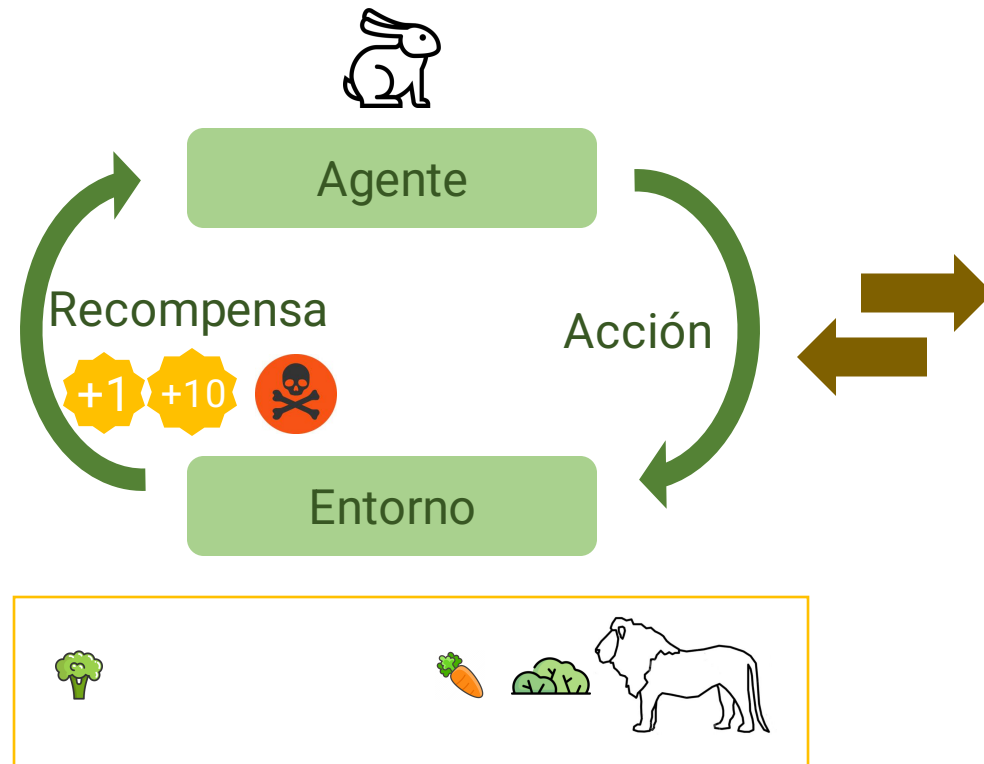
Markov Decision Process



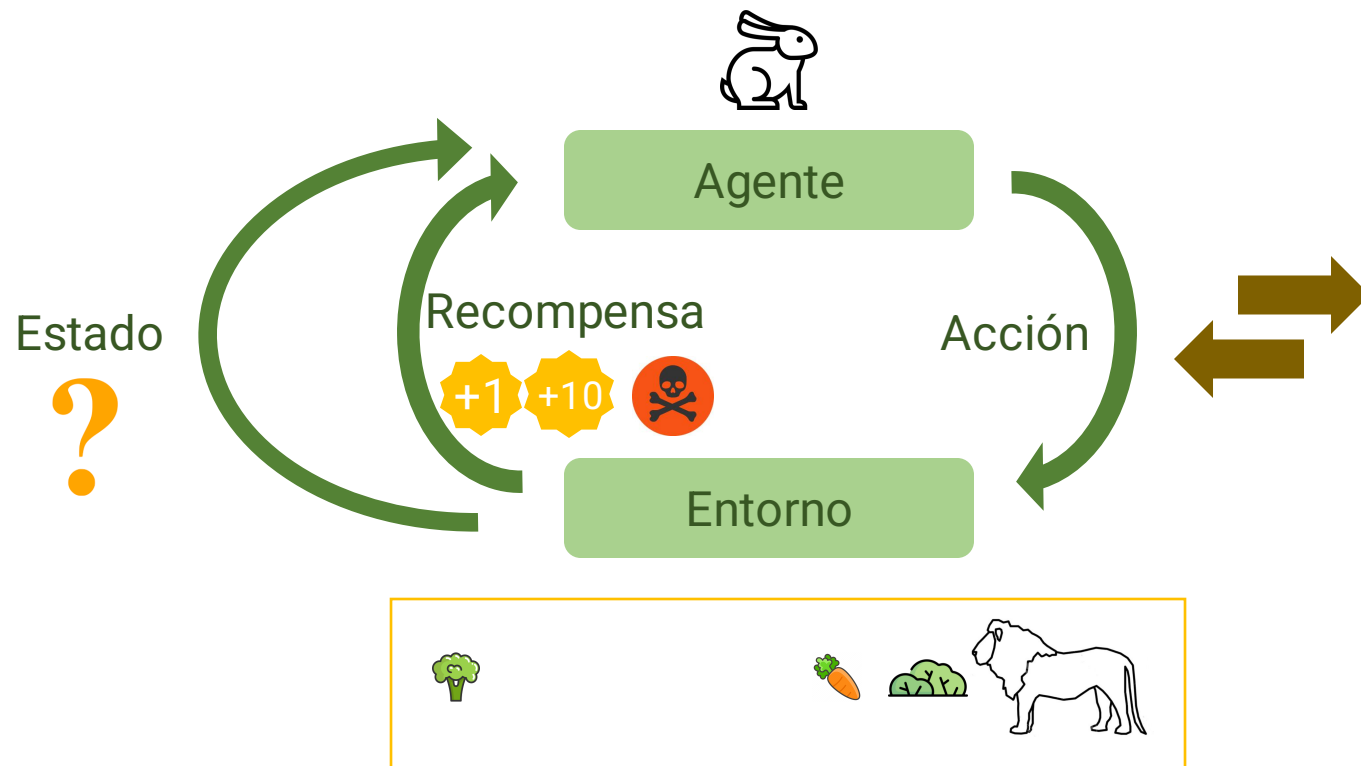
Markov Decision Process



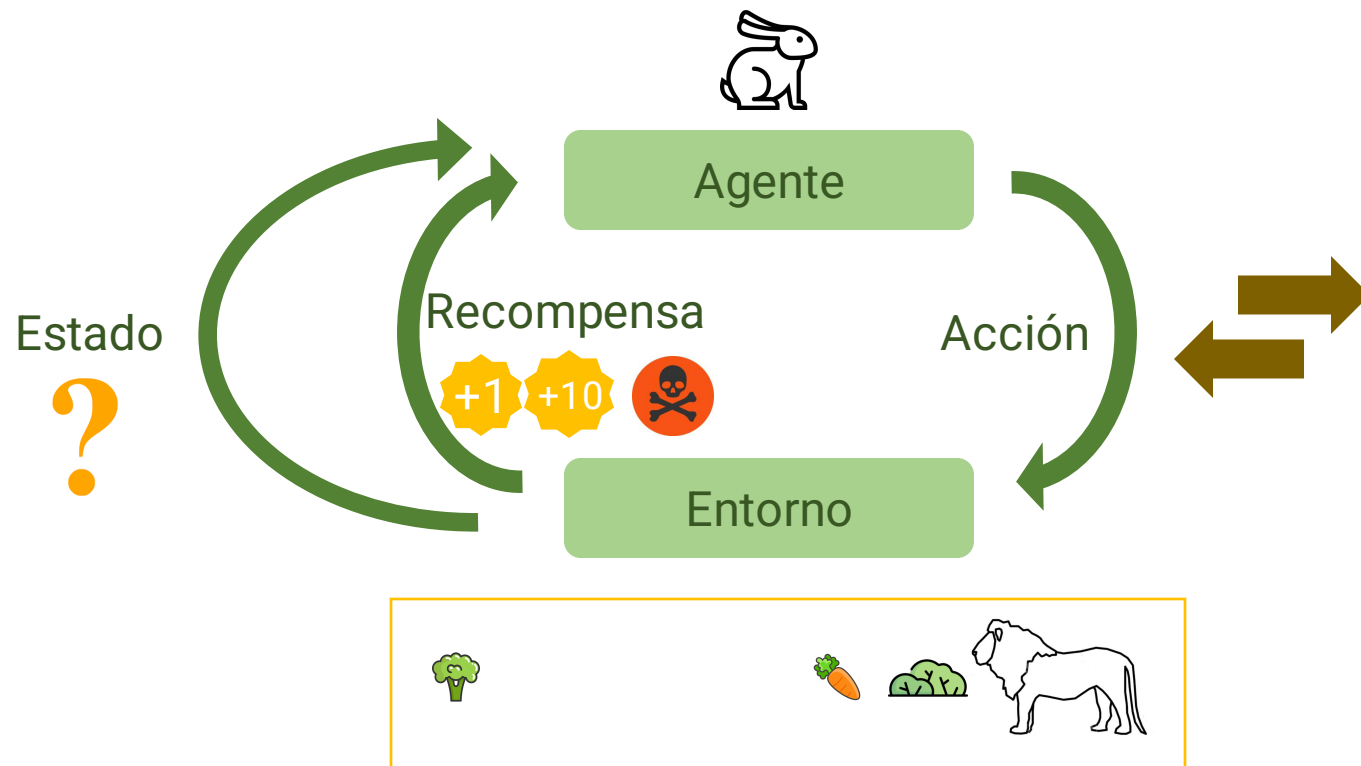
Markov Decision Process



Markov Decision Process



Markov Decision Process



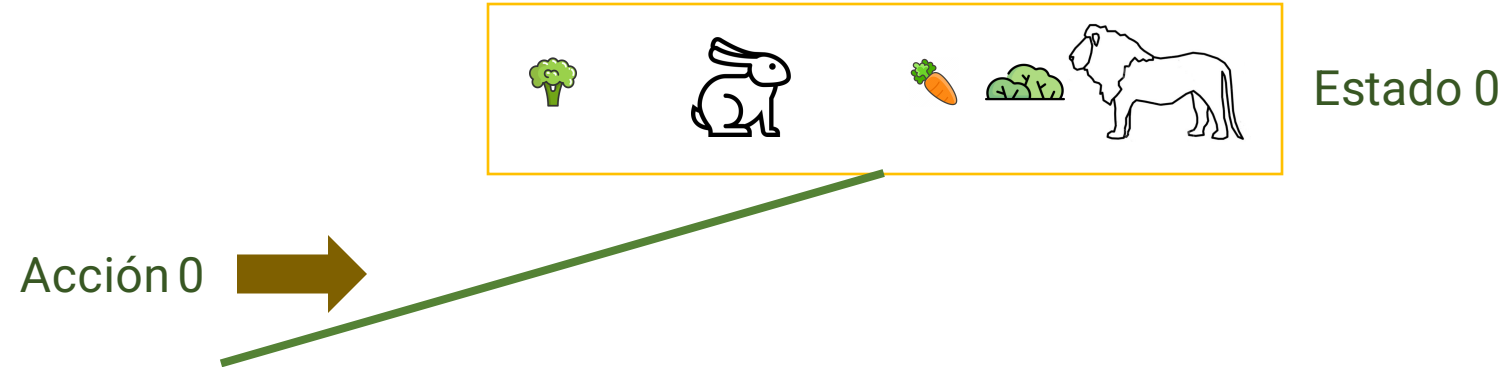
Ahora veremos cómo cambia la situación a medida que el conejo actúa.
Llamaremos a estas situaciones **estados**.

Markov Decision Process



Estado 0

Markov Decision Process



Markov Decision Process

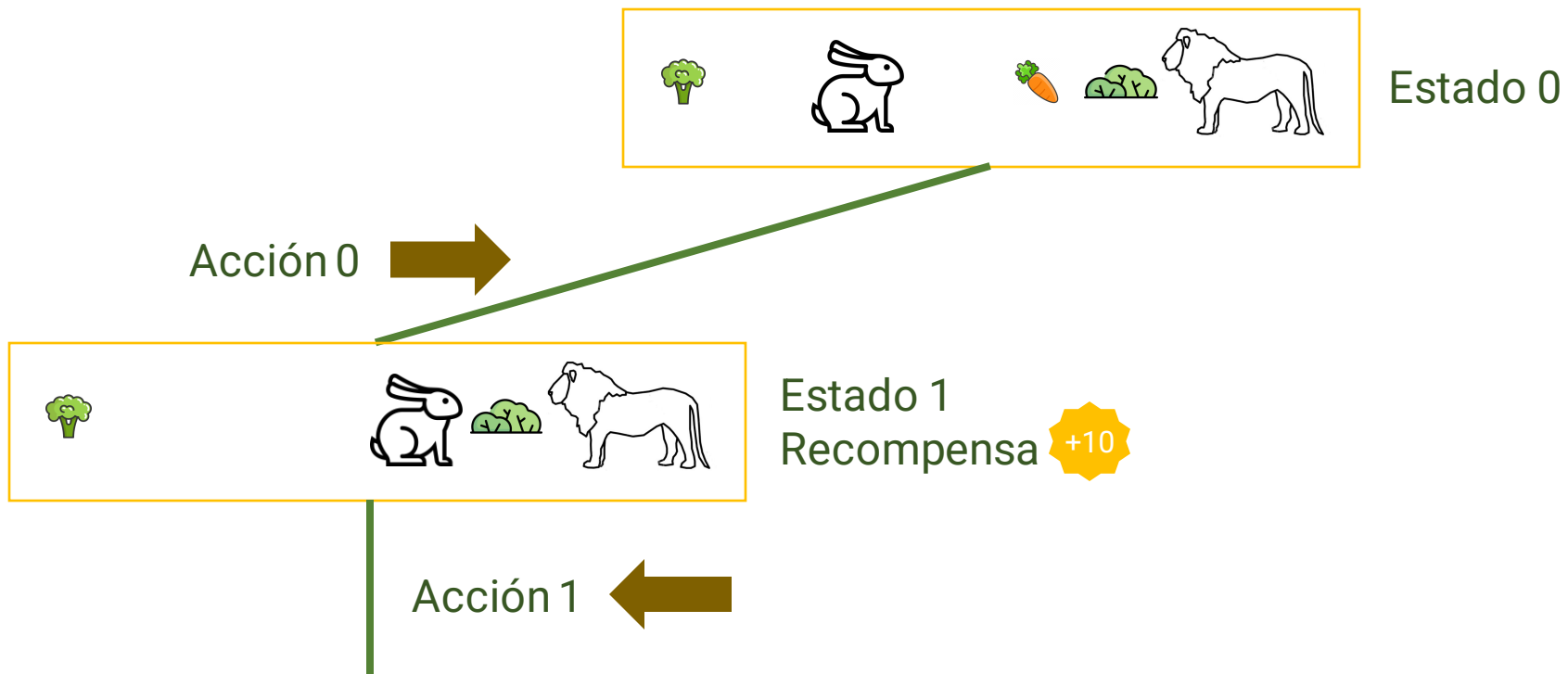


Acción 0 



Estado 1
Recompensa +10

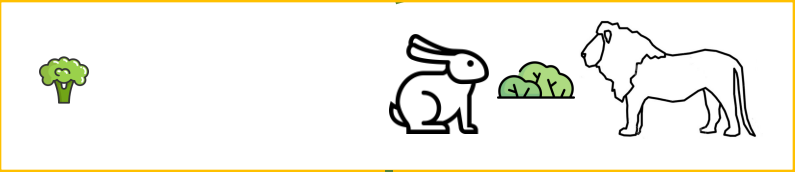
Markov Decision Process



Markov Decision Process



Acción 0 →



Recompensa +10



← Acción 1

Penalización

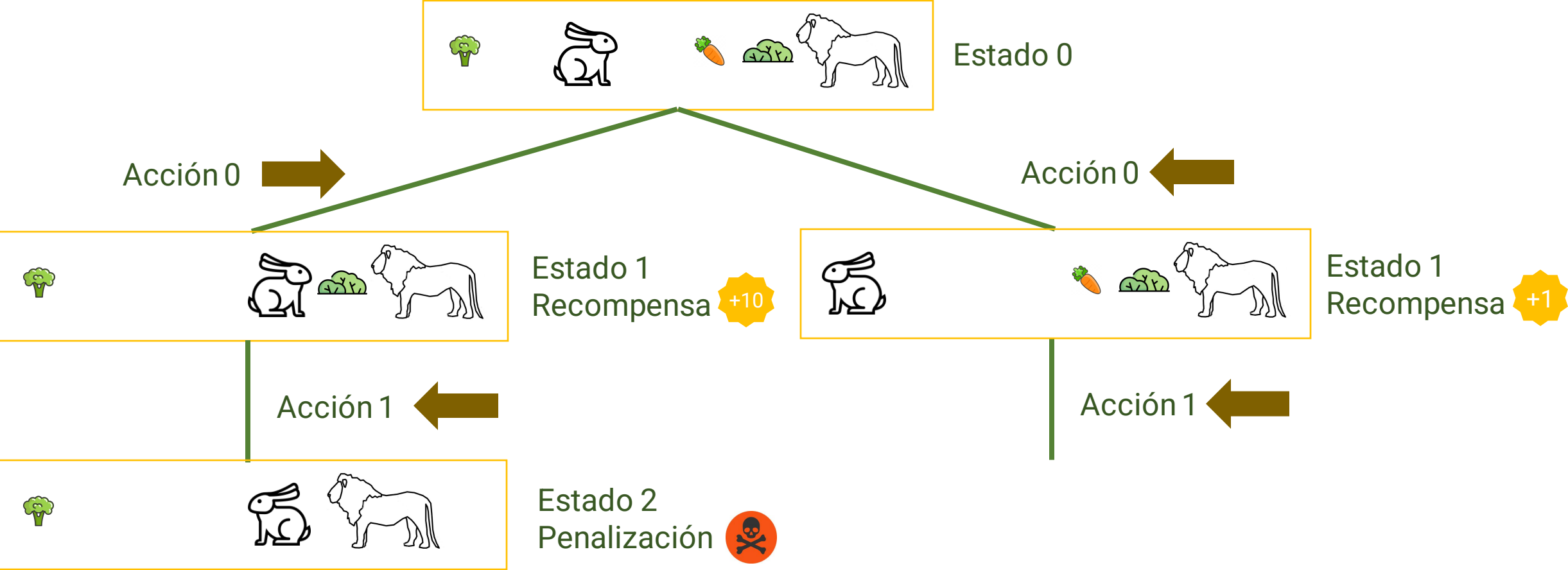
Markov Decision Process



Markov Decision Process



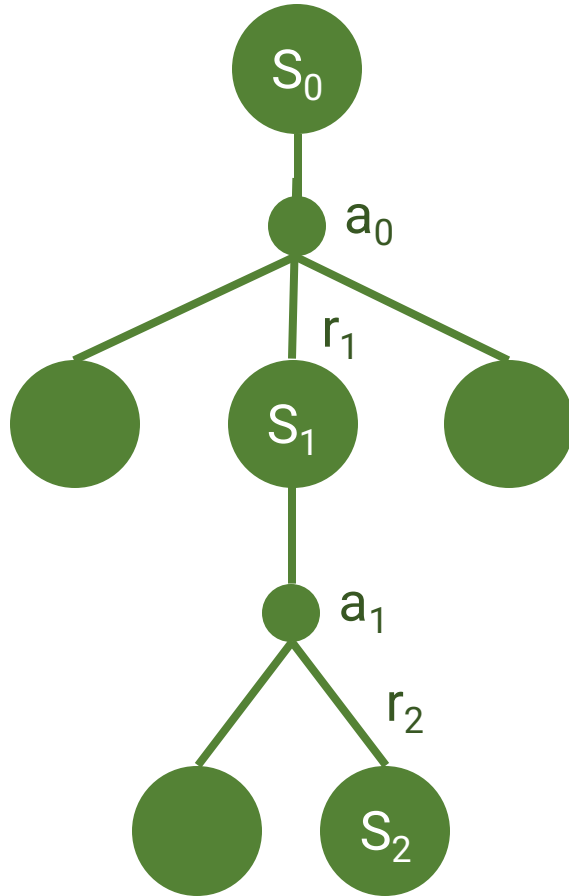
Markov Decision Process



Markov Decision Process



Markov Decision Process



La interacción del agente con el entorno genera una trayectoria de experiencias que consta de estados, acciones y recompensas.

Las **acciones** influyen en las **recompensas inmediatas**, así como en los **estados futuros** y, a través de ellos, en las **recompensas futuras**.

3. State space y action space

State space y action space

Action space






Conjunto de posibles acciones que el agente puede realizar

State space

Conjunto de posibles situaciones en las que el agente se puede encontrar

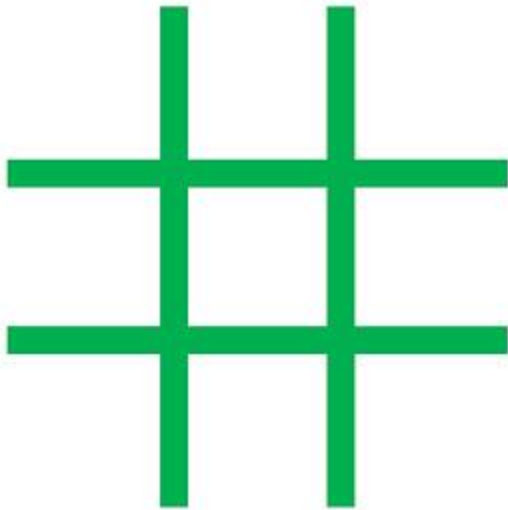
State space y action space

Podemos visualizar un ejemplo sencillo de **state space** en una cuadrícula:

	0	1	2	3
0				
1				
2				

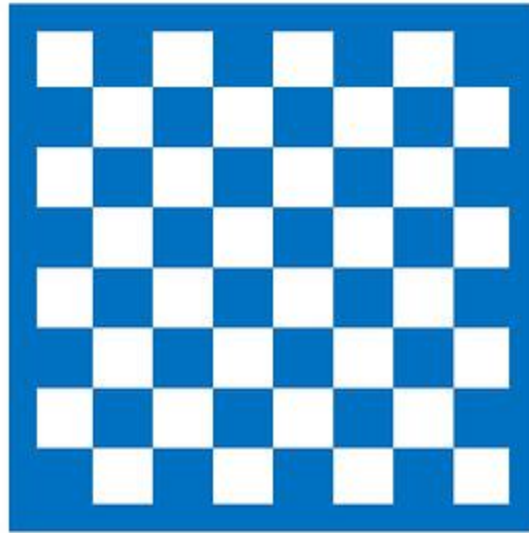
STATE SPACE COMPLEXITY

TIC-TAC-TOE



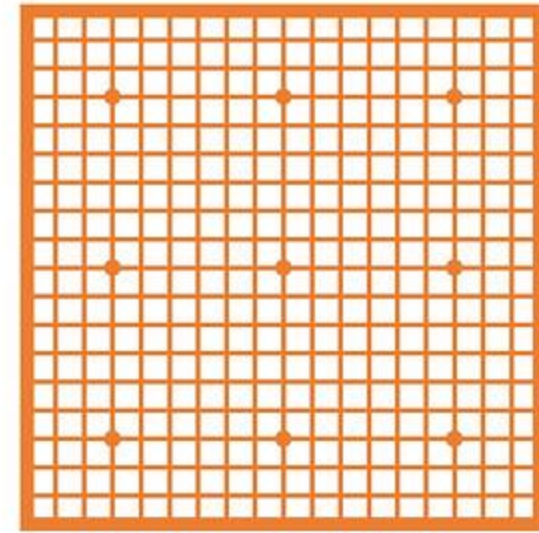
3x3

CHESS



8x8






GO









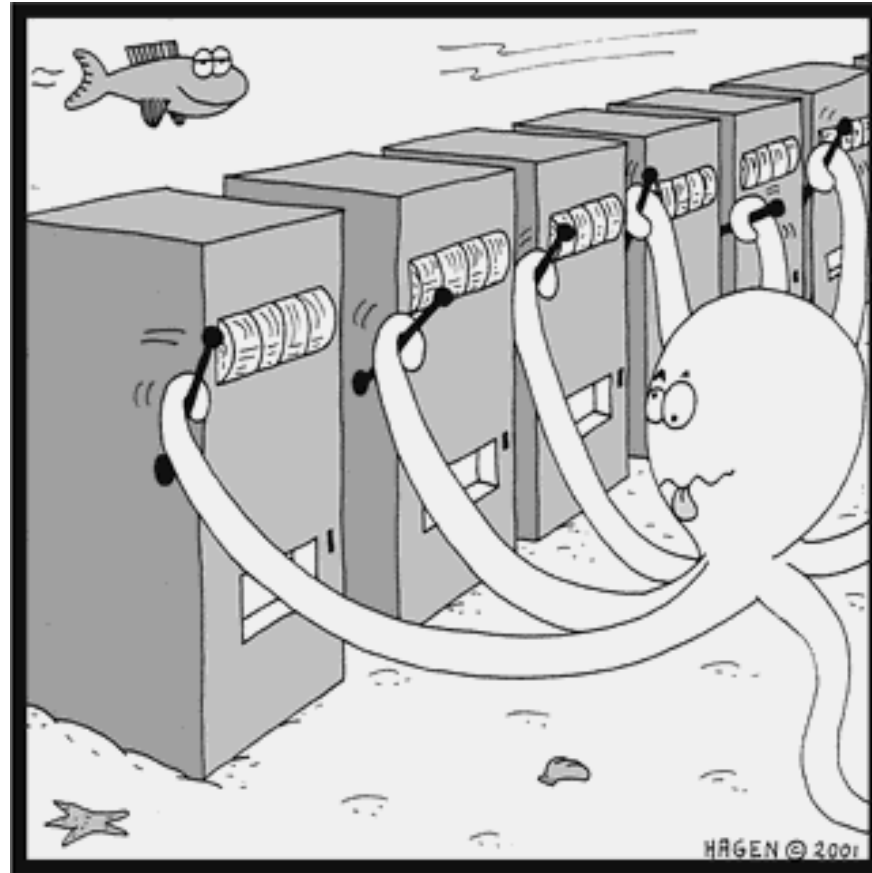
18x18

State space y action space

El **state space** incluye la posición del agente y la de todos los elementos del entorno:

	0	1	2	3
0				
1				
2				

	0	1	2	3
0				
1				
2				



El k-armed bandit problem es equivalente a un proceso de decisión de Markov de UN ESTADO

State space y action space

En este curso supondremos que el conjunto de acciones y estados sea finito

En realidad, existen algoritmos que pueden manejar conjuntos infinitos y conjuntos incontables

4. Self-driving cab problem

Self-driving cab problem

Objetivo principal:

- Recoger a los pasajeros en un lugar y dejarlos en otro

Objetivos secundarios:

- Dejar a los pasajeros en el lugar correcto
- Ahorrar tiempo a los pasajeros, tardando el mínimo tiempo posible
- Tener en cuenta la seguridad de los pasajeros y las normas de tránsito



Cómo siempre, empezamos definiendo los elementos de nuestro problema



Self-driving cab problem



Agente

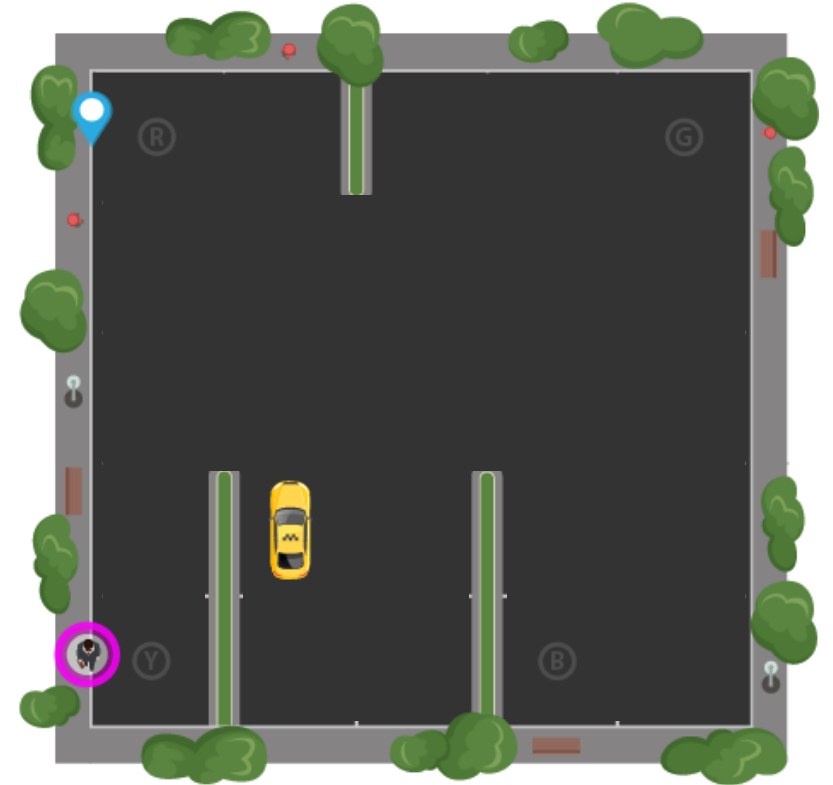
El taxi (la IA que controla el taxi automático)



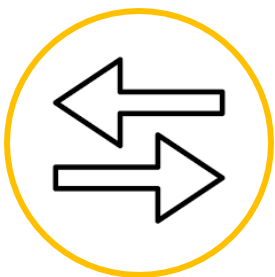
Entorno

Un aparcamiento
+ todos los elementos con los que el taxi
tiene que interactuar:

- El pasajero
- Los posibles lugares de recogida y destino de los pasajeros
- Paredes, que el taxi no puede cruzar



Self-driving cab problem



Acciones

Tenemos 6 acciones posibles:

1. Sur
2. Norte
3. Este
4. Oeste
5. Recoger
6. Dejar

Este es el **action space**

El taxi no puede realizar ciertas acciones en ciertos estados debido a las paredes



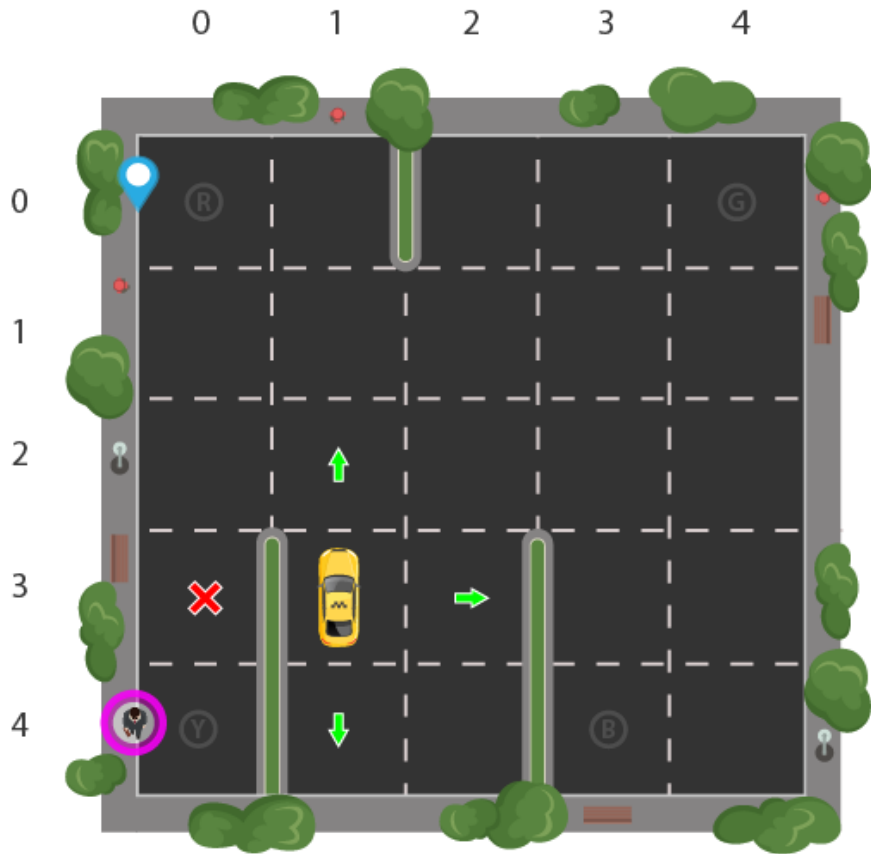
Self-driving cab problem



Recompensas

- El agente debe recibir una recompensa positiva alta por dejar al pasajero en el lugar correcto
- El agente debe ser penalizado si intenta dejar a un pasajero en un lugar equivocado
- El agente debe recibir una pequeña penalización por no llegar al destino después de cada movimiento

Self-driving cab problem: state space



El **state space** considera:

- Las 25 posibles posiciones del taxi
- Las 5 posiciones actuales del pasajero (4 puntos de recogida + dentro del taxi)
- Las 4 posiciones de destino de un pasajero

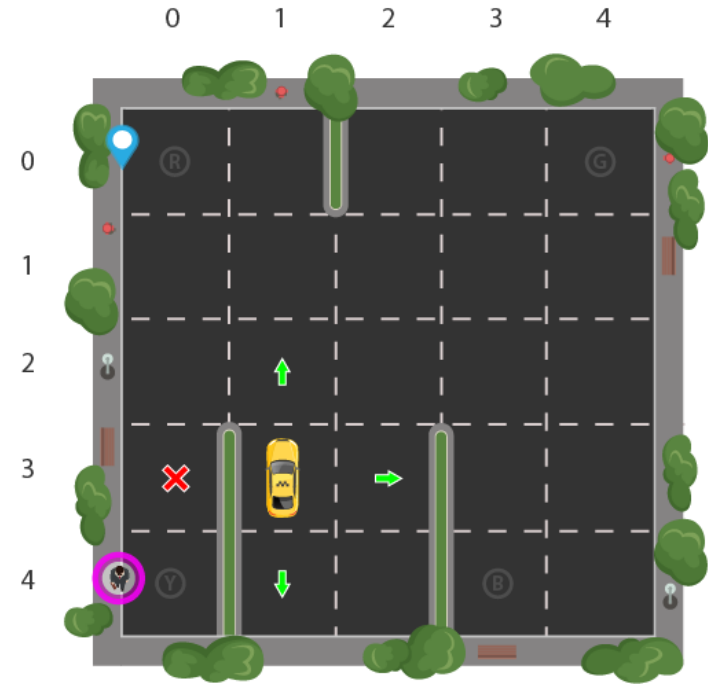
Por tanto, existen:

$$25 \times 5 \times 4 = 500 \text{ estados posibles}$$

Self-driving cab problem: tabla de recompensas

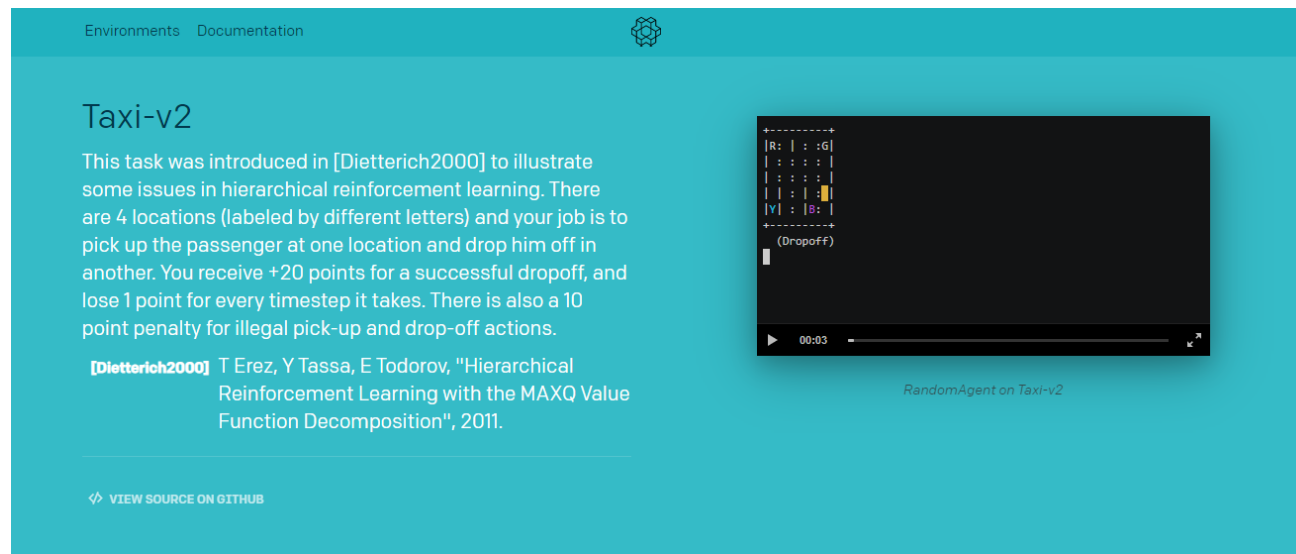
		Acciones					
		Sur	Norte	Este	Oeste	Recoger	Dejar
Estados	0	$R_{0,s}$	$R_{0,n}$	$R_{0,e}$	$R_{0,o}$	$R_{0,r}$	$R_{0,d}$
	1	$R_{1,s}$	$R_{1,n}$	$R_{1,e}$
	2

	498
	499	$R_{499,s}$	$R_{499,n}$	$R_{499,e}$	$R_{499,o}$	$R_{499,r}$	$R_{499,d}$



OpenAI Gym

- Contiene diferentes **entornos** de juego, que podemos conectar a nuestro código para entrenar a nuestro agente
- Proporciona toda la información que nuestro agente requiera, como posibles **acciones**, **recompensas** y **estados**



Documentación: <https://www.gymnasium.dev/>

Práctica Self-driving cab problem



5. Q-learning

Q-learning – Q-table

		Acciones					
Estados		Sur	Norte	Este	Oeste	Recoger	Dejar
	0	0	0	0	0	0	0
	1	0	0	0	0	0	0
	2	0	0	0	0	0	0

	498	0	0	0	0	0	0
	499	0	0	0	0	0	0

El valor de cada celda (llamado **q-value**) se actualizará cada vez que el agente se encuentre en el estado que corresponde a la fila y realice la acción que corresponde a la columna

Si la acción ha sido beneficiosa para el objetivo final, el q-value aumentará

Q-learning – Fórmula q-value

Fórmula para actualizar los q-value:

$$Q(state, action) \leftarrow (1 - \alpha)Q(state, action) + \alpha \left(reward + \gamma \max_a Q(next\ state, all\ actions) \right)$$

- **α** : Tasa de aprendizaje ($0 < \alpha \leq 1$): magnitud de actualización
- **γ** : Factor de descuento ($0 \leq \gamma \leq 1$): determina cuánta importancia queremos dar a las recompensas futuras

Q-learning – Fórmula q-value

Fórmula para actualizar los q-value:

$$Q(state, action) \leftarrow (1 - \alpha)Q(state, action) + \alpha \left(reward + \gamma \max_a Q(next\ state, all\ actions) \right)$$

Se actualiza el valor $Q_{(state, action)}$ teniendo en cuenta:

- El peso $(1 - \alpha)$ del valor Q anterior
- El **valor aprendido**

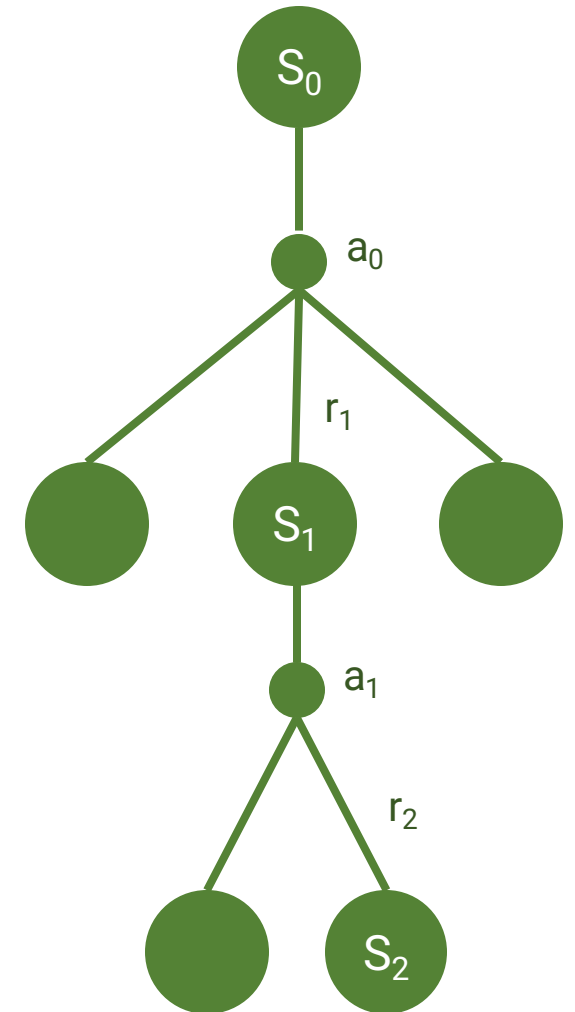
Se aprende la acción adecuada a tomar en el estado actual observando:

- La recompensa para la combinación de estado/acción actual
- Las recompensas máximas para el estado siguiente

Eventualmente, esto hará que nuestro taxi considere la ruta con las mejores **recompensas consecutivas**

Q-learning - Pasos

1. Inicializamos la tabla Q con todos ceros
2. Empezamos a explorar acciones: para el estado inicial (S_0), seleccionamos cualquiera de todas las acciones posibles (a_0)
3. Como resultado de la acción (a_0), nos movemos al siguiente estado (S_1)
4. Para todas las acciones posibles del estado (S_1), seleccionamos la que tenga el valor Q más alto
5. Actualizamos los valores de la tabla Q usando la fórmula q-value
6. Establecemos el siguiente estado como el estado actual
7. Si se alcanza el estado objetivo, finalizamos y repetimos el proceso



Self-driving cab problem: Q-learning

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0

	327	0	0	0	0	0	0

	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0

	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017

	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Después de suficiente exploración aleatoria de acciones, los valores Q tienden a converger

Se incluye un trade-off entre:

- **Exploración:** elegir una acción aleatoria
- **Explotación:** elegir acciones basadas en valores Q ya aprendidos

Práctica Self-driving cab problem

