UA

Portfolio ASM

Daniel Asensi Roch
DNI: 48776120C

30 de noviembre de 2023

Índice general

Capítulo 1

Mundo de Wumpus

1.1. Introducción

En el ámbito de la inteligencia artificial, la resolución de problemas complejos y dinámicos representa un desafío constante que impulsa el desarrollo de nuevas metodologías y enfoques. Una de estas metodologías es el uso de agentes BDI (Belief-Desire-Intention) en plataformas de simulación, como la plataforma GAMA. Esta práctica se centra en la implementación y evaluación de un agente BDI para resolver el desafío presentado por el mundo de Wumpus.

1.1.1. Planteamiento del problema

El mundo de Wumpus es un entorno clásico en la inteligencia artificial, que presenta un escenario de cueva cuadriculada donde un agente debe recolectar tesoros mientras evita trampas y enfrenta al Wumpus, una criatura peligrosa. Este entorno es ideal para explorar el comportamiento de los agentes inteligentes y su capacidad para tomar decisiones basadas en la percepción y el razonamiento.

Los agentes BDI son un enfoque de modelado y diseño de agentes inteligentes que se basa en tres componentes fundamentales: creencias, deseos e intenciones. Estos componentes permiten al agente formar una representación interna del mundo, establecer metas y tomar decisiones racionales para alcanzar sus objetivos. En el contexto del mundo de Wumpus, el agente BDI utiliza sus creencias para interpretar percepciones como brisas o resplandores, indicativos de trampas o tesoros cercanos, respectivamente. Los deseos del agente pueden incluir recolectar tesoros y evitar peligros, mientras que sus intenciones guiarán sus acciones concretas en el entorno.

1.1.2. Objetivos

Los objetivos de esta práctica son múltiples y se detallan a continuación:

- Familiarización con Agentes BDI: Adquirir un conocimiento sólido sobre el concepto de agentes BDI (Belief-Desire-Intention), comprendiendo cómo estas entidades teóricas pueden ser utilizadas para modelar y resolver problemas complejos en entornos dinámicos.
- Resolución del Mundo de Wumpus: Utilizar el entorno del mundo de Wumpus como un caso de estudio para explorar y demostrar la eficacia de los agentes BDI. Este objetivo incluye el desarrollo de estrategias para que el agente recolecte tesoros y evite peligros como trampas y el Wumpus.
- Toma de Decisiones Basada en Creencias, Deseos y Metas: Profundizar en el entendimiento de cómo un agente BDI toma decisiones racionales basadas en sus creencias (información sobre el entorno), deseos (objetivos a alcanzar) y metas (planes de acción concretos).
- Evaluación de Rendimiento y Eficacia: Evaluar críticamente el rendimiento y la eficacia del agente BDI implementado, analizando cómo se adapta y responde a diferentes configuraciones y desafíos presentados en el mundo de Wumpus.

1.2. Estado del Arte

La combinación de agentes BDI y el mundo de Wumpus representa un área de estudio fascinante en la inteligencia artificial. Los agentes BDI, con su estructura de creencias, deseos e intenciones, son particularmente adecuados para navegar y tomar decisiones en el entorno desafiante que presenta el mundo de Wumpus. Esta intersección ha sido explorada para estudiar cómo los agentes pueden formar representaciones internas del mundo, planificar bajo incertidumbre y adaptarse a entornos dinámicos.

1.2.1. Agentes BDI

- **Orígenes y Desarrollo Teórico:** Los agentes BDI, basados en los conceptos de creencias (Beliefs), deseos (Desires) e intenciones (Intentions), tienen sus raíces en la teoría filosófica de la acción humana, especialmente en los trabajos de Bratman sobre la teoría de la acción intencional. Su desarrollo en la inteligencia artificial comenzó en la década de 1980, con investigadores como Michael Wooldridge y Nicholas Jennings contribuyendo significativamente a su formalización y aplicación práctica. [1]

- **Principios Clave:** Los agentes BDI se caracterizan por su capacidad para representar y razonar sobre el mundo (creencias), formular objetivos (deseos) y elaborar planes para alcanzar esos objetivos (intenciones). Esta estructura permite un modelado más cercano a la toma de decisiones humana, lo que los hace adecuados para entornos complejos y cambiantes. [2]

- **Aplicaciones y Avances Recientes:** Los agentes BDI han encontrado aplicaciones en diversas áreas, desde videojuegos y simulaciones hasta sistemas de soporte a la decisión en negocios y logística. Los avances recientes incluyen la integración de técnicas de aprendizaje automático para mejorar la adaptabilidad y eficiencia de estos agentes. [2]

1.2.2. El Mundo de Wumpus

Como se expone en el siguiente artículo sobre el Mundo de Wumpus [3]

- **Origen y Evolución:** El mundo de Wumpus fue introducido por Gregory Yob en 1975 como un juego de computadora simple. Se ha convertido en un entorno de prueba estándar en la inteligencia artificial para demostrar y evaluar la capacidad de razonamiento y toma de decisiones de los agentes inteligentes.

- **Características del Entorno:** El mundo de Wumpus es un laberinto cuadriculado donde un agente debe navegar, recolectar tesoros y evitar peligros como trampas y el Wumpus. Este entorno presenta desafíos únicos como la necesidad de inferir la ubicación de peligros basándose en percepciones indirectas (como el olor o una brisa).

- **Uso en Investigación y Educación:** Ha sido ampliamente utilizado en la educación en inteligencia artificial para enseñar conceptos como la lógica, la planificación y el

razonamiento bajo incertidumbre. Además, ha servido como un campo de pruebas para nuevas teorías y modelos en la IA.

1.3. Metodología

La metodología empleada en esta práctica se centra en el uso de un agente BDI para navegar y resolver desafíos en el mundo de Wumpus. El agente está programado para reaccionar a diferentes estímulos del entorno, como la presencia de oro, brisas, olores y trampas, y tomar decisiones basadas en sus creencias, deseos e intenciones.

1.3.1. Análisis de Estrategias

El agente utiliza varias estrategias para navegar por el entorno y alcanzar sus objetivos. Estas estrategias se basan en la percepción del entorno y en la toma de decisiones inteligente para evitar peligros y alcanzar el objetivo de recolectar oro.

Estrategia para Moverse

La estrategia de movimiento del agente en el mundo de Wumpus es crucial para explorar eficientemente el entorno y alcanzar los objetivos de evitar peligros y recolectar oro. Esta estrategia se basa en una serie de percepciones y decisiones que guían al agente a través del laberinto.

1. **Movimiento Aleatorio Inicial:** Al inicio, el agente añade un deseo de movimiento aleatorio ***(move_random_desire)***. Esto le permite comenzar a explorar el entorno de manera no dirigida, lo cual es esencial en las etapas iniciales cuando no se ha detectado oro ni peligros.

2. **Percepción y Reacción a Estímulos:** El agente está programado para percibir diferentes estímulos del entorno, como áreas de brillo ***(glitterArea)***, brisas ***(breezeArea)*** y olores ***(odorArea)***. Cada uno de estos estímulos afecta su estrategia de movimiento:

   - Al detectar brillo, el agente se enfoca en esa ubicación y cambia su estrategia de movimiento para acercarse al oro.
   - Al percibir brisas u olores, el agente activa un deseo de escape para evitar peligros, lo que altera temporalmente su patrón de movimiento.

3. **Estrategia de Patrullaje:** Cuando el agente no está en modo de escape o búsqueda de oro, sigue un plan de patrullaje ***(patrolling)*** que implica moverse a una de las celdas vecinas de manera aleatoria. Este movimiento se realiza a través de la selección aleatoria de un vecino y el desplazamiento hacia esa celda.

4. **Cambio de Estrategia según el Contexto:** El agente es capaz de cambiar su estrategia de movimiento en función de las percepciones y los deseos activos. Por ejemplo, al detectar un peligro, cambia a un modo de escape, y al detectar oro, cambia a una estrategia de búsqueda enfocada.

Estrategia para evitar muertes

1. **Detección de Peligros:** El agente está programado para detectar áreas de brisa y olor, indicativas de trampas y la presencia del Wumpus, respectivamente. Al percibir estos estímulos, el agente activa un deseo de escape ***(want_escape)*** para evitar estos peligros.

2. **Retroceso Inteligente:** En respuesta a la detección de peligros, el agente ejecuta un plan de escape ***(escape)*** que lo hace retroceder a la celda anterior, evitando así entrar en zonas peligrosas.

3. **Contabilización de Trampas Evitadas:** El agente lleva un registro de las trampas evitadas ***(trampasEvitadas)***, lo que permite evaluar la eficacia de la estrategia de evasión.

Estrategias para conseguir el oro

1. **Detección de Oro:** El agente percibe áreas con brillo ***(glitterArea)***, que indican la proximidad del oro. Esta percepción activa un deseo específico ***(getGold)*** para buscar el oro.

2. **Navegación Hacia el Oro:** Una vez activado el deseo de obtener oro, el agente ejecuta un plan ***(exploreAroundGlitter)*** para explorar las celdas cercanas a la ubicación detectada del brillo, aumentando así las posibilidades de encontrar el oro.

3. **Finalización de la Búsqueda:** Al encontrar el oro ***(goldArea)***, el agente elimina todas las intenciones y deseos actuales y pausa la simulación, indicando el éxito en la consecución del objetivo.

1.4. Resultados

La implementación del agente BDI en el entorno del mundo de Wumpus ha demostrado ser efectiva en lograr los objetivos principales: recolectar oro y evitar la muerte por peligros como trampas o el Wumpus. Sin embargo, el análisis de la estrategia y el comportamiento del agente revela que, aunque es exitoso, no opera de manera óptima. A continuación, se detallan los resultados clave y las áreas de mejora identificadas.

1.4.1. Partes positivas de la implementación

- **Éxito en la Evitación de Peligros:** El agente ha demostrado una capacidad notable para evitar peligros, como se evidencia en el registro de trampas evitadas (trampasEvitadas). Esto indica que la estrategia de percepción y reacción ante estímulos peligrosos es efectiva. **La media de los steps realizados para encontrar el oro en 20 simulaciones han sido 1300.**
- **Recolección de Oro:** El agente ha sido capaz de localizar y recolectar el oro en el entorno del mundo de Wumpus. La estrategia de moverse hacia áreas de brillo y explorar alrededor de estas zonas ha resultado ser una táctica exitosa. Como podemos ver en la figura 1.2.

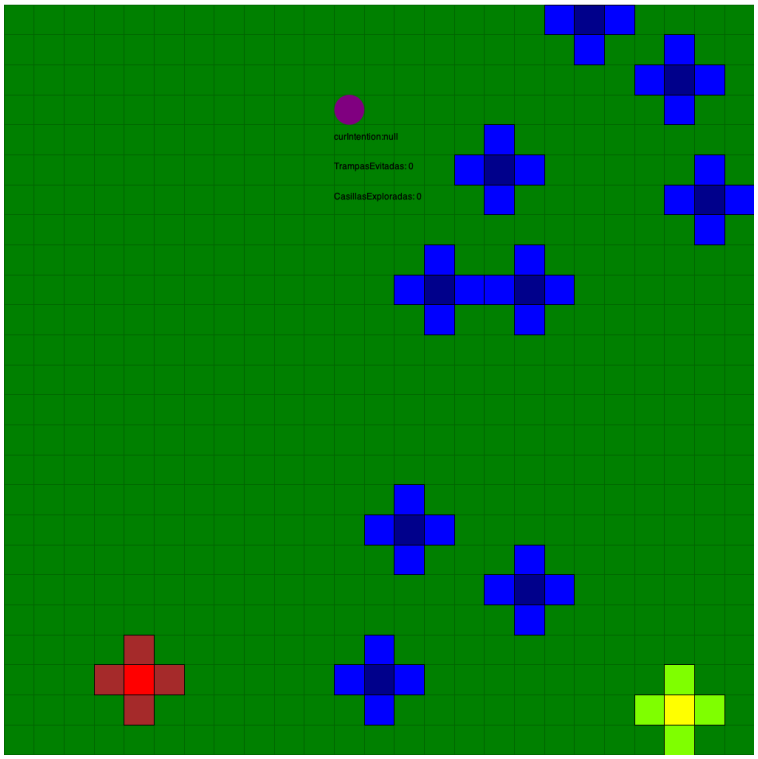

Figura 1.1: Inicio del mapa

Una vez ejecutada la simulación obtenemos que el agente ha llegado al oro.

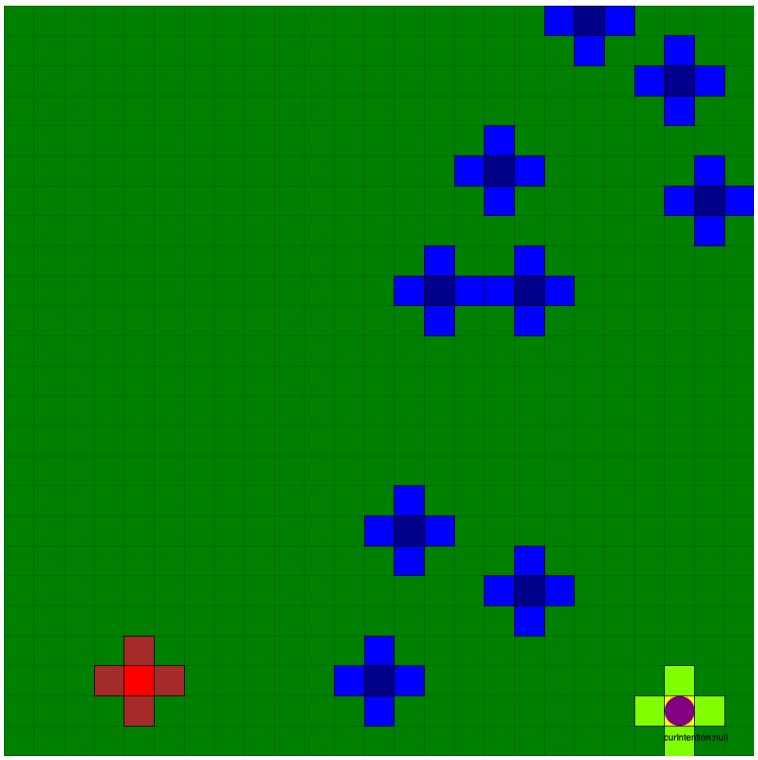

Figura 1.2: Mapa solucionado

Por otro lado, también hemos obtenido la cantidad de movimientos realizados por el agente, los cuales aparecen marcados en rojo y en azul la cantidad de trampas evitadas 1.3

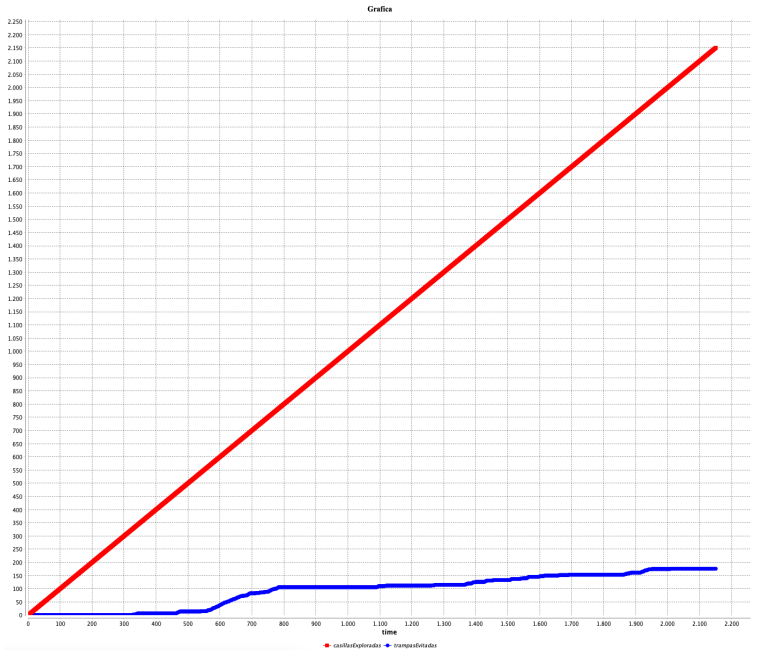

Figura 1.3: Gráfica de ciclos, movimientos y trampas evitadas

Además, para mayor control del agente se le ha añadido a su aspecto un texto que nos da información de sus intenciones, como se puede ver en la siguiente figura 2.4.

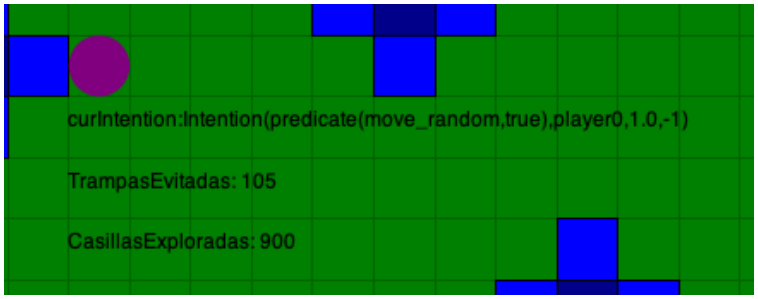

Figura 1.4: Información en aspecto del agente

1.4.2. Partes negativas de la implementación

- **Falta de Optimización en el Movimiento:** A pesar de lograr sus objetivos, el agente no muestra un patrón de movimiento optimizado. La estrategia de movimiento aleatorio, aunque efectiva para la exploración inicial, puede llevar a una exploración redundante o ineficiente de las celdas.
- **Ineficiencia en la Búsqueda de Oro:** Mientras que el agente puede encontrar y recolectar el oro, la estrategia no está optimizada para hacerlo de la manera más rápida o eficiente. El agente a menudo explora celdas innecesarias o toma rutas indirectas hacia el oro.

1.4.3. Potenciales mejoras

- **Potencial para Mejoras:** Los resultados indican que hay un margen significativo para mejorar la eficiencia del agente. Esto podría incluir la implementación de algoritmos de búsqueda más sofisticados, como A* o búsqueda en profundidad, para optimizar la ruta hacia el oro y minimizar la exploración innecesaria. [4]
- **Balance entre Seguridad y Eficiencia:** Un aspecto crucial en la mejora del agente será encontrar un equilibrio óptimo entre evitar peligros y buscar rutas eficientes. Esto requerirá un ajuste más fino de las percepciones y las decisiones del agente.

1.5. Código Fuente

Al tener la plantilla generada por el profesor, todos los alumnos, adjuntaré el código implementado por mí, en la especie ***Player***

1.5.1. Especie jugador inicialización

```
/**Agente del jugador */
species player skills: [moving] control: simple_bdi {


  // Posicion de inicio del jugador
  gworld maCellule <- one_of(gworld);
  // Posicion anterior del jugador
  gworld pastCellule <- maCellule;


  init {
    location<-maCellule.location;
    do add_desire(move_random_desire);

  }

```

1.5.2. Predicados

```
    predicate move_random_desire <- new_predicate("move_random");
    predicate want_escape <- new_predicate("want_escape", true);
    predicate getGold <- new_predicate("getGold", true);

    int  trampasEvitadas <- 0;
    int casillasExplorada <- 0;

```

1.5.3. Sensores de percepción

```
    /**Detecta una zona de brillo y para al agente encima */
  perceive target:glitterArea in: 1{

    focus id:"glitterLocation" var:location strength:10.0;

    ask myself{
      do remove_intention(want_escape, true);
      do remove_intention(move_random_desire, true);
    }

  }

```

```
13
14  //Detecta el tesoro
15  perceive target:goldArea in: 1{
16
17    ask myself{
18      do remove_belief( get_predicate(get_belief_with_name("glitterLocation"))
     );
19      do remove_intention(getGold,true);
20      do remove_intention(want_escape, true);
21      do remove_intention(move_random_desire, true);
22      ask world{
23        do pause;
24      }
25
26    }
27
28  }
29
30  /**Detecta una zona de brisa y hace retroceder al agente una casillas */
31  perceive target:breezeArea in: 1{
32
33    ask myself{
34      do remove_intention(move_random_desire, true);
35      do add_desire(want_escape);
36    }
37  }
38
39  /**Detecta una zona de edor y hace retroceder al agente una casillas */
40  perceive target:odorArea in: 1{
41
42    ask myself{
43      do remove_intention(move_random_desire, true);
44      do add_desire(want_escape);
45    }
46  }
```

1.5.4. Planes

```
1
2  //Creencia de busqueda de oro
3  rule belief: new_predicate(glitterLocation) new_desire: getGold;
4
5
6  //Exploracion celulas cercanas al oro
7  plan exploreAroundGlitter intention: getGold {
8
9    map glitter <- get_predicate(get_belief_with_name("glitterLocation")).
    values("location_value");
10   gworld celdaGlitter <- gworld(glitter['location_value']);
11   gworld celdaActual <- gworld({location.x, location.y});
12
```

```
13
14    if (celdaGlitter = celdaActual){
15      list<gworld> vecinos <- [];
16      ask celdaGlitter{
17        vecinos <- neighbors;
18      }
19
20      gworld vecino <- one_of(vecinos);
21      casillasExplorada <- casillasExplorada + 1;
22      do goto target: vecino on: gworld speed: 24.0;
23    }
24    else{
25      casillasExplorada <- casillasExplorada + 1;
26      do goto target: celdaGlitter on: gworld speed: 24.0;
27    }
28
29
30  }
31
32
33
34  /**Hace escapar al agente retrocediendo una casilla */
35  plan escape intention:want_escape {
36    do goto target: pastCellule on: gworld speed: 24.0;
37    trampasEvitadas <- trampasEvitadas + 1;
38    casillasExplorada <- casillasExplorada + 1;
39    do remove_intention(want_escape, false);
40    do add_desire(move_random_desire);
41  }
42
43
44
45  /**Plan de patrulla para moverse por el mapa libremente */
46  plan patrolling intention:move_random_desire {
47
48    gworld playerPosition <-gworld({location.x, location.y});
49    pastCellule <- playerPosition;
50
51        list<gworld> vecinos <- [];
52        ask playerPosition {
53            vecinos <- neighbors;
54        }
55
56        /**Me guardo mi posicion anterior */
57        gworld vecino <- one_of(vecinos);
58        casillasExplorada <- casillasExplorada + 1;
59        do goto target: vecino on: gworld speed: 24.0;
60  }
```

Capítulo 2

Control de un modelo

2.1. Introducción

En el ámbito de la inteligencia artificial y los sistemas multiagente, la modelización y simulación se presentan como herramientas fundamentales para la comprensión y el análisis de comportamientos complejos en entornos controlados. Este trabajo se centra en el desarrollo y la optimización de un modelo basado en agentes utilizando GAMA, un entorno de modelado y simulación para sistemas multiagente.

El modelo desarrollado, denominado robots", explora las dinámicas de interacción entre agentes autónomos en un entorno simulado. Estos agentes, representados por entidades móviles, interactúan dentro de un espacio delimitado, donde se aplican reglas específicas para su movimiento y comportamiento.

2.1.1. Objetivos

Los objetivos de este proyecto se pueden desglosar de la siguiente manera:

- **Desarrollo de un Modelo de Sistemas Multiagente:** Implementar un modelo en GAMA que simule el comportamiento de agentes autónomos. Este modelo debe ser capaz de representar interacciones complejas y proporcionar un marco para observar las dinámicas emergentes del sistema.
- **Optimización de Parámetros:** Ajustar y optimizar los parámetros clave del modelo, como la distancia de seguridad, el cambio de dirección y la variación aleatoria, para mejorar la eficiencia y efectividad de las interacciones de los agentes.
- **Análisis de Comportamientos Emergentes:** Observar y analizar los patrones de comportamiento emergentes en la simulación, con un enfoque especial en cómo las pequeñas variaciones en los parámetros pueden influir en el comportamiento general del sistema.
- **Evaluación de Estrategias de Evasión de Colisiones:** Examinar la eficacia de las estrategias implementadas para la evasión de colisiones entre agentes, evaluando su impacto en la supervivencia y eficiencia de los mismos dentro del entorno simulado.
- **Aplicación de Técnicas de Optimización:** Utilizar técnicas de optimización, como el algoritmo de optimización por enjambre de partículas (PSO), para encontrar la configuración óptima de parámetros que minimice los eventos no deseados (como colisiones) y maximice la efectividad del modelo.

2.2. Modelo Neuronal y Aprendizaje por Refuerzo

En esta sección, se propone un modelo neuronal teórico compatible con los estados s y las acciones a para su uso en el aprendizaje por refuerzo. Además, se diseña una función de recompensa $r(s_t, a_t, s_{t+1})$ y se discuten sus implicaciones, incluyendo la emergencia de conductas macroscópicas y la robustez del modelo.

2.2.1. Modelo Neuronal Propuesto

Para integrar un modelo de aprendizaje por refuerzo en nuestro sistema de agentes autónomos, se propone el uso de una red neuronal artificial (RNA) como aproximador de la función de valor. Esta RNA toma como entrada el estado actual del agente s_t y produce un vector de valores para cada acción posible en ese estado, lo que permite al agente decidir la acción a_t a realizar.

La estructura de la red consistiría en varias capas ocultas con activaciones no lineales, como la función ReLU, para capturar relaciones complejas entre el estado del agente y las acciones posibles. La salida de la red estaría compuesta por un conjunto de neuronas, cada una correspondiente a una acción posible, y la elección de acción se realizaría a través de un enfoque como la política softmax, donde las probabilidades de las acciones están influenciadas por los valores estimados por la red.

2.2.2. Función de Recompensa

La función de recompensa $r(s_t, a_t, s_{t+1})$ se diseña para guiar al algoritmo de aprendizaje por refuerzo hacia un controlador eficiente. Esta función otorga una recompensa positiva por acciones que evitan colisiones y mantienen una distancia segura de otros agentes, mientras penaliza las colisiones y el acercamiento excesivo. Por ejemplo:

- Se otorga una recompensa positiva cuando un agente mantiene una distancia segura de los demás.
- Se otorga una penalización cuando ocurre una colisión o cuando un agente se acerca demasiado a otro.

Esta estructura de recompensa incentiva el aprendizaje de estrategias de navegación seguras y eficientes, favoreciendo la evasión de colisiones.

2.3. Metodología

Esta sección detalla la metodología adoptada para desarrollar y optimizar el modelo de sistemas multiagente en GAMA, enfocándose en las estrategias de implementación y la aplicación del algoritmo de Optimización por Enjambre de Partículas (PSO).

2.3.1. Desarrollo del Modelo en GAMA

El primer paso en la metodología fue la implementación del modelo de agentes *robots* en el entorno GAMA. Este proceso implicó definir las características de los agentes, sus reglas de movimiento y las condiciones de interacción. Se establecieron parámetros clave como la distancia de seguridad, el cambio de dirección y la variación aleatoria para guiar las acciones de los agentes. Además, se programaron condiciones específicas para la finalización de la simulación, como el alcance de un número máximo de iteraciones o cuando todos los agentes alcanzan un estado terminal.

2.3.2. Optimización de Parámetros con PSO

Como se explica en el artículo [5]. Una parte crucial de la metodología fue la optimización de los parámetros del modelo utilizando el algoritmo de Optimización por Enjambre de Partículas (PSO). PSO es una técnica de optimización metaheurística inspirada en el comportamiento social de los enjambres, como los peces y las aves. En PSO, un *enjambre* de soluciones candidatas, denominadas partículas, se mueve a través del espacio de búsqueda de la solución óptima. Cada partícula ajusta su posición en el espacio de búsqueda basándose tanto en su experiencia personal (mejor posición encontrada) como en la experiencia del enjambre (mejor posición encontrada por el enjambre).

En el contexto de nuestro modelo, PSO se utilizó para encontrar la combinación óptima de parámetros que minimizara los eventos no deseados (como colisiones) y maximizara la eficacia del comportamiento del agente. Se realizaron múltiples iteraciones del algoritmo, cada una con diferentes valores de parámetros, para evaluar su impacto en el desempeño general del modelo.

Este enfoque permitió no solo ajustar el modelo para un rendimiento óptimo, sino también proporcionar una comprensión más profunda de cómo cada parámetro influía en el comportamiento del sistema. Los resultados de la optimización PSO ofrecieron perspectivas valiosas para el ajuste fino del modelo y la mejora de las estrategias de evasión de colisiones.

2.4. Resultados

En esta sección se presentan los resultados obtenidos tras realizar varias optimizaciones mediante el algoritmo de Optimización por Enjambre de Partículas (PSO). El objetivo principal de estas optimizaciones era determinar los parámetros más efectivos para la función de recompensa y la política de comportamiento de los agentes en el modelo de simulación.

2.4.1. Optimización de Parámetros

Después de ejecutar múltiples iteraciones de PSO, se identificaron los siguientes parámetros como óptimos para nuestro modelo:

- **Distancia de Seguridad (saveDistance):** Se estableció en 4.98. Este parámetro define la distancia mínima a la que un agente debe mantenerse de otros agentes para evitar colisiones.
- **Cambio de Dirección (headingChange):** Fijado en 70.0. Este valor determina el grado de cambio en la dirección del agente cuando se encuentra en riesgo de colisión.
- **Variación Aleatoria (randomVariation):** Ajustado a 4.5. Esta variable introduce un elemento de aleatoriedad en el movimiento de los agentes, permitiendo comportamientos menos predecibles y potencialmente más eficaces para evitar colisiones.

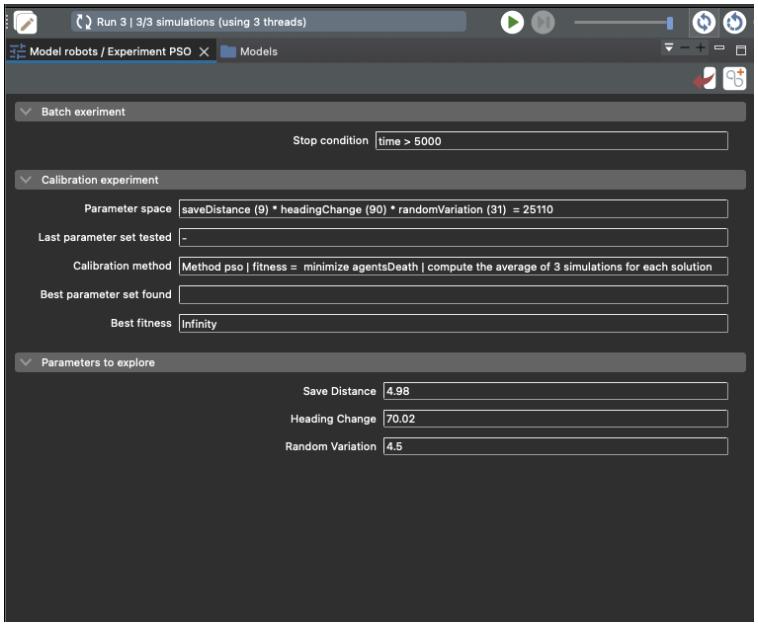

Figura 2.1: Obtención de mejores parámetros

Estos parámetros fueron integrados en la función *applyPolicy* de nuestro modelo, la cual se encarga de ajustar la dirección de los agentes en función de su proximidad a otros agentes. La

lógica implementada busca que, cuando un agente se aproxima peligrosamente a otro (menos de la distancia de seguridad), realice un cambio significativo en su dirección, ajustado por una variación aleatoria para evitar comportamientos predecibles y rígidos.

2.4.2. Análisis de la Implementación y Resultados

Aunque la implementación no es perfecta, los resultados obtenidos han demostrado ser efectivos para prolongar la supervivencia de los agentes en el entorno simulado. Los agentes, equipados con la política de comportamiento optimizada, han mostrado una notable capacidad para evitar colisiones durante periodos prolongados.

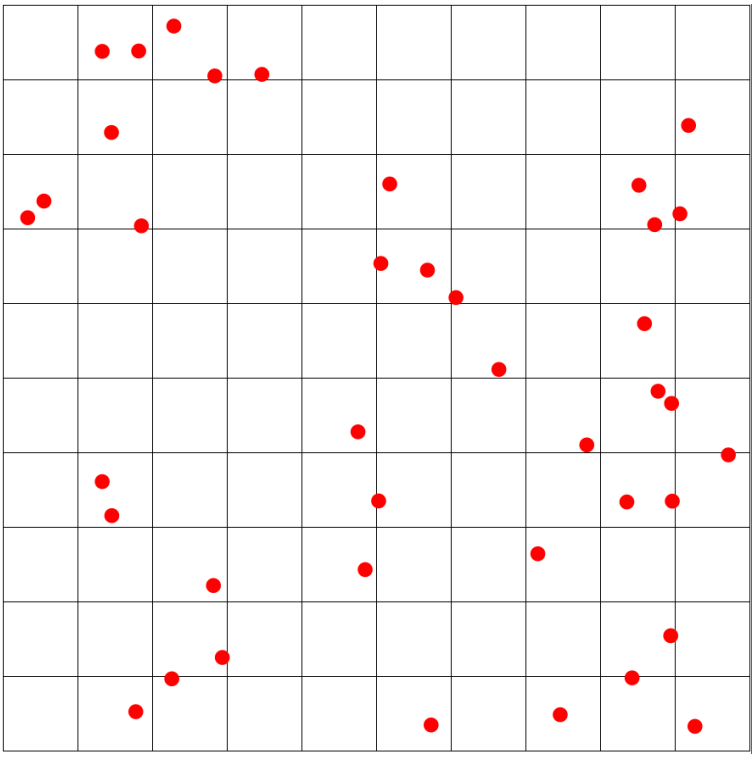

Figura 2.2: Todos los agentes vivos

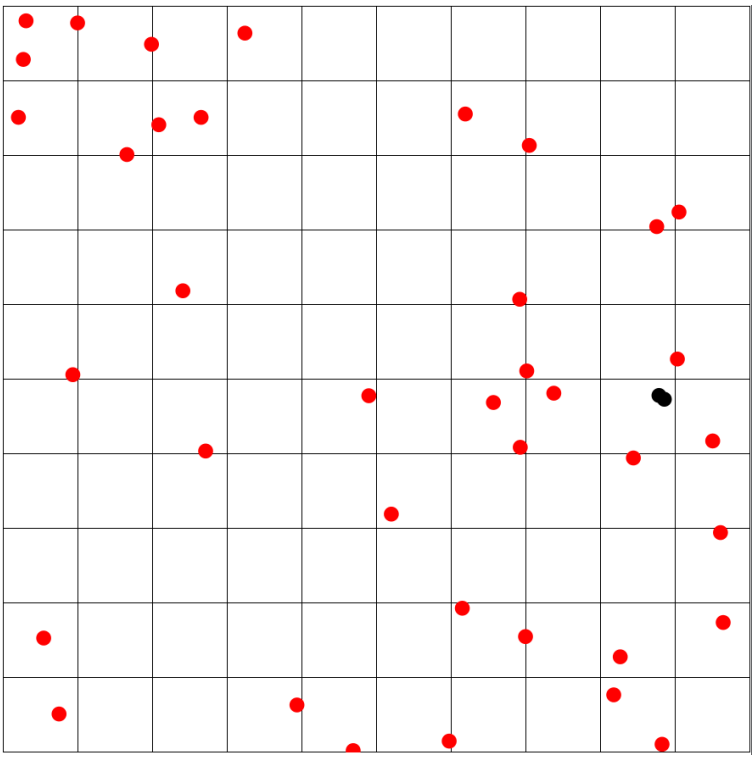

Figura 2.3: Ha ocurrido un choque

2.4.3. Representación Gráfica del Comportamiento de los Agentes

La eficacia de los parámetros optimizados se puede visualizar claramente en la siguiente gráfica. Esta representa la duración de la supervivencia de los agentes en el tiempo, demostrando cómo los ajustes en los parámetros han mejorado significativamente su capacidad para evitar colisiones y mantenerse "vivos.[en] el entorno de simulación.

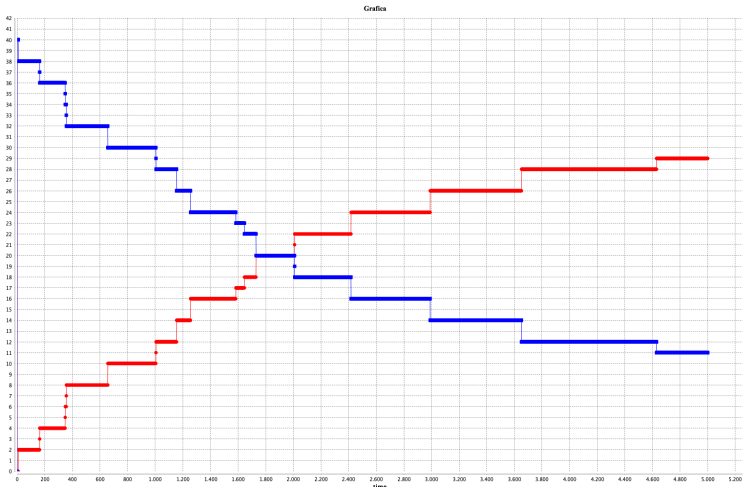

Figura 2.4: Gráfica de agentes vivos y muertos durante la simulación 5000 iteraciones

2.4.4. Conexión al experimento

Por otro lado, uno de los objetivos de la práctica era el posibilitar la conexión a un experimento existente usando Python. Para ello se ha desarrollado un Script en Python que facilita esta conexión.

Este Script se ha configurado de tal manera que se solicita a cada uno de los agentes su posición en coordenadas en el mapa. Y la cantidad de agentes vivos. Esto se puede ver en las imágenes 2.5 y 2.6.

```python
# Sacamos los nombres de todos los agentes
expression_future = asyncio.get_running_loop().create_future()
await client.expression(experiment_id, r"do get_agents_pos;")
gama_response = await expression_future
# Limpiamos la respuesta
agentsPositions = extraer_coordenadas(gama_response["content"])
print("Posiciones de los agentes: ", agentsPositions[0])



if gama_response["type"] != MessageTypes.CommandExecutedSuccessfully.value:
    print("error while trying to run the experiment", gama_response)
    return


expression_future = asyncio.get_running_loop().create_future()
await client.expression(experiment_id, r"agentsAlive")
gama_response = await expression_future
print("asking simulation the value of: agentsAlive=",  gama_response["content"])
```

Figura 2.5: Código para obtener agentes vivos y sus posiciones

```
> python3 ModelConnection.py
connecting to Gama server
initialize a gaml model
received {'type': 'SimulationStatusInform', 'content': {'message': ' Simulation 0 ready'}, 'exp_id': '1159'}
received {'type': 'SimulationStatusInform', 'content': {'message': 'Experiment ready'}, 'exp_id': '1159'}
received {'type': 'CommandExecutedSuccessfully', 'content': '1159', 'command': {'type': 'load', 'model': '/Users/daniel/Desktop/Master IA/AGENTES Y S
ISTEMAS MULTIAGENTE/Agentes/CMG/models/robots.gaml', 'experiment': 'experimento1', 'console': True, 'status': True, 'dialog': True, 'runtime': True}}
initialization successful, running the model
received {'type': 'CommandExecutedSuccessfully', 'content': '', 'command': {'type': 'play', 'exp_id': '1159'}}
received {'type': 'CommandExecutedSuccessfully', 'content': '[{72.85343350996183,56.09480366681105,0.0}, {83.9828336020006,50.8957622034921,0.0}, {84
.17790303237278,63.950511096656975,0.0}, {9.11777617252308,90.73885851694946,0.0}, {24.3222700450033307,33.2923329840259260,0.0}, {46.39573540406527,15
.518744428943378,0.0}, {49.64381126372529,13.167367509393994,0.0}, {51.3662037548965906,85.62614158186110,0.0}, {59.907926586327754,68.80220076285948,0
.0}, {83.89062359428116,93.30138186081825,0.0}, {83.21322728083456,67.13255921585932,0.0}, {36.12633668320963,26.696892101115186,0.0}, {32.47380823725
578,3.511777367873065,0.0}, {51.18344266275322,61.39374818872166,0.0}, {16.967228304634857,3.411676447247170,0.0}, {40.643914366777687,77.0830233665721
4,0.0}, {46.09179200572289,54.84403925604112,0.0}, {90.47199543351293,9.989888400192804,0.0}, {98.09586982505972,64.85213666390672,0.0}, {36.6933644
6004259,27.336377094322785,0.0}, {91.61899614479455,14.273061047484708,0.0}, {1.4344502498486805,57.892235341192020,0.0}, {6.1482903610377057,1.092019400
83460872,0.0}, {44.49544787223781864,5.526898398744456,0.0}, {4.2783860425896044,84.84052792172004,0.0}, {11.963583040884386,1.250315313401275310,0.0}, {3
7.039024345304190,50.596154536488570,0.0}, {53.699205618406340,38.068365146015720,0.0}, {34.102473647377650,82.726590749263920,0.0}, {44.673527872278070,6.0
702011716202650,0.0}, {39.558810694401870,56.872774958476406,0.0}, {48.935391839172240,71.311059370899430,0.0}, {98.697139682193990,1.794253292302578040,0.0
}, {7.767215395091514,86.342471616938810,0.0}, {19.017117218503344,25.429677428563277,0.0}, {12.269480288448102,73.512667077324280,0.0}, {28.018337080885364,28.066837697416602,0.0}, {50.649036495645110,35.484751543375550,0.0}, {2.733642432949444,37.461675571771465,0.0}, {55.109338489563940,39.154911865882350,0.0}]', 'command': {'type': 'expression', 'exp_id': '1159', 'expr': 'do get_agents_pos;'}}
Posiciones de los agentes:  (72.85343350996183, 56.09480366681105)
received {'type': 'CommandExecutedSuccessfully', 'content': '38', 'command': {'type': 'expression', 'exp_id': '1159', 'expr': 'agentsAlive'}}
asking simulation the value of: agentsAlive= 38
```

Figura 2.6: Posiciones de los agentes y cantidad de agentes vivos

2.4.5. Conclusión

Los resultados obtenidos a través de la optimización con PSO indican una mejora significativa en la capacidad de los agentes para navegar de manera segura y eficiente en el entorno simulado. Mientras que hay margen de mejora y refinamiento, los parámetros actuales proporcionan una base sólida para futuras iteraciones y ajustes del modelo.

Capítulo 3

Evaluación de la asignatura

3.1. Evaluación del curso y autorreflexión

¿Qué te gustó?

Lo que más me gustó de trabajar en este proyecto fue la oportunidad de aplicar conceptos teóricos de inteligencia artificial en un entorno práctico y desafiante. La implementación de un agente BDI en el mundo de Wumpus me permitió explorar de manera concreta cómo las creencias, deseos e intenciones pueden influir en la toma de decisiones de un agente. Además, me fascinó ver cómo un conjunto de reglas y comportamientos relativamente simples pueden dar lugar a un comportamiento emergente complejo y adaptativo en el agente.

¿Qué desafíos encontraste?

Uno de los principales desafíos que enfrenté durante este proyecto fue la adaptación a la plataforma GAMA y su lenguaje de programación específico, GAML. Aprender GAML, con su sintaxis y estructura únicas, fue inicialmente una tarea ardua, especialmente al modelar comportamientos complejos de agentes como los BDI. Además, la depuración y optimización de los modelos en GAMA requerían un análisis detallado y experimentación iterativa, lo que a veces se veía complicado por la limitada documentación y recursos de aprendizaje. Este proceso no solo implicaba comprender teóricamente los conceptos avanzados de inteligencia artificial, sino también aplicarlos efectivamente en un entorno de simulación específico, lo que representó un reto significativo pero a la vez enriquecedor en mi desarrollo profesional.

¿Cómo crees que podrías aplicar lo que aprendiste en el futuro?

Los conocimientos y habilidades adquiridos en este proyecto son altamente transferibles a otros ámbitos de la inteligencia artificial y la robótica. Por ejemplo, la comprensión de cómo diseñar agentes que pueden tomar decisiones basadas en información incompleta o incierta es crucial en el desarrollo de sistemas autónomos, como vehículos autónomos o robots de exploración. Además, la experiencia en la implementación de algoritmos de búsqueda y navegación puede ser aplicada en el desarrollo de software para la planificación de rutas en logística o en aplicaciones de mapeo. Finalmente, este proyecto ha reforzado mi capacidad para abordar problemas complejos y desarrollar soluciones creativas, habilidades que son valiosas en cualquier campo tecnológico o de investigación.

Lo que no me gustó:

Un aspecto menos favorable del curso fue la intensa carga de trabajo requerida. Si bien comprendo la importancia de abordar un amplio espectro de temas y aplicaciones, a veces sentí que la cantidad de trabajo comprometía la capacidad de profundizar en temas específicos. Además, esperaba una mayor explicación y aplicación práctica de las tecnologías de enjambre. Dado que estas tecnologías son fundamentales en la inteligencia de enjambres y en sistemas multiagente, una cobertura más detallada habría sido valiosa para comprender mejor su implementación y potencial en contextos reales.

Bibliografía

[1] S. Bussmann, N. R. Jennings, M. Wooldridge, S. Bussmann, N. R. Jennings, and M. Wooldridge, "Agent-based production control," *Multiagent Systems for Manufacturing Control: A Design Methodology*, pp. 9–52, 2004.

[2] Contributors to Wikimedia projects, "Belief–desire–intention software model - Wikipedia," Oct. 2023, [Online; accessed 12. Nov. 2023]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Belief-desire-intention_software_model&oldid=1178367320

[3] J. Perry and H. E. Sevil, "A multi-agent adaptation of the rule-based wumpus world game," *International Journal of Artificial Intelligence and Soft Computing*, vol. 7, no. 4, pp. 299–312, 2022.

[4] V. T. T. Mariano, F. d. J. N. Cárdenas, E. A. Hernández *et al.*, "Análisis de algoritmos de búsqueda en espacio de estados." *Ciencia Huasteca Boletín Científico de la Escuela Superior de Huejutla*, vol. 3, no. 5, 2015.

[5] F. Marini and B. Walczak, "Particle swarm optimization (pso). a tutorial," *Chemometrics and Intelligent Laboratory Systems*, vol. 149, pp. 153–165, 2015.