Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Guadalajara

23/11/24

Modelación de sistemas multiagentes con gráficas computacionales

Evidencia 2 - Entrega Intermedia

Autores:

Fernanda Diaz Gutierrez | A01639572

Miguel Angel Barrientos Ballesteros | A01637150

Carlos Iván Armenta Naranjo | A01643070

Jorge Javier Blazquez Gonzalez | A01637706

Gabriel Alvarez Arzate | A01642991

Evidencia 2: Entrega Intermedia

1) Conformación del equipo de trabajo

a) Fortalezas y Áreas de oportunidad

- **Fernanda Diaz Gutierrez:** Experiencia en comunicación con servidores (Flask), y la creación de endpoints para comunicar 2 sistemas. Facilidad para programar agentes en Python, pero sin experiencia previa en modelación en Unity con lectura de archivos JSON.
- **Miguel Angel Barrientos Ballesteros:** Experiencia en desarrollo de videojuegos en Unity, conocimiento avanzado del manejo de prefabs, configuración de interacciones y sistemas de colisiones. Facilidad de programación en Python.
- **Carlos Iván Armenta Naranjo:** Tiene experiencia realizando la parte de visión computacional y su comunicación con Unity, además de experiencia integrando y realizando endpoints en express, y manejo e integración de llamadas a APIs externas.
- **Jorge Javier Blazquez Gonzalez:** Experiencia en creación de scripts de C# que implementen funciones para los objetos de Unity. Manejo avanzado de configuración y diseño de ambientes virtuales en Unity.
- **Gabriel Alvarez Arzate:** Tiene experiencia en la creación de ambientes en Unity, en lo que corresponde a crear GameObjects y en diseño de modelos en el ambiente de Unity.

2) Expectativas y Compromisos

- **Expectativas del equipo:**
    - Agentes autónomos.
    - Comunicación entre agentes efectiva.
    - Manejo de ambientes en Unity.
    - Comprensión y aplicación de ontologías para la modelación de agentes.
    - Aplicación de visión computacional exitosa utilizando OpenAI en los agentes modelados.
    - Comunicación efectiva entre Unity y Python usando un servidor en Flask.
- **Compromisos:**
    - Trabajar en la creación de una solución creativa y eficiente para las necesidades del socio formador.
    - Cumplir con los plazos de entrega ante el socio formador.
    - Documentar eficientemente el proceso de desarrollo.

3) Creación de Herramientas de Trabajo Colaborativo

a) **Repositorio de GitHub:** https://github.com/cyberpeony/MultiAgentWarehouseSimulation

b) **Herramienta de comunicación:** Para la comunicación entre miembros del equipo, se utilizó Discord. Se utilizó Discord por su facilidad de comunicación inmediata en diferentes vías, sea por canales de video, audio o mensajería.

4) Propuesta Formal del Reto

a) Descripción del problema a resolver:

i) Breve introducción al problema a resolver.

El reto consiste en implementar un modelo de seguridad y vigilancia utilizando cámaras, drones y la participación de un guardia. El objetivo es desarrollar un sistema multiagente para simular situaciones de detección de figuras sospechosas, y visualizarlo en tiempo real en un ambiente 3D en unity. En el caso de nuestro proyecto, el ambiente virtual se sitúa en un exterior/estacionamiento de una empresa transportista de cargas.

El sistema simula la participación del guardia, las 7 cámaras y el dron (que también tiene cámara) en donde dichos agentes están activamente implementando algoritmos de visión computacional para buscar figuras sospechosas en el lugar. Durante la simulación se observa la interacción de los agentes ya mencionados. Cuando una cámara o el dron observa alguna discrepancia en el ambiente, se enviará una alerta al oficial y el enfrentará la situación tomando control del dron.

ii) Importancia del proyecto y su impacto.

Este proyecto tiene mucho peso significativo en la innovación en el sector de la seguridad, tanto pública como privada. El implementar tecnologías como estas en los sistemas de seguridad solo traería muchas ventajas a la mitigación de cualquier situación de peligro que se presente, y poder abordar la situación inicialmente con un dron equipado con las herramientas básicas para enfrentar situaciones hasta medianamente complicadas y que podrían poner en riesgo a los cuerpos de seguridad, de esta forma se podría cubrir la vigilancia de un muy alto porcentaje del área y enfrentar las situaciones que se presenten sin tener que exponer al personal de seguridad a grandes peligros que podrían poner en riesgo hasta su propia vida.

b) Identificación de los agentes involucrados.

i) Agentes identificados y sus interacciones con otros agentes:

(1) Dron: Función principal y capacidades.

El agente “DroneAgent” tiene como función principal patrullar una zona determinada en búsqueda de objetos sospechosos. Su flujo general es:

1. Seguir un path predeterminado de patrullaje mediante un spline. Al iniciar, comunica al GuardAgent que ha comenzado.
2. Si ve un objeto sospechoso mediante su visión computacional (o si recibe un mensaje de un CameraAgent, avisando de un objeto sospechoso), se sale de su ruta de patrullaje.

3. Inspecciona al objeto sospechoso acercándose a él. Mientras inspecciona, avisa al GuardAgent mediante comunicación entre agentes, para que active la alarma. También avisa a WindAgent mediante comunicación, para que este pueda intervenir en su física al dirigirse al objeto.
4. Determina si el objeto fue enemigo o amigo. De ser enemigo, “cede el control” al GuardAgent. De lo contrario, regresa a su path de patrullaje predeterminado.
5. Además, puede recibir instrucciones del GuardAgent al “ceder el control”, como atacar y aterrizar.

(2) Cámaras fijas: Descripción del rol en la detección de movimiento.

El agente diseñado para las cámaras fijas, CameraAgent, tiene como funcionalidad principal utilizar su visión computacional para detectar objetos sospechosos. Su flujo general es:

1. Inspeccionar fijamente en espera de algún spawn de un objeto sospechoso.

2. De detectar un objeto sospechoso mediante su visión computacional, avisar al DroneAgent utilizando comunicación entre agentes, para que inspeccione y siga su flujo de acciones.

3. Seguir inspeccionando, reset a valores booleanos de detección.

Es importante mencionar que este agente tiene 7 instancias, por lo que este flujo sucede para cada cámara.

(3) Personal de seguridad: Toma de decisiones en situaciones de alarma.

El agente de GuardAgent tiene como funcionalidad principal activar la alarma en caso de ser notificado por otro agente (DroneAgent) acerca de un objeto sospechoso, así como tomar control del DroneAgent en caso de que el objeto resulte un enemigo. Su flujo general es:

1. Idle hasta no recibir un mensaje de advertencia de objeto sospechoso del DroneAgent.
2. Al recibir dicho mensaje, activa la alarma.
3. Espera al veredicto del DroneAgent, para saber si el objeto fue enemigo o no. De ser enemigo, “toma control” del DroneAgent y le da las instrucciones de ataque y aterrizaje. De no ser enemigo, apaga la alarma y comunica al ambiente (TMP_Text) que fue una falsa alarma.
4. Reset de variables involucradas, y reinicio de actividades.

(4) Viento: Altera la ruta del dron al ir a inspeccionar un objeto sospechoso.

El agente WindAgent tiene como funcionalidad principal alterar la física del DroneAgent cuando este se desvíe para ir a inspeccionar un objeto sospechoso. Ahora mismo no ha sido implementado, pero se espera que su flujo general sea:

1. Idle hasta que el DroneAgent no comunique que irá a inspeccionar un objeto sospechoso.
2. Interviene una vez que se comunicó dicho mensaje: Mueve aleatoriamente valores de x,y y z de su posición durante su recorrido, para simular variaciones en la física debido al viento. Esto sucede tanto de ida, como de vuelta.
3. Una vez que el objeto vuelve a su path de patrullaje en el spline (desde que aterriza), entra en Idle de nuevo.

c) Propiedades del ambiente y de los agentes

i) Propiedades del ambiente:

El entorno elegido fue un estacionamiento de una empresa transportista de cargas. Este ambiente se eligió dado su espacio abierto y con distintas alturas (para poner cámaras, la plataforma del dron y el guardia). El ambiente tiene las siguientes propiedades:

- **Inaccesible:** Los agentes no pueden acceder a toda información sobre el estado del este, en especial coordenadas de otros agentes, o variables del viento.

- **No determinista:** Si se considera que los objetos sospechosos empiezan en coordenadas aleatorias, puede decirse que el ambiente no es del todo determinista. Además, las reacciones de agentes como dron y cámaras dependen de la visión computacional, lo que provoca que los estados del ambiente no puedan ser predecidos en todo momento.

- **Episódico:** Los agentes deciden siempre únicamente basados en la información del ambiente en el episodio actual. No consideran posibilidades futuras o información que aún no tienen.

- **Dinámico:** El ambiente tiene procesos corriendo por sí solos. Específicamente, la aparición de figuras sospechosas. Esto implica que no todo cambio en este es causado por las acciones de los agentes, lo que vuelve al ambiente dinámico.

- **Discreto:** Existe una cantidad fija de posibles acciones, lo que vuelve al ambiente discreto.

ii) Propiedades de los agentes:

- **Reactividad**: Los agentes mantienen una interacción continua con el ambiente: ven los cambios en figuras sospechosas, las analizan, atacan y se mueven. Esto les permite decidir sus acciones acorde al estado actual del ambiente.

- **Proactividad**: Los agentes tienen baja proactividad en este sistema. Estos no suelen hacer planes a futuro. Siguen rutinas y reaccionan a situaciones nuevas.

- **Habilidad Social**: El sistema tiene un ambiente multiagente. Los agentes no pueden cumplir sus objetivos sin tomar en cuenta a los demás. Es necesaria la comunicación entre cámaras y dron para avisar sobre figuras sospechosas, así como entre dron y guardia para decidir cómo afrontar las situaciones.

d) Diagrama de clase del agente

i) Representación gráfica mostrando los agentes y sus interacciones clave

En los diagramas de clases siguientes se presenta la estructura de los agentes: DroneAgent, GuardAgent y CameraAgent, donde la primera casilla muestra el nombre del agente, la segunda detalla las variables o atributos, la tercera las acciones que puede realizar en el entorno, la cuarta describe los métodos que utiliza para ejecutar dichas acciones y la quinta las reglas para ejecutar las acciones.

DroneAgent:

DroneAgent

State Description:

- **recorded_detections** (int): Cantidad de detecciones sospechosas que se detectaron por el DroneAgent en la simulación.
- **recorded_patrols** (int): Cantidad de veces que el DroneAgent completó su path de patrullaje (se suma al aterrizar).
- **utility** (int): Métrica, para observar la utilidad del DroneAgent al terminar la simulación.
- **myself** (droneAgentOnto): Representación de sí mismo (ontología):
  - **has_id** (int): ID del DroneAgent.
  - **has_path** (Path): Para verificar si tiene un spline para entrar como default path.
    - **path_bool** (bool): True en caso de que sí, False de lo contrario.
  - **has_suspicion** (Suspicion): Para revisar si tiene una sospecha de objeto desconocido.
    - **suspicion_bool** (bool): True si hay sospecha, False si no.
  - **has_target** (Target): Revisa si existe un objetivo asignado al DroneAgent.
    - **target_coordinates** (list): Coordenadas del objetivo.
- **land** (bool): Indica si el dron debe aterrizar.
- **guard_partner** (GuardAgent): Referencia al GuardAgent.
- **camera0_partner** a **camera6_partner** (CameraAgent): Referencias a los CameraAgent.

Actions:

- **patrol()**: El dron inicia un path default de patrullaje (usando splines en Unity) y añade a su action list: "patrol".
- **land()**: El dron regresa a su plataforma de despegue/aterrizaje en (0,0), y aterriza. Añade a su action list: "land".
- **inspect()**: El dron se sale del spline para inspeccionar un objeto sospechoso que fue detectado. Añade a su action list: "inspect". Envía un mensaje al GuardAgent de "suspicion = True".
- **Attack()**: El dron ejecuta un ataque según las órdenes recibidas del guardia. Añade a su action list: "attack". Cambia su sospecha a False, y land a True.

Methods:

- **setup():** Inicializa las variables del agente, incluyendo su representación onto (myself), sus referencias a los otros agentes, su lista de acciones y su lista de reglas.
- **step():** Ejecuta el ciclo de razonamiento y acción del agente, su flujo es:
  - **take_msg():** Para recibir mensajes y actualizar su estado.
  - **see():** Percibe el entorno y evalúa si existe algo sospechoso.
  - **next():** Selecciona la siguiente acción en base a las reglas.
  - **action(act):** Ejecuta la acción seleccionada.
- **take_msg():** Interpreta mensajes del buffer del entorno, reacciona a los mensajes de inspect o attack.
- **see(e):** Percibe el entorno actual, cambia has_suspicion en base a lo observado.
- **next():** Evalúa las reglas y elige la siguiente acción.
- **action(act):** Ejecuta la acción seleccionada.

Rules:

- **rule_patrol(act)**: Evalúa si el dron puede iniciar su ruta de patrullaje (spline):
  - No debe haber sospechas (has_suspicion.suspicion_bool == False).
  - No debe estar aterrizando (land == False).
  - La acción debe ser "patrol".
- **rule_land(act)**: Evalúa si el dron puede aterrizar:
  - No debe haber sospechas (has_suspicion.suspicion_bool == False).
  - El dron debe estar en modo aterrizaje (land == True).
  - La acción debe ser "land".

GuardAgent:

GuardAgent

State Description:

- **recorded_attacks** (int): Número de ataques ejecutados por el guardia mientras controla al DroneAgent.
- **utility** (int): Métrica, para observar la utilidad del GuardAgent al terminar la simulación.
- **enemy** (bool): Indica si se confirmó la presencia de un enemigo (malo), tras comunicarse con DroneAgent.

Actions:

- **takeControl(enemy)**: Toma el control del DroneAgent (dándole instrucciones por comunicación entre agentes) si se confirma la presencia de un objeto enemigo.
  - Envía dichas órdenes si enemy == True. En dado caso, activa la trueAlarm, lo añade a su actionList.
  - Si enemy==false, Activa la alarma Falsa. Lo añade a su actionList.
- **soundAlarm()**: Activa la alarma para notificar a todos los agentes del sistema. Añade "soundAlarm" a su actionList.

Methods:

- **setup():** Inicializa las variables del agente, incluyendo su número de ataques, enemy, lista de acciones y lista de reglas.
- **step():** Ejecuta el ciclo de razonamiento y acción del agente, su flujo es:
  - **take_msg():** Para recibir mensajes y actualizar su estado.
  - **see():** Percibe el entorno.
  - **next():** Selecciona la siguiente acción en base a las reglas.
  - **action(act):** Ejecuta la acción seleccionada.
- **take_msg():** Interpreta mensajes y reacciona como a "soundAlarm" y "checkAlarm".
- **see(e):** Percibe el entorno actual.
- **next():** Evalúa las reglas y elige la siguiente acción.
- **action(act):** Ejecuta la acción seleccionada.

Rules:

- **rule_takeControl(act)**:
  - Si hay un enemigo (**enemy == True**), y su **act == takeControl**, permite la acción de tomar control.

CameraAgent:

CameraAgent

State Description:

- **recorded_detections** (int): Cantidad de objetos sospechosos que el CameraAgent detectó en la simulación.
- **utility** (int): Métrica, para observar la utilidad del CameraAgent al terminar la simulación.
- **enemy** (bool): Indica si la cámara detectó a un enemigo.

Actions:

- **No existen acciones aún, por determinar***

Methods:

- **setup():** Inicializa las variables del agente, incluyendo su número de detecciones, enemy, utility, lista de acciones y lista de reglas.
- **step():** Ejecuta el ciclo de razonamiento y acción del agente, su flujo es:
  - **take_msg():** Para recibir mensajes y actualizar su estado.
  - **see():** Percibe el entorno.
  - **next():** Selecciona la siguiente acción en base a las reglas.
  - **action(act):** Ejecuta la acción seleccionada.
- **take_msg():** Interpreta mensajes entrantes.
- **see(e):** Percibe el entorno actual. Si vio algo, envía un mensaje al droneAgent para que vaya a inspeccionar. Suspicion == True. Aumentan +1 sus recorded_detections.
- **next():** Evalúa las reglas y elige la siguiente acción.
- **action(act):** Ejecuta la acción seleccionada.

Rules:

- **No existen reglas aún, por determinar***

VientoAgent:

Pendiente de realizar, esperando a que se implemente (revisar plan de trabajo).

e) Métricas de utilidad y éxito

i) Utilidad:

El éxito se medirá usando indicadores que fueron especificados como variables o atributos de los agentes, especialmente aquellos que comienzan con la keyword “recorded”. Esto es: “recorded_attacks”, “recorded_detections”, “recorded_patrols” y “utility”. Estas variables son actualizadas durante los steps de la simulación, llevándonos a un “resumen” de la eficiencia y de las estadísticas de las decisiones tomadas por los agentes a lo largo de toda la simulación.

1. **DroneAgent:** Su éxito se mide por la cantidad de detecciones sospechosas obtenidas (recorded_detections) y el número de patrullajes que se completaron (recorded_patrols). Un dron se considera exitoso si logra mantener una alta actividad de patrols (recorridos) y logra detectar los objetos sospechosos de manera correcta y precisa.

2. **GuardAgent:** Su éxito se mide a través de los ataques correctos llevados a cabo ante los objetos sospechosos en la simulación (recorded_attacks). Esta variable nos dice cuántas veces el guardAgent respondió acorde a lo esperado a un enemy que se confirmó.

3. **CameraAgent (7):** Su éxito se mide similar a las del dron, en base a la cantidad de detecciones que logró y registró en (recorded_detections), y cuántas resultaron en un inspect() útil por parte de un dron.

De manera más específica, se evaluará la utilidad del sistema usando indicadores que utilizan las variables mencionadas anteriormente, pero más enfocadas a obtener resultados evaluables y comparables de simulación en simulación.

Para algunas de estas estadísticas aún se necesita detallar o expandir las variables de “record” en algunos agentes, misma mejora que será implementada para la evidencia final. De manera general, se busca evaluar:

1. **Precisión de las detecciones:**

$$Precisión = \frac{recorded-detections-útiles}{recorded-detections-totales}$$

- Para medir la cantidad de detecciones del DroneAgent y los CameraAgents que sí resultaron en una amenaza de tipo “enemy”.

2. **Eficiencia del DroneAgent:**

$$Eficiencia = \frac{recorded-detections}{recorded-patrols}$$

- Para medir la cantidad de sospechas que encuentra el DroneAgent por cada patrullaje (1 ruta).

3. **Tasa de FalseAlarms:**

$$FalseAlarms = \frac{recorded-detections-falsas}{recorded-detections-totales}$$

- Para medir cuántas de las sospechas resultaron en un “FalseAlarm” u objeto de tipo “no enemy”.

4. **Índice de “attacks” efectivos (para los GuardAgents cuando toman el control):**

$$EfficientAttacks = \frac{recorded-attacks}{recorded-detections-confirmadas}$$

- Para evaluar cuántos ataques realizados por los guardias fueron producto de un enemy real (correctos y eficientes).

ii) Criterios: Indicadores específicos

Para las métricas específicas detalladas anteriormente, se establecieron valores para medir su éxito a nivel sistema. Estos valores pueden ser modificados una vez que la versión final de la simulación esté lista. Por el momento, los valores se detallan a continuación:

- **Precisión:** La proporción de detecciones se espera sea mayor o igual a 50%, esto considerando la naturaleza aleatoria de los objetos sospechosos.
- **Eficiencia (DroneAgent):** Se espera al menos 1 detección útil por ruta de patrullaje. Esto nos diría que al menos cada ruta del dron nos está arrojando una detección relevante.
- **FalseAlarms:** Se espera que sea menor o igual a 40%, esto considerando la naturaleza aleatoria de los objetos sospechosos. Esto es 40 de cada 100 detecciones hechas por los agentes, son falsas (u objetos no enemigos).
- **EfficientAttacks:** Mayor o igual a 90%, pues se espera que la gran mayoría de los ataques sí sucedan si se detectó una amenaza real.

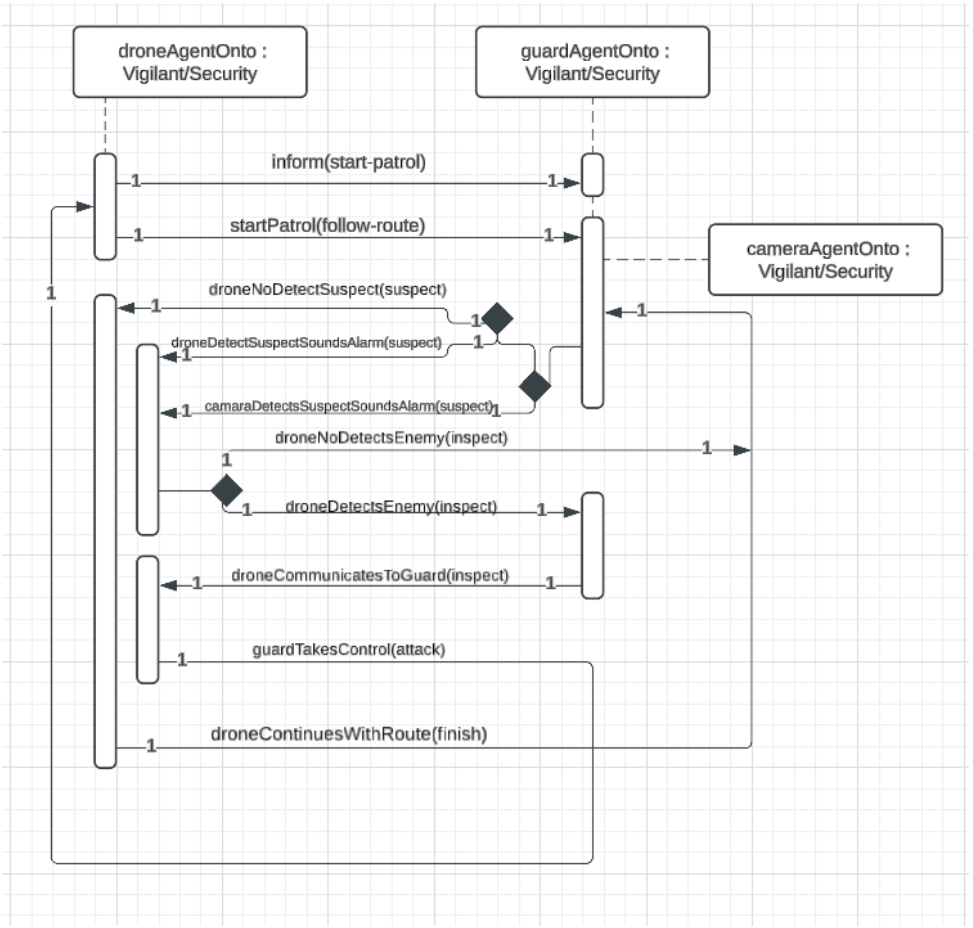
f) Diagramas de secuencia de Protocolos de Interacción

La secuencia de acciones comienza cuando el dron inicia su ruta predefinida. Al iniciar, se notifica al guardia, y las cámaras de seguridad se activan para integrarse al flujo de vigilancia.

Mientras el dron y las cámaras monitorean el entorno, si no detectan ningún objeto sospechoso, el dron continúa su ruta programada hasta completarla. Sin embargo, si el dron detecta algo sospechoso, le notifica al guardia, quien activa una alarma, y posteriormente el dron se dirige hacia el objeto para inspeccionarlo.

En caso de identificar que el objeto no representa una amenaza, el dron regresa a patrullar comunicándole al guardia sobre la falsa alarma. Por el contrario, si se confirma que es un enemigo, el dron avisa inmediatamente al guardia y espera su instrucción.

Una vez notificado, el guardia toma el control del dron, ordena un ataque al enemigo y, tras completar la acción, el dron regresa a su plataforma de inicio para reiniciar todo el flujo.



g) Plan de trabajo y aprendizaje adquirido

i) Plan de trabajo

Actividades pendientes	Responsable	Fecha de inicio	Fecha de fin	Intervalo de esfuerzo (horas)
Implementación del servidor de Flask	Fernanda Díaz	23/11/2024	28/11/2024	12-15 horas
Integración del servidor de Flask en los Agentes en Python.	Fernanda Díaz y Miguel Ángel Barrientos	23/11/2024	28/11/2024	10-12 horas

Integración del servidor de Flask en Unity.	Gabriel Álvarez Arzate	23/11/2024	28/11/2024	8-10 horas
Creación del Agente del Viento en Python.	Fernanda Diaz	23/11/2024	28/11/2024	8-12 horas
Creación de función de loop para cambio de Display en la simulación de Unity con Computer Vision	Carlos Iván Armenta y Jorge Javier Blázquez.	23/11/2024	28/11/2024	12-15 horas

ii) Aprendizaje Adquirido

Hasta el momento se han tenido diversos tipos de aprendizajes entre los cuales se encuentran los siguientes:

- **Creación de Agentes Inteligentes:**

Definición de agentes haciendo uso de una ontología, para estructurar sus conocimientos, habilitando la comunicación entre agentes para simular un ambiente dinámico. Además del modelado del comportamiento y capacidades de los Agentes de manera autónoma.

- **Modelado de un entorno en Unity:**

Diseño del entorno utilizando prefabs, splines y gameObjects para representar escenarios y objetos del mundo real (en este caso un enemigo, un objeto “amigo”/gato, un estacionamiento de una empresa transportista de cargas, etc.), con integración de cámaras tanto dinámicas como estáticas, añadiendo interfaz virtual para probar la interacción de los agentes y sus características.

- **Visión Computacional con OpenAI Vision:**

Implementación de modelos de visión computacional en tiempo real desde el entorno de Unity procesando imágenes para reconocer objetos del entorno permitiendo la integración de decisiones basadas en datos visuales.

- **Comunicación entre las partes de la simulación (Unity y Python):**

Se utilizaron dos métodos de comunicación:

1. Entre Unity y Python un servidor de Flask local, realizando la creación de endpoints para coordinar la comunicación dinámica entre ambas herramientas.
2. Para comunicar los datos de la visión computacional en tiempo real con Unity, se realizó la creación de un virtual environment para instalar dependencias y se usó un puerto de un localhost para levantar la comunicación de ambos lados a través de sockets, donde se manda tanto la imagen como la respuesta del modelo. Esto asegura la comunicación en tiempo real.

- **Integración global y flujo de información:**

Los agentes interactúan con el entorno de Unity basándose en datos visuales procesados por el modelo de visión computacional los cuales son usados por agentes para interactuar y tomar sus decisiones y comportamientos.