INSTITUTO TECNOLOGICO DE CULIACÁN

INGENIERIA EN SISTEMAS COMPUTACIONALES

INTELIGENCIA ARTIFICIAL

AGENTES DELIBERATIVOS (BASADOS EN OBJETIVOS)

MAESTRO

ZURIEL DATHAN MORA FELIX

AYALA RODRÍGUEZ JOSÉ ERNESTO

LIZARRAGA VALENZUELA JESUS EDUARDO

HORA:

11:00-12:00

17 de febrero del 2025

Culiacán, Sinaloa

AGENTES DELIBERATIVOS (BASADOS EN OBJETIVOS)

¿Qué son los agentes deliberativos basados en objetivos?

Los agentes deliberativos basados en objetivos son sistemas de inteligencia artificial que combinan características de los agentes reactivos y los basados en modelos. Estos agentes tienen la capacidad de razonar sobre sus objetivos y planificar acciones para alcanzarlos, utilizando experiencias pasadas para tomar decisiones. Además, evalúan los datos del entorno y comparan diferentes enfoques que los ayudan a lograr el resultado deseado, siempre eligiendo el camino más eficiente.

Características

- **Planificación antes de actuar:** No responden de inmediato, sino que analizan y planifican la mejor acción para alcanzar su objetivo.
- **Uso de un modelo del mundo:** Tienen una representación interna del entorno que les permite evaluar las consecuencias de sus acciones.
- **Toman decisiones basadas en objetivos:** No actúan por simple reacción, sino que seleccionan acciones con base en una meta específica.
- **Evaluación de múltiples opciones**: Comparan diferentes caminos o estrategias antes de decidir cuál ejecutar.
- **Capacidad de adaptación:** Si el entorno cambia, pueden modificar su planificación para seguir cumpliendo su objetivo.
- **Mayor uso de recursos computacionales:** Debido a su razonamiento complejo, requieren más memoria y procesamiento en comparación con agentes reactivos.

Ventajas

- **Toma de decisiones más eficiente:** Evalúan diferentes opciones antes de actuar, eligiendo la mejor para alcanzar su objetivo.
- **Mayor flexibilidad y adaptabilidad:** Pueden ajustar sus planes si el entorno cambia, en lugar de reaccionar de forma automática.
- **Uso de modelos del mundo:** Pueden predecir el impacto de sus acciones, lo que les permite planificar con anticipación.
- **Mejor rendimiento en entornos complejos:** Son ideales para problemas que requieren planificación y razonamiento, como la robótica o la logística.

Desventajas

- **Mayor consumo de recursos computacionales:** Debido a la planificación y el razonamiento, requieren más memoria y procesamiento que los agentes reactivos.
- **Dificultad para operar en entornos altamente dinámicos:** Si el entorno cambia muy rápido, el agente podría quedarse "pensando" en lugar de reaccionar a tiempo.
- **Complejidad en su diseño e implementación:** Requieren modelos precisos del entorno y algoritmos avanzados para la toma de decisiones.
- **Latencia en la respuesta:** Como analizan antes de actuar, pueden tardar más en responder en comparación con agentes reactivos.

Ejemplo en código

Ejemplo de un **agente deliberativo basado en objetivos** que planea el camino más corto desde un punto de inicio hasta un objetivo usando el algoritmo **A*:**

```python
import heapq

class AgenteDeliberativo:
    def __init__(self, mapa, inicio, objetivo):
        self.mapa = mapa  # Representación del entorno (grafo)
        self.inicio = inicio
        self.objetivo = objetivo

    def heuristica(self, nodo):
        # Distancia Manhattan como heurística
        x1, y1 = nodo
        x2, y2 = self.objetivo
        return abs(x1 - x2) + abs(y1 - y2)

    def planificar_camino(self):
        frontera = [(0, self.inicio)]
        costo_acumulado = {self.inicio: 0}
        camino = {self.inicio: None}

        while frontera:
            _, nodo_actual = heapq.heappop(frontera)

            if nodo_actual == self.objetivo:
                return self.reconstruir_camino(camino)

            for vecino in self.mapa[nodo_actual]:
                nuevo_costo = costo_acumulado[nodo_actual] + 1  # Costo de moverse

                if vecino not in costo_acumulado or nuevo_costo < costo_acumulado[vecino]:
                    costo_acumulado[vecino] = nuevo_costo
                    prioridad = nuevo_costo + self.heuristica(vecino)
                    heapq.heappush(frontera, (prioridad, vecino))
                    camino[vecino] = nodo_actual

        return None  # No hay camino

    def reconstruir_camino(self, camino):
        nodo = self.objetivo
        ruta = []
        while nodo:
            ruta.append(nodo)
            nodo = camino[nodo]
        return ruta[::-1]  # Devuelve la ruta en orden correcto
```

```python
# Mapa como un grafo de conexiones
mapa = {
    (0,0): [(0,1), (1,0)],
    (0,1): [(0,0), (0,2)],
    (0,2): [(0,1), (1,2)],
    (1,0): [(0,0), (1,1)],
    (1,1): [(1,0), (1,2)],
    (1,2): [(0,2), (1,1)]
}

agente = AgenteDeliberativo(mapa, (0,0), (1,2))
ruta_optima = agente.planificar_camino()
print("Ruta planificada:", ruta_optima)
```

AGENTES HÍBRIDOS

¿Qué son los agentes híbridos?

Los agentes híbridos en Inteligencia Artificial (IA) son sistemas que combinan múltiples enfoques de IA para mejorar su rendimiento y capacidad de toma de decisiones. En lugar de basarse en un solo paradigma, como agentes basados en reglas o redes neuronales, los agentes híbridos integran diferentes técnicas para aprovechar lo mejor de cada una.

Características de los agentes híbridos:

1. **Combinación de enfoques**: Integran métodos como lógica simbólica, aprendizaje automático, planificación, algoritmos evolutivos y sistemas basados en reglas.
2. **Mejora en la toma de decisiones**: Permiten una mejor adaptación a entornos complejos y cambiantes.
3. **Escalabilidad**: Son más eficientes en problemas grandes que requieren múltiples estrategias de resolución.
4. **Interoperabilidad**: Pueden interactuar con diferentes tipos de sistemas de IA.

Ventajas:

Mayor adaptabilidad: Pueden actuar en entornos dinámicos, combinando respuestas rápidas (reactivas) con planificación estratégica (deliberativa).

Equilibrio entre rapidez y precisión: Pueden responder de inmediato a eventos inesperados mientras optimizan su desempeño a largo plazo.

Mejor gestión de la incertidumbre: Integran modelos probabilísticos o de aprendizaje automático para mejorar su toma de decisiones en situaciones desconocidas.

Mayor eficiencia computacional: Al dividir tareas entre distintos mecanismos, optimizan el uso de recursos y reducen la carga computacional en comparación con agentes puramente deliberativos.

Flexibilidad en implementación: Se pueden diseñar con arquitecturas modulares para combinar técnicas como redes neuronales, planificación basada en reglas y lógica simbólica.

Mejor rendimiento en problemas complejos: Son ideales para aplicaciones como robótica autónoma, asistentes inteligentes y sistemas de diagnóstico, donde se requieren múltiples estrategias de toma de decisiones.

Desventajas:

Mayor complejidad en el diseño e implementación: Integrar múltiples enfoques de IA requiere arquitecturas más sofisticadas, lo que hace que el desarrollo y la integración sean más complicados.

Mayor consumo de recursos computacionales: Al combinar modelos deliberativos y aprendizaje automático, pueden requerir mayor capacidad de procesamiento y memoria, afectando el rendimiento en tiempo real.

Dificultad en la optimización: Ajustar los distintos módulos para que trabajen de manera eficiente puede ser complicado, especialmente cuando hay conflictos entre los enfoques reactivos y deliberativos.

Mayor necesidad de mantenimiento y actualización: Si el agente utiliza modelos del mundo o aprendizaje automático, estos deben actualizarse periódicamente para mantener su precisión y relevancia.

Dependencia de un buen diseño arquitectónico: Un diseño deficiente puede generar conflictos entre los diferentes módulos, afectando la eficiencia y el rendimiento del sistema.

Ejemplo en código:

Este agente combina una estrategia **deliberativa** (planea ir hacia la meta) con una estrategia **reactiva** (evita obstáculos).

```python
class AgenteHibrido:
    def __init__(self):
        self.posicion = (0, 0)  # Posición inicial
        self.meta = (3, 3)  # Objetivo
        self.obstaculo = (2, 2)  # Obstáculo fijo

    def plan_deliberativo(self):
        """Decide el mejor movimiento hacia la meta."""
        x, y = self.posicion
        if x < self.meta[0]:
            return (x + 1, y)  # Moverse a la derecha
        elif y < self.meta[1]:
            return (x, y + 1)  # Moverse hacia arriba
        return self.posicion  # Si ya está en la meta, no moverse

    def reaccionar_a_obstaculo(self, nueva_posicion):
        """Si el nuevo movimiento choca con un obstáculo, cambia de ruta."""
        if nueva_posicion == self.obstaculo:
            print("⚠️ Obstáculo detectado, cambiando ruta...")
            return (self.posicion[0], self.posicion[1] + 1)  # Moverse hacia arriba en vez de adelante
        return nueva_posicion

    def mover(self):
        """Combina planificación y reacción para moverse."""
        nueva_posicion = self.plan_deliberativo()
        nueva_posicion = self.reaccionar_a_obstaculo(nueva_posicion)
        self.posicion = nueva_posicion
        print(f"📍 Nueva posición: {self.posicion}")

# Simulación del agente moviéndose hasta la meta
agente = AgenteHibrido()
while agente.posicion != agente.meta:
    agente.mover()
```

Funcionamiento:

Plan deliberativo (`plan_deliberativo`)

- El robot siempre intenta moverse hacia su meta siguiendo el camino más directo.
- Primero avanza en X (horizontalmente), y si no puede, avanza en Y (verticalmente).

🔹 Reacción a obstáculos (`reaccionar_a_obstaculo`)

- Si el robot intenta moverse y encuentra un obstáculo, cambia su movimiento para evitarlo.
- En este caso, si hay un obstáculo, intenta moverse hacia arriba en lugar de seguir avanzando.

🔹 Movimiento (`mover`)

- Combina planificación y reacción para mover al agente.
- Llama primero a la función deliberativa y luego a la función reactiva.
- Imprime la nueva posición del robot en la cuadrícula.

Ejemplo:

El robot comienza en la posición `(0,0)` y su meta es `(3,3)`.
Hay un obstáculo en `(2,2)`, así que debe evitarlo.

Ejecución del código:

1.- El robot planea moverse a `(1,0)`, lo hace sin problemas.
2.- Luego a `(2,0)`, también es un movimiento válido.
3.- Después intenta ir a `(2,2)`, ¡pero hay un obstáculo!
4.- Reacciona cambiando su movimiento a `(2,1)`, evitando el obstáculo.
5.-Continúa moviéndose hasta llegar a `(3,3)`, su meta.