Tecnológico de Monterrey

Evidencia 2. Presentación del reto
Modelación de sistemas multiagentes con gráficas computacionales

Alumnos:
Daniel Queijeiro Albo A01710441
Daniel Contreras Chávez A01710608

Instituto Tecnológico y de Estudios Superiores de Monterrey,
Campus Querétaro

28/11/2024

Índice

Introducción

Para este reto tuvimos que implementar una simulación basada en agentes y modelos de la librería Mesa de Python, donde usando nuestro conocimiento obtenido de las clases de multiagentes y gráficas recreamos las reglas del juego de mesa “Flash Point: Fire Rescue” para que nuestros agentes pudieran interactuar el tablero y lograr el objetivo del juego y ganar.

Trasfondo

“Flash Point: Fire Rescue” es un juego de mesa en el que los jugadores asumen el rol de bomberos encargados de rescatar a las víctimas atrapadas en un edificio en llamas. Inspirados en este juego, nuestra simulación presenta una narrativa donde un pueblo descubre una casa habitada por brujitas. Al intentar quemarla, las brujitas, que en realidad son buenas y están preparando consomé, necesitan de sus secuaces duendes para salvarlas.

Estrategia utilizada

Para garantizar que nuestros agentes pudieran cumplir eficazmente con sus objetivos, implementamos la siguiente estrategia:

1. Asignación de Puntos de Interés: Si al inicio de su turno el agente no tiene un punto de interés asignado, se le es asignado uno.

2. Movimiento: A través del cálculo de la distancia euclidiana se determina el camino más corto entre el punto de interés y el agente, así los agentes se desplazan de manera eficiente hacia sus puntos de interés, minimizando el tiempo y los recursos utilizados.

3. Acciones Alternativas: En caso de que no haya puntos de interés disponibles, los agentes priorizan apagar fuegos activos, contribuyendo a la contención general del incendio.

Esta estrategia se diseñó para maximizar la eficiencia de los agentes, asegurando que cada uno contribuya de manera significativa al objetivo común.

Métodos y herramientas

Dentro de los métodos que empleamos destacan:

- Movimiento y verificación: Midiendo la distancia euclidiana entre el agente y su objetivo, fijandonos si hay una pared en su camino la pueda romper.
- Asignación y toma de decisiones: En base al contexto del tablero los agentes pueden ir hacia un punto de interés, apagar fuegos o romper paredes.

Para el desarrollo del proyecto usamos las siguientes herramientas:

- Mesa: Utilizamos la librería Mesa de Python para el modelado y la simulación de agentes. Mesa proporciona una estructura robusta para definir comportamientos, interacciones y reglas del entorno.
- Unity: Empleamos Unity para la visualización gráfica de la simulación. Mediante la importación de modelos 3D, logramos representar de manera visual las acciones y movimientos de los agentes.
- Flask: Implementamos Flask como servidor para gestionar la comunicación entre el modelo de simulación y la interfaz de Unity.

Conexión Cliente-Servidor

Con nuestro servidor activo tenemos la posibilidad de hacer dos peticiones GET.

GET (/api/map):

- Descripción: Este endpoint proporciona la información del mapa en formato JSON.
- Uso: Unity utiliza estos datos para generar el tablero de la simulación, asegurando que cualquier configuración del tablero pueda ser recreada visualmente.

GET (/api/simulation):

- Descripción: Este endpoint ejecuta la simulación de los agentes y devuelve toda la información relevante sobre sus acciones y estados en una sola respuesta.
- Uso: Unity interpreta estos datos para animar los movimientos y decisiones de los agentes, permitiendo una representación de la simulación.

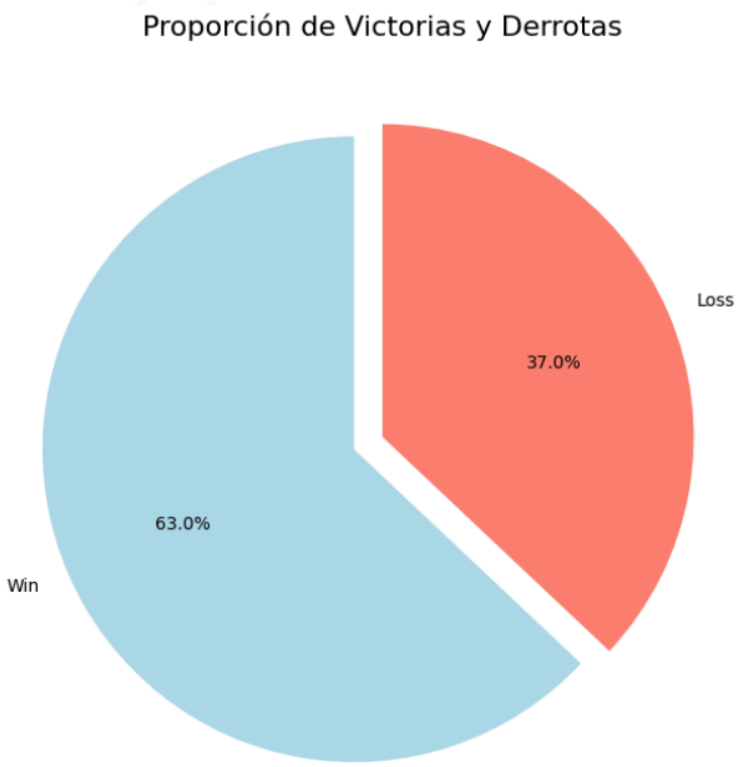
Visualización

Enlace al video grabado de la simulación en Unity:

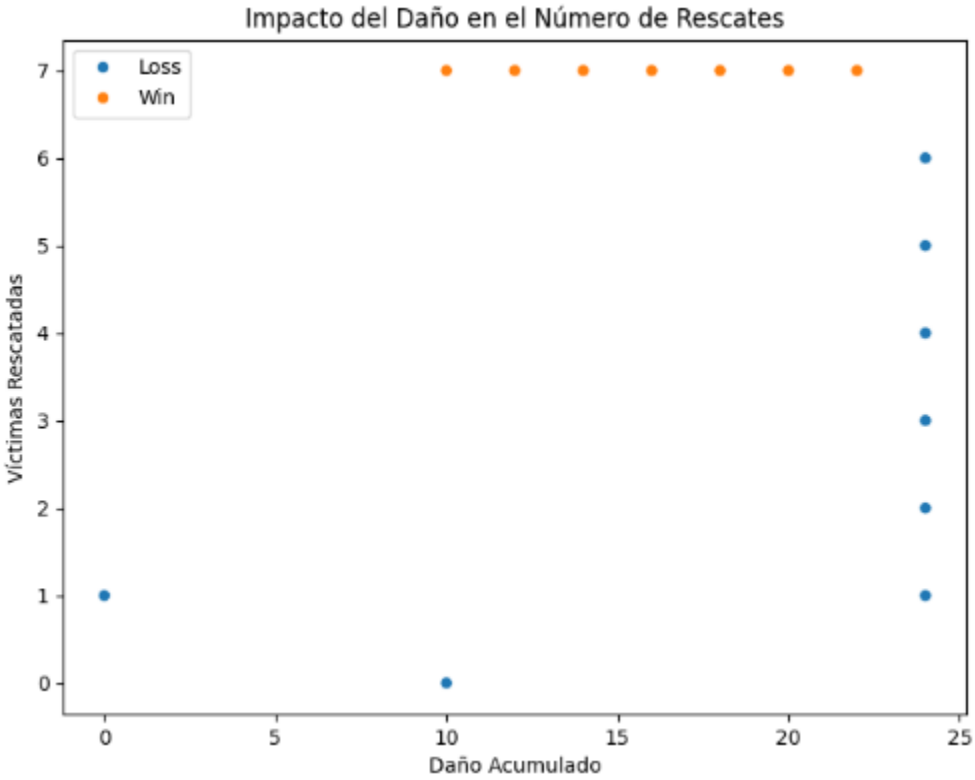
VideoJuego.mp4

Resultados

Para los resultados hicimos un total de 100 iteraciones de la simulación y graficamos los resultados con librerías de Python.

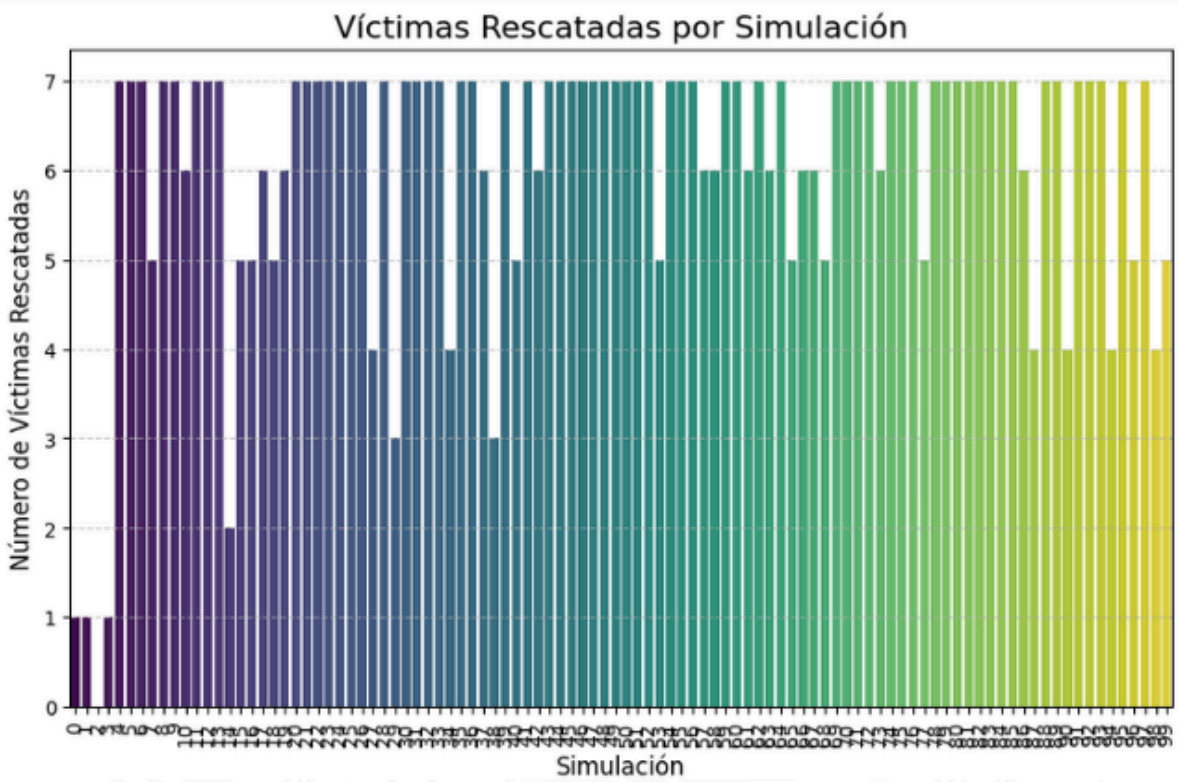


En este primer gráfico podemos ver que dentro de las 100 iteraciones tuvimos una tasa de victorias del 63%



En este gráfico podemos notar que los agentes empiezan a ganar desde que el edificio acumula 10 de daño hasta incluso casos de 23 de daño justo antes de perder por el daño acumulado.

También vemos casos donde “pierden” por sobrepasar los 600 pasos limite de la simulación.



Aquí podemos observar que en promedio los agentes lograron rescatar por lo menos a 4 víctimas por simulación. Además podemos ver que la mayoría de las simulaciones, independientemente de ser victorias o derrotas, se concentran entre las 6 y 7 víctimas rescatadas.

Posibles mejoras

Estamos conscientes que tenemos áreas de mejora para el proyecto. El modelado de agentes y la visualización en Unity tienen elementos faltantes para poder brindar una buena experiencia de juego.

Principalmente con los agentes podemos terminar de implementar todas las reglas del juego original, tal como que los agentes sean derribados por explosiones, que los agentes no terminen su turno en celdas con fuego, que las explosiones matan a víctimas.

Y en cuanto a Unity hace falta animaciones que permitan entender todo lo que están haciendo los agentes, tal como derribar paredes, recoger víctimas, entrar y salir del edificio.

Conclusiones

Nuestra implementación demuestra exitosamente la viabilidad de utilizar sistemas multi-agente para modelar escenarios complejos. Con una tasa de victoria del 63%, el sistema muestra resultados prometedores en el logro de sus objetivos. La combinación de Mesa para simulación y Unity para visualización es una gran manera para modelar y analizar el comportamiento de los agentes que creamos, aunque si logramos identificar áreas clave para mejorar la eficiencia y experiencia de simulación.