

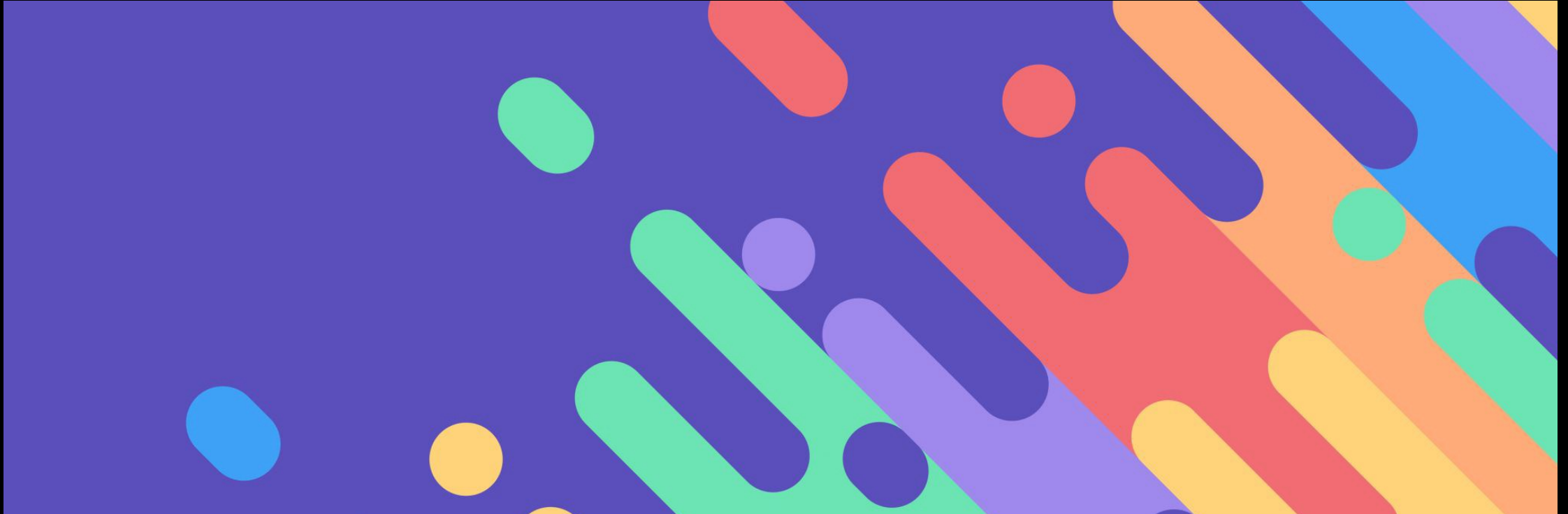
# APRENDIZAJE POR REFUERZO



Inteligencia Artificial

CEIA - FIUBA

Dr. Ing. Facundo Adrián  
Lucianna



---

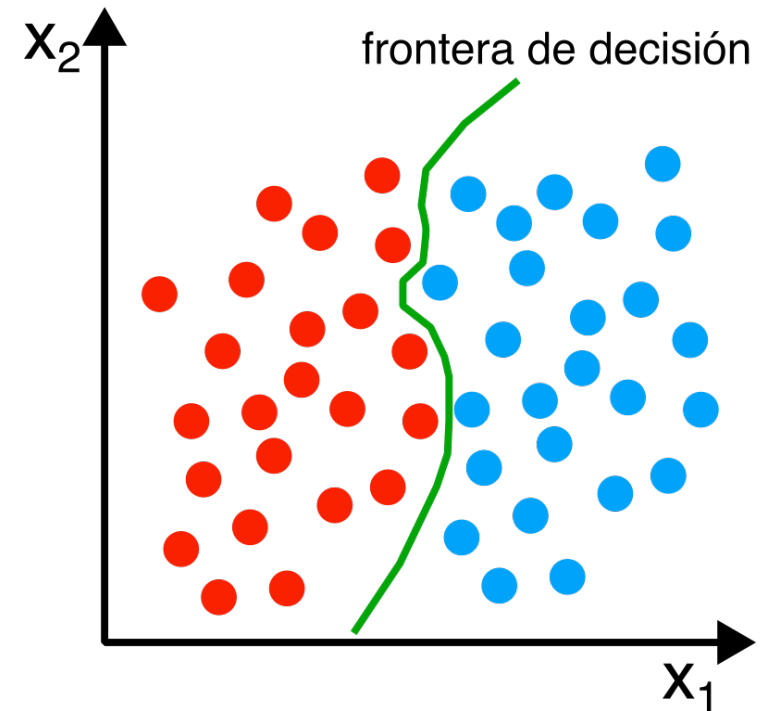
LO QUE VIMOS LA CLASE ANTERIOR...

---

# CLASIFICACIÓN

Es más común encontrarnos con problema de clasificación que de regresión:

- Una persona llega a una guardia con un set de síntomas atribuidos a una de tres condiciones médicas.
- Un servicio de banca online debe determinar si una transacción en el sitio es fraudulenta o no, usando como base la dirección IP, historia de transacciones, etc.
- En base a la secuencia de ADN de un número de pacientes con y sin una enfermedad dada, un genetista debe determinar que mutaciones de ADN genera un efecto nocivo relacionado a la enfermedad o no.



---

# REGRESIÓN LOGÍSTICA

Para clasificar, en vez de modelar la salida, modelamos la probabilidad de que una observación es de una clase u otra. Para ello, podemos modelar a la probabilidad usando una función que nos asegure que siempre tendremos valores entre 0 y 1.

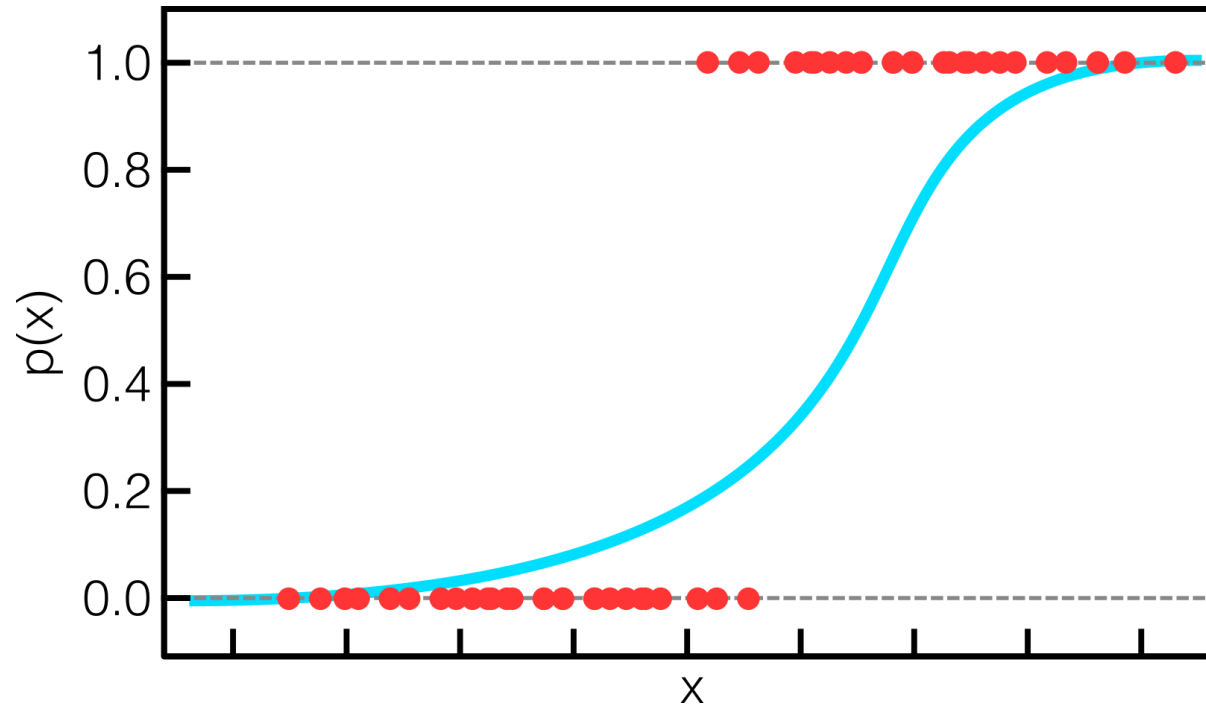
En regresión logística, esto lo resolvemos usando una función sigmoide:

$$p(x) = \frac{e^{b+w_0x}}{1 + e^{b+w_0x}} = \frac{1}{1 + e^{-(b+w_0x)}}$$

---

# REGRESIÓN LOGÍSTICA

Lo que visualmente se observa:



---

# CLASIFICADOR BAYESIANO INGENUO

## Teorema de Bayes

Este teorema es uno de los teoremas más importantes de probabilidad, y uno que hasta el día de hoy genera divisiones en el plano filosófico por su implicancia

Este describe la probabilidad de un evento, basado en conocimiento previo de condiciones que pueden estar relacionados con el evento.

*Por ejemplo, si se sabe que el riesgo de desarrollar problemas de salud aumenta con la edad, el teorema de Bayes permite evaluar con mayor precisión el riesgo para un individuo de una edad conocida condicionándolo en relación con su edad, en lugar de asumir que el individuo es típico de la población en su conjunto.*

---

# CLASIFICADOR BAYESIANO INGENUO

Una de las aplicaciones de este teorema es el denominado clasificador bayesiano ingenuo.

Este clasificador utiliza la probabilidad de observar atributos, dado un resultado, para estimar la probabilidad de observar el resultado  $y_j$ , dado un conjunto de atributos.



---

# APRENDIZAJE POR REFUERZO



---

# APRENDIZAJE POR REFUERZO

En las últimas clases vimos dos áreas conocidas de aprendizaje supervisados. Dejaremos a Aprendizaje no supervisado para AMq1, y vayamos al **Aprendizaje por Refuerzo**.

*El aprendizaje por refuerzo es un enfoque de aprendizaje automático donde un agente aprende a tomar decisiones óptimas al interactuar con un entorno, maximizando una señal de recompensa a lo largo del tiempo.*

Para este tipo de aprendizaje es más fácil de pensar usando el concepto de agente.

---

# APRENDIZAJE POR REFUERZO

Un agente puede aprender a jugar al ajedrez con aprendizaje supervisado, proporcionándole ejemplos de situaciones de juego con los mejores movimientos para dichas situaciones. Pero si no hay un profesor que proporcione ejemplos, ¿qué puede hacer el agente?

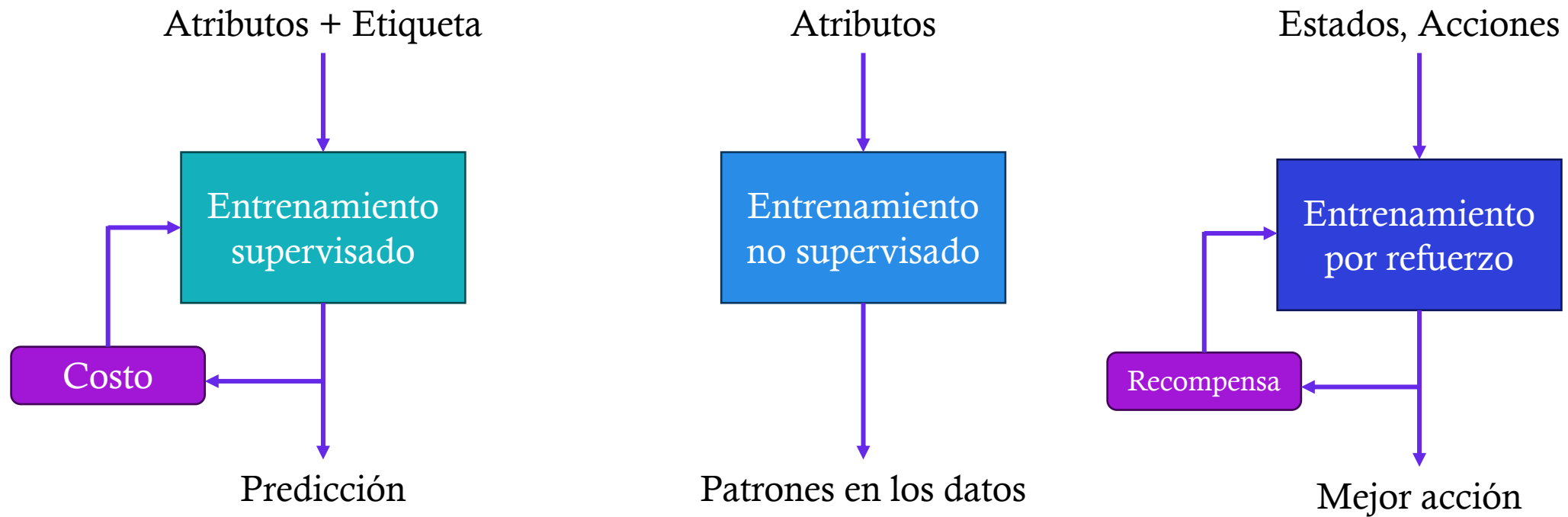
**Intentando movimientos aleatorios**, el agente puede eventualmente **construir un modelo de predicción** de su entorno: cómo estará el tablero después de que haga un movimiento e incluso cómo es probable que el oponente responda en una situación dada.

El problema es el siguiente: sin cierta realimentación de lo que es bueno y de lo que es malo, el agente no tendrá razones para decidir qué movimiento hacer.

El agente necesita saber que algo bueno ha ocurrido cuando gana y que algo malo ha ocurrido cuando pierde. Esta clase de realimentación se denomina **recompensa**, o **refuerzo**.

---

# APRENDIZAJE POR REFUERZO



---

# APRENDIZAJE POR REFUERZO

**Aprendizaje por refuerzo** se utiliza más comúnmente para resolver una clase diferente de problemas del mundo real, como una tarea de control o una tarea de decisión, en las que se opera un sistema que interactúa con el mundo real.

Es útil para una variedad de aplicaciones como:

- Operar un dron o un vehículo autónomo
- Manipular un robot para navegar por el entorno y realizar diversas tareas.
- Gestionar una cartera de inversiones y tomar decisiones comerciales.
- Jugar juegos como Go, Ajedrez, videojuegos.



---

# PROCESO DE DECISIÓN DE MÁRKOV

---

# PROCESO DE DECISIÓN DE MÁRKOV

Un proceso de decisión de Márkov (MDP) es un proceso de control estocástico en tiempo discreto. Proporciona un marco matemático para modelar la toma de decisiones en situaciones en las que los resultados son en **parte aleatorios** y en parte están bajo el **control del agente**.

En cada paso temporal, el proceso se encuentra en el estado  $s$ , y agente puede elegir cualquier acción  $a$  que esté disponible en el estado  $s$ . El proceso responde en el siguiente paso temporal pasando aleatoriamente a un nuevo estado  $s'$ , y ofreciendo al responsable de la toma de decisiones la recompensa correspondiente  $R(s, a, s')$ .

La probabilidad de que el proceso pase a su nuevo estado  $s'$  está influida por la acción elegida. En concreto, viene dada por la función de transición de estado  $P(s, a, s')$ . Así, el siguiente estado  $s'$  depende del estado actual  $s$  y de la acción del decisor  $a$ . Pero dado  $s$  y  $a$ , es condicionalmente independiente de todos los **estados y acciones anteriores**.

Los procesos de decisión de Márkov son una extensión de las cadenas de Márkov; la diferencia es la adición de acciones y recompensas.

---

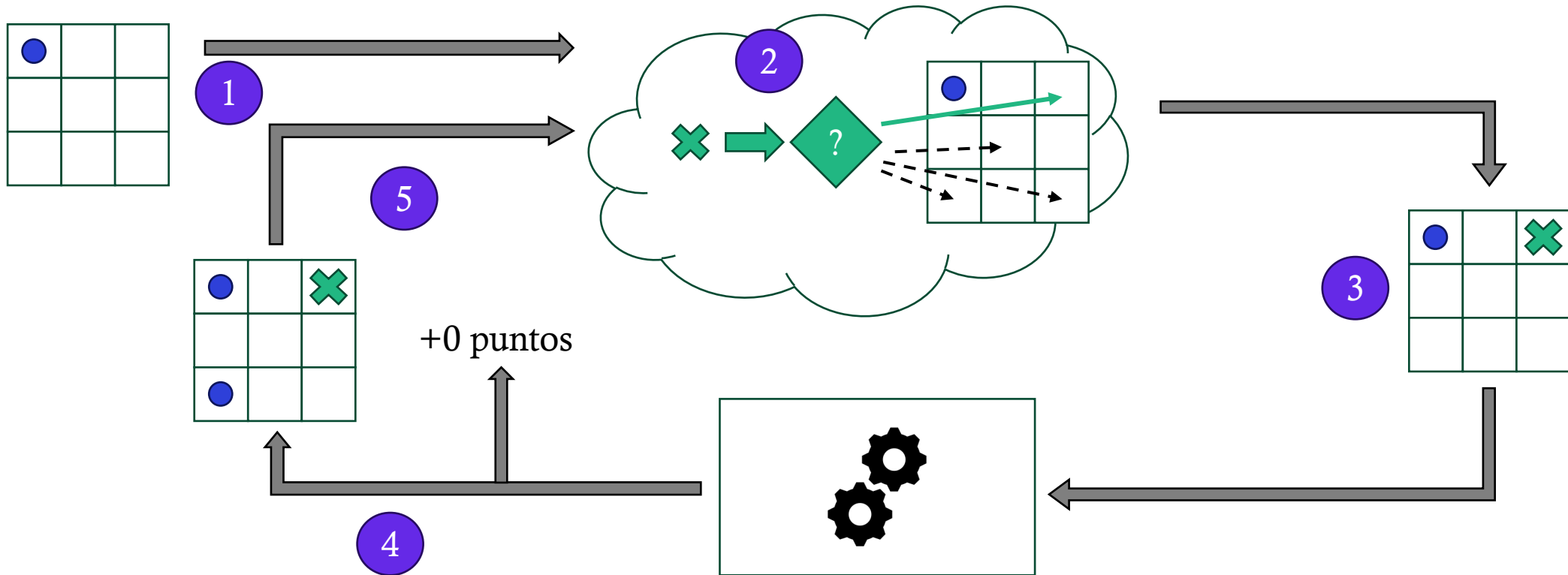
# PROCESO DE DECISIÓN DE MÁRKOV

MDP tiene 5 componentes:

- **Agente**
- **Ambiente**
- **Estado**
- **Acción**
- **Recompensa**: es el refuerzo positivo o negativo que el agente recibe del entorno como resultado de sus acciones. Es una forma de evaluar la "bondad" o la "maldad" de una acción en particular.

# PROCESO DE DECISIÓN DE MÁRKOV

Veamos un ejemplo usando tatetí en ejemplo de funcionamiento de MDP:





---

# PROCESO DE DECISIÓN DE MÁRKOV

La ejecución de un MDP puede ser descrita como una secuencia de ocurrencias en términos de estado, acción y recompensa sobre una secuencia de pasos temporales.

En tareas que tienen un final, la secuencia es episódica. En un juego de tatetí, un episodio es cada partida que se juega. **Cada episodio es independiente del siguiente**. Por lo que el funcionamiento de Aprendizaje por refuerzo (AR) se repite en episodios, donde cada episodio, se repite varios pasos de tiempo.

Por otro lado, las tareas de AR no tienen fin se conocen como *Tareas Continuas* y pueden continuar para siempre.

---

# PROCESO DE DECISIÓN DE MÁRKOV

El **agente** y el **entorno** controlan las transiciones estado-acción:

El MDP opera alternando entre el **agente** realizando una acción y luego el **entorno** haciendo algo. En cada paso de tiempo:

- Dado el estado actual, la siguiente acción la decide el **agente** de un grupo de posibles acciones.
- Dado el estado actual y la siguiente acción elegida por el agente, la transición al siguiente estado y la recompensa están controladas por el **entorno**. Esta parte del entorno tomando una decisión es lo que el agente ve como que sus acciones son probabilísticas.

---

# PROCESO DE DECISIÓN DE MÁRKOV

¿Cómo el agente toma una decisión?

Este es precisamente el problema de aprendizaje por refuerzo que queremos resolver. Para ello se utiliza tres conceptos:

- **Retorno**: A medida que el agente ejecuta pasos de tiempo, acumula recompensas en cada paso de tiempo. La recompensa acumulada es lo que se llama retorno.
- **Política**: La política es la estrategia que se sigue para elegir una acción.
- **Valor**: Indica el Retorno esperado siguiendo alguna Política

---

# PROCESO DE DECISIÓN DE MÁRKOV

## Retorno

Cuando se calcula el retorno, en lugar de sumar todas las recompensas, se aplica un factor de descuento  $\gamma$  para ponderar las recompensas posteriores a lo largo del tiempo ( $\gamma$  está entre 0 y 1). Estos se conocen como recompensas con descuento:

$$Return = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^n r_n$$

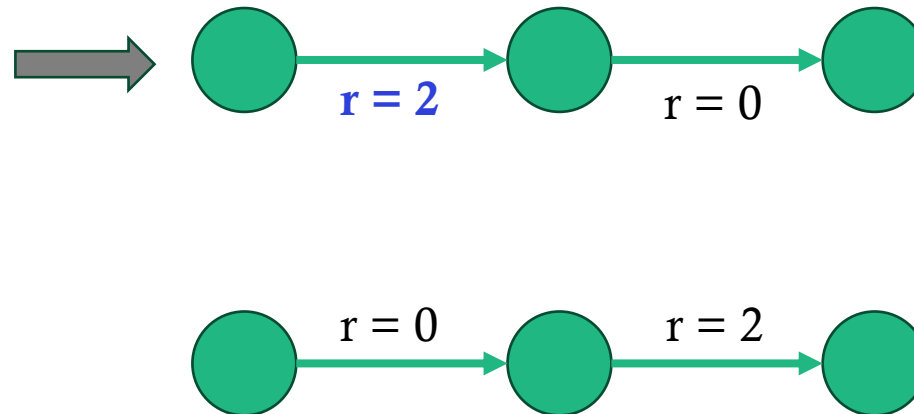
De esta manera, las recompensas acumulativas no crecen infinitamente a medida que el número de pasos de tiempo se vuelve muy grande.

---

# PROCESO DE DECISIÓN DE MÁRKOV

## Retorno

La recompensa inmediata es más valiosa que la recompensa posterior:

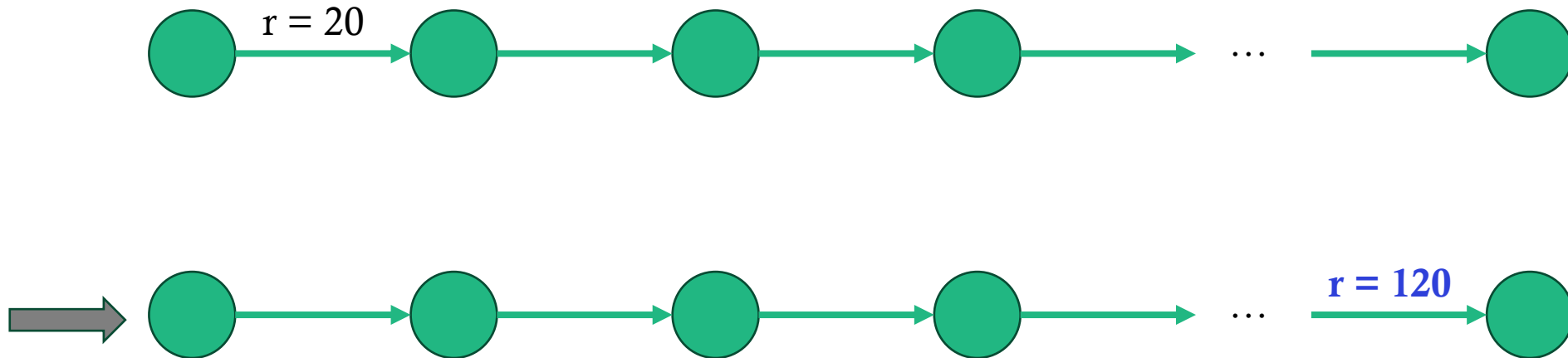


---

# PROCESO DE DECISIÓN DE MÁRKOV

## Retorno

Si el agente tiene que elegir entre obtener alguna recompensa ahora u obtener una recompensa mucho mayor más adelante, lo más probable es que la recompensa mayor sea preferible. Esto se debe a que queremos que el agente mire los rendimientos totales en lugar de las recompensas individuales.



---

# PROCESO DE DECISIÓN DE MÁRKOV

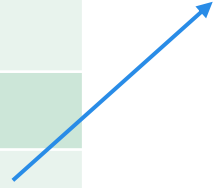
## Política

Esto es básicamente que define el modelo, es la acción que va a tomar dado un estado que se encuentre.

La forma más trivial es volver a la tabla de estado-acción que vimos en la clase 2.

	a1	a2	a3
s1	$\pi(a_1 s_1)$	...	...
s2	...	...	...
s3	...	...	...
s4	...	...	$\pi(a_3 s_4)$

Probabilidad  
de realizar la  
acción  $a_3$   
cuando se  
encuentra en  
el estado  $s_4$



---

# PROCESO DE DECISIÓN DE MÁRKOV

## Política

Las políticas pueden ser:

- **Deterministas**: Una política determinista es una política en la que el agente siempre elige la misma acción fija cuando alcanza un estado particular.
- **Estocásticas**: La política estocástica es una política en la que el agente varía las acciones que elige para un estado, en función de cierta probabilidad para cada acción. Podría hacer esto mientras juega, por ejemplo, para que no se vuelva completamente predecible.



---

# PROCESO DE DECISIÓN DE MÁRKOV

## Política

El agente realmente no tiene una política útil cuando comienza y no tiene idea de qué acción debe tomar en un estado determinado. Luego, al utilizar el algoritmo de aprendizaje por refuerzo, aprende lentamente una política útil que puede utilizar.

Hay tantas políticas posibles, ¿cuál debería utilizar el Agente?

*El objetivo del agente es seguir una Política que maximice su Retorno.* Entonces, de todas las políticas que el agente podría seguir, quiere elegir la mejor, es decir. el que le da mayor retorno.

---

# PROCESO DE DECISIÓN DE MÁRKOV

## Valor

Digamos que el agente se encuentra en un estado particular. Además, digamos que al agente se le ha dado de alguna manera una política,  $\pi$ .

Si parte de ese estado y siempre elige acciones basadas en esa política, ¿cuál es el rendimiento que podría esperar obtener?

Este rendimiento promedio a largo plazo, o rendimiento esperado, se conoce como **valor** de ese estado en particular según la política  $\pi$ .

---

# PROCESO DE DECISIÓN DE MÁRKOV

## Valor

Tenemos dos tipos de Valores:

- **Valor de Estado:** el rendimiento esperado de un estado determinado, mediante la ejecución de acciones basadas en una política determinada  $\pi$  desde ese estado en adelante.
- **Valor de Estado-Acción (valor Q):** el rendimiento esperado al realizar una acción determinada desde un estado determinado y luego ejecutar acciones basadas en una política determinada  $\pi$  después de eso.

---

# PROCESO DE DECISIÓN DE MÁRKOV

Relación entre los tres:

- La **Recompensa** es la recompensa inmediata obtenida por una sola acción.
- El **Retorno** es el total de todas las recompensas con descuento obtenidas hasta el final de ese episodio.
- El **Valor** es el rendimiento medio (también conocido como rendimiento esperado) de muchos episodios.

Se puede pensar en el **Valor** de la siguiente manera. El agente aprende de la experiencia. A medida que interactúa con el entorno y completa episodios, obtiene los retornos de cada episodio.

A medida que acumula más experiencia (es decir, obtiene retornos por más y más episodios), tiene una idea de qué estados y qué acciones en esos estados producen el mayor retorno.

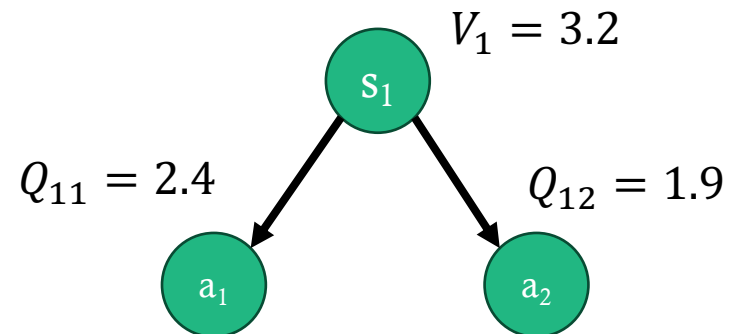
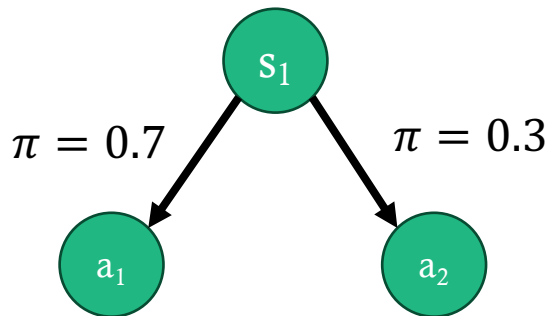
Almacena esta *experiencia* como **Valor**

---

# PROCESO DE DECISIÓN DE MÁRKOV

Utilizando la función de Valor para comparar políticas

Dadas dos políticas, podemos determinar las funciones Valor de Estado o Valor Estado-Acción correspondientes para cada una de esas políticas, siguiendo la política y evaluando los rendimientos.



---

# PROCESO DE DECISIÓN DE MÁRKOV

Utilizando la función de Valor para comparar políticas

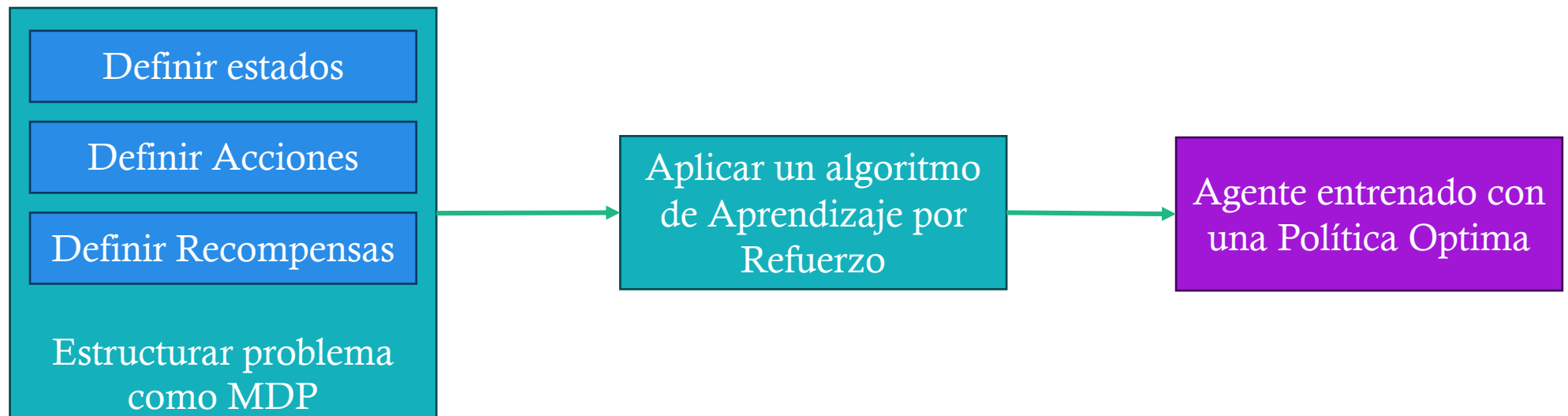
Dado que podemos comparar políticas para determinar cuáles son *buenas* y cuáles son *malas*, también podemos utilizar eso para encontrar la *mejor* política. Esto se conoce como **Política Óptima**.

---

# PROCESO DE DECISIÓN DE MÁRKOV

Resolviendo el problema de Aprendizaje por Refuerzo encontrando la política óptima

Entonces, podemos entender como entrenar un agente usando Aprendizaje por Refuerzo, usando MDP. Lo que se busca encontrar la **Política Óptima** para el agente. Una vez que tiene la Política Óptima, simplemente usa esa política para elegir acciones de cualquier estado.





---

# CATEGORÍAS DE SOLUCIONES



---

# CATEGORÍAS DE SOLUCIONES

## Predicción vs Control

Los problemas se pueden dividir en dos tipos:

- **Problemas de predicción:** Se proporciona una política como entrada y el objetivo es generar la función de Valor. No es necesario que sea la Política Óptima.
- **Problemas de control:** No se proporciona información y el objetivo es explorar el espacio de políticas y encontrar la Política Óptima.

---

# CATEGORÍAS DE SOLUCIONES

Hay muchos algoritmos que buscan encontrar la **Política Óptima** para el agente.  
Hay dos grandes categorías:

- Basado en modelo
- Sin modelo

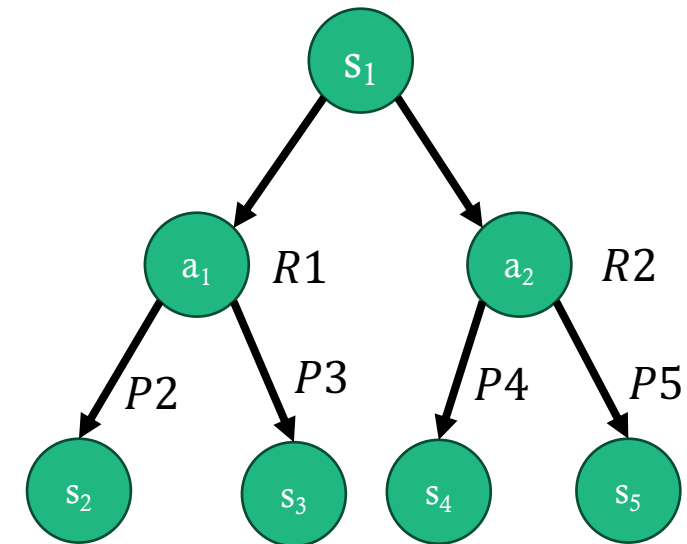
---

# CATEGORÍAS DE SOLUCIONES

## Basado en modelo

Los enfoques basados en modelos se utilizan cuando se conoce el funcionamiento interno del entorno. Podemos decir de manera confiable qué próximo estado y recompensa generará el entorno cuando se realice alguna acción desde algún estado actual.

Estos modelos son los que se conocen como planificación. Son aquellos que ya vimos en clase 2 y 3 (y que hay más avanzados). No es AR.



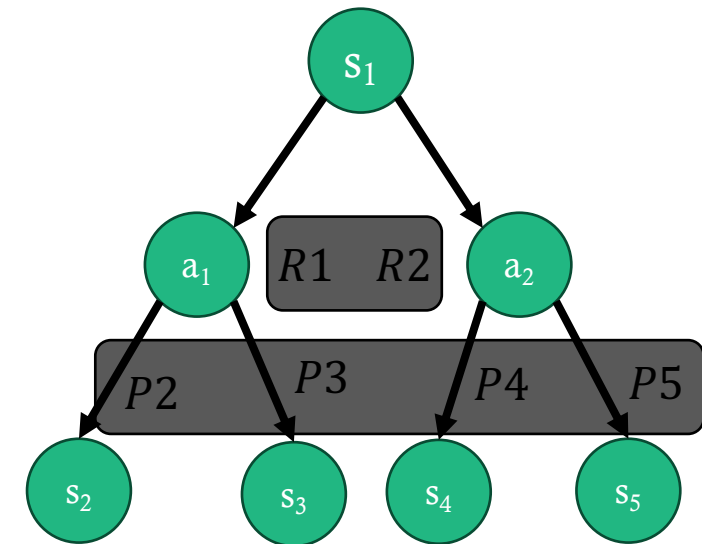
---

# CATEGORÍAS DE SOLUCIONES

Libre de modelos

Dado que el funcionamiento interno del entorno es invisible para el agente,

*¿Cómo observa el algoritmo el comportamiento del entorno?*



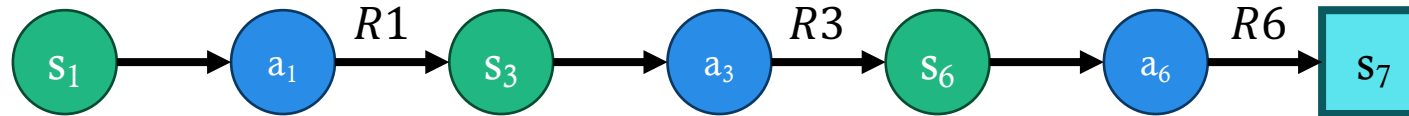


---

# CATEGORÍAS DE SOLUCIONES

## Libre de modelos

A medida que el agente da cada paso, sigue un camino. La trayectoria del agente se convierte en los *datos de entrenamiento* del algoritmo.





---

# ECUACIÓN DE BELLMAN

---

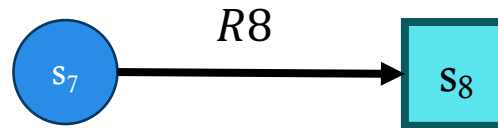
# ECUACIÓN DE BELLMAN

La ecuación de Bellman es un concepto fundamental en el campo del Aprendizaje por Refuerzo.

*Esencialmente, describe cómo el valor de estar en un estado particular bajo una política específica se relaciona con el valor de estar en el próximo estado y las recompensas esperadas.*

Vayamos paso a paso para crear una intuición de esto.

Consideremos la recompensa al realizar una acción desde un estado para alcanzar un estado terminal:

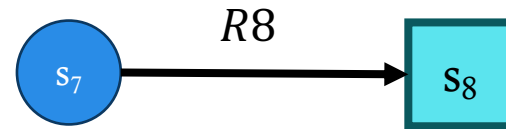




---

# ECUACIÓN DE BELLMAN

El Retorno de estado es igual a la Recompensa obtenida por tomar esa acción.



$G_7$

Retorno de  $s_7$  = Recompensa de la acción tomada desde  $s_7$

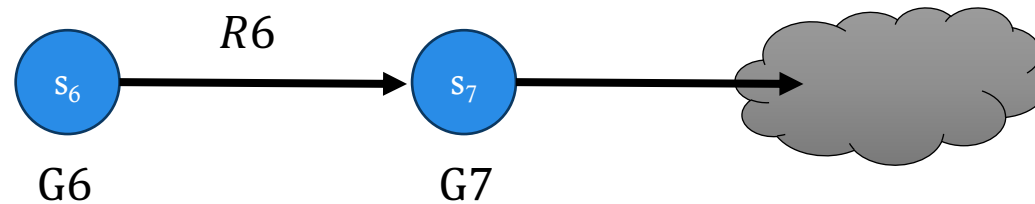
$$G_7 = R_8$$

---

# ECUACIÓN DE BELLMAN

Veamos ahora el estado anterior  $S_6$ . El retorno de  $S_6$  es la recompensa obtenida al realizar la acción para llegar a  $S_7$  más cualquier retorno con descuento que obtendríamos de  $S_7$ .

Lo importante es que ya no se necesita conocer los detalles de los pasos individuales realizados más allá del  $S_7$ .



$$\begin{aligned} \text{Retorno de } s_6 &= \text{Recompensa de } s_7 + \text{Retorno (descontado) de } s_7 \text{ en adelante} \\ G_6 &= R_6 + \gamma G_7 \end{aligned}$$

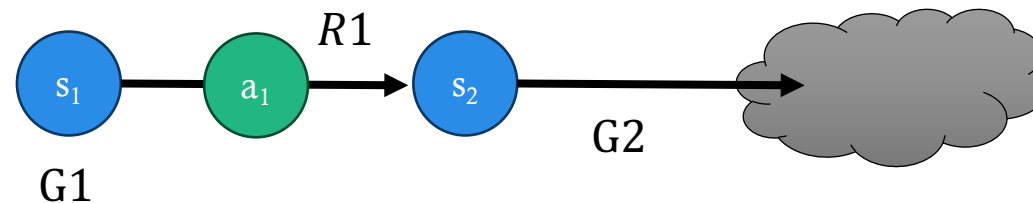
---

# ECUACIÓN DE BELLMAN

En general, el retorno de cualquier estado se puede descomponer en dos partes:

- La recompensa inmediata de la acción para llegar al siguiente estado
- Más el retorno descontado del siguiente estado siguiendo la misma política para todos los pasos posteriores.

Esta es la *ecuación de Bellman*.



Retorno de  $s_1$  = Recompensa de  $s_1$  + Retorno (descontado) de  $s_2$

$$G1 = R1 + \gamma G2$$

---

# ECUACIÓN DE BELLMAN

En general, el retorno de cualquier estado se puede descomponer en dos partes:

- La recompensa inmediata de la acción para llegar al siguiente estado
- Más el retorno descontado del siguiente estado siguiendo la misma política para todos los pasos posteriores.

Esta es la *ecuación de Bellman*.

$$G_t = R_t + \gamma G_{t+1}$$

---

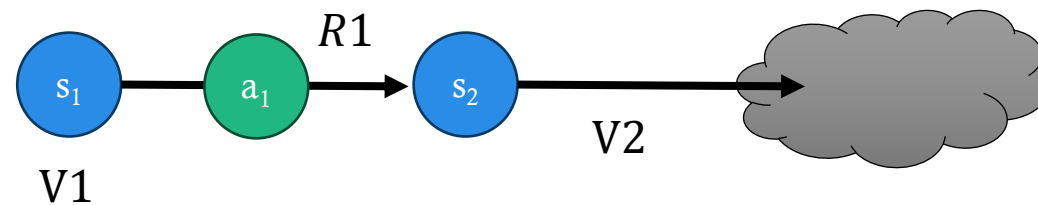
# ECUACIÓN DE BELLMAN

## Ecuación para Valor de estado

El retorno es la recompensa con descuento por un solo camino. El **Valor de Estado** se obtiene tomando *el promedio del rendimiento a lo largo de muchos caminos* (la expectativa de rendimiento).

El **Valor de Estado** se puede descomponer en dos partes:

- La recompensa inmediata de la siguiente acción para llegar al siguiente estado
- El **Valor** descontado de ese siguiente estado siguiendo la política para todos los pasos posteriores.



Valor de  $s_1$  = Recompensa de  $s_1$  + Valor (descontado) de  $s_2$

$$V(s_1) = E[R_1 + \gamma V(s_2)]$$

---

# ECUACIÓN DE BELLMAN

## Ecuación para Valor de estado

El retorno es la recompensa con descuento por un solo camino. El **Valor de Estado** se obtiene tomando *el promedio del rendimiento a lo largo de muchos caminos* (la expectativa de rendimiento).

El **Valor de Estado** se puede descomponer en dos partes:

- La recompensa inmediata de la siguiente acción para llegar al siguiente estado
- El **Valor** descontado de ese siguiente estado siguiendo la política para todos los pasos posteriores.

$$V(s_t) = E[R_t + \gamma V(s_{t+1})]$$

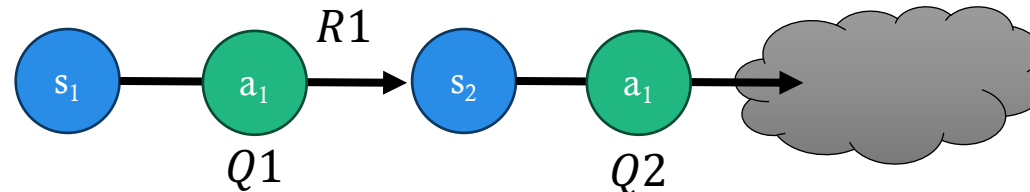
---

# ECUACIÓN DE BELLMAN

## Ecuación para Valor de estado-acción

El **Valor de Estado-Acción** también se puede descomponer en dos partes:

- La recompensa inmediata de la siguiente acción para llegar al siguiente estado
- El **Valor** descontado de ese siguiente estado siguiendo la política para todos los pasos posteriores.



Valor de  $s_1$ :  $a_1$  = Recompensa de aplicar la siguiente acción + Valor (descontado) de  $s_2$ :  $a_2$

$$Q(s_1, a_1) = E[R_1 + \gamma Q(s_2, a_2)]$$

$$Q(s_t, a_t) = E[R_t + \gamma Q(s_{t+1}, a_{t+1})]$$

---

# ECUACIÓN DE BELLMAN

¿Por qué es útil la ecuación de Bellman?

- **El retorno se puede calcular de forma recursiva sin llegar al final del episodio:** Los episodios pueden ser muy largos (y costosos de recorrer) o pueden ser interminables. En cambio, podemos utilizar esta relación recursiva. Si conocemos el retorno del siguiente paso, entonces podemos usar para inmediatamente atrás.
- **Podemos trabajar con estimaciones, en lugar de valores exactos:** Hay dos formas de calcular lo mismo:
  - Una es el retorno del estado actual
  - Otra es la recompensa de un paso más el retorno del siguiente paso.

Dado que es muy costoso medir el retorno real desde algún estado (hasta el final del episodio), se usa retornos estimados. Se calcula las estimaciones de las dos maneras mencionadas y se comprueba qué tan correctas son estas estimaciones comparando los dos resultados.

La diferencia nos dice cuánto **error cometimos en las estimaciones**. Esto ayuda a mejorar las estimaciones revisándolas de una manera que reduzca ese error (aprendizaje).





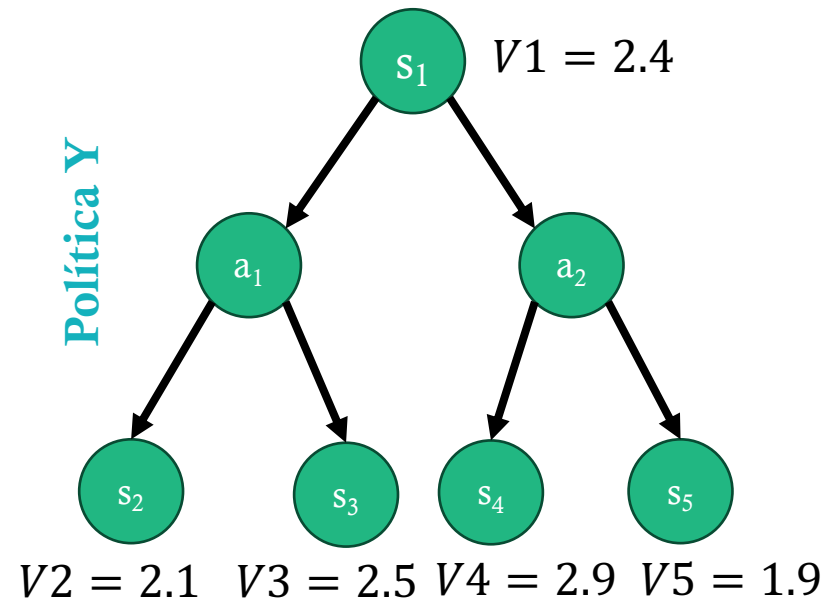
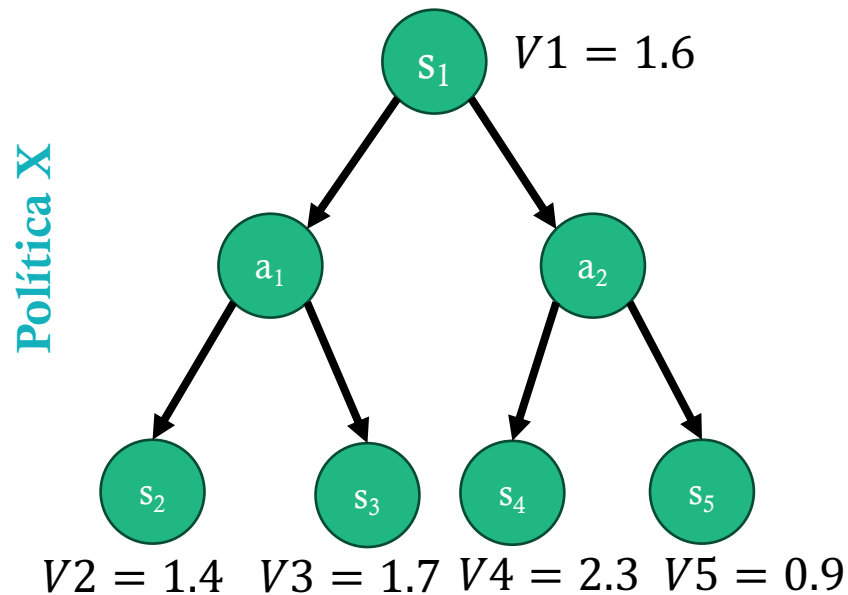
---

# ALGORITMOS BASADOS EN POLÍTICA O EN VALOR

# POLÍTICA O VALOR

Si usamos un algoritmo que se basa en valor para entrenarse:

- Se dice la política Y es mejor que la política X si la **función de valor** de Y es mayor que la de X.



---

# POLÍTICA O VALOR

Si usamos un algoritmo que se basa en valor para entrenarse:

- Si se sigue encontrando políticas cada vez mejores, eventualmente se será capaz de encontrar la mejor política, es decir la **Política Óptima**.

Política X

	a1	a2
s <sub>1</sub>	1.8	1.2
s <sub>2</sub>	1.5	1.3
s <sub>3</sub>	1.2	1.6
s <sub>4</sub>	0.8	1.8

Política Y

	a1	a2
s <sub>1</sub>	2.8	2.2
s <sub>2</sub>	2.5	1.9
s <sub>3</sub>	2.3	1.8
s <sub>4</sub>	2.7	3.0

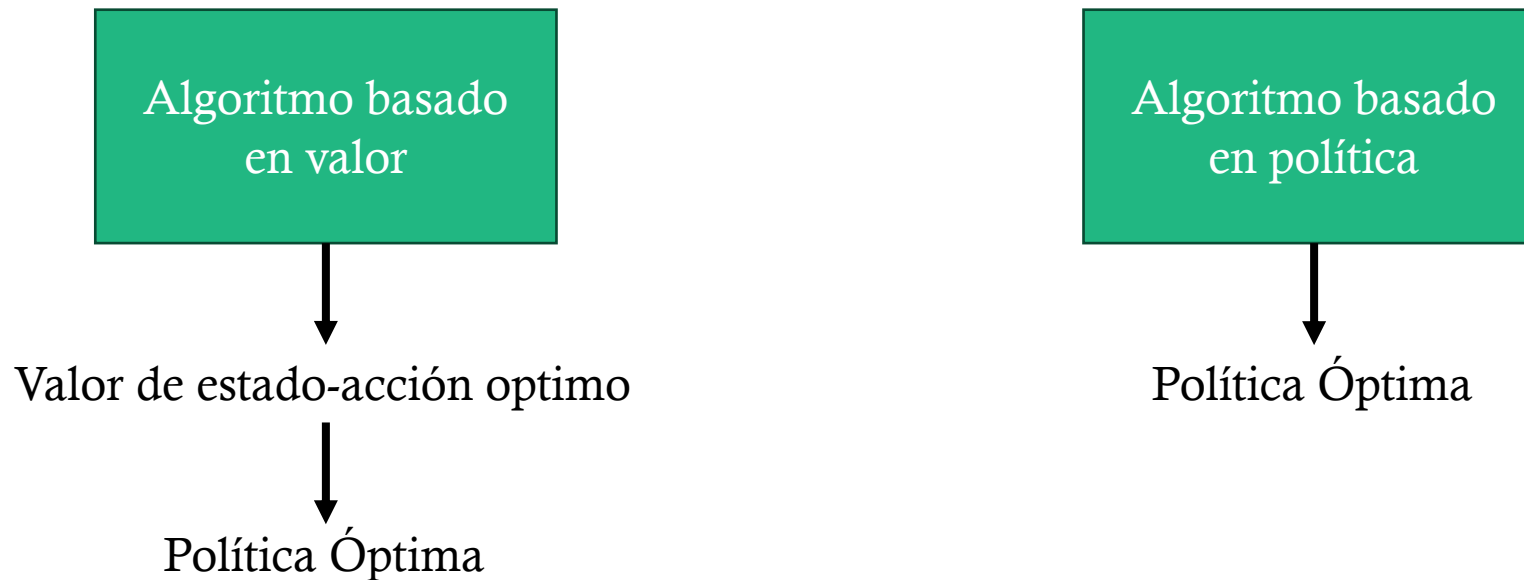
Política Z

	a1	a2
s <sub>1</sub>	3.0	3.3
s <sub>2</sub>	3.2	3.4
s <sub>3</sub>	3.6	3.1
s <sub>4</sub>	3.0	2.8

---

# POLÍTICA O VALOR

Dado esta equivalencia de encontrar el Valor de Estado-Acción óptimo es lo mismo que encontrar la Política Óptima, hay algoritmos de AR que:

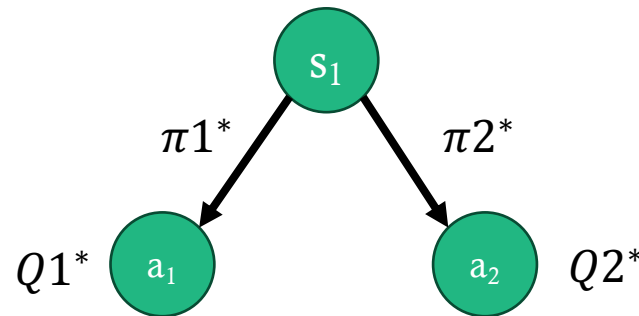


---

# POLÍTICA O VALOR

¿Pero, como hace para seleccionarse la política a raíz del valor?

Una vez que se encuentra el Valor Óptimo de Estado-Acción, se puede obtener fácilmente la Política Óptima eligiendo la acción con el valor de Estado-Acción más alto.



$\pi^* = 1$  para la acción que corresponda al  $\max(Q1^*, Q2^*)$

$\pi^* = 0$  para las restantes

---

# POLÍTICA O VALOR

¿Pero, como hace para seleccionarse la política a raíz del valor?

Generalmente, la Política Óptima es en este proceso **determinista** ya que siempre elige la mejor acción.

Sin embargo, la Política Óptima puede ser estocástica si hay un empate entre dos valores  $Q$ . En ese caso, la Política Óptima elige cualquiera de las dos acciones correspondientes con igual probabilidad.

*En juego con oponentes, una política óptima estocástica es necesaria porque una política determinista daría como resultado que el agente realice movimientos predecibles que su oponente podría derrotar fácilmente.*

---

# POLÍTICA O VALOR

¿Pero, como hace para seleccionarse la política a raíz del valor?

Un punto importante que, para buscar la Política Óptima, necesitamos usar algoritmos basados en Valores de Estado-Acción, ya que el valor de Estado solo, no nos permite determinar que acciones.

*Los algoritmos basados en Valor de Estado se usan en problemas de predicción, mientras que los primeros en problemas de control.*



---

# SOLUCIONES ITERATIVAS



---

# SOLUCIONES ITERATIVAS

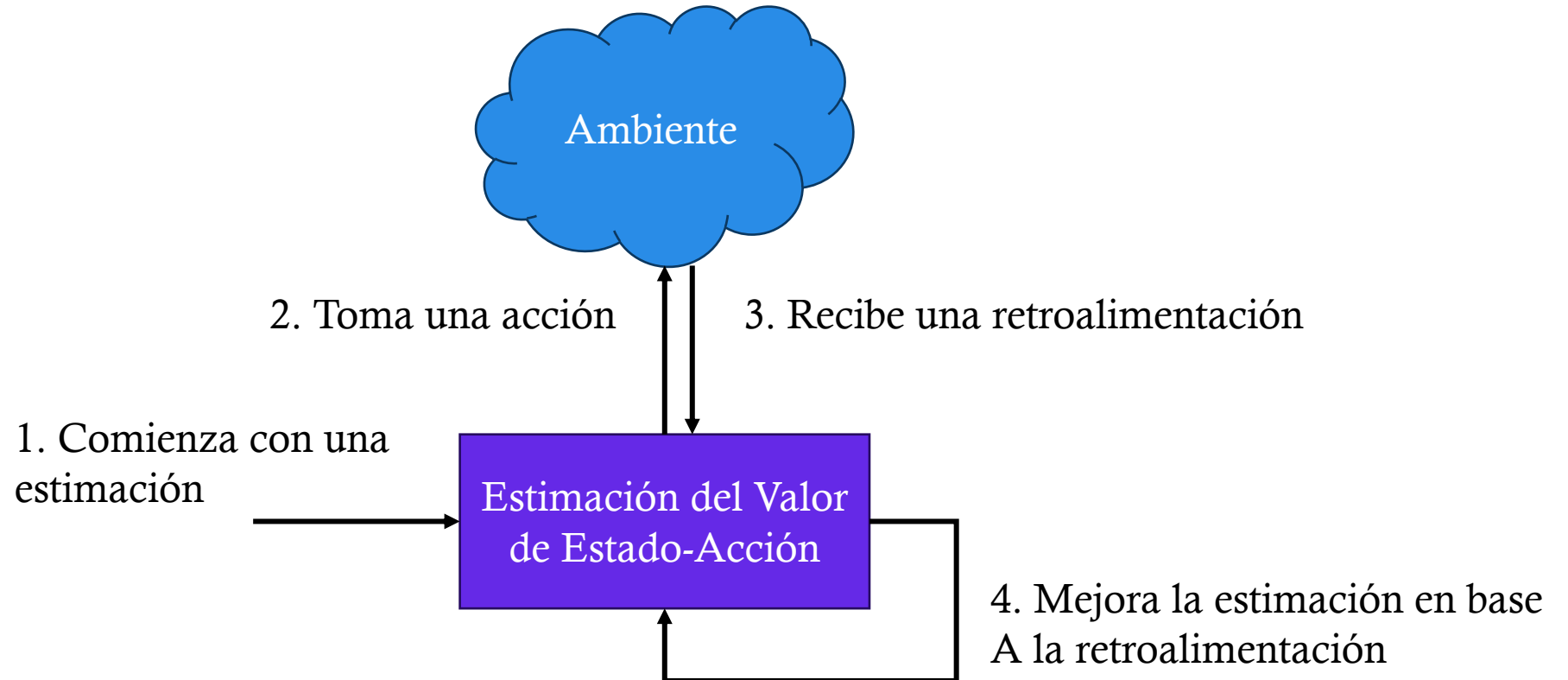
Existen varios algoritmos basados en valores y basados en políticas. Aunque podemos reducir a unos pocos principios esenciales que todos emplean.

En un nivel alto, todos los algoritmos, tanto los basados en valores como los basados en políticas, arrancan con una estimación inicial que van ajustando en iteraciones posteriores. Para ello, realizan cuatro operaciones básicas:

1. Inician estimaciones
2. Toma una acción
3. Obtiene retroalimentación de ambiente
4. Mejora la estimación

---

# SOLUCIONES ITERATIVAS



---

# SOLUCIONES ITERATIVAS

Inicializa estimaciones

El algoritmo basado en **valores** utiliza una tabla de **valor de estado-acción óptimo estimado**, mientras que un algoritmo basado en **políticas** utiliza una **tabla de política óptima estimada** con probabilidades para cada acción en cada estado.

	a1	a2
s <sub>1</sub>	0	0
s <sub>2</sub>	0	0
s <sub>3</sub>	0	0
s <sub>4</sub>	0	0

Por ejemplo, se inicializa todo en cero

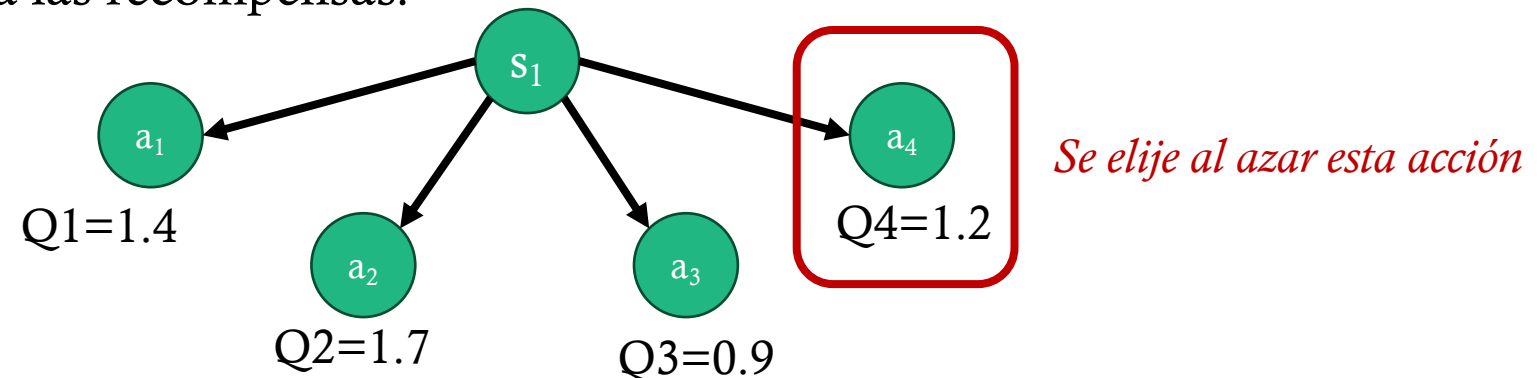
---

# SOLUCIONES ITERATIVAS

## Toma una acción

El agente necesita encontrar el equilibrio adecuado entre **Exploración** y **Explotación**, para tomar una acción.

**Exploración**: cuando se comienza a aprender, no se tiene idea de qué acciones son *buenas* y cuáles son *malas*. Entonces se pasa por un proceso de descubrimiento en el que se prueba diferentes acciones al azar y se observa las recompensas.



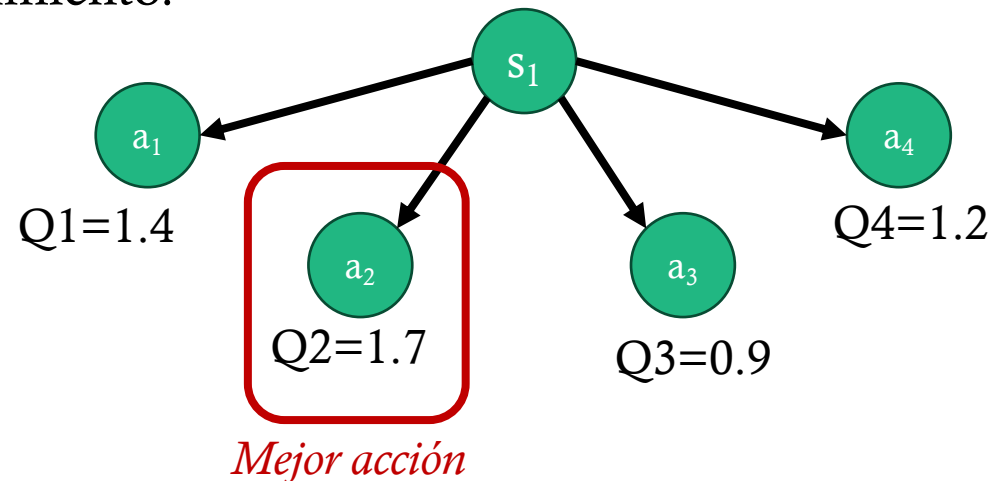
---

# SOLUCIONES ITERATIVAS

## Toma una acción

El agente necesita encontrar el equilibrio adecuado entre **Exploración** y **Explotación**, para tomar una acción.

**Explotación**: en el otro extremo, cuando el modelo está completamente entrenado, ya se ha explorado todas las acciones posibles, por lo que se puede elegir las mejores acciones que producirán el máximo rendimiento.



# SOLUCIONES ITERATIVAS

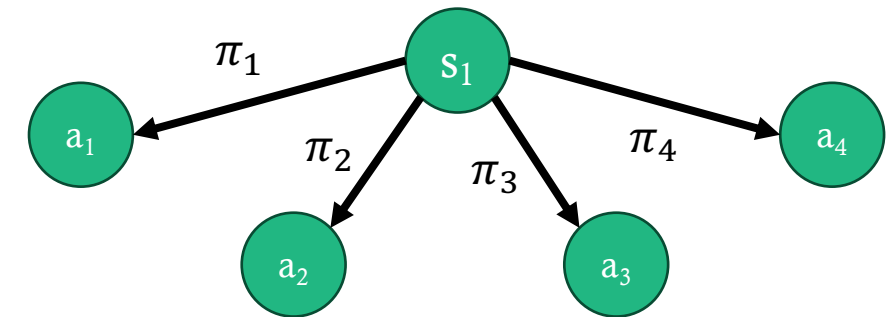
## Toma una acción

Los agentes basados en políticas y los agentes basados en valores utilizan diferentes métodos para lograrlo.

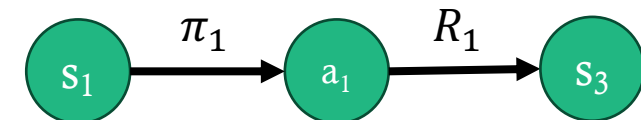
**Basado en política:** Utiliza sus propias estimaciones para elegir una acción. La tabla de políticas de un agente basado en políticas ya tiene una estimación continua de la política óptima, que le indica la probabilidad deseada de todas las acciones que puede realizar en cualquier estado determinado.

Entonces simplemente elige una acción basada en las probabilidades de esa política óptima estimada.

Cuanto mayor sea la probabilidad de una acción, más probabilidades habrá de que la elijan.



	a1	a2	a3	a4
s1	$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$



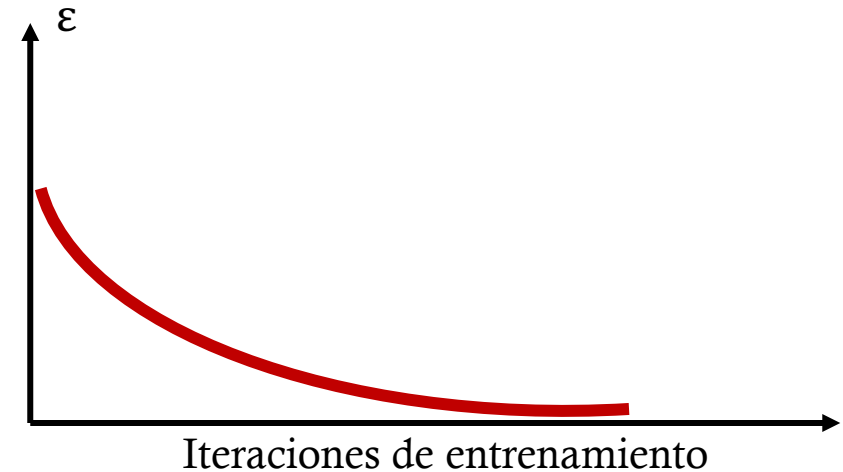
---

# SOLUCIONES ITERATIVAS

## Toma una acción

Los agentes basados en políticas y los agentes basados en valores utilizan diferentes métodos para lograrlo.

**Basado en valores:** Un agente basado en valor adopta una estrategia dinámica conocida como  $\epsilon$ -Greedy. Utiliza una tasa de exploración  $\epsilon$  que se ajusta a medida que avanza el entrenamiento para garantizar una mayor exploración en las primeras etapas del entrenamiento y cambia hacia una mayor explotación en las etapas posteriores.



$$\epsilon = \epsilon_0 e^{-\lambda t}$$

Si  $U[0,1] < \epsilon$ , **exploramos**

Si no, **explotación**

# SOLUCIONES ITERATIVAS

## Toma una acción

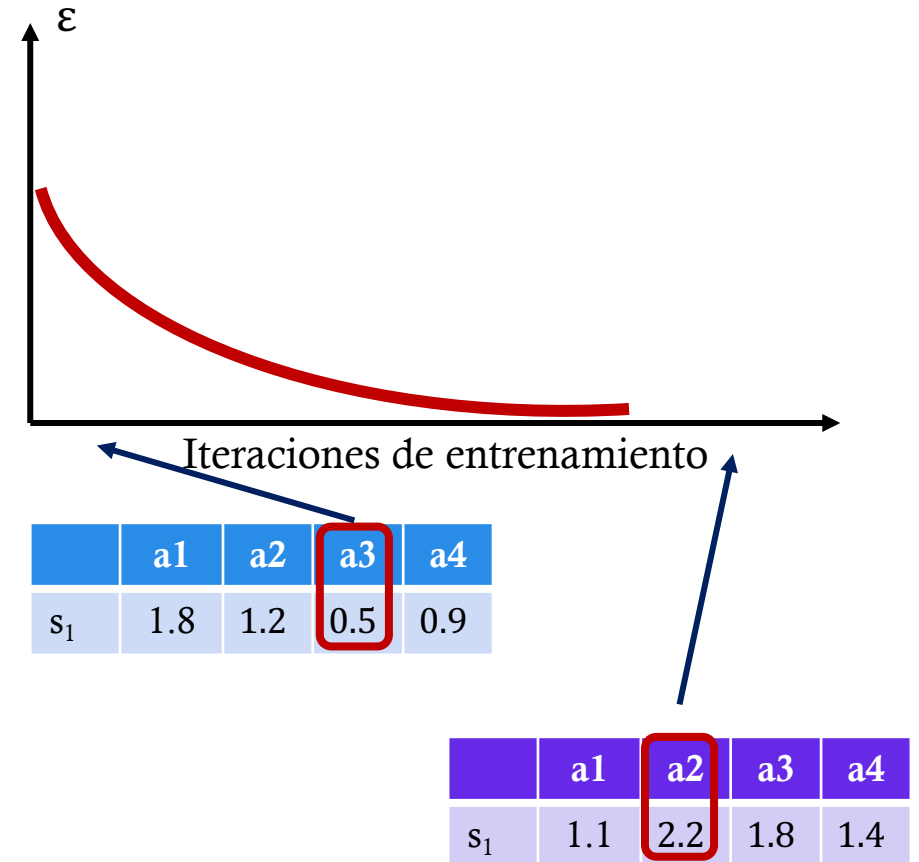
Los agentes basados en políticas y los agentes basados en valores utilizan diferentes métodos para lograrlo.

**Basado en valores:** Se establecemos  $\epsilon$  inicialmente en 1.

Luego, al comienzo de cada episodio, se reduce  $\epsilon$  a cierta tasa.

Cada vez que elige una acción en cada estado, se **explora** con una probabilidad  $\epsilon$ . Se **explota** con una probabilidad  $1-\epsilon$ .

Dado que  $\epsilon$  es mayor en las primeras etapas, es más probable que el agente **explore**. A medida que  $\epsilon$  disminuye, la probabilidad de exploración disminuye y el agente se vuelve **codicioso** al explotar cada vez más el entorno.





---

# SOLUCIONES ITERATIVAS

Obtiene retroalimentación de ambiente

El agente realiza la acción que ha seleccionado y obtiene retroalimentación del entorno. El agente recibe retroalimentación del entorno en forma de recompensa.

---

# SOLUCIONES ITERATIVAS

Mejora la estimación

La forma de mejorar la estimación de vuelta va a depender si es un algoritmo basado en política o valor.

**Basado en política:** El agente actualiza la tabla de políticas si obtiene una recompensa positiva, para aumentar la probabilidad de la acción que acaba de realizar.

---

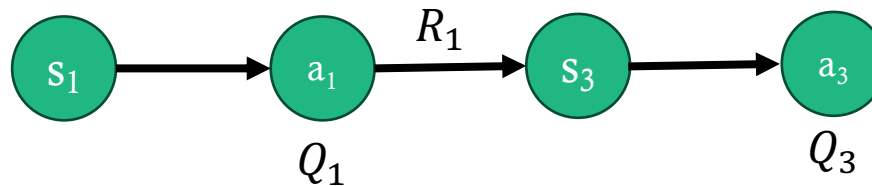
# SOLUCIONES ITERATIVAS

## Mejora la estimación

La forma de mejorar la estimación de vuelta va a depender si es un algoritmo basado en política o valor.

**Basado en valores:** Un agente establece que según la recompensa que recibe, al analizar la ecuación de Bellman establece si el valor debería ser mayor o menor.

Recordemos que la ecuación de Bellman nos establecía:



$$Q(s_1, a_1) = R_1 + \gamma Q(s_3, a_3)$$

---

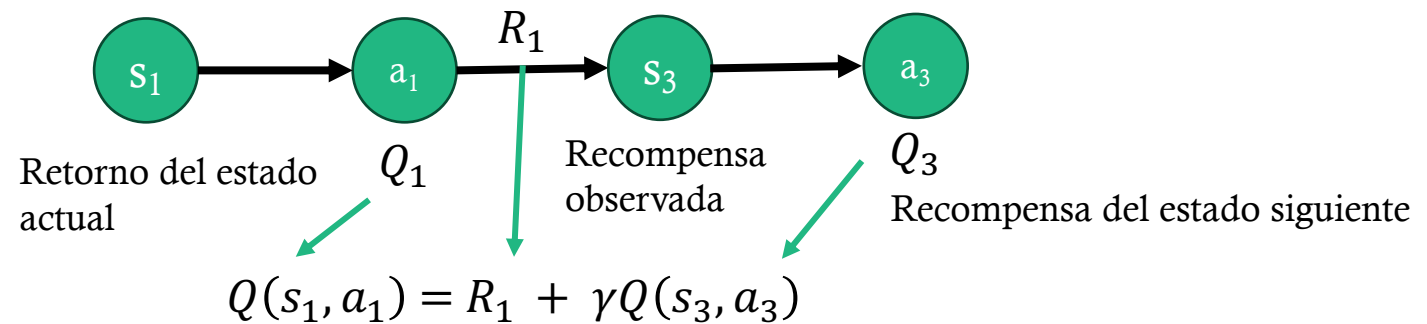
# SOLUCIONES ITERATIVAS

## Mejora la estimación

La forma de mejorar la estimación de vuelta va a depender si es un algoritmo basado en política o valor.

**Basado en valores:** Un agente establece que según la recompensa que recibe, al analizar la ecuación de Bellman establece si el valor debería ser mayor o menor.

Recordemos que la ecuación de Bellman nos establecía:



---

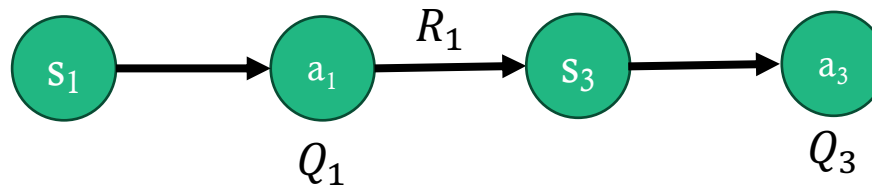
# SOLUCIONES ITERATIVAS

## Mejora la estimación

La forma de mejorar la estimación de vuelta va a depender si es un algoritmo basado en política o valor.

**Basado en valores:** Un agente establece que según la recompensa que recibe, al analizar la ecuación de Bellman establece si el valor debería ser mayor o menor.

Recordemos que la ecuación de Bellman nos establecía:



Error = Recompensa observada + Retorno del siguiente estado – Retorno del estado actual

$$Error = (R_1 + \gamma Q(s_3, a_3)) - Q(s_1, a_1)$$

---

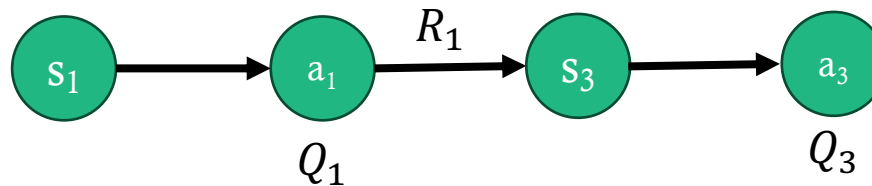
# SOLUCIONES ITERATIVAS

## Mejora la estimación

La forma de mejorar la estimación de vuelta va a depender si es un algoritmo basado en política o valor.

**Basado en valores:** Un agente establece que según la recompensa que recibe, al analizar la ecuación de Bellman establece si el valor debería ser mayor o menor.

Recordemos que la ecuación de Bellman nos establecía:



Así el agente aprende:  $Q(s_1, a_1) = Q(s_1, a_1) + \alpha \text{ Error}$

---

# SOLUCIONES ITERATIVAS

## Mejora la estimación

La idea central de diferentes algoritmos es cómo se mejora las estimaciones. Estas variaciones están relacionadas principalmente con tres factores:

- **Frecuencia:** El número de pasos hacia adelante dados antes de una actualización.
  - Por episodio.
  - Por cada paso
  - n-pasos
- **Profundidad:** El número de pasos hacia atrás para propagar una actualización.
  - Por episodio.
  - Por cada paso
  - n-pasos
- **Formula:** Fórmula que se utiliza para calcular la estimación actualizada.
  - Diferentes algoritmos usan diferentes variantes de la ecuación de Bellman.
  - Si se basa en políticas, se aumenta o disminuye la probabilidad en base a la recompensa recibida.

CAN IT RUN DOOM?

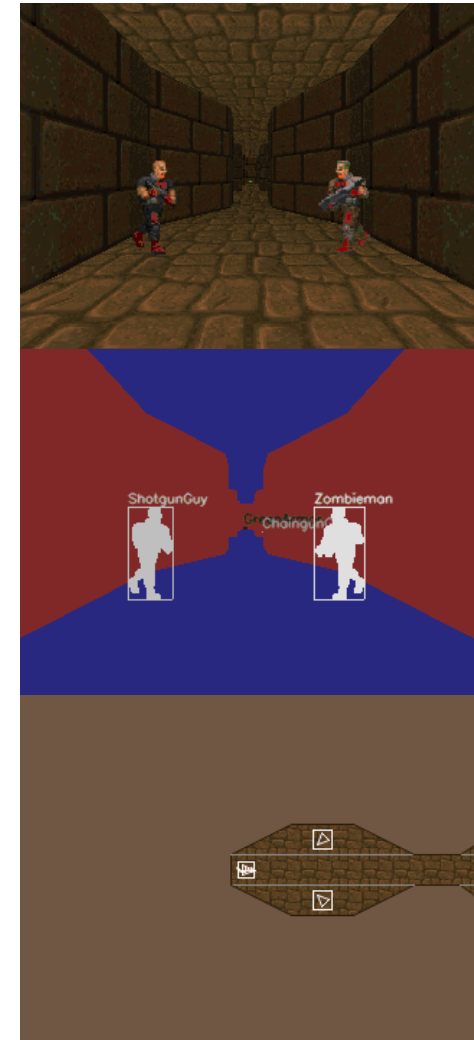
PLAY



# CAN IT PLAY DOOM? KINDA...

Algo que tiene interesante el AR es que, dado que se usa en problemas de control, se necesita crear escenarios simulados. Hay varias implementaciones disponibles, entre ellas:

- **Gymnasium**: Un fork de Gym de OpenAI con múltiples entornos de simulación en 2D y 3D, y videos juegos de Atari. Además de miles de entornos desarrollados por third parties o herramientas para desarrollar uno propio.
- **VizDoom**: Librería para desarrollar agentes que juegan Doom usando información visual. Tiene un wrapper de compatibilidad con Gymnasium, pero al ser un desarrollo previo, tiene su propia API en C++, Python y Julia.
  - Está diseñado para ofrecer altas opciones de personalización
  - Admite modos para un jugador y multijugador
  - Creación de escenarios personalizados basado en el editor de mapa de Doom.
  - Basado en **ZDoom**.



---

# CAN IT PLAY DOOM? KINDA...

Armemos una implementación de un algoritmo de AR usando **VizDoom** como entorno de simulación. Dado que vamos a implementar un modelo sencillo, vamos a usar un escenario simple.

Tenemos el siguiente escenario (muy simplificado):

**Objetivo:** Matar al demonio con la menos cantidad de balas posibles.

*El demonio siempre está en el mismo lugar y fijo, se muere de una sola bala.*

*El jugador solo se puede mover de forma lateral, y solo en 11 posiciones discretas.*

Cada episodio puede tener 15 movimientos, y termina si el jugador hace 15 movimientos o el demonio muere.



---

# CAN IT PLAY DOOM? KINDA...

Armemos una implementación de un algoritmo de AR usando **VizDoom** como entorno de simulación. Dado que vamos a implementar un modelo sencillo, vamos a usar un escenario simple.

El agente tiene las siguientes acciones:

- Moverse un bloque a la derecha o a la izquierda de donde esta.
- Disparar.

El agente no puede percibir nada.



---

# CAN IT PLAY DOOM? KINDA...

Armemos una implementación de un algoritmo de AR usando **VizDoom** como entorno de simulación. Dado que vamos a implementar un modelo sencillo, vamos a usar un escenario simple.

El escenario da los siguientes castigos y recompensas:

- **Castigo:**
  - La distancia horizontal en bloques entre el jugador y el demonio. -1 puntos entre bloque de distancia.
  - Si dispara y no le pega al demonio, -20 puntos
- **Recompensa:**
  - 1000 puntos cada vez que le pega al demonio.



---

# CAN IT PLAY DOOM? KINDA...

## Q-Learning

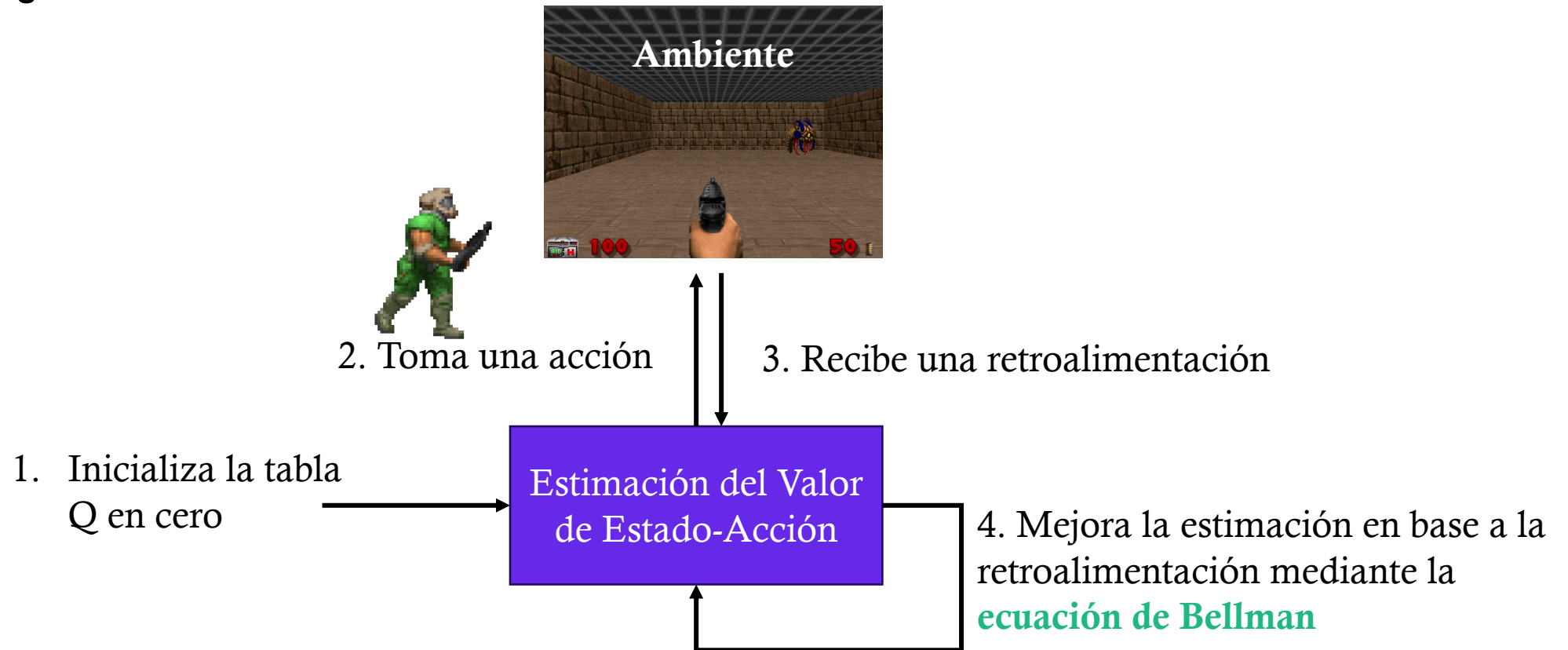
El algoritmo que vamos a implementar es el llamado Q-Learning. Este es muy sencillo y por consiguiente no útil en aplicaciones más complejas, por eso este escenario esta simplificado. Lo importante es que es la base para algoritmos más avanzados de AR.

Este algoritmo está basado en **Valor de Estado-Acción**. Para este problema tenemos 11 estados (uno por posición que el agente puede ubicarse) y 3 acciones (moverse a la derecha, a la izquierda y disparar).

Estado	Derecha	Izquierda	Disparo
0			
1			
...	...	...	...
10			

# CAN IT PLAY DOOM? KINDA...

## Q-Learning





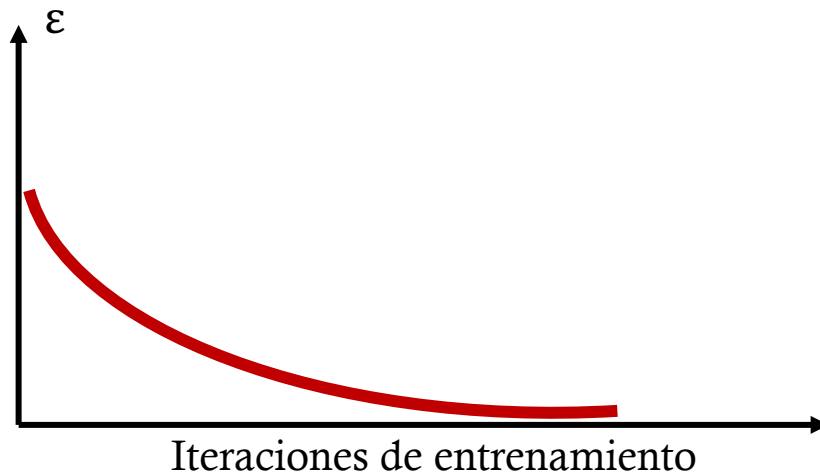
---

# CAN IT PLAY DOOM? KINDA...

## Q-Learning

Vimos que los algoritmos basados en valor realizan **exploración** y **explotación** en cada episodio. De la tabla Q, para un estado, se elige la mejor acción (mayor valor) o un valor al azar mediante la estrategia  $\epsilon$ -Greedy.

Para este problema en particular vamos a usar 100 episodios y  $\lambda = 0.03$



$$\epsilon = \epsilon_0 e^{-\lambda t}$$

Si  $U[0,1] < \epsilon$ , **exploramos**

Si no, **explotación**

---

# CAN IT PLAY DOOM? KINDA...

## Q-Learning

Habíamos visto que actualizábamos la tabla Q, la cual va a aproximarse a la tabla de valores de la **Política Óptima**, la actualización para diferentes valores es:

$$Q(s_i, a_j) = Q(s_i, a_j) + \alpha \text{Error}$$

Si el aplicar un estado y acción  $Q(s_i, a_j)$ , nos lleva a un nuevo estado  $s_k$ . El algoritmo de Q-Learning implementa la formula como:

$$Q(s_i, a_j) = Q(s_i, a_j) + \alpha \left[ \underbrace{(R_{ij} + \gamma \max(Q(s_k, :)))}_{\text{Diferencia de estimaciones usando el mejor caso}} - Q(s_i, a_j) \right]$$
$$Q(s_i, a_j) = (1 - \alpha)Q(s_i, a_j) + \alpha (R_{ij} + \gamma \max(Q(s_k, :)))$$

Para este ejercicio usamos  $\alpha=0.1$  y  $\gamma=0.99$



---

# CAN IT PLAY DOOM? KINDA...

## Q-Learning

Por ejemplo, si realizamos un entrenamiento, llegamos a la siguiente tabla:

Estado	Derecha	Izquierda	Disparo
0	0	0	0
1	-0.1	48.8	-0.2
2	0.7	432.8	-0.2
3	114.2	277.5	1000
4	989.8	415.9	-1.2
5	974.8	296.3	-1.3
6	952.8	6.4	-1.8
7	312.9	-0.5	-1.7
8	-0.4	-0.4	-1.2
9	-0.3	-0.3	-0.6
10	0	0	0

La recompensa promedio  
para este agente es: 5790

*La recompensa promedio para  
un agente aleatorio es: -5113*