

Tema 2.5 Aprendizaje por refuerzo

Tema 2. Resolución de problemas con búsqueda

- Representación de problemas y espacio de estados
- Búsqueda de soluciones
 - Búsqueda ciega
 - Búsqueda heurística
 - Búsqueda con adversario
 - **Aprendizaje de heurísticas**
 - Planificación
 - Búsqueda local
- Aplicaciones

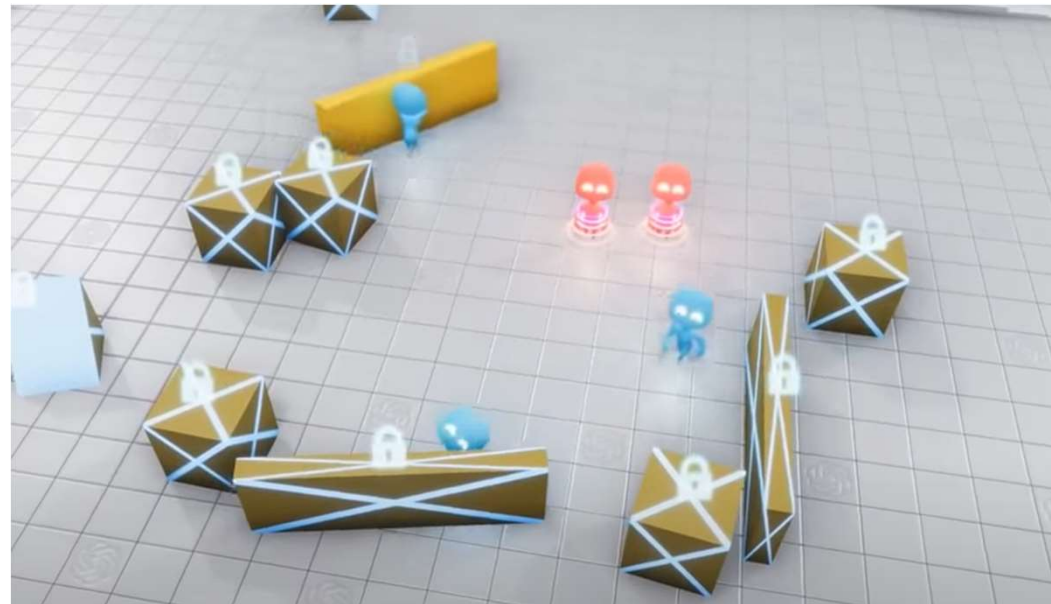
- Algoritmos de Búsqueda → **modelo** de transición completo (espacio de estados con costes predefinidos)
 - **Definición de Heurísticas en problemas de búsqueda**
 - Conocimiento experto reflejado en una función heurística que representa el conocimiento de la bondad de un estado → el programador conoce el juego y define esa función → necesito el modelo (transición y éxito)
 - Aprendizaje
 - Jugar partidas y aprendizaje por inducción - En IA2 veréis técnicas de aprendizaje offline
 - Algoritmo de **búsqueda de Montecarlo (MCTS)** → no usa modelo → aprende de simulaciones
 - Vamos a introducir el **aprendizaje por refuerzo** → libre de modelo
- Online vs offline learning

- <https://youtu.be/5SkQuT3kZOc?t=279> (a partir del minuto 4.39) – experimento del escondite DotCSV

- Segunda parte: estrategias locas aprendidas

<https://youtu.be/5SkQuT3kZOc?t=368>

<https://youtu.be/5SkQuT3kZOc?t=539>



- Aprendizaje supervisado

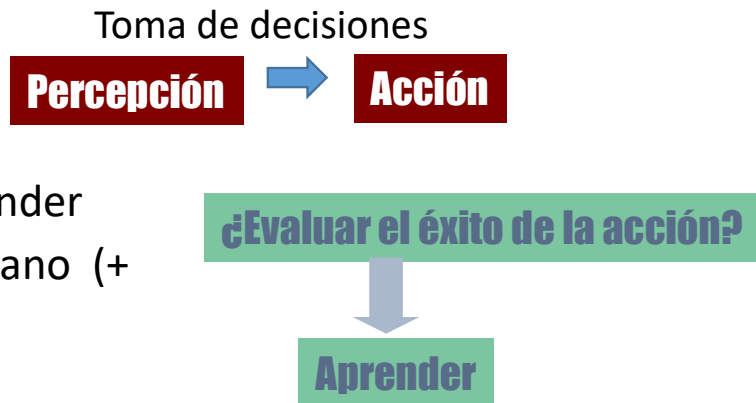
- Aprender con **entrenamiento**
- Ejemplos positivos y negativos de lo que quiero aprender
- Aprender por **observación** del comportamiento humano (+ razonamiento por analogía)

- Aprendizaje con refuerzo

- **Experiencia**
- **Simulaciones** o partidas contra otros AI y testers humanos durante el desarrollo

- Aprendizaje no **supervisado**

- **Clustering**



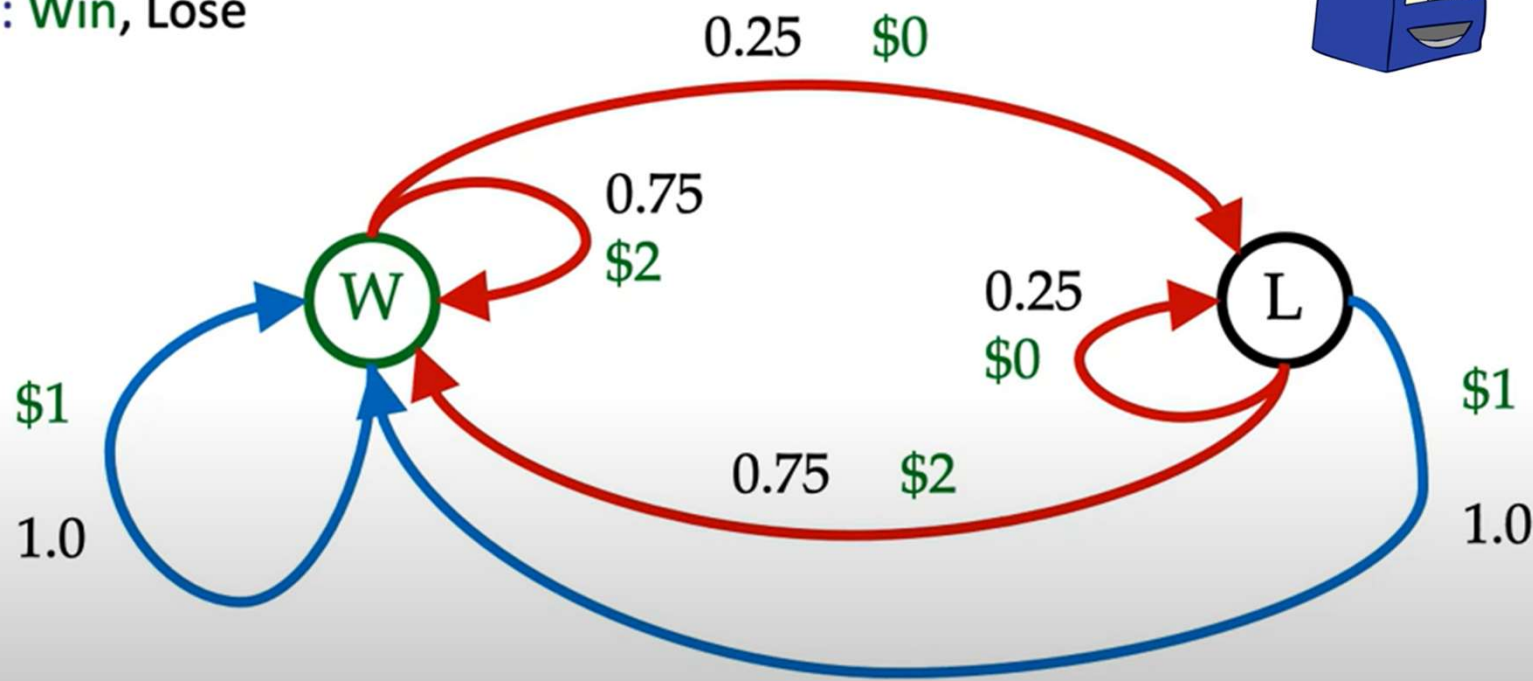
- Se da este nombre a un conjunto de técnicas de **aprendizaje basado en experiencia (prueba y error)**
- **No es supervisado**
 - **Se obtiene un** resultado por sus acciones pero nadie le dice qué debería haber hecho como alternativa a una mala acción.
 - No se dispone de **pares entrada/salida correctos**, ni existen correcciones explícitas para acciones subóptimas.
 - El agente debe usar **prueba y error** para evaluar su propio comportamiento y aprender de sus errores.
- Es popular en juegos de estrategia.

- Un agente en un estado debe realizar una tarea y dispone de un **conjunto de acciones** para resolverla
- El agente “escoge” la acción o el conjunto de acciones que cree que le permiten resolver la tarea
- El agente recibe **algo** de información del entorno sobre qué tal lo ha hecho
- Ni la recompensa ni el estado siguiente obtenido son conocidos a priori por el agente, y dependen únicamente del estado actual y de la acción tomada.
- Es decir, antes de aprender, el agente no sabe qué pasará cuando toma una acción determinada en un estado particular.
- Un buen aprendizaje permite que el agente “sepa” las consecuencias de las acciones tomadas, reconociendo las acciones que sobre estados concretos le llevan a conseguir mayor recompensa
- El agente utiliza esa información para modificar sus acciones futuras
- El objetivo es maximizar alguna función de recompensa a largo plazo



Relacionado con los MDP (Markov Decision Processes)

- Actions: *Blue, Red*
- States: *Win, Lose*



- Resolver MDP es **planificación offline (no lo haces realmente)** mientras que RL sí lo haces.

- Elegiríamos jugar a la máquina roja todas las veces

	Value
Play Red	15
Play Blue	10

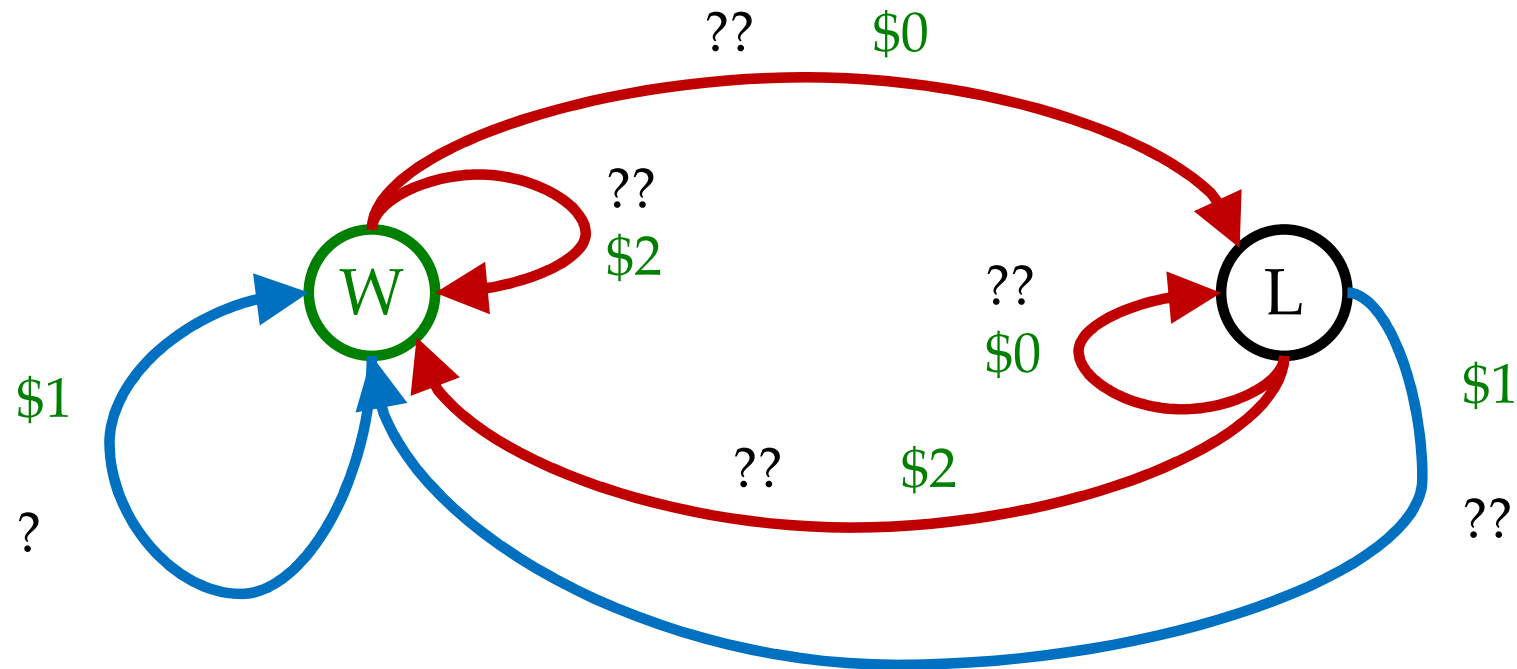
Let's Play!



\$2 \$2 \$0 \$2 \$2
\$2 \$2 \$0 \$0 \$0

Online Planning

- Rules changed! Red's win chance is different.



Let's Play!



\$1 \$1 \$1



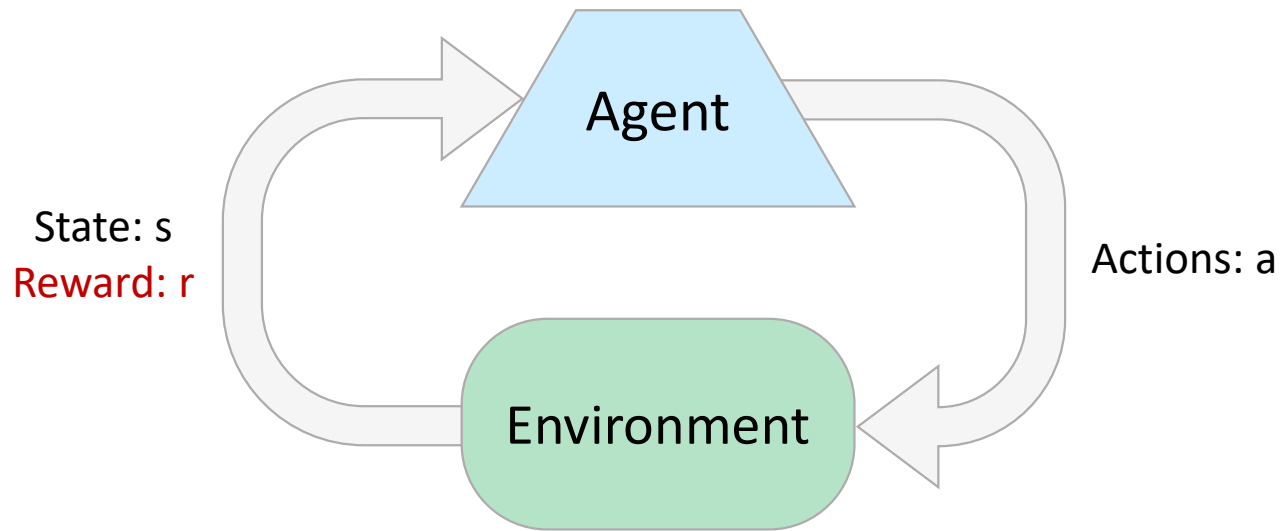
\$0 \$0 \$0 \$2

What Just Happened?

- That wasn't planning, it was learning!
 - Specifically, reinforcement learning
 - There was an MDP, but you couldn't solve it with just computation
 - You needed to actually act to figure it out
- Important ideas in reinforcement learning that came up
 - **Exploration**: you have to try unknown actions to get information
 - **Exploitation**: eventually, you have to use what you know
 - **Regret**: even if you learn intelligently, you make mistakes
 - **Sampling**: because of chance, you have to try things repeatedly
 - **Difficulty**: learning can be much harder than solving a known MDP

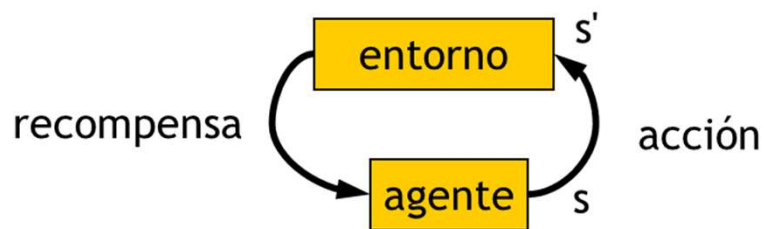


Reinforcement Learning



- Basic idea:
 - Receive feedback in the form of **rewards**
 - Agent's utility is defined by the reward function
 - Must (learn to) act so as to **maximize expected rewards**
 - All learning is based on observed samples of outcomes!

Ejemplo



El entorno es capaz de dar una cuantificación numérica del éxito o fracaso de las acciones del agente (esto no siempre es fácil)

¿Qué tamaño tiene esta tabla?

¿De cuántos estados estamos hablando? $< 3^9$

¿cuántas acciones? < 9

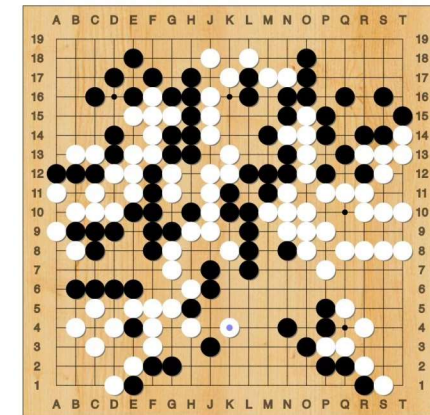
E \ A					
			...		
	0	0	...	0	0
	0	0	...	0	0
...
	0	1	...	0	0
	0	0	...	0	1
...

En juegos más complejos (GO) no se puede almacenar la tabla en memoria

Deep Mind combina el reciente interés en Deep learning con el clásico algoritmo de **aprendizaje por refuerzo q-learning**, inventado hace más de 25 años.

- Combina RL con Deep learning
- Tablero al final de la 4ª partida (única ganada por humano)
- La [reciente victoria de AlphaGo](#) (marzo 2016) el modelo de *machine learning* creado por Google para jugar al Go, frente a [Lee Sedol](#), el mejor jugador del mundo por 4 partidas a 1
- El sistema aprende a partir de datos y de hipótesis generadas por sí mismo, creando jugadas que no han existido previamente y tratando de resolverlas jugando contra versiones diferentes de sí

Requiere de una enorme potencia de computación → uso intensivo de la [Google Cloud Platform](#)



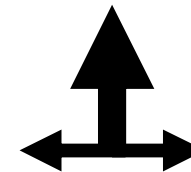
- Un **conjunto de estados** que describan el entorno
- Un **conjunto de acciones** que el agente puede realizar para modificar su entorno (cambiar de estado)
- Una **función de refuerzo (recompensa)** que indica al agente el resultado obtenido al realizar una acción sobre el estado
- Una **estrategia de exploración** para probar distintas acciones en el juego
 - **Vincular estados con acciones para** maximizar ganancia a largo plazo.
 - Los algoritmos están muy relacionados con programación dinámica.

→	→	→	+1
↑			-1
↑ START			

actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT



reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

- Estados
- Acciones
- Estrategia de exploración
- Recompensas
- Aprender cuál es la solución → en cada estado se aprende la mejor decisión

→	→	→	+1
↑		→	-1
→	→	→	↑

Recompensa de cada paso -2

→	→	→	+1
↑		↑	-1
↑	→	↑	←

Recompensa de cada paso -0.1

- Algoritmo que pertenece a la familia de técnicas de RL llamadas TD: Temporal Difference Algorithms
- Combina ideas de MonteCarlo y Programación Dinámica
 - Como el algoritmo de MC: no necesita modelo. Aprende de la experiencia
- Máquina de estados
 - **Un estado codifica todo lo que sea relevante sobre el entorno del personaje: posición, salud, proximidad al enemigo,..**
 - Cada estado tiene un **conjunto de acciones** que se pueden ejecutar.
 - El personaje **ejecuta una acción en el estado actual** y la función de refuerzo le da un valor de feedback $[-1,1]$ (0 si no sabemos)
 - **El valor de refuerzo no tiene que ser el mismo si se repite la acción en el mismo estado** (puede usar otra información contextual)

- **Cada episodio** es una sesión de entrenamiento en la que el agente explora el entorno y **recibe la recompensa** hasta que alcanza el estado objetivo.
- Matriz o función de **recompensa directa** **$R = \text{Estados} \times \text{Acciones}$**
- El entrenamiento mejora el “cerebro” del agente
matriz $Q = \text{Estados} \times \text{Acciones}$
 - Cuantos más episodios de entrenamiento mejor será la matriz Q
 - La matriz Q me dice qué acción elegir
 - Estado actual = inicial
 - Para el estado actual elegir la acción con valor Q máximo (valores de calidad)
 - estado actual = al que me lleve la acción
 - Continuar hasta que el estado actual = objetivo.

$$r\left(\begin{array}{|c|c|c|} \hline x & o & \\ \hline \hline \hline \hline \end{array}, \begin{array}{|c|c|c|} \hline x & o & \\ \hline \hline x & & \\ \hline \hline \hline \end{array}\right) = 0$$

Estado intermedio →

Al principio no tiene recompensa directa pero es un buen movimiento porque a largo plazo vamos a recibir una recompensa → propaga hacia atrás.

$$r\left(\begin{array}{|c|c|c|} \hline x & o & o \\ \hline \hline x & & \\ \hline \hline \hline \end{array}, \begin{array}{|c|c|c|} \hline x & o & o \\ \hline \hline x & & \\ \hline \hline x & & \\ \hline \hline \hline \end{array}\right) = 1$$

La bondad de una acción no está clara en el momento de la acción

Sólo damos recompensa cuando ocurre algo significativo el personaje aprenderá que toda la secuencia de acciones que me han llevado a la última también son buenas, aunque no hubo feedback explícito cuando se realizaron.

Q-learning sólo se preocupa por aprender en su turno

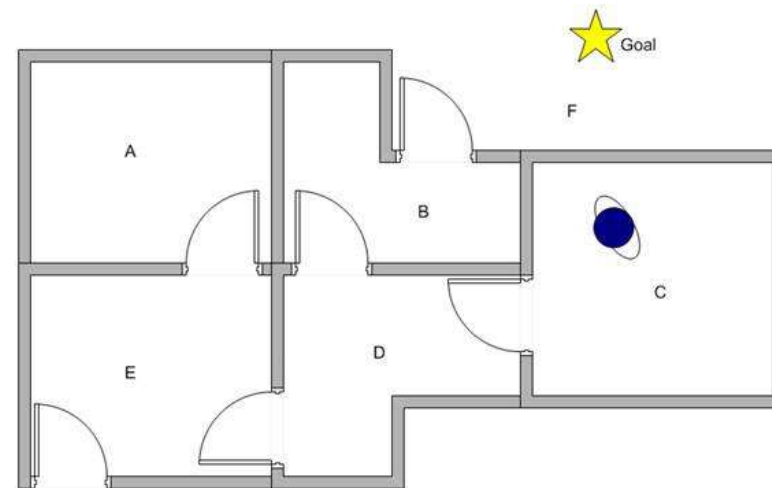
Tupla de experiencia

$$\left(\begin{array}{|c|c|c|} \hline x & o & \\ \hline \hline \hline \hline \end{array}, \begin{array}{|c|c|c|} \hline x & o & \\ \hline \hline x & & \\ \hline \hline \hline \end{array}, 0, \begin{array}{|c|c|c|} \hline x & o & o \\ \hline \hline x & & \\ \hline \hline \hline \end{array}\right)$$

(Estado, Acción, Recompensa, Estado siguiente)

(S, A, R, S')

- Robot que aprende a salir del edificio. **Exploración.** No conoce el entorno.
- ¿Cómo hacemos que nuestro agente aprenda de la experiencia?
- Queremos modelar la evacuación de un agente desde cualquier habitación.
- Desde C a F

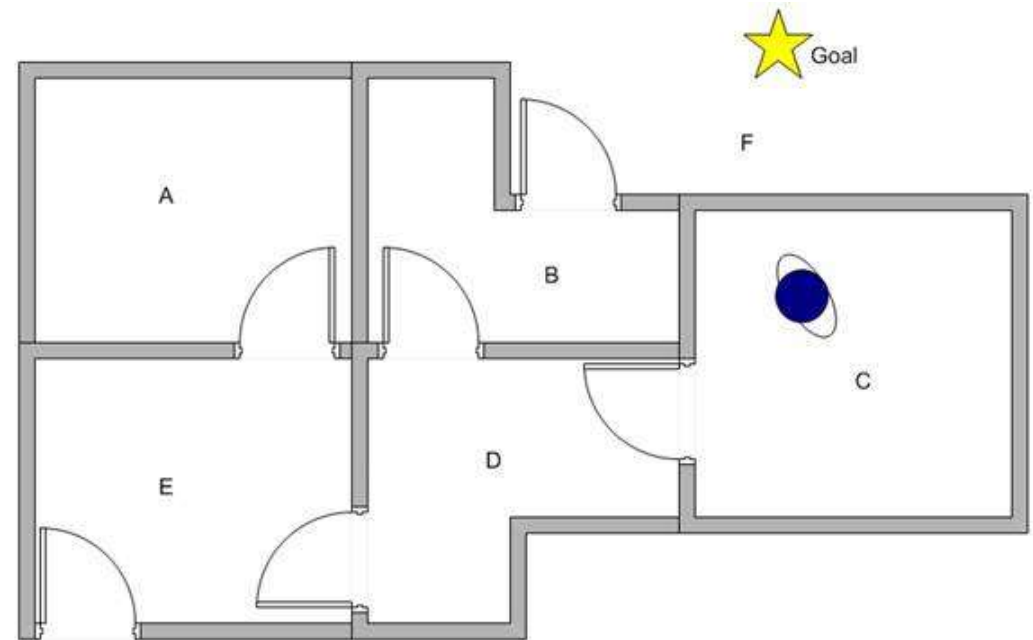


Ejemplo: $Q = 0$ “no sabe nada”

$Q = C$

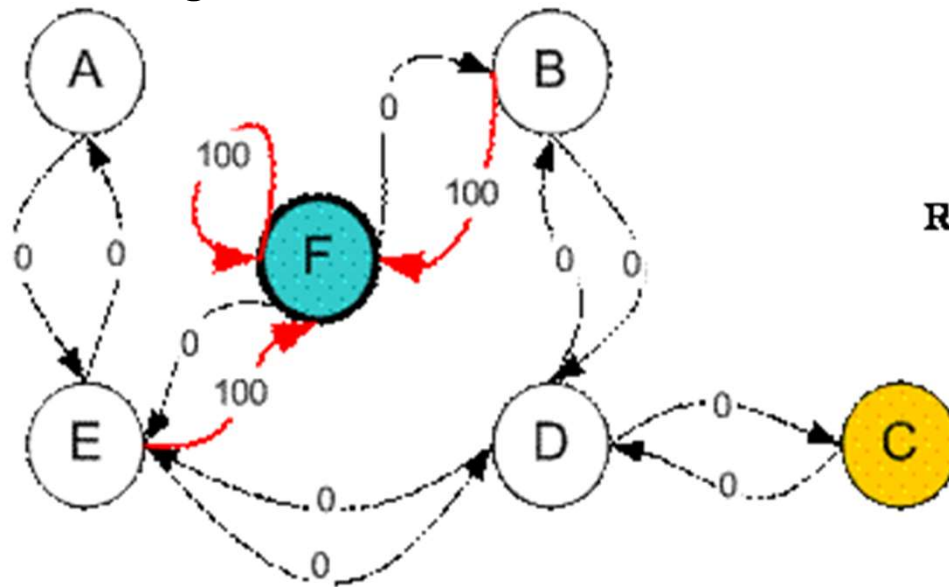
Estados

	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0



Ejemplo Recompensa sólo al alcanzar el objetivo

- Cada habitación (incluyendo F (exterior)) es un estado y cada acción es un movimiento entre habitaciones. Diagrama de estados



$R =$

<i>state \ action</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>F</i>
<i>A</i>	-	-	-	-	0	-
<i>B</i>	-	-	-	0	-	100
<i>C</i>	-	-	-	0	-	-
<i>D</i>	-	0	0	-	0	-
<i>E</i>	0	-	-	0	-	100
<i>F</i>	-	0	-	-	0	100

- El agente empieza en C. Puede ir a D pero no a B....
- Los números son los valores instantaneos de recompensa que se pueden representar también en una matriz R

Q-learning

- Tupla de experiencia: <estado actual, acción, valor refuerzo, estado siguiente >

$\langle s, a, r, s' \rangle$

Buscar en la **matriz de calidad (Q-values)**

Se usan para

actualizar los Q-values según la matriz de recompensas que indica lo buena que ha sido la acción en el estado.

Resultado
de la acción

- La matriz **Q-values (valores de calidad)** guarda para cada estado y acción $Q(s,a)$ lo aprendido hasta el momento
- La actualización de los Q-values se hace con la siguiente regla

$$Q(s,a) = (1 - \alpha) Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$$

Los parámetros del algoritmo son α (ratio de aprendizaje [0,1]) y γ (tasa de descuento)

- La componente $\alpha \cdot r$ actualiza el valor Q usando el **valor de refuerzo (r)** → **matriz de Recompensas**
- La componente $\alpha \gamma \max(Q(s',a'))$ busca todas las posibles acciones que se pueden hacer en el estado s' y se elige la del valor Q más alto. Esto ayuda a tener en cuenta la bondad del estado siguiente.

- α **ratio o velocidad de aprendizaje** (*learning rate*).
 - Valor entre 0 y 1 que indica cuánto aprender de cada experiencia.
 - Con 0 no aprendemos nada de una nueva experiencia y con 1 olvida todo lo que sabía hasta ahora y nos fiamos completamente de la nueva experiencia.
- γ **tasa de descuento** (*discount factor*).
 - Valor entre 0 y 1 que indica la importancia del largo plazo.
 - 0 significa que sólo nos importan los refuerzos inmediatos, y 1 significa que los refuerzos inmediatos no importan, sólo importa el largo plazo.
 - Este factor nos ayuda a mezclar recompensas directas con recompensas a largo plazo y producir la recompensa mixta.
- En ambos parámetros los extremos (0 o 1) son poco útiles.

- La matriz de calidad **$Q(\text{estado}, \text{accion})$** representa lo que ha aprendido en varias experiencias (de recorrer el edificio).
- **Al principio matriz cero de 6 filas (si no sabemos el nº de estados/acciones se va construyendo la matriz a partir de una celda)**
- Para el ejemplo simplificamos la fórmula de Q learning con ratio de aprendizaje **$\alpha = 1$**

$$Q(s,a) = (1 - \alpha) Q(s,a) + \alpha(r + \gamma \max(Q(s',a')))$$

$$Q(s,a) = r + \gamma \max(Q(s',a'))$$

- **Tasa de descuento $\gamma = [0,1]$.**
- **Valores más pequeños hacen que se tenga en cuenta sólo la recompensa inmediata.**
- **Valores altos usan recompensa futura (retrasa la recompensa)**

Ejemplo: 1er episodio de aprendizaje

■ Valor $\gamma = 0.8$

Estado inicial = B (acciones ir a D o ir a F)

$$Q = \begin{matrix} & \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{100} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

$$R = \begin{matrix} & \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix} & \begin{bmatrix} - & - & - & - & 0 & - \\ - & - & - & 0 & - & 100 \\ - & - & - & 0 & - & - \\ - & 0 & 0 & - & 0 & - \\ 0 & - & - & 0 & - & 100 \\ - & 0 & - & - & 0 & 100 \end{bmatrix} \end{matrix}$$

□ Desde B: estrategia de exploración aleatoria \rightarrow ir a F

$$Q(s,a) = r + \gamma \max(Q(s',a'))$$

$$Q(state, action) = R(state, action) + \gamma \cdot \text{Max}[Q(next\ state, all\ actions)]$$

$$Q(B, F) = R(B, F) + 0.8 \cdot \text{Max}\{Q(F, B), Q(F, E), Q(F, F)\} = 100 + 0.8 \cdot 0 = 100$$

□ Desde F: tres acciones irB, irE, irF

□ Fin del primer episodio

Ejemplo: 2º episodio de aprendizaje

- Estado inicial aleatorio = D
- Estrategia de exploración aleatoria
irB, irC, irE

□ Dos posibles acciones irD, irF.

□ Cómputo del valor Q

$$Q(\text{state}, \text{action}) = \mathbf{R}(\text{state}, \text{action}) + \gamma \cdot \text{Max}[Q(\text{next state}, \text{all actions})]$$

$$Q(D, B) = \mathbf{R}(D, B) + 0.8 \cdot \text{Max}\{Q(B, D), Q(B, F)\} = 0 + 0.8 \cdot \text{Max}\{0, 100\} = 80$$

□ Seguimos porque B no es estado objetivo

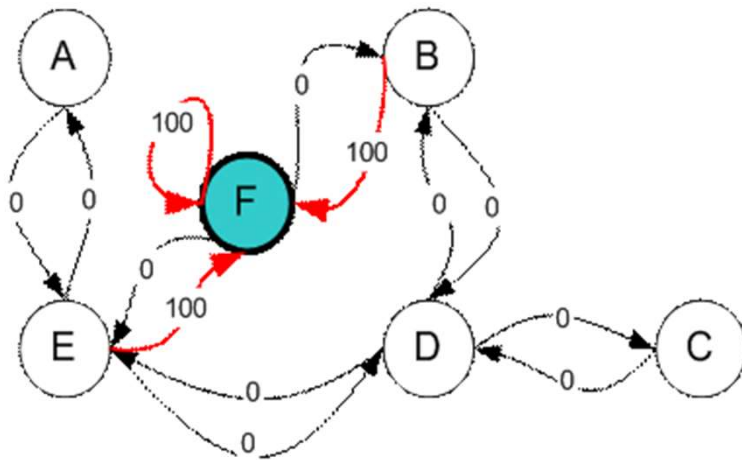
$\mathbf{R} =$

state \ action	A	B	C	D	E	F
A	-	-	-	-	0	-
B	-	-	-	0	-	100
C	-	-	-	0	-	-
D	-	0	0	-	0	-
E	0	-	-	0	-	100
F	-	0	-	-	0	100

$\mathbf{Q} =$

	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	100
C	0	0	0	0	0	0
D	0	80	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

- Desde B (irD, irF) la exploración nos lleva a elegir F
- F tiene tres acciones: irB, irE, irF. (todas 0 en la matriz Q)
- El nuevo valor del resultado de Q es 100 y no cambia la matriz Q.



$$Q = \begin{matrix} & \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

$$Q(\text{state}, \text{action}) = \mathbf{R}(\text{state}, \text{action}) + \gamma \cdot \text{Max}[Q(\text{next state}, \text{all actions})]$$

$$Q(B, F) = \mathbf{R}(B, F) + 0.8 \cdot \text{Max}\{Q(F, B), Q(F, E), Q(F, F)\}$$

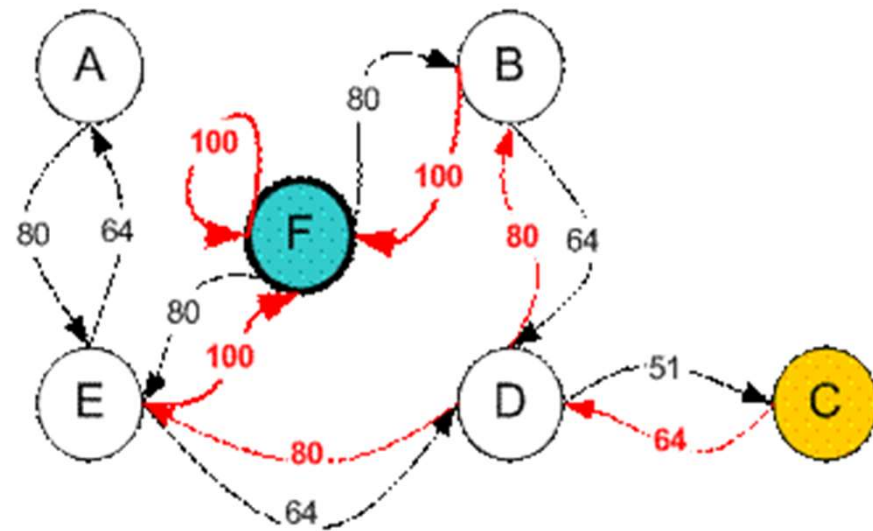
$$= 100 + 0.8 \cdot \text{Max}\{0, 0, 0\} = 100$$

Ejemplo... después de n episodios la matriz Q converge

$state \backslash action$	A	B	C	D	E	F	$state \backslash action$	A	B	C	D	E	F
A	-	-	-	-	400	-	A	-	-	-	-	80	-
B	-	-	-	320	-	500	B	-	-	-	64	-	100
C	-	-	-	320	-	-	C	-	-	-	64	-	-
D	-	400	256	-	400	-	D	-	80	51	-	80	-
E	320	-	-	320	-	500	E	64	-	-	64	-	100
F	-	400	-	-	400	500	F	-	80	-	-	80	100

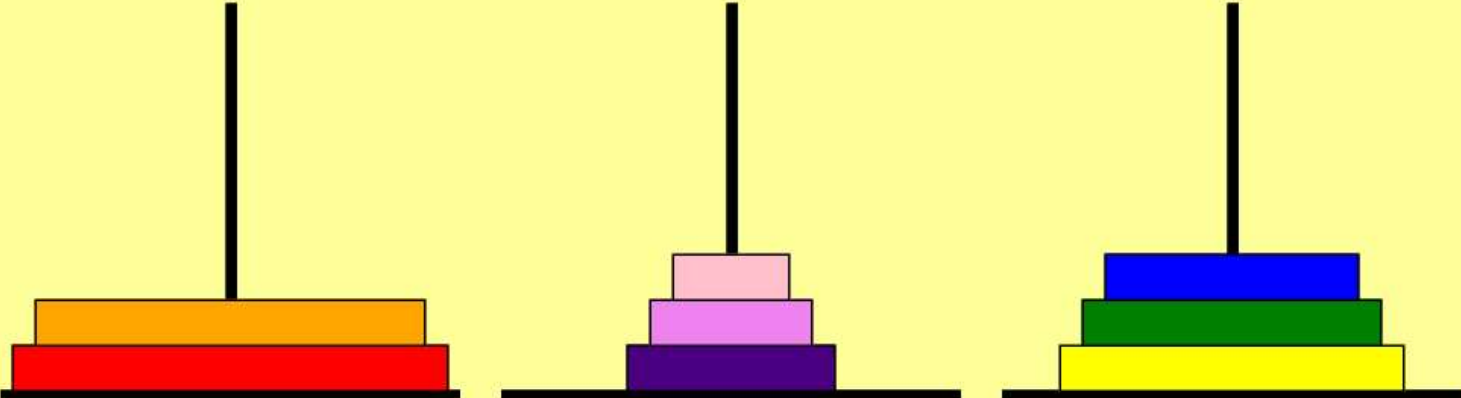
- El agente puede encontrar el mejor camino siguiendo las acciones que con valor Q máximo.

- C-D-B-F
- C-D-E-F



Ejemplo de aplicación de RL en el juego de las torres de Hanoi

Tower of Hanoi



TOWER 1 TOWER 2 TOWER 3

No. of disks 8 ▼

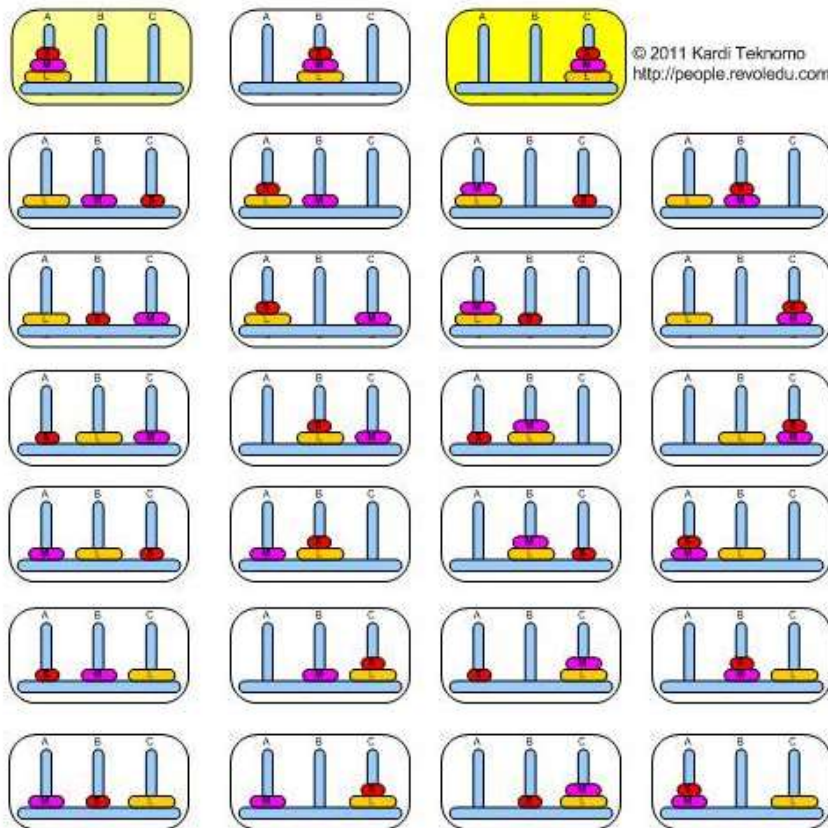
Minimum no. of moves 255

Your no. of moves 56

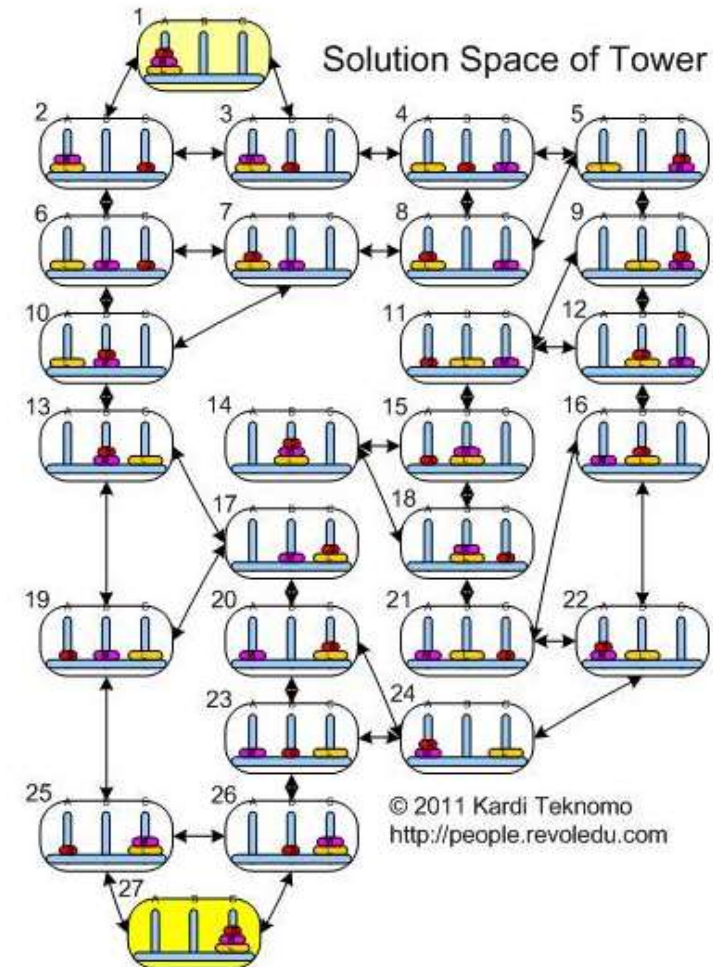
Instructions Restart Undo Solve

Diagram below shows all 27 possible state of the Tower of Hanoi.

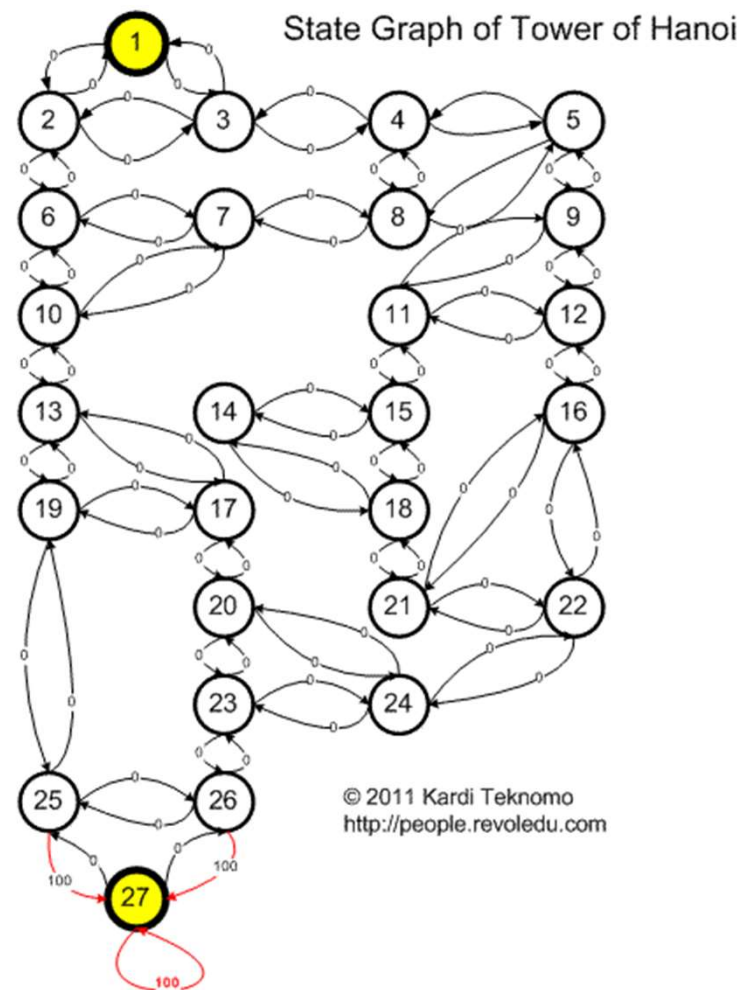
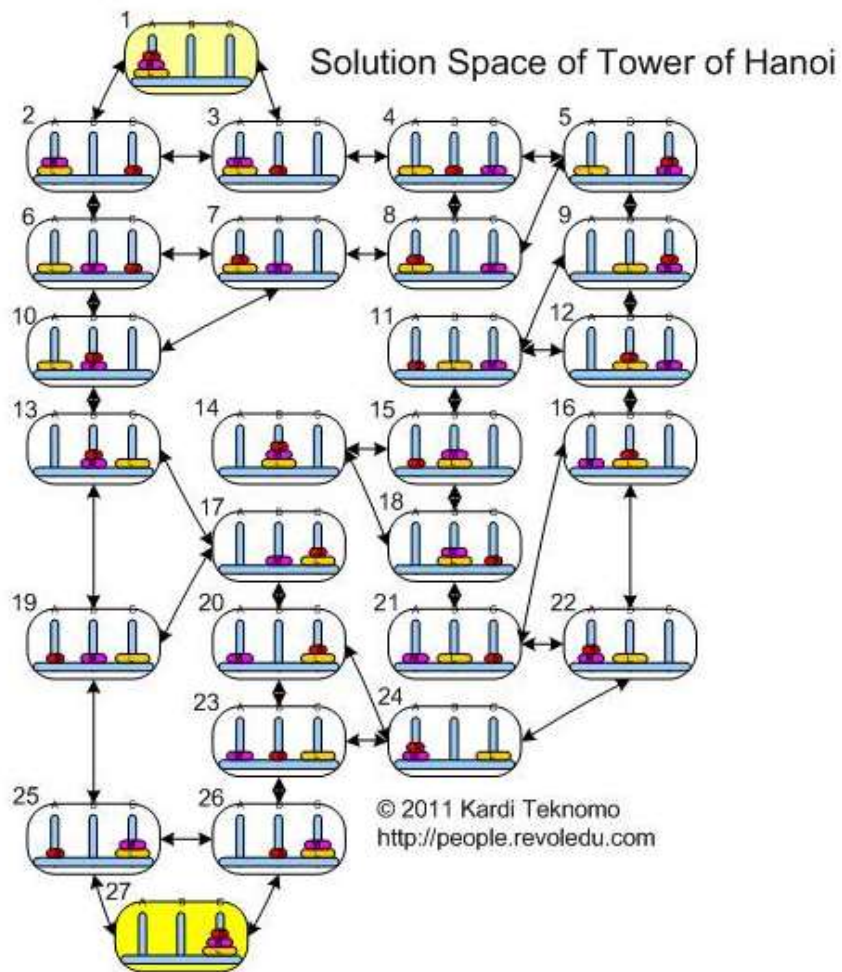
All Possible States of Tower of Hanoi



Solution Space of Tower of Hanoi



Ejemplo de aplicación de RL en el juego de las torres de Hanoi



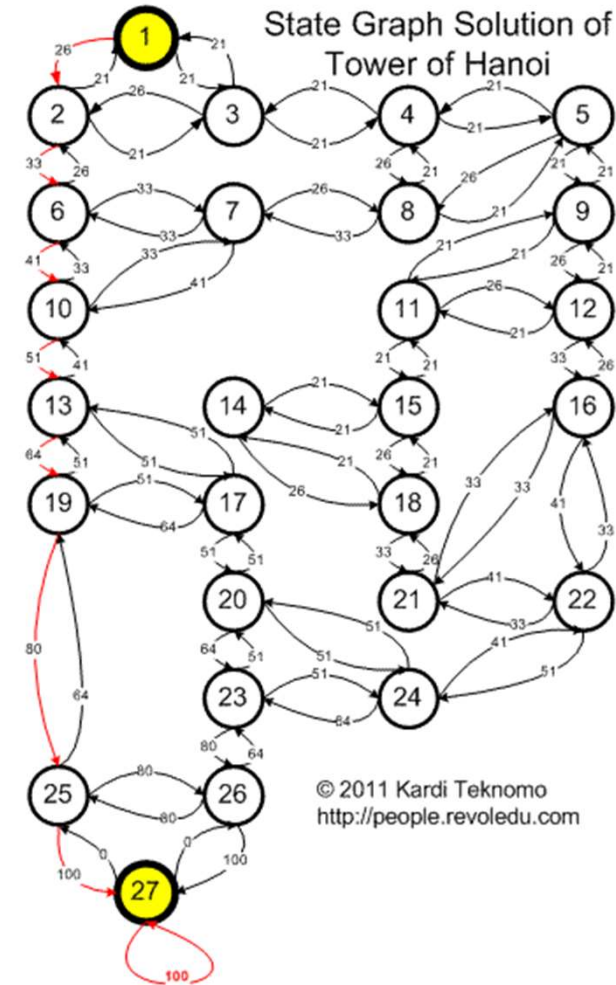
Matriz R

Reward	Action																										
State	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	-	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	0	-	0	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	0	0	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	-	-	0	-	0	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	-	-	-	0	-	-	-	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	-	0	-	-	-	-	0	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	-	-	-	-	-	0	-	0	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	-	-	-	0	0	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9	-	-	-	-	0	-	-	-	-	-	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10	-	-	-	-	-	0	0	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	-	-	-	-	-	-	-	-	0	-	-	0	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	0	-	0	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	0	-	0	-	-	-	-	-	-	-	-
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	0	-	-	-	-	-	-	-	-	-
15	-	-	-	-	-	-	-	-	-	-	0	-	-	0	-	-	-	0	-	-	-	-	-	-	-	-	-
16	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	0	0	-	-	-	-	-	-
17	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	0	0	-	-	-	-	-	-	-
18	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	-	-	-	-	-	0	-	-	-	-	-	-
19	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	0	-	-	-	-	-	-	-	0	-	-
20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	0	0	-	-	-
21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	0	-	-	-	0	-	-	-	-	-
22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	0	-	-	0	-	-	-
23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	0	-	0	-
24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	0	0	-	-	-	-
25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	0	100
26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	0	-	100
27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100

Remember the R matrix is simply an adjacency matrix of the state graph above, with special attention to the links toward the goal. All links toward the goal (including an additional self loop) must have high value (100) while all other links have zero values. Non-connected nodes are not considered, and therefore the value is infinity.

Matriz Q normalizada después del aprendizaje

Norm On	Action	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	True	Max State
1		26%	27%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26%	2	
2		21%	27%	-	-	33%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	33%	6	
3		21%	26%	-	-	27%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26%	2	
4		-	27%	-	-	27%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26%	8	
5		-	-	27%	-	-	-	-	-	26%	27%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26%	8	
6		-	26%	-	-	-	33%	-	-	-	47%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	47%	10	
7		-	-	-	27%	-	33%	-	-	26%	47%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	47%	10	
8		-	-	-	27%	27%	-	33%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	33%	7	
9		-	-	-	27%	-	-	-	-	-	-	27%	26%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26%	12	
10		-	-	-	33%	33%	-	-	-	-	-	-	26%	57%	-	-	-	-	-	-	-	-	-	-	-	-	-	57%	13	
11		-	-	-	-	-	-	27%	-	-	-	26%	-	-	-	27%	-	-	-	-	-	-	-	-	-	-	-	26%	12	
12		-	-	-	-	-	-	27%	-	-	27%	-	-	-	-	33%	-	-	-	-	-	-	-	-	-	-	-	33%	16	
13		-	-	-	-	-	-	-	47%	-	-	-	-	-	-	-	57%	-	64%	-	-	-	-	-	-	-	-	64%	19	
14		-	-	-	-	-	-	-	-	26%	-	-	-	-	-	27%	-	26%	-	-	-	-	-	-	-	-	-	26%	18	
15		-	-	-	-	-	-	-	-	27%	-	-	27%	-	-	-	-	26%	-	-	-	-	-	-	-	-	-	26%	18	
16		-	-	-	-	-	-	-	-	-	26%	-	-	-	-	-	-	-	26%	-	-	-	-	-	-	-	-	47%	22	
17		-	-	-	-	-	-	-	-	-	-	26%	-	-	-	-	-	-	26%	-	-	-	-	-	-	-	-	64%	19	
18		-	-	-	-	-	-	-	-	-	-	57%	-	-	-	-	-	-	64%	57%	-	33%	47%	-	-	-	-	64%	19	
19		-	-	-	-	-	-	-	-	-	-	-	27%	27%	-	-	-	-	-	33%	-	-	-	-	-	-	-	80%	25	
20		-	-	-	-	-	-	-	-	-	-	57%	-	-	-	-	-	-	57%	-	-	-	-	-	-	80%	-	64%	23	
21		-	-	-	-	-	-	-	-	-	-	-	33%	-	-	33%	-	26%	-	-	-	47%	-	-	-	-	-	47%	22	
22		-	-	-	-	-	-	-	-	-	-	-	33%	-	-	-	-	-	-	33%	-	-	-	57%	-	-	-	57%	24	
23		-	-	-	-	-	-	-	-	-	-	-	-	57%	-	-	-	-	-	57%	-	-	-	-	80%	-	-	80%	26	
24		-	-	-	-	-	-	-	-	-	-	-	-	-	57%	-	-	-	-	57%	-	-	47%	64%	-	-	-	64%	23	
25		-	-	-	-	-	-	-	-	-	-	-	-	-	-	64%	-	-	-	-	-	-	-	-	-	80%	-	100%	27	
26		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	64%	-	-	-	-	-	-	-	80%	-	-	100%	27	
27		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100%	-	-	-	-	-	-	-	-	-	100%	27	



Q-learning aprende una heurística (óptima) para llegar de cualquier nodo al objetivo.

A* es un algoritmo general capaz de encontrar solución entre cualquier par de nodos.

- Los ejemplos tienen un número discreto de estados, y un número discreto de acciones y con números relativamente pequeños y manejables
- **Estados continuos**
 - Sensores de distancia, o en tableros como ajedrez o el Go, los estados son discretos, pero su número es tan grande, que es inviable usar una tabla.
 - Se sustituyen las columnas de la tabla R de recompensas por un mecanismo que aproxime funciones (como NN)
 - Se utiliza **una red de neuronas (NN) separada para cada acción**, cuya entrada es la lista de valores del estado, y cuya salida es valor de refuerzo de la acción.
 - Repetimos esto en las redes de todas las acciones para tener la lista de refuerzos para elegir la mejor acción.
 - Para aprender, en lugar de actualizar la tabla Q, entrenamos las redes.
 - Se puede usar **Deep learning (redes profundas)** y combinar **todas las acciones en una misma red** con una salida para cada acción.
- **Acciones continuas**
 - Control de un automóvil. Las acciones incluyen el ángulo de giro del volante y la presión de los pedales que son valores continuos también.
 - Ya no nos sirve una red de neuronas para cada acción, porque hay infinitas acciones. Tampoco nos sirve una red de neuronas con varias salidas porque necesitaría infinitas salidas → **Sistemas Actor-Crítico (actor-critic)**.

- Teknomo, Kardi. 2005. Q-Learning by Examples.
<http://people.revoledu.com/kardi/tutorial/ReinforcementLearning/index.html>
- **Russell, S., Norvig, P.** Artificial Intelligence. A Modern Approach.
Prentice Hall, 2013, 3ª edición.