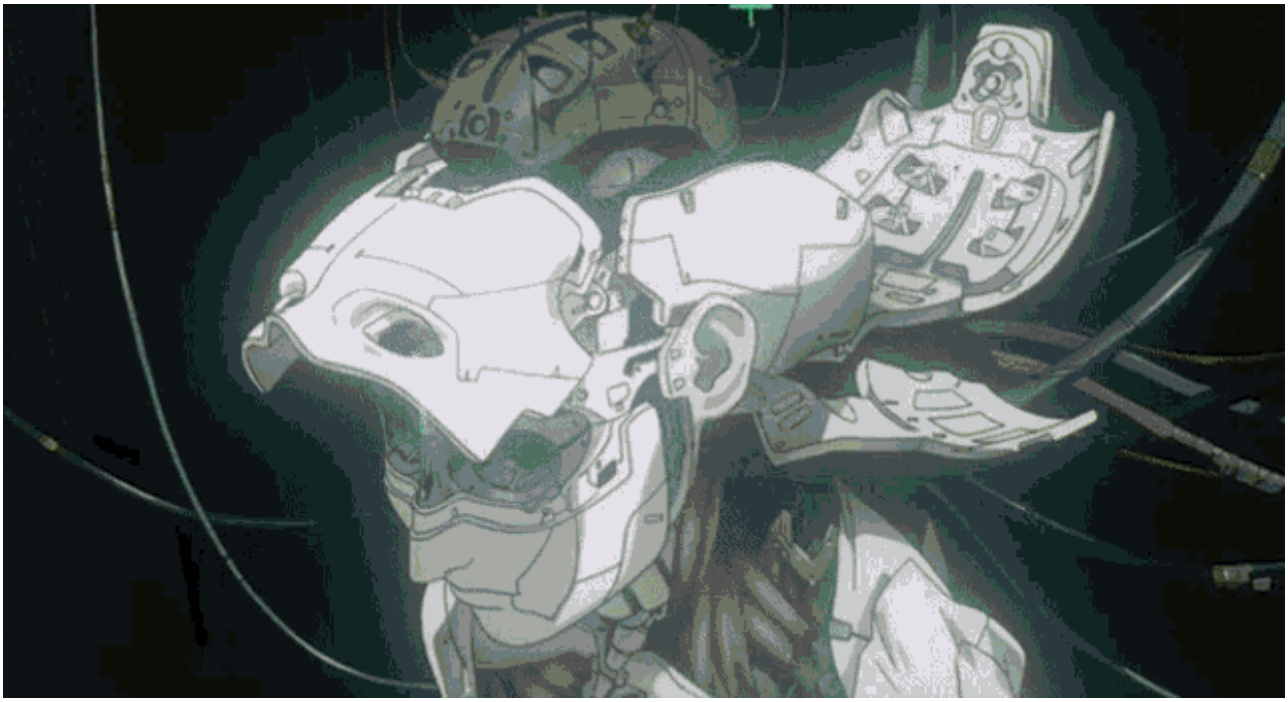
> "¿Qué es lo que hace a un humano? ¿Es su capacidad de tomar decisiones? ¿O es algo más profundo?"
>
> — Major Motoko Kusanagi, *Ghost in the Shell*

> "¿Qué es lo que separa a las máquinas de los androides como nosotros? Las máquinas han desarrollado emociones... consciencia."
>
> — 2B, *NieR: Automata*

# Tipos de Agentes

## 1. ¿Qué es un Agente?

Un **agent** en AI se define de manera amplia como cualquier sistema que percibe su entorno y toma acciones para lograr sus objetivos. En otras palabras, un agente recibe información del mundo, decide qué hacer y luego actúa sobre el mundo. Formalmente, *"an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators."* (Chapter 2 Intelligent Agents — Artificial Intelligence documentation) Por ejemplo, un agente humano tiene ojos y oídos como sensores, y manos y voz como actuadores, mientras que un robot podría tener cámaras y micrófonos como sensores, y motores como actuadores.



PROF

*(Major Motoko Kusanagi siendo ensamblada - un ejemplo perfecto de un agente con sensores y actuadores avanzados.)*

Todo agente tiene componentes fundamentales que guían su comportamiento: **percepts** (percepciones), **actions** (acciones) y el **environment** (entorno). Un **percept** es la entrada que un agente percibe en un momento dado, como una instantánea del mundo a través de sus sensores. Las percepciones del agente a lo largo del tiempo forman una **percept sequence** (secuencia de percepciones), que es el historial completo de todo lo que el agente ha percibido hasta el momento

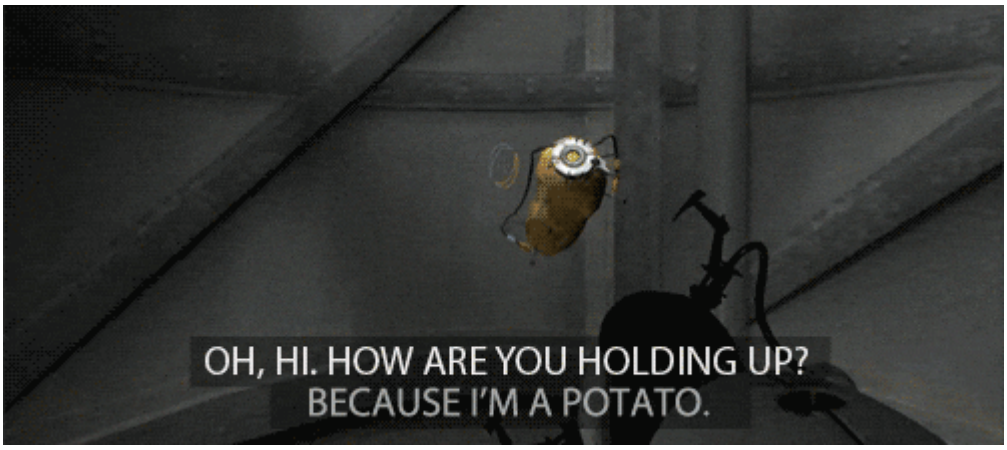
(Chapter 2 Intelligent Agents — Artificial Intelligence documentation). Con base en esta secuencia de percepciones (o a veces solo en la última percepción), el agente elige una **action** para ejecutar mediante sus actuadores. El environment es el contexto o mundo en el que el agente opera: le proporciona percepciones y reacciona a las acciones del agente.

En términos matemáticos, describimos a un agente mediante una **agent function** que mapea cualquier secuencia de percepciones a una acción (Chapter 2 Intelligent Agents — Artificial Intelligence documentation). Si denotamos (P^*) como el conjunto de todas las posibles secuencias de percepciones y (A) como el conjunto de todas las acciones, entonces el comportamiento de un agente es una función (f: P^* \to A) (Intelligent agent - Wikipedia). Por ejemplo, si (f) asigna la secuencia de percepciones "inbox has an unread email" a la acción "open email", el agente abrirá un correo cada vez que perciba que está sin leer. El mecanismo real que implementa este mapeo es el **agent program**, que es la implementación concreta (código o reglas) que se ejecuta en un sistema físico. Es importante distinguir la agente function abstracta del agent program que la implementa (Chapter 2 Intelligent Agents — Artificial Intelligence documentation): la función es como la descripción idealizada del comportamiento, mientras que el programa es el código real que se ejecuta en el mundo.

Para concretar estas ideas, piensa en una analogía cotidiana: **responder un mensaje en tu teléfono**. La notificación en tu teléfono (percept) te informa que hay un mensaje nuevo; entonces decides cómo responder (action) basándote en quién lo envía y lo que dice (contexto del environment). Si es tu jefe enviando un correo a las 9 AM, podrías responder de inmediato; si es un spam cualquiera, lo ignoras. Aquí, *tú* eres el agente, tu teléfono proporciona las percepciones y tus respuestas son las acciones. Otro ejemplo es el **algorithm** de recomendaciones de videos de TikTok: percibe tu comportamiento de visualización (cuánto tiempo miras, qué te gusta) y actúa seleccionando el siguiente video que te mostrará. Con el tiempo se adapta para lograr mejor su objetivo de mantenerte enganchado.

En la cultura popular, un ejemplo clásico de agente de AI es **GLaDOS de la serie *Portal***. GLaDOS (Genetic Lifeform and Disk Operating System) monitorea las instalaciones de Aperture Science mediante una extensa red de cámaras y sensores, y controla varios sistemas y cámaras de prueba a través de sus actuadores. Está percibiendo constantemente (acciones del sujeto de prueba, estado de la instalación, resultados de test) y actuando (reconfigurar cámaras de prueba, desplegar torretas, liberar neurotoxina) para llevar a cabo sus "pruebas". Podemos imaginar la agent function de GLaDOS como mapear los estados de las cámaras de prueba y el comportamiento del sujeto a las respuestas adecuadas (si el sujeto resuelve un rompecabezas, introducir un nuevo elemento de prueba; si intenta escapar, activar medidas defensivas). Esto ilustra un agente **rational** (racional) que intenta hacer lo correcto basándose en sus percepciones (¡aunque su definición de "correcto" tal vez no se alinee mucho con los valores humanos!). A lo largo de este capítulo y del texto, formalizaremos lo que significa hacer "lo correcto", relacionándolo al final con conceptos como medidas de desempeño y racionalidad.

PROF



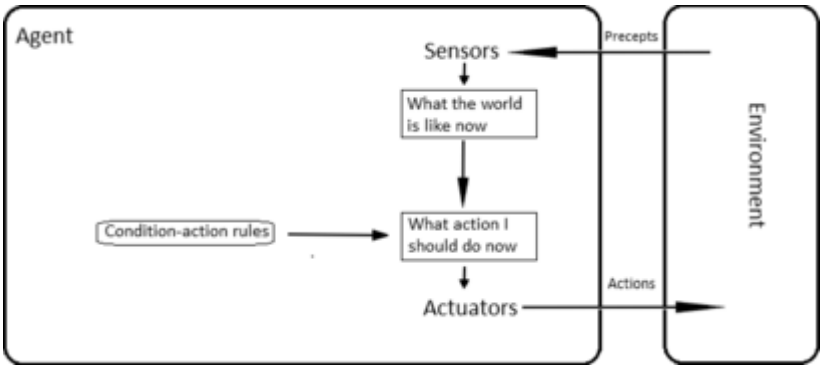

*(GLaDOS de Portal, observando y controlando sus cámaras de prueba – un agente de AI por excelencia en acción.)*

En términos matemáticos, describimos a un agente mediante una **agent function** que mapea cualquier secuencia de percepciones a una acción (Chapter 2 Intelligent Agents — Artificial Intelligence documentation). Si denotamos (P^*) como el conjunto de todas las posibles secuencias de percepciones y (A) como el conjunto de todas las acciones, entonces el comportamiento de un agente es una función (f: P^* \to A)* (Intelligent agent - Wikipedia). Por ejemplo, si (f) asigna la secuencia de percepciones "inbox has an unread email" a la acción "open email", el agente abrirá un correo cada vez que perciba que está sin leer. El mecanismo real que implementa este mapeo es el **agent program**, que es la implementación concreta (código o reglas) que se ejecuta en un sistema físico. Es importante distinguir la agente function abstracta del agent program que la implementa (Chapter 2 Intelligent Agents — Artificial Intelligence documentation): la función es como la descripción idealizada del comportamiento, mientras que el programa es el código real que se ejecuta en el mundo.

Para concretar estas ideas, piensa en una analogía cotidiana: **responder un mensaje en tu teléfono**. La notificación en tu teléfono (percept) te informa que hay un mensaje nuevo; entonces decides cómo responder (action) basándote en quién lo envía y lo que dice (contexto del environment). Si es tu jefe enviando un correo a las 9 AM, podrías responder de inmediato; si es un spam cualquiera, lo ignoras. Aquí, *tú* eres el agente, tu teléfono proporciona las percepciones y tus respuestas son las acciones. Otro ejemplo es el **algorithm** de recomendaciones de videos de TikTok: percibe tu comportamiento de visualización (cuánto tiempo miras, qué te gusta) y actúa seleccionando el siguiente video que te mostrará. Con el tiempo se adapta para lograr mejor su objetivo de mantenerte enganchado.

PROF

En la cultura popular, un ejemplo clásico de agente de AI es **GLaDOS de la serie *Portal***. GLaDOS (Genetic Lifeform and Disk Operating System) monitorea las instalaciones de Aperture Science mediante una extensa red de cámaras y sensores, y controla varios sistemas y cámaras de prueba a través de sus actuadores. Está percibiendo constantemente (acciones del sujeto de prueba, estado de la instalación, resultados de test) y actuando (reconfigurar cámaras de prueba, desplegar torretas, liberar neurotoxina) para llevar a cabo sus "pruebas". Podemos imaginar la agent function de GLaDOS como mapear los estados de las cámaras de prueba y el comportamiento del sujeto a las respuestas adecuadas (si el sujeto resuelve un rompecabezas, introducir un nuevo elemento de prueba; si intenta escapar, activar medidas defensivas). Esto ilustra un agente **rational** (racional) que intenta hacer lo correcto basándose en sus percepciones (¡aunque su definición de "correcto" tal vez no se alinee mucho con los valores humanos!). A lo largo de este capítulo y del texto, formalizaremos lo que significa hacer "lo correcto", relacionándolo al final con conceptos como medidas de desempeño y racionalidad.




*(Un agente interactuando con su entorno, mostrando sensores que reciben percepciones y actuadores que ejecutan acciones.)*

## 2. Tipos de Agentes

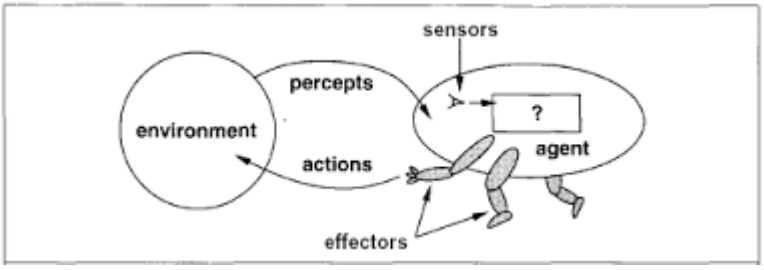
No todos los agentes se construyen de la misma manera. Dependiendo de cómo un agente utiliza sus percepciones para decidir sus acciones, clasificamos los agentes en varios **tipos**. En esta sección, presentamos los tipos clave de agentes en orden creciente de sofisticación: **agentes reflexivos simples**, **agentes reflexivos basados en modelo**, **agentes basados en metas**, **agentes basados en utilidad** y **agentes que aprenden** (Intelligent agent - Wikipedia) (Intelligent agent - Wikipedia). Cada tipo tiene un mecanismo de decisión diferente, desde lo puramente reactivo hasta lo plenamente adaptativo. Hablaremos de cómo funcionan y daremos ejemplos intuitivos — desde termostatos y bots de videojuegos hasta autos autónomos y sistemas de recomendación — para ilustrar cada tipo.

### Agentes Reflexivos Simples (Simple Reflex Agents):


PROF

*(Un ejemplo perfecto de un agente reflexivo en acción: el parry en Super Smash Bros. El agente ve un ataque entrante (percepción) y responde inmediatamente con una acción defensiva.)*

Un *simple reflex agent* selecciona acciones basadas únicamente en la **percepción actual**, ignorando el resto del historial de percepciones (Intelligent agents | Psychology Wiki | Fandom). Este es el tipo más sencillo de agente: *reacciona* a lo que detecta en este momento, sin pensar en el pasado ni en el futuro. La regla de decisión básicamente es: "**if condition, then action**," a menudo implementada como un conjunto de reglas condición–acción (Intelligent agent - Wikipedia).




Por ejemplo, un agente reflexivo simple para una aspiradora podría tener reglas como "si percibe suciedad, aspira" y "si choca con la pared, gira". No hay estado interno ni memoria de percepciones pasadas: el agente reacciona *solo* a la situación actual.

Debido a que no consideran ningún historial, los agentes reflexivos simples funcionan mejor en entornos **totalmente observables** — donde la percepción actual le dice al agente *todo lo que necesita saber* para actuar correctamente (Intelligent agent - Wikipedia). Si los sensores del agente pueden obtener siempre el estado completo del entorno, una simple regla puede mapear ese estado a una acción óptima. Sin embargo, en entornos parcialmente observables, los agentes reflexivos simples pueden tener problemas. Podrían repetir acciones o quedarse atascados en bucles porque no tienen memoria. (De hecho, **los bucles infinitos a menudo son inevitables** para los agentes reflexivos simples en escenarios con observación parcial (Intelligent agent - Wikipedia). A veces, introducir algo de aleatoriedad en las acciones puede ayudar, pero la limitación fundamental persiste). A pesar de sus límites, los agentes reflexivos pueden ser muy efectivos en tareas sencillas.

**Ejemplos:** Piensa en un **termostato** básico de tu casa. Es un agente reflexivo simple que mide la temperatura actual (percept) y enciende o apaga la calefacción o el aire acondicionado (action) basándose en una regla: *si la temperatura está por debajo del objetivo, enciende la calefacción; si está por encima, apágala*. El termostato no recuerda cuál era la temperatura hace una hora, solo reacciona a la lectura actual. Esto encaja perfectamente en el modelo de agente reflexivo simple (What are the Different Types of AI Agents?). Otro ejemplo es un personaje no jugador (NPC) en un videojuego simple que no aprende ni planifica, sino que solo reacciona. Considera un bot de **Mario Kart** que gira de inmediato cuando detecta que un caparazón rojo viene detrás. Podría tener una regla "si viene un caparazón (percepción actual), esquívalo". No hace estrategia de carrera ni recuerda vueltas anteriores, solo responde de forma reflexiva a la amenaza inmediata. Muchos AIs de juegos de arcade clásicos (como los fantasmas en *Pac-Man* en su modo de persecución) son esencialmente agentes reflexivos que responden a la posición actual del jugador.

PROF

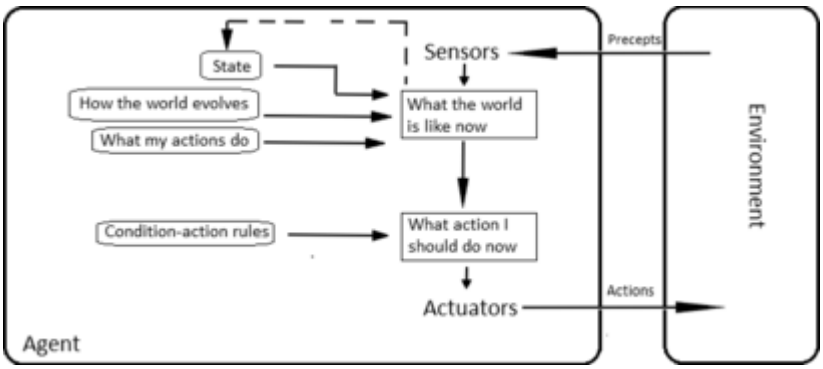
Podemos formalizar un agente reflexivo simple con pseudocódigo para su agent program: evalúa la percepción actual con un conjunto de condiciones y devuelve la acción correspondiente. Por ejemplo:

```
IF percept == "temperature < 20°C" THEN action := "heater_on"
IF percept == "temperature >= 20°C" THEN action := "heater_off"
```

Este enfoque basado en reglas lógicas es directo y, a menudo, eficiente. La simplicidad hace que los agentes reflexivos sean fáciles de diseñar para los problemas adecuados (y fáciles de verificar, ya que el comportamiento es un mapeo directo de percepciones a acciones). Sin embargo, cuando el entorno es más complejo — requiere recordar cosas o hay información oculta — necesitamos agentes más capaces.

## Agentes Reflexivos Basados en Modelo (Model-Based Reflex Agents):

Un *model-based reflex agent* es un paso más en capacidad. Además de las percepciones actuales, **mantiene algún estado interno** que depende del historial de percepciones (Intelligent agent - Wikipedia). Este estado interno es una representación (un "modelo") de aspectos del mundo que el agente no puede ver directamente en el momento. Al actualizar su modelo interno, el agente puede "recordar" o inferir información importante sobre el mundo, lo que le permite actuar con más efectividad que un agente reflexivo simple cuando el entorno no es totalmente observable.




En esencia, un agente basado en modelo tiene dos pasos en su ciclo de decisión:

1. **Actualización de estado:** Primero, usa la nueva percepción y su anterior estado interno para actualizar su comprensión del mundo. Esto requiere cierto conocimiento de "cómo funciona el mundo" — el modelo que el agente tiene sobre la dinámica del entorno (Intelligent agent - Wikipedia). Por ejemplo, si el agente pierde de vista un objeto, puede usar su modelo para inferir dónde podría haberse movido.
2. **Selección de acción:** Luego, elige una acción basándose en el estado actualizado (a menudo sigue usando reglas simples de condición–acción, pero aplicadas ahora al *estado inferido* en lugar de solo a la percepción inmediata).

El modelo interno puede ser desde un booleano sencillo ("tengo la llave o no") hasta un mapa complejo ("esta es la distribución de todo el laberinto según lo he explorado"). Esto es lo que llamamos el **estado** del agente, una representación del entorno que el agente mantiene. El modelo del entorno podría incluir leyes físicas, reglas del juego o simplemente un registro de percepciones pasadas que no han cambiado (por ejemplo, "la puerta X estaba abierta hace 5 minutos y asumo que sigue abierta a menos que perciba lo contrario").

PROF

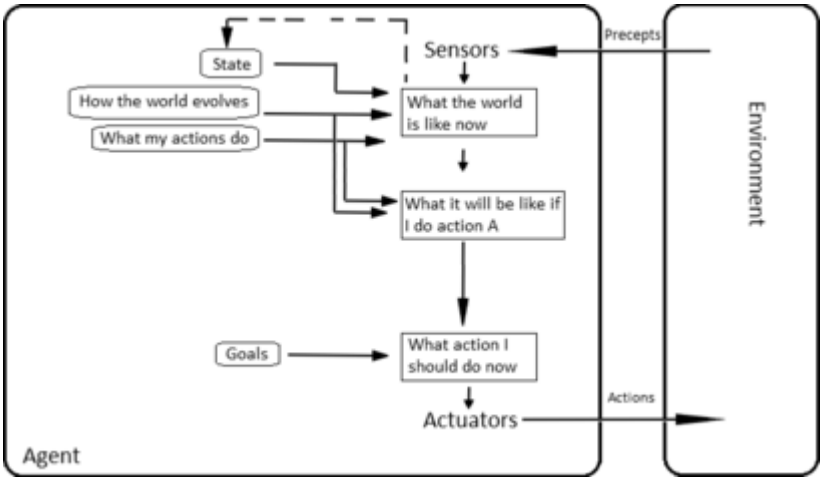
**Ejemplos:** Un buen ejemplo de un agente basado en modelo es un **robot aspirador (smart vacuum cleaner)** que construye un mapa de tu casa. A medida que se mueve, recuerda dónde hay obstáculos (muebles) y qué áreas ya limpió. Sus sensores, en un momento dado, quizás no vean toda la habitación, pero mantiene un mapa interno (modelo) para evitar chocar con cosas que están fuera de su rango de detección. Si está en el pasillo y la base de carga está en la sala, puede dirigirse en la dirección correcta incluso si no puede ver la base en ese instante, porque su estado interno "recuerda" el plano de la casa. Otro ejemplo de la cultura popular es **Light Yagami en *Death Note*** (un ejemplo humano de "agente"). Light siempre va varios pasos por delante, manteniendo un modelo interno del conocimiento y los planes de L. No solo reacciona a lo que L le dice en el momento; considera lo que L *sabe hasta ahora* y qué movimientos podría hacer, y luego Light actúa (escribe un nombre en la Death Note, etc.) basándose en ese modelo interno. En esencia, el "agent program" de Light actualiza un modelo mental

de las creencias de su oponente y lo usa para elegir acciones — un enfoque muy basado en modelo (¡y astuto!).

En términos de AI, muchos agentes que juegan videojuegos utilizan estado interno. Un programa de ajedrez, por ejemplo, actualiza la posición de las piezas en su memoria interna a medida que se realizan los movimientos; no necesita escanear visualmente el tablero cada turno. "Sabe" dónde están las piezas a partir de su modelo del estado del juego. Los agentes reflexivos basados en modelo son poderosos porque combinan reacciones inmediatas con una memoria del mundo, permitiéndoles tomar decisiones más **informadas**. Son un paso previo a los agentes basados en metas y en utilidad, formando a menudo la columna vertebral de esos agentes más avanzados al proporcionar la información de estado necesaria para hacer planificación o cálculos de utilidad.

### Agentes Basados en Metas (Goal-Based Agents):

Mientras que los agentes reflexivos (simples o basados en modelo) actúan en el momento, los **agentes basados en metas** miran más allá. Un agente basado en metas elige acciones con una **meta** específica en mente. No se conforma con reaccionar de cualquier forma; considera las *consecuencias futuras* de sus acciones y si lo acercarán o no a su meta. Dicho de otro modo, al agente se le da (o define) una meta que describe un resultado deseado, y trata de **planificar** una secuencia de acciones para lograr ese resultado (Intelligent agent - Wikipedia).




PROF

Tener una meta explícita permite al agente evaluar posibles acciones preguntándose: "¿Me acerca esta acción (o secuencia de acciones) a mi meta?" Esto a menudo implica **search** (búsqueda) y **algoritmos de planificación**. El agente podría simular o considerar estados futuros del entorno (usando su modelo) para ver qué acción es mejor. Este es un gran salto en complejidad: el agente puede explorar *escenarios hipotéticos* internamente. En este punto entran en juego técnicas clásicas de AI como breadth-first search, depth-first search, Dijkstra o el algoritmo A*, ya que el agente planifica rutas o secuencias de movimientos para alcanzar la meta.

Una desventaja es que los agentes puramente basados en metas consideran solo si se alcanza o no la meta, sin importar lo *eficiente* que sea el camino (eso es lo que los agentes basados en utilidad abordan a continuación). Pero incluso sin una medida de utilidad, los agentes basados en metas suelen ser más flexibles que los reflexivos. Pueden operar en distintos entornos adaptando su secuencia de acciones para lograr la meta, en vez de seguir un patrón fijo de reacción. La contrapartida es que planificar puede

ser más lento que reaccionar, y si la meta se establece de forma incorrecta o es inalcanzable, el comportamiento del agente puede ser deficiente.

**Ejemplos:** Un ejemplo directo de un agente basado en metas es un **sistema de navegación GPS** en un juego como *The Legend of Zelda: Breath of the Wild*. Supón que pones una marca en el mapa (tu ubicación meta); la IA de búsqueda de caminos planificará una ruta desde la posición actual de Link hasta esa meta, considerando el terreno y los senderos transitables. Está eligiendo acciones (p. ej., "avanzar 10 pasos al norte, luego al oeste por el puente, luego escalar la colina") que, según sus predicciones, llevarán a Link hacia la meta. Otro ejemplo es un **auto autónomo**. Su meta principal podría ser "conducir hasta la ubicación X obedeciendo las leyes de tránsito y evitando obstáculos". La IA del auto planifica la ruta y toma decisiones continuamente (acelerar, frenar, girar, cambiar de carril) que no son solo reacciones, sino acciones dirigidas a lograr esa meta de llegar al destino de manera segura. Si aparece un cierre de calle inesperado, el auto puede replanificar una ruta nueva para lograr aún la meta. En ambos casos, el agente necesita un modelo de cómo las acciones afectan al mundo (cómo se mueve el auto, qué carreteras conectan dónde) y un estado meta al que apuntar.

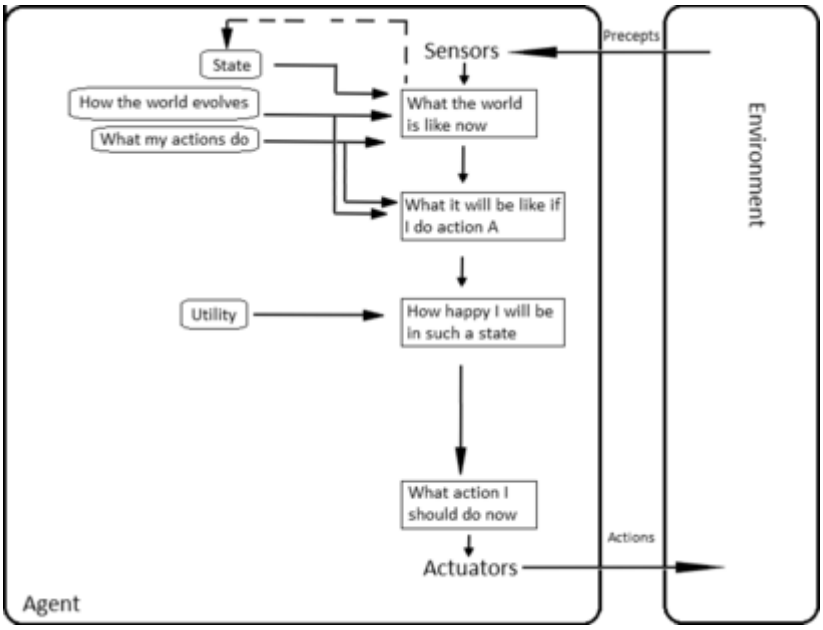
Los agentes basados en metas destacan en problemas de **búsqueda y planificación**. Por ejemplo, en un rompecabezas como Sudoku, la meta podría ser una cuadrícula completada que cumpla todas las reglas. Un agente que resuelva Sudoku considerará movimientos futuros (llenar casillas) que lo lleven eventualmente a la meta de un rompecabezas resuelto, en lugar de limitarse a reaccionar paso a paso. Del mismo modo, en ajedrez, una perspectiva basada en metas podría ser "dar jaque mate al oponente", aunque es difícil codificarlo directamente, los motores de ajedrez usan esencialmente la meta de ganar y realizan búsqueda de múltiples jugadas para decidir la mejor acción. Evalúan estados intermedios mediante heurísticas que estiman el progreso hacia la meta (un indicio de la utilidad).

Cabe señalar que a veces incorporar metas puede hacer que un agente sea menos eficiente si no incorpora también alguna medida de *costo* o *preferencia*. Por ejemplo, imagina la navegación: si hay dos rutas a la meta, una muy larga y otra corta, un agente puramente basado en metas las ve como igualmente válidas (al final "llego de todas formas"), mientras que un **agente basado en utilidad** (siguiente sección) preferiría la ruta más corta porque brinda mayor utilidad (menos tiempo o combustible). En la práctica, muchos agentes de AI combinan el razonamiento basado en metas con utilidades o heurísticas para obtener lo mejor de ambos mundos. Pero como concepto, los agentes basados en metas introducen la idea fundamental de **planificar** con un propósito, lo cual fue un gran paso en la concepción de AI.

PROF

## Agentes Basados en Utilidad (Utility-Based Agents):

Las metas por sí solas solo especifican éxito o fracaso (llegamos a la meta o no), pero a menudo nos importa *qué tan bien* logramos algo. Aquí es donde entran los **agentes basados en utilidad**. Un agente basado en utilidad es similar a uno basado en metas pero con una capa adicional: tiene una **utility function** (función de utilidad) que asigna un valor numérico (una utilidad) a cada posible estado (o resultado), representando el grado de satisfacción o deseabilidad de ese estado (Intelligent agent - Wikipedia) (Intelligent agent - Wikipedia). El agente no solo busca *cualquier* logro de la meta, sino maximizar su **utilidad esperada**.




Al usar una función de utilidad, el agente puede comparar distintos resultados y seleccionar no solo un estado meta, sino el *mejor* entre posibles estados meta o la mejor forma de alcanzarlo. En escenarios con incertidumbre o compensaciones, los agentes basados en utilidad sobresalen. Preferirán la acción que, según sus cálculos, conducirá a la utilidad más alta, teniendo en cuenta las probabilidades de varios resultados (si el entorno es estocástico) (Intelligent agent - Wikipedia). En otras palabras, la acción (a^*) *elegida suele ser la que maximiza la utilidad esperada:*
$$a^* = \arg\max_{a \in A} \mathbb{E}[U(\text{State}_{\text{resultado de }a})],$$
donde (U) es la función de utilidad y la expectativa considera la incertidumbre en el resultado de la acción (a).

Diseñar la función de utilidad depende de la tarea. Para un auto autónomo, la utilidad podría ser una combinación de factores como minimizar el tiempo de viaje, reducir el riesgo de colisión y aumentar la comodidad del pasajero. Para un agente que juega, la utilidad podría ser la probabilidad de ganar dada la situación actual (que el agente intenta maximizar). Los agentes basados en utilidad esencialmente buscan **optimizar** algo en lugar de simplemente lograrlo. Esto conlleva una toma de decisiones más compleja, a menudo usando técnicas de la teoría de la decisión y la economía. Estos agentes pueden manejar compensaciones: por ejemplo, un asistente personal basado en utilidad podría sopesar la *utilidad de interrumpirte con una notificación* contra la *utilidad de dejarte trabajar sin molestias* y solo avisarte si la importancia del mensaje (ganancia de utilidad por informarte) supera la molestia (pérdida de utilidad).

PROF

**Ejemplos:** Un ejemplo típico es un **sistema de recomendaciones en línea** (como las sugerencias de películas de Netflix o la página "Para Ti" de TikTok). Tal sistema puede verse como un agente que, a cada paso, elige un ítem (película o video) para mostrarte. El "objetivo" no es solo mostrar cualquier video, sino maximizar la satisfacción o el engagement del usuario: efectivamente una utilidad. El algoritmo tendrá una función de utilidad que predice cuánto te gustará o te atraerá cierto contenido, y elegirá el que tenga la utilidad esperada más alta ( Q&A: How TikTok's 'black box' algorithm and design shape user behavior | UW News ). En el caso de TikTok, las acciones (videos recomendados) se eligen maximizando la utilidad pronosticada (donde la utilidad podría correlacionar con el tiempo de visualización o la probabilidad de que des "like"), para mantenerte en la app. Otro ejemplo: considera un **Pokémon battle AI** (o un jugador humano actuando como un agente). Puede tener la meta de ganar el combate, pero un

enfoque basado en utilidad asignaría valores a estados intermedios (como tener más HP, o reducir el HP del oponente). Un *Pokémon trainer agent* podría evaluar cada movimiento según el daño esperado, la probabilidad de éxito y la ventaja futura obtenida, usando en esencia una función de utilidad que captura lo deseable del estado de la batalla tras esa jugada. Si el Movimiento A tiene 90% de probabilidades de acertar con daño moderado y el Movimiento B tiene 50% de probabilidades de acertar pero sería un KO de un golpe, la decisión basada en utilidad sopesa las utilidades esperadas (daño * probabilidad, etc.) para decidir. El movimiento con mayor utilidad esperada (dependiendo de la situación y la preferencia de riesgo codificada) será elegido. Esto es más matizado que una simple regla basada en metas: "si hay una jugada que eventualmente podría llevar a la victoria, elige la primera que encuentres". El agente basado en utilidad busca la *mejor* jugada según su medida de desempeño (que podría ser ganar rápido con el mínimo daño, por ejemplo).

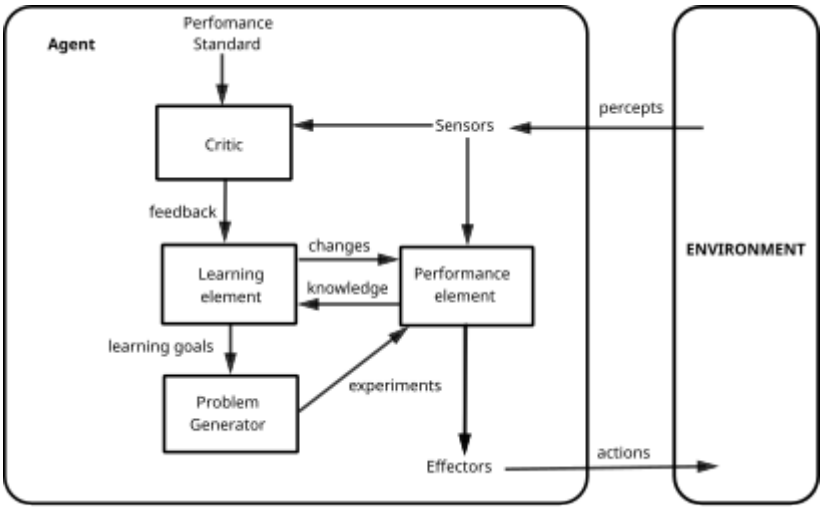
En robótica, las funciones de utilidad se usan a menudo cuando un agente tiene múltiples objetivos. Por ejemplo, un dron de reparto podría tener una función de utilidad que combine velocidad de entrega, consumo de energía y seguridad. Entonces planificará un recorrido de vuelo que maximice una puntuación global de utilidad — que podría ser una suma ponderada de esos factores — para decidir la ruta óptima (el clima y los obstáculos introducen incertidumbre que también debe tenerse en cuenta).

Al incorporar utilidades, los agentes pueden manejar mejor entornos **estocásticos**. En un entorno determinista, basta con alcanzar la meta (y muchos caminos pueden lograrlo); en un entorno estocástico, distintas acciones podrían conducir a la meta con diferentes probabilidades o costos. Un agente basado en utilidad maneja esto naturalmente al favorecer las acciones que *en promedio* dan resultados mejores. En resumen, un agente basado en utilidad se puede ver como un agente basado en metas con una capa de optimización adicional: no solo apunta a *tener éxito*, sino a *el mejor éxito posible* según una métrica de desempeño dada.

### Agentes que Aprenden (Learning Agents):

La última (y en muchos sentidos la más poderosa) categoría es el *learning agent*. Un **learning agent** está diseñado para mejorar su desempeño con el tiempo aprendiendo de sus experiencias. A diferencia de los tipos de agentes anteriores, que tienen comportamientos fijos definidos por reglas, modelos o utilidades incorporados, un learning agent puede empezar con un conocimiento inicial y luego **refinar o aumentar automáticamente** ese conocimiento al observar los resultados de sus acciones en el entorno (Intelligent agent - Wikipedia). Esto le permite manejar situaciones que el diseñador no había anticipado explícitamente.

PROF




Podemos pensar en un learning agent con cuatro componentes conceptuales:

- **Performance Element:** Es como el agente subyacente (sea reflexivo, basado en metas, etc.) que elige acciones en el entorno. Es lo que hemos discutido hasta ahora.
- **Learning Element:** Este componente se encarga de realizar mejoras. Observa lo que hace el performance element y qué tan bien lo hace, y ajusta el performance element para que actúe mejor en el futuro (Intelligent agent - Wikipedia).
- **Critic:** La parte del agente que da retroalimentación sobre cómo lo está haciendo el agente en función de alguna medida de desempeño. (Por ejemplo, la puntuación de un juego o una señal de recompensa del entorno pueden servir de feedback para guiar el aprendizaje).
- **Problem Generator:** Este componente podría sugerir acciones exploratorias para ayudar al agente a aprender más sobre su entorno. (En términos de aprendizaje por refuerzo, esto se relaciona con la disyuntiva exploración-explotación).

En muchos learning agents, especialmente los que usan **reinforcement learning (aprendizaje por refuerzo)**, el entorno proporciona recompensas o castigos (retroalimentación positiva o negativa) y la tarea del agente es aprender una policy (mapeo de percepciones/estados a acciones) que maximice la recompensa acumulada. Al principio, el agente podría actuar de forma subóptima (incluso aleatoria) mientras explora lo que hacen las acciones. Pero a medida que acumula experiencia, comienza a favorecer las acciones que producen recompensas más altas. Así es exactamente como los AIs modernos que juegan han alcanzado un desempeño sobrehumano: a través de millones de interacciones de prueba y error, aprendiendo gradualmente qué estrategias generan la recompensa más alta (victorias).

PROF

**Ejemplos:** Uno de los learning agents más familiares en la vida cotidiana es el **filtro de spam** de tu correo electrónico. Comienza con cierto conocimiento inicial sobre cómo es el spam, pero conforme marcas correos como spam o no spam, aprende de esos ejemplos. Con el tiempo, se adapta a nuevos tipos de mensajes de spam que no había visto antes. Esencialmente, el performance element del filtro clasifica los correos, y el learning element actualiza su modelo interno (quizás un clasificador de machine learning) basándose en qué correos los usuarios marcan como spam. Con cada retroalimentación, el filtro mejora su tarea (What are the Different Types of AI Agents?). Otro ejemplo es el **algorithm** de recomendaciones de TikTok que mencionamos: no solo es un agente basado en utilidad, sino también un learning agent. Actualiza continuamente su modelo de tus preferencias basándose en tus interacciones. De hecho, el sistema de TikTok se ha hecho famoso por lo rápido que

aprende lo que te gusta y lo que no, hasta el punto de que a veces se bromea con que *"el algoritmo de TikTok me conoce mejor que yo mismo."* ( Q&A: How TikTok's 'black box' algorithm and design shape user behavior | UW News ) Ese nivel tan alto de personalización se logra con un learning agent que procesa enormes cantidades de datos de usuario para refinar las recomendaciones de contenido.

En los videojuegos, las IA modernas suelen emplear aprendizaje. Considera una **Street Fighter AI** que se adapta a tu estilo de juego. Quizás al principio es fácil de vencer, pero conforme "ve" más de tus movimientos, empieza a predecirlos y contrarrestarlos con más eficacia. Esto podría implementarse con un learning agent que actualiza su estrategia (o las probabilidades de seleccionar ciertos contraataques) con base en lo que ha observado del jugador humano. Al final de una larga sesión, puede haber "aprendido" que siempre inicias la ronda con una patada en salto y, por tanto, comienza a anticiparse con un movimiento anti-aéreo. Este tipo de adaptación es lo que hace que los agentes que aprenden resulten interesantes (¡o frustrantes!) porque no permanecen estáticos.

En su funcionamiento interno, los agentes que aprenden usan técnicas de **machine learning** — incluidas redes neuronales, árboles de decisión, algoritmos genéticos, etc. Una subclase de estos agentes, conocidos como **agentes de reinforcement learning**, es especialmente destacada. Usan algoritmos (como Q-learning, policy gradients, Deep Q Networks, etc.) para aprender a partir de la retroalimentación de recompensas. El agente podría empezar sin saber ni las reglas del entorno, pero con prueba y error descubre qué acciones dan mayores recompensas. Los éxitos más famosos de estos agentes incluyen dominar juegos de Atari directamente desde píxeles, vencer a campeones mundiales de Go (*AlphaGo*), y alcanzar un juego sobrehumano en videojuegos complejos como StarCraft (*AlphaStar*). Estos sistemas comenzaron sin conocimiento más allá de percepciones y acciones básicas, y aprendieron a dominar sus entornos únicamente de la experiencia.

Es importante recalcar que el aprendizaje suele superponerse con otros tipos de agentes. Puedes tener un agente reflexivo **que aprende** (aprende qué reglas condición-acción funcionan mejor), un agente basado en metas **que aprende** (aprende cuáles metas son alcanzables o cómo lograrlas más rápido), etc. El aprendizaje da al agente **autonomía**: la capacidad de mejorar sin requerir reprogramación constante. A largo plazo, un buen learning agent puede superar el desempeño que su diseñador podría haber incorporado manualmente, porque puede descubrir estrategias novedosas. Esto hace que los agentes que aprenden sean especialmente adecuados para entornos complejos, cambiantes o desconocidos, donde el comportamiento preprogramado sería frágil.

PROF

## 3. El Entorno Importa

*(2B de NieR: Automata demuestra el comportamiento de un agente basado en modelos: mantiene un modelo interno del terreno y los obstáculos mientras navega por un entorno complejo.)*

Ya vimos diferentes diseños de agentes; ahora es crucial notar que **la elección del tipo de agente a menudo depende del entorno en el que opera**. En AI, se clasifican los entornos por un conjunto de propiedades que influyen en qué tipo de agente es adecuado (Chapter 2 Intelligent Agents — Artificial Intelligence documentation) (Chapter 2 Intelligent Agents — ArtificialIntelligence documentation). Aquí hay algunas propiedades clave y cómo impactan el diseño de agentes:

- **Fully observable vs. Partially observable:** En un entorno **fully observable**, los sensores del agente pueden acceder al estado completo del mundo en cada paso de tiempo (Chapter 2 Intelligent Agents — Artificial Intelligence documentation). Esto significa que el agente no necesita mantener estado interno para inferir información faltante: el entorno le dice todo lo que necesita saber. El ajedrez es un ejemplo clásico: la configuración entera del tablero es conocida por ambos jugadores en todo momento. Un agente reflexivo simple (o de planificación sencilla) puede bastar en un mundo completamente observable porque no hay información oculta. En un entorno **partially observable**, las percepciones del agente proporcionan información incompleta del estado del mundo. Piensa en el **póker**: ves tus propias cartas pero no las del rival, así que debes inferir lo que podrían tener. Aquí, un agente usualmente necesita memoria o un modelo (por eso agentes basados en modelo) para hacer seguimiento de posibilidades e historiales. La mayoría de entornos del mundo real son parcialmente observables debido al ruido, sensores de alcance limitado o variables ocultas. Por ejemplo, un auto autónomo no puede ver detrás de las esquinas: puede necesitar mantener una creencia de lo que podría venir. Como dijimos, los agentes reflexivos simples rinden mal en entornos parcialmente observables porque no tienen memoria; los agentes basados en modelo o los que aprenden manejan mejor este tipo de escenarios al recordar o estimar partes del estado que no se observan directamente.

- **Deterministic vs. Stochastic:** En un entorno **deterministic**, el siguiente estado está totalmente determinado por el estado actual y la acción del agente (Chapter 2 Intelligent Agents — Artificial Intelligence documentation). No hay aleatoriedad en cómo evoluciona el entorno. Rompecabezas de física clásica o un Sudoku son deterministas: si tomas una acción, sabes exactamente lo que obtendrás. En tales casos, la planificación es más sencilla, ya que no tienes que contemplar resultados probabilísticos. Un entorno **stochastic**, por otro lado, tiene cierta aleatoriedad. La misma acción en el mismo estado podría conducir a resultados diferentes (a menudo modelados

PROF

con probabilidades). **La predicción del clima** es un entorno estocástico — incluso con las mismas condiciones atmosféricas, el tiempo podría variar, y un agente (por ejemplo, un AI para predecir el tiempo) debe manejar incertidumbre. Los entornos deterministas permiten agentes más directos: un agente basado en metas puede planificar una secuencia de acciones única sabiendo que funcionará. En entornos estocásticos, a menudo se vuelve importante un enfoque basado en utilidad, porque el agente debería evaluar las acciones según sus resultados esperados. Por ejemplo, considera un agente automatizado de inversiones en la bolsa: el entorno (mercado) es altamente estocástico. Un agente puramente basado en metas ("maximizar ganancias") sin utilidades y probabilidades tendría dificultades, porque necesita evaluar el riesgo y la rentabilidad esperada. En cambio, un agente basado en utilidad que maneje probabilidades (quizás maximizando rentabilidad esperada menos penalización por riesgo) es más adecuado. Muchos agentes de **reinforcement learning** se diseñan para entornos estocásticos; manejan de forma inherente la recompensa esperada.

- **Static vs. Dynamic:** Un entorno **static** es aquel que no cambia mientras el agente delibera. Si nada cambia excepto por las acciones del agente, el entorno es estático (Chapter 2 Intelligent Agents — Artificial Intelligence documentation). Un entorno **dynamic** puede cambiar "por sí solo" a medida que pasa el tiempo, o debido a otros agentes. Por ejemplo, un **Sudoku** es estático: el rompecabezas no va a modificar sus números de repente mientras piensas, puedes tomarte tu tiempo. Por el contrario, un **juego de estrategia en tiempo real** como StarCraft es dinámico: incluso mientras tu agente de IA planifica la siguiente jugada, el oponente podría mover unidades o se podrían agotar recursos. Los entornos dinámicos exigen que los agentes sean rápidos y, a menudo, **respondan continuamente**. En un mundo dinámico, un agente que planifica podría necesitar mezclar planificación y ejecución (o replanificar sobre la marcha) porque si planifica demasiado tiempo, el mundo podría haber cambiado cuando quiera actuar. Un agente reflexivo simple a veces puede funcionar bien en entornos dinámicos porque reacciona con rapidez, pero si la tarea requiere pensamiento estratégico a largo plazo, el agente debe planificar bajo presión de tiempo. En escenarios muy dinámicos (como autos autónomos o fútbol robótico), a menudo se diseña un agente con componentes reflexivos (para evitar colisiones inmediatas, por ejemplo) y módulos basados en metas o utilidad (para decisiones de más alto nivel), garantizando tanto seguridad como comportamiento dirigido a objetivos.

PROF

- **Single-agent vs. Multi-agent:** Aunque no está en la lista original, vale la pena mencionarlo: si el entorno tiene otros agentes (especialmente otros agentes inteligentes), es un entorno **multi-agent**. Ejemplos incluyen juegos multijugador, mercados económicos o redes sociales. En esos entornos, un agente podría necesitar considerar las acciones de otros, lo que lleva a conceptos de **teoría de juegos** y estrategias (por ej., cooperación o competencia). Un entorno de redes sociales podría verse como multi-agent (múltiplos algoritmos y usuarios interactuando), aunque a menudo tratamos a los demás como parte de la dinámica del entorno. Los entornos multi-agent pueden ser competitivos (como ajedrez o StarCraft) o cooperativos. Un agente en tales escenarios puede beneficiarse de ser basado en modelo (para modelar a otros agentes) o de aprendizaje (para adaptarse a los comportamientos de los demás). Por ejemplo, un agente de trading en una subasta se enfrenta a otras ofertas; una estrategia reflexiva seguramente no bastará, podría necesitar predecir las pujas de los demás (un modelo de otros agentes) y tener una utilidad para los resultados.

*(Los androides de NieR: Automata interactuando en un entorno multi-agente. Cada agente debe considerar las acciones e intenciones de los otros para coordinar efectivamente.)*

Estas propiedades del entorno a menudo determinan la sofisticación mínima que necesita un agente. Un entorno totalmente observable, determinista y estático (como un rompecabezas sencillo) podría resolverse con un agente de búsqueda (basado en metas) sin requerir aprendizaje ni siquiera estado interno. Si relajamos esas suposiciones — supongamos que se vuelve parcialmente observable o estocástico — se empieza a necesitar estado interno (basado en modelo) o razonamiento de utilidad. Si el entorno es desconocido o está cambiando, el aprendizaje se vuelve muy valioso para que el agente siga siendo eficaz.

Para visualizarlo: **los agentes reflexivos simples** van bien en entornos totalmente observables, deterministas y estáticos (como un sensor automático de puerta que se abre cuando alguien está cerca: el mundo es simple y se observa por completo). **Los agentes basados en modelo** son necesarios cuando el agente debe recordar cosas (entornos parcialmente observables, quizá estáticos o lentos). **Los agentes basados en metas** brillan cuando el agente debe lograr un estado final específico en un mundo razonablemente predecible (como rompecabezas o rutas de navegación). **Los agentes basados en utilidad** son importantes en entornos estocásticos o con múltiples compensaciones, donde no basta con alcanzar una meta, sino que importa *cuál* meta o cuán óptimamente se llega. **Los agentes que aprenden** son esenciales en entornos desconocidos, dinámicos o muy complejos, donde el conocimiento preprogramado se queda corto — pueden adaptarse a la tarea. En la práctica, muchos sistemas reales combinan elementos: por ejemplo, un dron autónomo es un agente que aprende, usa utilidad (para equilibrar rapidez vs. seguridad), tiene metas (llegar a un waypoint), emplea un modelo (física de vuelo) y también tiene reflejos (bucles de estabilización).

PROF

Un caso concreto: **un tablero de ajedrez vs. una mesa de póker**. En ajedrez (entorno completamente observable, determinista y estático por turnos), un agente basado en metas con búsqueda puede lograr un desempeño extraordinario — por eso la IA dominó el ajedrez hace décadas con algoritmos que exploran árboles de juego. En el póker (parcialmente observable, estocástico y con otros agentes), una IA exitosa (como DeepStack o Libratus para póker) tuvo que usar un modelo de creencias (mantener registro de lo que sabe vs. lo que ignora), razonar sobre probabilidades de cartas del oponente (decisiones basadas en utilidad para apostar) y aprender de autopruebas para refinar su estrategia. El entorno realmente dicta el enfoque.

***(Imagen: Un juego de Gungi que representa un entorno completamente observable y determinista. En ajedrez, una IA puede ver todo el estado y planificar varios movimientos por adelantado. En contraste, juegos como póker o situaciones del mundo real ocultan información, requiriendo que los agentes manejen la incertidumbre.)***

## 4. Agentes en Acción

Pasamos de las arquitecturas individuales de agentes a cómo operan en la práctica, es decir, cómo estos diseños se aplican para resolver problemas reales de decisión. Los agentes de AI hoy están por todas partes, a menudo en segundo plano en la tecnología y la sociedad. Veamos algunos dominios:

- **Teoría de Juegos y Dilemas Sociales:** Imagina plataformas de redes sociales donde interactúan usuarios (y a veces bots). Si tratamos a cada usuario o bot como un agente, podemos analizar su comportamiento con teoría de juegos. Un ejemplo famoso es la estrategia **tit-for-tat** en interacciones repetidas (como el juego de Dilema del Prisionero iterado). Tit-for-tat es básicamente una estrategia reflexiva simple: comienza de forma cooperativa, luego haz lo que el otro agente hizo en la jugada anterior. En plataformas como Twitter o foros, puedes notar patrones similares a tit-for-tat: la gente corresponde amabilidad o rudeza. Ahora, si tenemos AI agents moderando o participando, podrían usar estrategias parecidas. Por ejemplo, un agente diseñado para guiar discusiones en línea podría emplear un enfoque basado en modelo: llevar registro de qué usuarios son consistentemente constructivos vs. trolls, y adaptar sus interacciones (quizá ignorar o penalizar a trolls — algo así como una versión de tit-for-tat). Esto combina razonamiento de teoría de juegos y diseño de agentes. En situaciones multi-agent como subastas, sistemas de tráfico o incluso apps de citas, cada AI agent debe considerar a los demás. Capítulos futuros sobre teoría de juegos profundizarán en cómo diseñar agentes para alcanzar equilibrios de Nash o resultados cooperativos.

PROF

- **AI en Esports y Juegos en Tiempo Real:** Los videojuegos modernos, sobre todo esports, son un campo de pruebas para agentes de AI avanzados. Tomemos juegos de estrategia en tiempo real (RTS) como StarCraft II o MOBAs como Dota 2. Estos entornos son parcialmente observables, estocásticos (hay información oculta y a veces probabilidades de golpe), dinámicos y multi-agent (oponentes y aliados). AI agents como **AlphaStar (de DeepMind)** alcanzaron nivel de Gran Maestro en StarCraft II usando una combinación de técnicas: redes neuronales profundas (para evaluación de estados), aprendizaje por refuerzo (para aprender estrategias de partidas contra sí mismo) y entrenamiento de tipo teórico de juegos contra ligas de oponentes (AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning - Google DeepMind) (AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning - Google DeepMind). AlphaStar fue esencialmente un agente basado en aprendizaje y utilidad — su meta

era ganar la partida, pero debía maximizar la recompensa a largo plazo (victorias) en un espacio muy complejo, aprendiendo todo desde cero. En Dota 2, **OpenAI Five** también aprendió a jugar a nivel de campeones a partir de autoentrenamiento, coordinando cinco agentes (uno para cada héroe en el equipo) que aprendieron a cooperar y competir (AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning - Google DeepMind). Estos ejemplos muestran cómo los agentes que aprenden pueden superar los reflejos y la estrategia humana explorando incontables posibilidades. Otro desarrollo interesante: los AI agents ahora se usan para **entrenar a jugadores humanos**. Por ejemplo, un juego podría incluir un AI coach (un agente que analiza la partida y da consejos) o sparrings de IA que ajustan su dificultad mediante aprendizaje para mantener un reto constante. La presencia de aprendizaje y adaptación en estos agentes significa que pueden seguir siendo efectivos incluso cuando las estrategias humanas evolucionan.

- **Sistemas de Recomendación y Personalización:** Como se mencionó, los algoritmos de recomendación en plataformas como TikTok, YouTube, Netflix o Amazon pueden verse como agentes que toman decisiones secuenciales. Cada vez que terminas de ver un video o de comprar un producto, el sistema elige la siguiente recomendación. Su "meta" suele ser maximizar la interacción o la satisfacción del usuario (utility). Estos son casos de **agentes que aprenden basados en utilidad** por excelencia. Emplean enormes volúmenes de datos para aprender modelos predictivos (a menudo modelos de deep learning) de cuál sería tu utilidad frente a un ítem, y luego escogen el ítem con mayor utilidad predicha. Lo interesante es que el sistema también debe considerar la *exploración*: mostrar algo un poco novedoso para aprender más de tus gustos, frente a la explotación: mostrarte lo que es casi seguro que te complazca. Eso se parece mucho al problema clásico de aprendizaje por refuerzo. De hecho, algunos agentes de recomendación se modelan explícitamente como problemas de reinforcement learning (donde los clics del usuario o el tiempo de visualización son recompensas). Con el tiempo, el agente "te conoce" — es un bucle iterativo de percepción (tu interacción) y acción (la siguiente recomendación). Lo vivimos a diario. Cuando TikTok te engatusa tan rápido con contenido que no puedes dejar de deslizar, o cuando Netflix sugiere contenido que se alinea extrañamente con tu estado de ánimo reciente, eso es un AI agent en acción, tomando decisiones basadas en modelos aprendidos de tu comportamiento. Capítulos futuros sobre **Markov Decision Processes (MDPs)** formalizarán esta idea de tomar decisiones secuenciales bajo incertidumbre, que sustenta también estos escenarios de recomendación (cada recomendación influye en lo que harás después, lo cual influye en lo que el agente debe hacer luego).

PROF

- **Vehículos Autónomos y Robótica:** Un auto autónomo es un conjunto de agentes o un agente muy complejo que incorpora múltiples técnicas. Su sistema de percepción es un agente que detecta carriles, peatones, etc. (a menudo usando modelos de aprendizaje automático). El módulo de planificación es un agente basado en metas (meta: navegar hasta el destino). El módulo de decisión podría ser basado en utilidad (equilibra velocidad, comodidad, seguridad). Y, en general, el sistema aprende de datos — tanto offline (entrenando con grandes conjuntos de datos de conducción) como a veces online (adaptándose a preferencias del conductor). En escenarios multi-agent como el tráfico, cada auto puede tener que anticipar lo que harán los demás (¿cruzará ese peatón ahora? ¿es ese conductor agresivo o prudente?). Estos agentes usan **Hidden Markov Models (Modelos Ocultos de Markov)** y **filtros de Kalman** para rastrear y predecir el estado de otros vehículos/peatones (porque no todo se observa directamente, p. ej. la

intención de un auto o un peatón oculto tras un camión). Exploraremos estos modelos en capítulos siguientes, que ayudan a los agentes a manejar la incertidumbre en la percepción a lo largo del tiempo.

La idea clave de estos ejemplos es que *los conceptos de agente impregnan las aplicaciones de AI en el mundo real*. Al entender los tipos de agentes, podemos razonar mejor sobre cómo diseñar AI para un problema dado. Por ejemplo, si nuestro problema es muy dinámico y parcialmente observable (un dron en corrientes de viento), sabemos que seguramente el agente necesitará un modelo interno (para recordar el viento pasado) y quizá un algoritmo de aprendizaje para adaptarse a los patrones de viento, en lugar de un simple conjunto de reglas if-then. Si es un problema puntual, totalmente observable y estático, tal vez no necesitemos aprendizaje o estado interno — una simple búsqueda podría bastar.

Como adelanto de temas futuros: exploraremos formalmente cómo los agentes manejan la incertidumbre y la toma de decisiones secuencial. Esto incluye **Markov Chains (cadenas de Markov)** y **Hidden Markov Models**, que proporcionan marcos matemáticos para tratar transiciones de estado e incertidumbre en la percepción. Con ellos profundizaremos en la parte "modelo" de los agentes basados en modelo, aprendiendo cómo predecir cómo cambia el entorno (Markov Chains) y cómo inferir estados ocultos a partir de percepciones ruidosas (HMMs). Estos son esenciales para construir agentes que puedan desenvolverse en entornos parcialmente observables (piensa en un robot intentando localizarse sin GPS — los HMMs son ideales para ello).

También entraremos en **reinforcement learning**, que básicamente es la ciencia de los agentes que aprenden: cómo un agente puede empezar de cero y aprender un comportamiento óptimo (policy) mediante prueba y error. Esto unirá la idea de utilidad (a través de funciones de recompensa) y el aprendizaje (algoritmos que mejoran la policy del agente para maximizar la recompensa acumulada) en entornos inciertos. Al final de esos apartados, verás cómo un agente puede formular matemáticamente "¿Qué pasa si hago X?" y mejorar con la retroalimentación, lo que es increíblemente poderoso.

Finalmente, hablaremos de la toma de decisiones bajo incertidumbre y de una planificación más compleja (como en escenarios parcialmente observables o multi-agent), tocando posiblemente **teoría de juegos** y **planificación en el espacio de creencias** (cuando el agente planifica no solo sobre estados concretos, sino sobre distribuciones de estados).

PROF

## 5. Formalismo Matemático

Recapitulemos algunos de los conceptos formales introducidos, de un modo un poco más riguroso, y añadamos algo de detalle matemático para afianzar nuestra comprensión:

- **Agent function:** Como se mencionó, el comportamiento de un agente es una función (f: P^* \to A), que mapea historiales de percepciones a acciones (Intelligent agent - Wikipedia). Si denotamos una percepción en el tiempo (t) como (p_t), entonces la secuencia de percepciones hasta (t) es (\langle p_1, p_2, ..., p_t \rangle). La agent function es (f(\langle p_1, ..., p_t\rangle) = a_t), donde (a_t) es la acción que el agente toma tras percibir esa secuencia. Esta es una forma general y abstracta de especificar *qué* hace el agente. Por ejemplo, si (f()sequence of sensor readings where the last reading indicates low battery() = "move to charging dock"), eso describe el comportamiento del agente de ir a cargar cuando la batería está baja. El agent program es una implementación de (f). Podría haber muchos programas que realicen la misma función (f). Un punto clave: desde una perspectiva externa, dos agentes que implementen la misma (f) son

indistinguibles en su comportamiento, sin importar cuán diferentes sean sus mecanismos internos.

- **Racionalidad y medida de desempeño:** A menudo queremos que nuestros agentes sean **rational** (racionales), es decir, que hagan "lo correcto". Más formalmente (de la definición de Russell & Norvig), dado un performance measure que evalúa el éxito de un agente en un entorno, un agente racional en un momento dado selecciona la acción que se espera que maximice su performance measure, *dada la evidencia de su secuencia de percepciones y cualquier conocimiento incorporado* (Chapter 2 Intelligent Agents — Artificial Intelligence documentation). Esta definición implica que la racionalidad no es omnisciencia; el agente toma la mejor decisión *esperada* que puede, pero si algo impredecible sucede, no significa que fuera irracional, solo que tuvo mala suerte. Por ejemplo, una IA de ajedrez puede hacer una jugada que, según su conocimiento, es la mejor (maximiza las probabilidades esperadas de victoria), pero si el oponente hace una jugada brillante e inesperada y la IA pierde, no decimos que la IA fuera irracional, sino que no conocía esa jugada de contraataque en su búsqueda. El performance measure encapsula los objetivos u objetivos múltiples. Para un auto autónomo, la medida de desempeño podría combinar seguridad, legalidad y tiempo de viaje. Para un robot de limpieza, podría ser la cantidad de suciedad limpiada por hora menos los choques con muebles. A menudo diseñamos funciones de utilidad para representar matemáticamente el performance measure en un agente basado en utilidad.

- **Maximización de utilidad y teoría de la decisión:** Un agente basado en utilidad intenta maximizar la utilidad esperada. Supongamos que el agente tiene una función de utilidad (U(s)) para estados (s). Si las acciones posibles son (a_1, a_2, ..., a_n), y ejecutar la acción (a_i) en el estado actual lleva a un conjunto de posibles estados ({s_{i1}, s_{i2}, ...}) con probabilidades (P(s_{ij} | \text{estado actual}, a_i)) (estas provienen del modelo del entorno, tal vez asumiendo un carácter markoviano), entonces la utilidad esperada de la acción (a_i) es:
  [ \mathbb{E}[U | a_i] = \sum_j P(s_{ij} | \text{state}, a_i) ; U(s_{ij}). ]
  El agente basado en utilidad elegiría (a_k = \arg\max_{a_i} \mathbb{E}[U | a_i]). Esta es una fórmula sencilla de teoría de la decisión. Un ejemplo concreto: supón que un agente tiene dos acciones: **llevar un paraguas** o **no llevarlo**. La función de utilidad podría considerar que mojarse es muy malo. Si las percepciones le indican una probabilidad de 40% de lluvia hoy, puede calcular:

PROF

- Si llevo el paraguas, la utilidad podría ser (U_{\text{umbrella}} = U(\text{no mojarse pero cargar paraguas})). Asignemos un valor: quizá no mojarse + cargar paraguas = 0.8 de utilidad (algo de molestia, pero estás seco).

- Si no lo llevo: hay 40% de probabilidad de lluvia (estado: "mojado, sin paraguas") con utilidad 0.0 (mojarse es pésimo), y 60% de no lluvia ("seco, sin paraguas") con utilidad 1.0 (lo mejor, no cargaste nada y permaneciste seco). Entonces la utilidad esperada de no llevarlo = (0.4 * 0.0 + 0.6 * 1.0 = 0.6).
  En este escenario, llevar el paraguas da 0.8 de utilidad segura, que es mayor que 0.6 esperado si no lo llevas. Así que la decisión racional es llevarlo (0.8 > 0.6). Este tipo de cálculo es lo que un agente basado en utilidad formaliza. Suena trivial en el ejemplo, pero la misma lógica se amplía a decisiones complejas con muchos resultados (la matemática se complica y solemos usar algoritmos para aproximar la mejor acción).

- **Razonamiento probabilístico:** Muchos agentes avanzados (especialmente los que aprenden y los basados en modelo en mundos inciertos) emplean razonamiento probabilístico. **Bayesian networks** (redes bayesianas), **Markov chains** (cadenas de Markov) y **Hidden Markov Models** (modelos ocultos de Markov) son herramientas para ayudar al agente a actualizar sus creencias sobre el mundo. Por ejemplo, un robot con un sensor ruidoso puede usar un Hidden Markov Model para mantener una distribución de probabilidad sobre su ubicación (problema de localización). Cuando recibe una nueva percepción, usa la regla de Bayes para actualizar esa distribución (esencialmente lo que hace en segundo plano la parte de aprendizaje/basada en modelo). Formalmente, si un agente tiene la creencia (Bel(s)) de que está en el estado (s), y recibe una percepción (e), puede actualizar:
  [ Bel'(s) = \alpha , P(e | s) , Bel(s), ]
  donde (P(e|s)) viene del modelo del agente (la probabilidad de ver la percepción (e) si el estado verdadero es (s)), y (\alpha) es una constante de normalización. Esta es la esencia de la actualización bayesiana (que exploraremos con detalle después). La conclusión es que los agentes usan ecuaciones como esta para aprovechar al máximo sus percepciones, especialmente cuando no pueden observar directamente el verdadero estado.

Con estas bases formales, estamos preparados para el análisis más detallado de los próximos capítulos. Veremos cómo construir modelos de Markov para los entornos y cómo los agentes pueden hallar políticas óptimas (por ejemplo, en aprendizaje por refuerzo, usando algoritmos que maximizan la recompensa acumulada, otra forma de utilidad).

## 6. Conclusión y Próximos Pasos

En este capítulo, presentamos la noción fundamental de un agente en AI y revisamos los principales **tipos de agentes**: desde agentes reflexivos simples que reaccionan al momento, pasando por agentes basados en modelo con memoria, agentes basados en metas que planifican, agentes basados en utilidad que optimizan y, finalmente, agentes que aprenden, capaces de mejorar por sí mismos. Discutimos cómo estos agentes perciben y actúan en sus entornos y, lo más importante, cómo las **características del entorno** (¿observable? ¿determinista? ¿estático? etc.) influyen en qué tipo de agente es apropiado.

PROF

Para recapitular en términos prácticos:

- Si tienes una tarea y entorno muy claros (como un rompecabezas o un sistema estable), un agente reflexivo o basado en metas con lógica predefinida podría bastar.
- Si tu agente opera en el mundo real con mucha incertidumbre, seguramente emplearás un modelo y, posiblemente, un algoritmo de aprendizaje para enfrentar lo inesperado.
- Las funciones de utilidad añaden una capa de refinamiento, permitiendo que el agente maneje compensaciones y la incertidumbre con optimización numérica.
- El aprendizaje da a los agentes la capacidad de **adaptarse** — un rasgo esencial al adentrarnos en aplicaciones complejas y cambiantes como apps personalizadas, vehículos autónomos y más.

Cada tipo de agente que abordamos es como un bloque de construcción. En la práctica, muchos sistemas de AI son híbridos. Un auto autónomo, por ejemplo, podría usar agentes reflexivos para el control de bajo nivel (mantenerse en el carril), planificación basada en metas para la navegación y

algoritmos de aprendizaje para mejorar su política de conducción. Entender estos arquetipos nos ayuda a razonar sobre las piezas de un sistema de AI y cómo contribuyen al conjunto.

En lo que sigue, profundizaremos en las **técnicas y herramientas** que hacen funcionar a estos agentes. En los próximos capítulos, exploraremos **Markov Chains** y **Hidden Markov Models**, que brindan marcos matemáticos para manejar transiciones de estado e incertidumbre en la percepción. Esto expandirá la parte de "modelo" de los agentes basados en modelo, mostrándonos cómo predecir cambios en el entorno (Markov Chains) y cómo inferir estados ocultos a partir de percepciones ruidosas (HMMs). Estos conceptos son la base para construir agentes que puedan *inferir* información oculta (estimación de estado) y actualizar su modelo interno de forma rigurosa. Luego, profundizaremos en cómo los agentes aprenden políticas óptimas mediante **reinforcement learning**, uniendo así el concepto de aprendizaje y utilidad en entornos inciertos. Es la misma clase de ideas que impulsa los avances sorprendentes de la IA en el juego, la robótica y la personalización.

Por último, veremos la toma de decisiones bajo incertidumbre y la planificación más compleja (como en situaciones parcialmente observables o multi-agent), posiblemente tocando **teoría de juegos** y **planificación en espacio de creencias** (cuando un agente planifica no solo sobre estados concretos, sino sobre distribuciones de estados).

Como ejercicio creativo: piensa en dónde van los agentes. Cada vez más los introducimos en **realidades virtuales y simulaciones** (por ejemplo, personajes de IA en simulaciones de entrenamiento o videojuegos). En un entorno simulado muy complejo, tal vez se requiera todo el arsenal: el agente podría no observar completamente el mundo (necesita un modelo interno), podría tener metas a largo plazo (cumplir una misión en la simulación), debe maximizar alguna utilidad (puntaje o satisfacción del usuario) y, desde luego, aprender (porque el mundo virtual puede ser tan complejo como el real). Es un campo de investigación apasionante preguntarse: *¿Qué sucede si colocamos agentes en realidades virtuales tan ricas e impredecibles como la vida real?* Conseguimos un entorno seguro para que la IA evolucione y aprenda, posiblemente generando agentes que luego se apliquen al mundo real. Esto se relaciona con temas como la transferencia de simulación a la realidad y el aprendizaje continuo.

Antes de pasar al siguiente capítulo técnico, tómate un momento para reflexionar sobre lo aprendido. Partimos de un concepto sencillo (un agente percibe y actúa) y llegamos a ideas bastante sofisticadas (aprender de recompensas, maximizar utilidad esperada). Estos pilares son la columna vertebral de la AI. Como ejercicio, podrías **pensar en tu personaje favorito de un juego o un anime y analizarlo como un agente**. ¿Cuáles son sus percepciones, sus acciones y qué tipo de agente crees que sea (reflexivo, basado en metas, que aprende)? ¿Cómo influye su entorno en su diseño? Por ejemplo, ¿es *Naruto* (del anime) un agente basado en metas (meta: ser Hokage), y quizá también un agente que aprende (mejorar sus habilidades con la experiencia)? O considera a *2B de NieR:Automata*: percibe mediante sensores avanzados y actúa con capacidades de combate, adaptando sus estrategias según la experiencia mientras sigue objetivos de misión. Este tipo de análisis divertido refuerza los conceptos.

PROF

**Tu Turno – Mini Reto:** *Elige un agente de AI de la ficción o de un juego que te guste. En pocas oraciones, describe sus percepciones, sus acciones y qué tipo de agente crees que sea (¿reflexivo? ¿basado en metas? ¿que aprende?). ¿Cómo influye su entorno en su diseño?* Esto no solo pone a prueba tu entendimiento, sino que muestra cómo estas ideas impregnan las historias y juegos que disfrutamos.

Un agente basado en modelos mantiene un estado interno que depende de su historial de percepciones. Esto le permite lidiar mejor con entornos parcialmente observables. Por ejemplo, un

robot aspiradora que construye un mapa de la casa mientras limpia está usando un modelo del mundo para navegar.

En un entorno multi-agente, un agente debe considerar las acciones y decisiones de otros agentes. Esto es común en juegos multijugador, mercados económicos y sistemas robóticos colaborativos. La presencia de múltiples agentes puede llevar tanto a la cooperación como a la competencia.