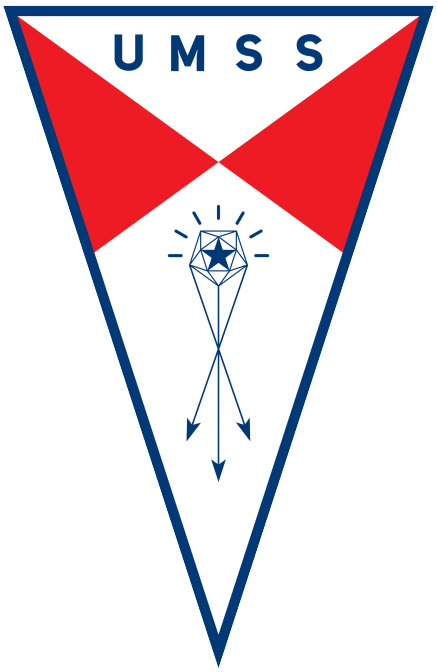


# Preparación de Sopa de Maní

Mateo Alejandro Merino Vidal
Valery Dariana Ortuño Panozo
Anahí Sanabria Ugarte
Leonel Zeballos Aldunate

17 de febrero de 2026

# 1. Introducción y Descripción del Entorno

La inteligencia artificial ha demostrado ser útil en tareas complejas de la vida cotidiana, como la preparación de alimentos en entornos automatizados. En este trabajo, se modela la preparación del tradicional plato boliviano **Sopa de Maní** como un problema de planificación, utilizando el formalismo STRIPS (Stanford Research Institute Problem Solver).

El objetivo es permitir que agentes inteligentes colaboren dentro de un entorno de cocina simulada para adquirir ingredientes, procesarlos y realizar la cocción adecuada, hasta llegar al estado final en el que la sopa esté servida.

Se emplean dos tipos de agentes con diferentes arquitecturas:

- El **Agente Comprador**, basado en objetivos, actúa de acuerdo con metas definidas y realiza acciones para satisfacer esas metas.
- El **Agente Cocinero**, basado en modelos, representa el conocimiento del entorno mediante estados y transiciones, tomando decisiones de acuerdo con la evolución del sistema.

El entorno en el que operan se modela como un *espacio de estados*, en el cual las acciones tienen precondiciones y efectos. Este modelo permite representar los cambios causados por cada acción de forma lógica y estructurada. Además, se utilizan herramientas de planificación automática para representar cómo los agentes logran sus objetivos a través de secuencias de acciones, utilizando un diagrama de estados implícito para guiar su comportamiento.

# 2. Descripción de los Agentes Inteligentes

### Agente Comprador

El **Agente Comprador** es un agente deliberativo basado en objetivos. Este tipo de agente compara continuamente el estado actual del entorno con un estado deseado. Si detecta una discrepancia (por ejemplo, falta de ingredientes), elabora un plan de acciones —en este caso, realizar compras— para reducir dicha diferencia. Su objetivo principal es garantizar la disponibilidad de todos los ingredientes necesarios para preparar la sopa. Funciona antes que el cocinero, asegurando que todo esté listo para la preparación.

| Acción | Descripción |
|---|---|
| Comprar(Maní) | Compra del ingrediente base |
| Comprar(Papas) | Compra de tubérculo |
| Comprar(Cebolla) | Compra de condimento |
| Comprar(Ajo) | Compra de condimento |
| Comprar(Aceite) | Compra de insumo para sofrito |
| Comprar(Sal) | Compra de condimento |
| Comprar(Perejil) | Compra para decoración |
| Comprar(Agua) | Compra o uso de agua disponible |
| Comprar(Carne) | Compra de proteína |
| Comprar(Olla) | Adquisición de herramienta de cocción |
| Comprar(Sartén) | Adquisición de sartén para sofreír |

Cuadro 1: Planificación del Agente Comprador

### Agente Cocinero

El **Agente Cocinero** es un agente basado en modelos. Su comportamiento está definido por la representación del estado interno del entorno (estado del sistema de cocina) y la forma en la que las acciones modifican ese estado. Este agente ejecuta operaciones de transformación física como pelar, picar, sofreír, mezclar y hervir, actuando solo cuando los ingredientes requeridos han sido adquiridos. Cada acción tiene condiciones previas que deben cumplirse y produce efectos que modifican el entorno.

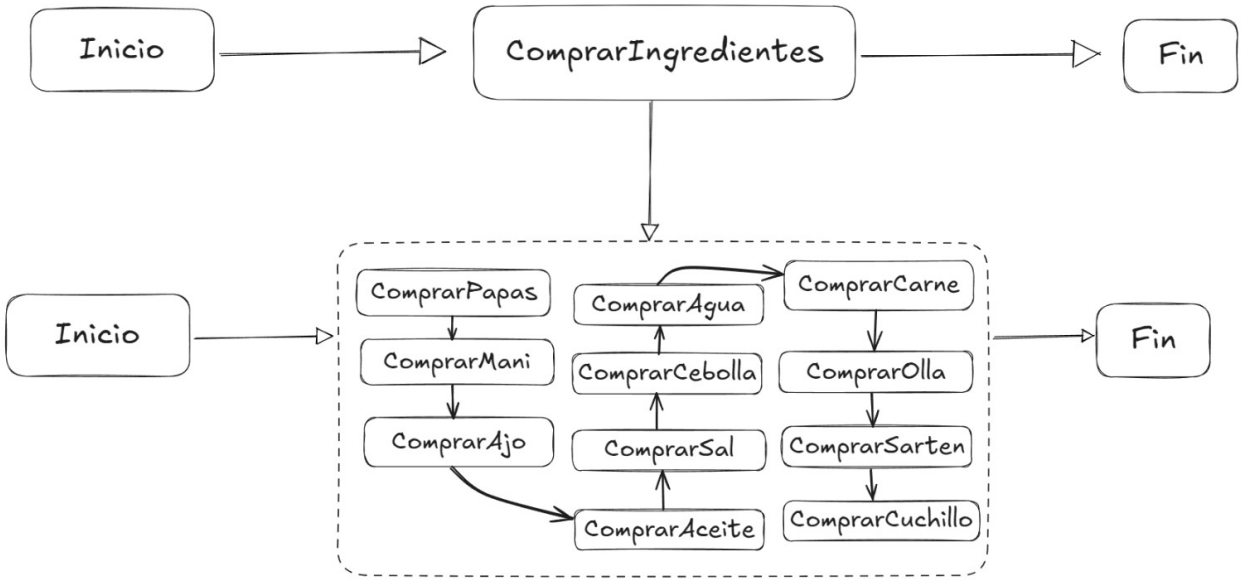


Figura 1: Diagrama del comprador

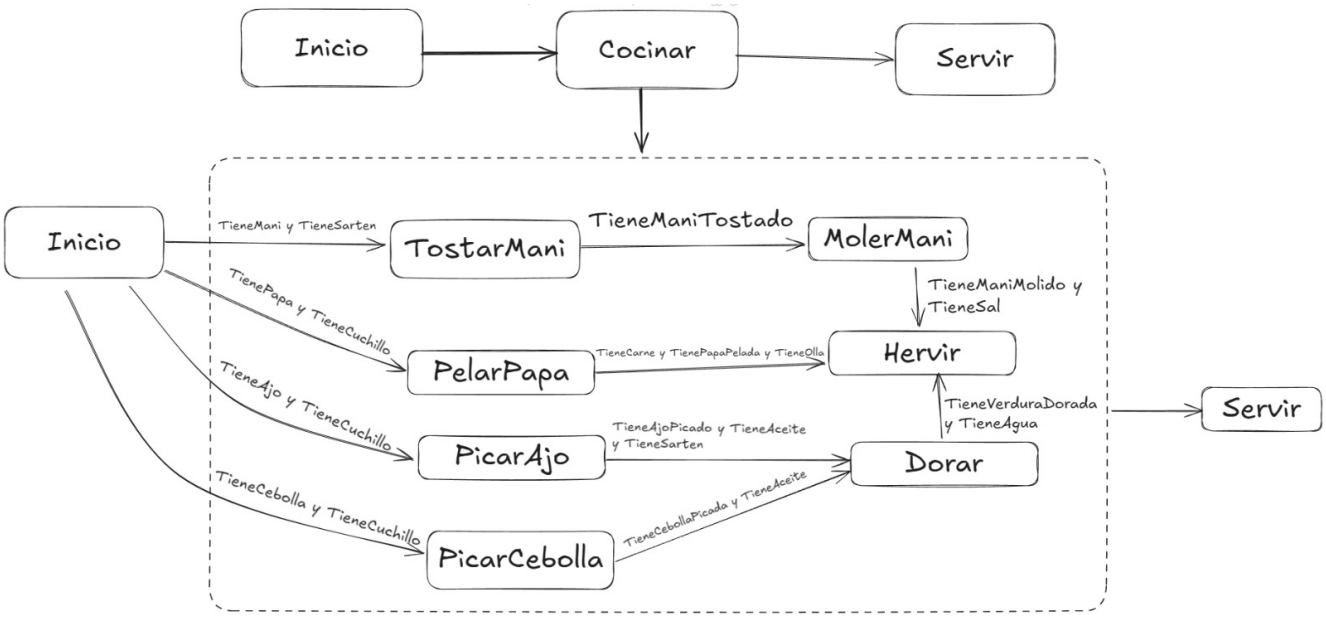


Figura 2: Diagrama del cocinero

## 3. Formulación en Predicados STRIPS

**Estado Inicial**

$$\neg Tiene(\text{Maní}) \land \neg Tiene(\text{Aceite}) \land \neg Tiene(\text{Papas}) \land \neg Tiene(\text{Cebolla}) \\ \land \neg Tiene(\text{Ajo}) \land \neg Tiene(\text{Agua}) \land \neg Tiene(\text{Carne}) \land \neg Tiene(\text{Sal}) \\ \land \neg Tiene(\text{Sartén}) \land \neg Tiene(\text{Olla}) \land \neg Tiene(\text{Cuchillo})$$

**Estado Objetivo**

$$\texttt{Servido(SopaDeManí)}$$

**Acciones del Agente Comprador (STRIPS)**

| | |
|---|---|
| Acción: | $Comprar(x)$ |
| PRECOND: | $Ingrediente(x) \land \neg Tiene(x)$ |
| EFECTO: | $Tiene(x)$ |

**Acciones del Agente Cocinero (STRIPS)**

Acción: $TostarMani$
PRECOND: $Tiene(Man) \land Tiene(Sartn)$
EFECTO: $TieneManiTostado$

Acción: $MolerMani$
PRECOND: $TieneManiTostado$
EFECTO: $TieneManiMolido$

Acción: $PelarPapa$
PRECOND: $Tiene(Papa) \land Tiene(Cuchillo)$
EFECTO: $TienePapaPelada$

Acción: $PicarAjo$
PRECOND: $Tiene(Ajo) \land Tiene(Cuchillo)$
EFECTO: $TieneAjoPicado$

Acción: $PicarCebolla$
PRECOND: $Tiene(Cebolla) \land Tiene(Cuchillo)$
EFECTO: $TieneCebollaPicada$

Acción: $Dorar$
PRECOND: $TieneAceite \land TieneAjoPicado \land TieneCebollaPicada \land Tiene(Sarten)$
EFECTO: $TieneVerduraDorada$

Acción: $Hervir$
PRECOND: $Tiene(Carne) \land Tiene(PapaPelada) \land Tiene(Olla) \land Tiene(ManiMolido) \land Tiene(Sal)$
$\land TieneVerduraDorada \land Tiene(Agua)$
EFECTO: $Cocido(Sopa)$

Acción: $Servir$
PRECOND: $Cocido(Sopa)$
EFECTO: $Servido(Sopa)$

## 4. Implementación

La solución fue implementada en el lenguaje de programación `Python`, modelando el problema como una planificación en el espacio de estados. Se definieron dos agentes: uno comprador con restricciones de presupuesto, y uno cocinero que ejecuta pasos tipo STRIPS para preparar la Sopa de Maní.

### Estructura General del Código

El sistema define los siguientes componentes:

- Clase `Estado`: representa el conjunto de hechos actuales.
- Clase `Operador`: modela acciones STRIPS con precondiciones, efectos positivos y negativos.
- `AgenteComprador`: decide qué ingredientes comprar según un presupuesto disponible.
- `AgenteCocinero`: ejecuta operaciones sobre los ingredientes para lograr la sopa servida.

### Implementación del Modelo en Python

```
#### Modelos base para el problema ####

#Clase que representa el estado del problema
class Estado:
    def __init__(self, hechos):
        self.hechos = set(hechos)  #Se almacenan como conjunto para facilitar
        ↪  operaciones de conjunto

    def cumple(self, condiciones):
        return condiciones.issubset(self.hechos)  #Verifica si un conjunto de
        ↪  condiciones esta contenido en el estado

    def aplicar(self, agrega, elimina):
        nuevos = (self.hechos - elimina).union(agrega)  #Elimina y agrega hechos
        return Estado(nuevos)  #Devuelve un nuevo estado actualizado

    def __contains__(self, item):
        return item in self.hechos


#Clase que modela una accion STRIPS
class Operador:
    def __init__(self, nombre, precond, agrega, elimina):
        self.nombre = nombre  #Nombre de la accion
        self.precond = set(precond)  #Precondiciones necesarias para ejecutar la
        ↪  accion
        self.agrega = set(agrega)  #Efectos positivos, hechos que se anaden
        self.elimina = set(elimina)  #Efectos negativos, hechos que se eliminan

    def es_aplicable(self, estado):
        return estado.cumple(self.precond)  #Verifica si el estado actual
        ↪  permite aplicar la accion

    def aplicar(self, estado):
```

```
        return estado.aplicar(self.agrega, self.elimina)  # Aplica los efectos
        ↪ de la accion sobre el estado


#### Agente comprador ####

class AgenteComprador:
    def __init__(self, inventario_inicial, ingredientes_necesarios, precios,
    ↪ presupuesto):
        self.estado = Estado(inventario_inicial)  #Ingredientes que ya se tienen
        self.meta = set(ingredientes_necesarios)  #Ingredientes que se necesitan
        ↪ para la sopa
        self.precios = precios  #Diccionario con precios de cada ingrediente
        self.presupuesto = presupuesto  #Cantidad de dinero disponible
        self.plan = []  #Plan de acciones del agente

    def planificar(self):
        faltantes = self.meta - self.estado.hechos  #Operacion de conjuntos para
        ↪ ver que ingredientes faltan
        total = sum(self.precios.get(ing, 0) for ing in faltantes)  #Suma del
        ↪ costo total de lo que falta
        if total > self.presupuesto:
            #Si no alcanza el presupuesto, no se puede comprar todo
            print(f"No hay suficiente presupuesto. Se necesitan Bs {total}, pero
            ↪ solo hay Bs {self.presupuesto}")
            return None

        #Comprar ingredientes faltantes
        for ingrediente in faltantes:
            costo = self.precios.get(ingrediente, 0)
            if self.presupuesto >= costo:
                self.estado.hechos.add(ingrediente)  #Anadir ingrediente al
                ↪ estado
                self.presupuesto -= costo  #Restar el costo del presupuesto
                self.plan.append(f"Comprar({ingrediente}, {costo})")  #Registrar
                ↪ accion en el plan
        return self.plan  #Devolver el plan de compras



#### Agente cocinero ####

class AgenteCocinero:
    def __init__(self, estado_inicial, objetivo, operadores):
        self.estado = Estado(estado_inicial)  #Estado inicial con todos los
        ↪ ingredientes comprados y disponibles
        self.objetivo = objetivo  #Meta a alcanzar, sopa servida
```

```
        self.operadores = operadores  #Lista de operadores
        self.plan = []  #Lista de pasos del plan

    def seleccionar_operador(self, meta):
        # Selecciona un operador que genere la meta como efecto
        for op in self.operadores:
            if meta in op.agrega:
                return op
        return None  #No se encontro operador para la meta

    def planificar(self):
      #Con pila por objetivos, comenzamos en el objetivo final
        pila = list(reversed(list(self.objetivo)))  #Pila de metas por cumplir
        while pila:
            tope = pila.pop()  #Obtener la siguiente meta de la pila
            if isinstance(tope, str):
                #Si la meta no se ha cumplido, buscar operador que la produzca
                if tope not in self.estado:
                    operador = self.seleccionar_operador(tope)
                    if operador:
                        pila.append(operador)  #Primero planificar la accion
                        pila.extend(operador.precond - self.estado.hechos)
                         ↪  #Agregar sus precondiciones
            elif isinstance(tope, Operador):
                if tope.es_aplicable(self.estado):
                    self.estado = tope.aplicar(self.estado)  #Aplicar la accion
                    self.plan.append(tope.nombre)  #Registrar accion en el plan
        return self.plan  #Devolver el plan completo


#### Ejecucion principal ####

if __name__ == "__main__":
    #Ingredientes necesarios para preparar la sopa
    ingredientes = {
        "Mani", "Papa", "Cebolla", "Ajo", "Aceite", "Agua",
        "Carne", "Sal", "Sarten", "Olla", "Cuchillo"
    }

    #Precios de cada ingrediente
    precios = {
        "Mani": 15, "Papa": 30, "Cebolla": 21, "Ajo": 4,
        "Aceite": 20, "Agua": 10, "Carne": 50, "Sal": 13,
        "Sarten": 129, "Olla": 250, "Cuchillo": 61
    }

    presupuesto_disponible = 300  #Presupuesto inicial para el agente comprador
```

```
#Ingredientes que ya se tienen (no necesitan ser comprados)
inventario_inicial = {"Papa", "Aceite", "Cuchillo", "Mani", "Sal", "Olla",
↪  "Agua"}

#Crear agente comprador
comprador = AgenteComprador(inventario_inicial, ingredientes, precios,
↪  presupuesto_disponible)
plan_compras = comprador.planificar()  # Generar plan de compra

if plan_compras is None:
    #Si no alcanza el presupuesto, detener el proceso
    print("\nNo se puede continuar con la planificación de la sopa.")
else:
    #Lista de operadores STRIPS que representan acciones de cocina
    operadores = [
        Operador("Tostar(Mani)", {"Mani", "Sarten"}, {"TieneManiTostado"},
        ↪  {"Mani"}),
        Operador("Moler(Mani)", {"TieneManiTostado"}, {"TieneManiMolido"},
        ↪  {"TieneManiTostado"}),
        Operador("Pelar(Papa)", {"Papa", "Cuchillo"}, {"TienePapaPelada"},
        ↪  {"Papa"}),
        Operador("Picar(Ajo)", {"Ajo", "Cuchillo"}, {"TieneAjoPicado"},
        ↪  {"Ajo"}),
        Operador("Picar(Cebolla)", {"Cebolla", "Cuchillo"},
        ↪  {"TieneCebollaPicada"}, {"Cebolla"}),
        Operador("Dorar(Ajo)", {"TieneAjoPicado", "Aceite", "Sarten"},
        ↪  {"TieneVerduraDorada"}, {"TieneAjoPicado"}),
        Operador("Dorar(Cebolla)", {"TieneCebollaPicada", "Aceite",
        ↪  "Sarten"}, {"TieneVerduraDorada"}, {"TieneCebollaPicada"}),
        Operador("Hervir(Ingredientes)", {"Carne", "TienePapaPelada",
        ↪  "Olla", "TieneManiMolido", "Sal", "TieneVerduraDorada", "Agua"},
                 {"Cocido(Sopa)"}, {"Carne", "TienePapaPelada",
                 ↪  "TieneManiMolido", "TieneVerduraDorada", "Agua"}),
        Operador("Servir(Sopa)", {"Cocido(Sopa)"}, {"Servido(Sopa)"},
        ↪  {"Cocido(Sopa)"})
    ]

    #Estado resultante luego de las compras
    estado_despues_de_compras = comprador.estado.hechos

    #Crear agente cocinero y planificar la preparación
    cocinero = AgenteCocinero(estado_despues_de_compras, {"Servido(Sopa)"},
    ↪  operadores)
    plan_cocina = cocinero.planificar()

    #Mostrar plan de compras
```

```
        print("=== Plan del Agente Comprador ===\n")
        cont = 1
        for paso in plan_compras:
            print(f"{cont}. ", paso)
            cont+=1

        #Mostrar presupuesto restante
        print(f"Presupuesto restante: Bs {comprador.presupuesto}\n")

        #Mostrar plan de cocina
        cont = 1
        print("=== Plan del Agente Cocinero ===")
        for paso in plan_cocina:
            print(f"{cont}. ", paso)
            cont+=1
```

### Ejecución y Coordinación de Agentes

En la ejecución principal, se define:

- Los ingredientes requeridos y sus precios.
- El presupuesto disponible.
- Los ingredientes ya disponibles en el hogar.

```
# === Main para poner en ejecucion a los agentes ===

if __name__ == "__main__":
    # Ingredientes necesarios para preparar la sopa
    ingredientes = {
        "Mani", "Papa", "Cebolla", "Ajo", "Aceite", "Agua",
        "Carne", "Sal", "Sarten", "Olla", "Cuchillo"
    }

    # Precios de cada ingrediente
    precios = {
        "Mani": 15, "Papa": 30, "Cebolla": 21, "Ajo": 4,
        "Aceite": 20, "Agua": 10, "Carne": 50, "Sal": 13,
        "Sarten": 129, "Olla": 250, "Cuchillo": 61
    }

    presupuesto_disponible = 300  # Presupuesto total para el agente comprador

    # Ingredientes que ya se tienen (no necesitan ser comprados)
    inventario_inicial = {"Papa", "Aceite", "Cuchillo", "Mani", "Sal", "Olla",
    ↪  "Agua"}
```

```
# Crear agente comprador
comprador = AgenteComprador(inventario_inicial, ingredientes, precios,
↪  presupuesto_disponible)
plan_compras = comprador.planificar()  # Generar plan de compra

if plan_compras is None:
    # Si no alcanza el presupuesto, detener el proceso
    print("\nNo se puede continuar con la planificación de la sopa.")
else:
    # Lista de operadores STRIPS que representan acciones de cocina
    operadores = [
        Operador("Tostar(Mani)", {"Mani", "Sarten"}, {"TieneManiTostado"},
        ↪  {"Mani"}),
        Operador("Moler(Mani)", {"TieneManiTostado"}, {"TieneManiMolido"},
        ↪  {"TieneManiTostado"}),
        Operador("Pelar(Papa)", {"Papa", "Cuchillo"}, {"TienePapaPelada"},
        ↪  {"Papa"}),
        Operador("Picar(Ajo)", {"Ajo", "Cuchillo"}, {"TieneAjoPicado"},
        ↪  {"Ajo"}),
        Operador("Picar(Cebolla)", {"Cebolla", "Cuchillo"},
        ↪  {"TieneCebollaPicada"}, {"Cebolla"}),
        Operador("Dorar(Ajo)", {"TieneAjoPicado", "Aceite", "Sarten"},
        ↪  {"TieneVerduraDorada"}, {"TieneAjoPicado"}),
        Operador("Dorar(Cebolla)", {"TieneCebollaPicada", "Aceite",
        ↪  "Sarten"}, {"TieneVerduraDorada"}, {"TieneCebollaPicada"}),
        Operador("Hervir(Ingredientes)", {"Carne", "TienePapaPelada",
        ↪  "Olla", "TieneManiMolido", "Sal", "TieneVerduraDorada", "Agua"},
                 {"Cocido(Sopa)"}, {"Carne", "TienePapaPelada",
                 ↪  "TieneManiMolido", "TieneVerduraDorada", "Agua"}),
        Operador("Servir(Sopa)", {"Cocido(Sopa)"}, {"Servido(Sopa)"},
        ↪  {"Cocido(Sopa)"})
    ]

    # Estado resultante luego de las compras
    estado_despues_de_compras = comprador.estado.hechos

    # Crear agente cocinero y planificar la preparación
    cocinero = AgenteCocinero(estado_despues_de_compras, {"Servido(Sopa)"},
    ↪  operadores)
    plan_cocina = cocinero.planificar()

    # Mostrar plan de compras
    print("=== Plan del Agente Comprador ===")
    cont = 1
    for paso in plan_compras:
        print(f"{cont}. ", paso)
        cont+=1
```

```python
# Mostrar presupuesto restante
print(f"\nPresupuesto restante: Bs {comprador.presupuesto}")

# Mostrar plan de cocina
cont = 1
print("\n=== Plan del Agente Cocinero ===")
for paso in plan_cocina:
    print(f"{cont}. ", paso)
    cont+=1
```

## 5. Pruebas y Resultados

Con el fin de validar el correcto funcionamiento del sistema multiagente diseñado, se realizaron pruebas sobre escenarios realistas en los que el agente comprador parte de un inventario parcial y un presupuesto limitado, mientras que el agente cocinero debe alcanzar un estado meta de sopa servida aplicando operadores STRIPS de forma secuencial.

### Agente Comprador

El agente comprador fue configurado como un agente deliberativo basado en objetivos. Su tarea consistió en adquirir todos los ingredientes y herramientas necesarias para preparar la sopa de maní, partiendo de un inventario inicial preexistente. En esta versión, se le asignó un presupuesto y una lista completa de ingredientes, pero sólo compró aquellos que no estaban ya disponibles en su inventario.

Durante la ejecución, el agente:

- Evaluó los ingredientes faltantes comparando el inventario actual con la meta.
- Calculó el costo total requerido para adquirir los ingredientes restantes.
- Verificó si el presupuesto disponible era suficiente.
- Generó un plan de compra secuencial, considerando el presupuesto restante después de cada adquisición.

Ejemplo de plan generado:

```
=== PLANIFICACIÓN DEL AGENTE COMPRADOR ===
- Comprar(Sarten) por Bs 129
- Comprar(Caracter) por Bs 50
- Comprar(Cebolla) por Bs 21
- Comprar(Ajo) por Bs 4
```

Este plan demuestra que el agente fue capaz de adquirir únicamente los recursos necesarios y dentro del presupuesto. Los ingredientes ya disponibles en el inventario inicial fueron correctamente ignorados en el cálculo de costos y no fueron incluidos en el plan de compra.

### Agente Cocinero

El agente cocinero, por su parte, fue diseñado como un agente deliberativo basado en modelos. Su función consistió en transformar el conjunto de ingredientes adquiridos en una secuencia lógica de acciones que conduzca al estado meta `Servido(Sopa)`.

Utilizó operadores definidos según el formalismo STRIPS, aplicando dinámicamente las acciones sólo cuando sus precondiciones eran satisfechas. El agente empleó una pila de metas, planificando desde el objetivo final hacia atrás, e incorporando las precondiciones requeridas en el proceso.

Plan de ejemplo generado:

```
=== PLANIFICACIÓN DEL AGENTE COCINERO ===
- TostarMani
- MolerMani
- PelarPapa
- PicarAjo
- PicarCebolla
- DorarCebolla
- Hervir
- Servir
```

Este plan refleja una secuencia de operaciones culinarias coherente y completa, respetando la causalidad entre acciones. Por ejemplo, no se intenta hervir la sopa hasta haber picado, sofrito y molido los ingredientes necesarios.

### Validación del Funcionamiento

Ambos agentes lograron sus objetivos respetando las restricciones impuestas:

- El agente comprador actuó racionalmente frente al presupuesto limitado, comprando sólo lo esencial.
- El cocinero aplicó correctamente las transformaciones necesarias mediante planificación simbólica STRIPS.
- Las herramientas (como la olla o el sartén) no fueron eliminadas del estado tras su uso, lo cual permitió su reutilización.
- Se validó que los ingredientes fueron consumidos correctamente al ser transformados por las acciones del agente cocinero.

En conjunto, el sistema multiagente mostró un comportamiento racional, robusto y determinista, logrando planificar de manera eficiente la elaboración completa de la sopa de maní.

### Conclusión general

Los resultados obtenidos permiten concluir que la implementación de un sistema multiagente con planificación deliberativa es una estrategia eficaz para resolver tareas complejas que requieren secuencias ordenadas de acciones, coordinación entre roles y toma de decisiones racionales.

En particular, la preparación automatizada de la **Sopa de Maní** evidenció cómo la cooperación entre dos agentes con funciones diferenciadas —uno encargado de adquirir recursos (comprador) y otro de transformarlos (cocinero)— puede llevar a cabo con éxito un proceso de múltiples etapas. Cada agente utilizó técnicas de búsqueda dirigidas por metas, aplicando modelos simbólicos con condiciones y efectos claros (tipo STRIPS).

Este enfoque no solo garantiza la coherencia lógica en la planificación, sino que también puede escalarse y adaptarse a otros contextos donde se requiera:

- División de tareas entre múltiples entidades autónomas.
- Gestión de recursos limitados (como presupuestos).
- Ejecución secuencial de acciones con dependencias causales.

En resumen, se demuestra que la inteligencia artificial simbólica —mediante planificación basada en estados, operadores y metas— es capaz de modelar y resolver problemas reales de forma efectiva, estructurada y explicable.