UACM

Universidad Autónoma de la Ciudad de México

NADA HUMANO ME ES AJENO

COLEGIO DE CIENCIA Y TECNOLOGÍA

LICENCIATURA EN MODELACIÓN MATEMÁTICA

Simulación Estocástica de Multitudes como un Sistema Complejo

T E S I S

QUE PARA OPTAR POR EL TÍTULO DE

LICENCIADO EN MODELACIÓN MATEMÁTICA

P R E S E N T A

MANUEL ANTONIO SALAS CHÁVEZ

D I R E C T O R

M. EN C. ENRIQUE CRUZ MARTÍNEZ

Ciudad de México, junio de 2026.

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

CA Coordinación Académica

SISTEMA BIBLIOTECARIO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO
COORDINACIÓN ACADÉMICA

RESTRICCIONES DE USO PARA LAS TESIS DIGITALES

DERECHOS RESERVADOS©

Agradecimientos

Deseo expresar mi más sincero agradecimiento a todas las personas que contribuyeron a mi formación académica y al desarrollo de este trabajo de titulación.

En primer lugar, agradezco al M. en C. Enrique Cruz Martínez, director de esta tesis, por su guía, paciencia y apoyo constante durante la realización de este proyecto. Sus conocimientos, observaciones y consejos fueron fundamentales para llevar este trabajo a buen término.

Asimismo, agradezco a los lectores de esta tesis: el M. en C. Agustín González Villanueva, el M. en C. Fernando René Martínez Ortiz y el M. en C. Rafael Torres Simón, por el tiempo dedicado a la revisión de este documento y por sus valiosas sugerencias, las cuales contribuyeron a enriquecer y fortalecer este trabajo.

También quiero agradecer a mis compañeros de los laboratorios LAMAT y LACECI por su compañerismo, apoyo y por compartir experiencias, conocimientos y momentos que hicieron más agradable y enriquecedora esta etapa de mi formación.

Finalmente, agradezco a todas las personas que me brindaron su apoyo y confianza a lo largo de este camino. Este logro también es resultado de su presencia e influencia en mi vida.

A todos ustedes, muchas gracias.

Con gratitud Antonii.

Índice general

Resumen

El presente trabajo desarrolla y emplea algoritmos para simular evacuaciones de personas usando una o más computadoras. La idea principal es entender cómo se mueve la gente cuando hay una situación de emergencia, como un incendio o un temblor, y ver qué rutas funcionan mejor para salir rápido entendido como la mejor ruta en una exploración parcial pero eficiente del espacio físico con obstáculo (Algoritmos: A*, A* cooperativo (CA*) y Monte Carlo (MC)).

Primero se explica la teoría de cómo modelar las multitudes con diferentes enfoques. En el caso de estudio se muestra cómo tomar una imagen digital (como el plano de un edificio) y transformarla en una matriz de celdas donde se distingue qué zonas son caminos libres y cuáles son obstáculos.

Se definen los individuos como una entidad computacional (agente) y se calculan las trayectorias usando los algoritmos A* y CA* para más de un agente. A lo largo del desarrollo se hacen las simulaciones únicamente en la planificación de rutas sobre una cuadrícula discreta, donde los agentes se mueven paso a paso sin considerar dinámicas de aceleración o inercia.

Se hicieron varias simulaciones cambiando la cantidad de personas, desde 1 hasta 200 y se compararon mapas generados por computadora con mapas de imágenes reales. Se presentan los resultados en imágenes, tablas e histogramas para comparar diferentes escenarios incluyendo saturación de espacios. Finalmente se muestra el trabajo a futuro.

Capítulo 1

Simulación de Multitudes y Evacuación: Un Enfoque Accesible

1.1. Introducción

La simulación de multitudes y evacuación es un campo que se dedica a entender cómo reaccionan y se comportan las personas en situaciones de emergencia, como incendios, terremotos o cualquier otro tipo de desastre. Este tipo de simulación tiene una gran importancia porque permite prever cómo se moverán las personas en lugares cerrados o públicos, lo cual es esencial para mejorar la seguridad en eventos masivos y en la planificación de edificios.

A través de simulaciones, no solo se hacen suposiciones sobre cómo las personas reaccionan, sino que se crean modelos matemáticos y computacionales que tratan de predecir lo que realmente sucedería en una emergencia. De esta manera, es posible analizar con mayor precisión qué rutas de evacuación son las más eficientes o cómo evitar la sobrepoblación en determinados espacios **(Hernández, 2015)**.

Algunos ejemplos de aplicaciones prácticas de la simulación de multitudes son:

- Eventos Masivos: En conciertos, festivales o partidos deportivos, la simulación ayuda a prever la distribución de la multitud para evitar aglomeraciones peligrosas.
- Evacuaciones de emergencia: Durante simulacros o situaciones reales, las simulaciones permiten analizar el comportamiento de las personas mientras se desplazan hacia las salidas. Estos modelos permiten evaluar aspectos clave como las rutas de evacuación utilizadas, los tiempos de salida y la identificación de posibles cuellos de botella, lo que contribuye a mejorar la seguridad y eficiencia del proceso de evacuación.
- Edificios de Oficinas: Antes de la construcción de un edificio, se pueden simular las rutas de evacuación para asegurar que todos los ocupantes puedan salir de manera segura en caso de incendio u otros incidentes.
- Centros Comerciales: Las simulaciones también se utilizan para estudiar las rutas de evacuación en grandes centros comerciales, especialmente durante eventos especiales con gran concurrencia de personas.

1.2. ¿Cómo Funciona la Simulación de Multitudes?

Cuando hablamos de simulación de multitudes, nos referimos a la tarea de comprender cómo las personas se mueven, cómo interactúan entre ellas y cómo toman decisiones dentro de un

espacio determinado. Para ello, existen diferentes enfoques, que se dividen principalmente en dos tipos: **modelos continuos** y **modelos discretos**.

1.2.1. Modelos Continuos

Imagina que la multitud es como un fluido, similar al agua que fluye por un río. En este caso, no tratamos a cada persona de manera individual, sino que estudiamos el comportamiento global del flujo de personas. Este enfoque es especialmente útil en situaciones con grandes cantidades de gente, como en estadios o grandes edificios, ya que permite entender cómo se mueve la multitud en su conjunto.

Un aspecto fundamental de este modelo es la densidad crítica, que se refiere al punto en que la multitud alcanza una concentración tal que el flujo de personas se vuelve más lento o incluso se bloquea. Este fenómeno ocurre cuando la densidad supera ciertos límites, lo que puede generar cuellos de botella y dificultar el movimiento fluido de las personas.

Un ejemplo de este modelo es el modelo de flujo de partículas, que describe cómo las partículas (o personas) se desplazan a través de un medio, con su comportamiento dependiente de la densidad del flujo.

1.2.1.1. Flujo de Personas

En un modelo continuo, el flujo de personas puede describirse mediante la ecuación de continuidad, que en física describe cómo una cantidad se conserva en un flujo. Para las multitudes, esta ecuación puede expresarse como:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0$$

donde:

- ρ es la densidad de personas (número de personas por unidad de área),
- $\mathbf{v}$ es la velocidad del flujo de personas,
- t es el tiempo.

Este modelo implica que la densidad de personas varía con el tiempo a medida que la multitud se mueve hacia las salidas o se dispersa **(Ruiz, 2017)**.
La ecuación de continuidad describe el principio de conservación de la masa, en este caso, la conservación del número total de personas dentro de una región cerrada. Es decir, cualquier cambio en la densidad de personas ρ en una región debe ser compensado por el flujo de personas que entra o sale de dicha región. De manera más formal, la ecuación establece que el número de personas dentro de un sistema cerrado se conserva en el tiempo, ya que:

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{v})$$

donde el término $\frac{\partial \rho}{\partial t}$ representa la tasa de cambio de la densidad de personas en el tiempo, y el término $\nabla \cdot (\rho \mathbf{v})$ es el flujo neto de personas que entran o salen de una región.

Un ejemplo de este tipo de modelo es el **modelo de flujo de partículas en un sistema cerrado**, en el que se modela cómo las personas, vistas como partículas, se mueven a través de un espacio con un flujo continuo hacia una salida. Este tipo de simulación es común en

eventos masivos como conciertos o partidos deportivos, donde se predice cómo se distribuye la multitud a lo largo del tiempo hacia las salidas disponibles. En este contexto, las observaciones pueden considerar canales de flujo con una geometría específica que afecta la distribución de las personas.

1.2.2. Modelos Discretos

En los modelos discretos, cada persona es tratada de manera independiente, como si fuera un agente en un videojuego. En este caso, cada persona toma decisiones sobre cómo moverse dependiendo de su entorno, las personas a su alrededor y su objetivo final: la salida. Este enfoque es más detallado y se usa cuando es necesario comprender cómo interactúan los individuos en una multitud. Un ejemplo típico de este tipo de simulación es el modelo basado en agentes (ABM) **(Helbing, 2012; Bonabeau, 2002)**.

Los modelos discretos se utilizan para simular y entender fenómenos complejos que involucran interacciones individuales, como el tráfico urbano, la propagación de enfermedades o la dinámica social. A continuación se presentan algunos ejemplos de la vida real en los que se utilizan modelos discretos:

Simulación de multitudes: En el ámbito de la evacuación de edificios o la organización de eventos masivos, los modelos basados en agentes ayudan a predecir cómo las personas se moverán y se organizarán para salir de manera eficiente, minimizando el riesgo de congestionamientos **(Helbing, 2000)**.

Transporte y tráfico urbano: Los modelos discretos se utilizan para simular el comportamiento de los conductores y peatones en las ciudades, optimizando la gestión del tráfico y reduciendo los tiempos de espera en los semáforos o los embotellamientos **(Nagel & Schreckenberg, 1992)**.

Propagación de enfermedades: En el campo de la epidemiología, los modelos basados en agentes permiten simular cómo se propaga una enfermedad infecciosa entre individuos, teniendo en cuenta el comportamiento social y las interacciones entre personas **(Epstein, 2005)**.

Mercados financieros: Los modelos discretos también se aplican en la simulación de mercados, donde los agentes (inversores) toman decisiones de compra o venta de acuerdo con las condiciones del mercado, con el fin de estudiar fenómenos como las burbujas especulativas o los choques financieros **(Farmer, 2005)**.

Estos modelos permiten hacer predicciones y tomar decisiones informadas en situaciones complejas donde el comportamiento colectivo depende de la interacción de los agentes individuales.

1.2.2.1. Interacción entre Personas

En un modelo discreto, cada persona toma decisiones basadas en las acciones de los demás. Un modelo comúnmente utilizado es el **modelo de fuerzas sociales**, que modela la interacción de cada individuo con su entorno y con otros agentes de la multitud. Las fuerzas que afectan el movimiento de las personas se pueden describir por una ecuación como:

$$\mathbf{F} = \mathbf{F}_{\text{atracción}} + \mathbf{F}_{\text{repulsión}} + \mathbf{F}_{\text{presión}}$$

donde:

- $\mathbf{F}_{\text{atracción}}$ representa la fuerza que mueve a la persona hacia la salida,
- $\mathbf{F}_{\text{repulsión}}$ representa la fuerza que evita el choque con otras personas,
- $\mathbf{F}_{\text{presión}}$ es la fuerza que tiene en cuenta la densidad de personas en un área.

Estas fuerzas determinan cómo cada persona se mueve dentro del espacio, tomando en cuenta las decisiones de la multitud y las limitaciones del entorno. En este contexto, un **agente** es un individuo autónomo dentro de la simulación, que posee la capacidad de percibir su entorno, tomar decisiones y actuar en consecuencia. Cada agente representa a una persona con características y comportamientos propios, lo que permite modelar interacciones complejas y dinámicas realistas en la multitud.

Un ejemplo de este modelo es el uso de simulaciones para prever el movimiento de personas en una evacuación, donde las personas se mueven hacia la salida pero también se ven afectadas por la presencia de otras personas **(Hernández, 2015)**.

Además, los modelos de simulación basados en agentes pueden incluir comportamientos complejos como el pánico o el cambio de dirección de los individuos cuando se encuentran con obstáculos o con personas que se comportan de manera errática **(Martínez, 2016)**.

Este modelo permite simular cómo las personas reaccionan ante las salidas y cómo evitan el choque con otras personas. También se puede modelar el pánico o la sobrepoblación, situaciones en las que las personas se empujan unas a otras. Estos modelos ayudan a entender cómo el comportamiento de un individuo puede afectar a toda la multitud **(Hernández, 2015)**.

1.3. Pensamiento Individual y Colectivo en las Multitudes

Cuando las personas se encuentran en una multitud, sus decisiones no solo dependen de lo que piensan individualmente, sino también de lo que sucede a su alrededor. El **pensamiento individual** es cuando una persona toma decisiones por sí misma basándose en su propio juicio, mientras que el **pensamiento colectivo** se refiere a cómo las personas, como grupo, toman decisiones influenciadas por las acciones de los primeros vecinos.

1.3.1. Pensamiento Individual

El pensamiento individual ocurre cuando cada persona toma decisiones por su cuenta, dependiendo de sus necesidades, emociones y percepciones. Por ejemplo, en un concierto, si comienza a moverse mucha gente hacia una salida, algunas personas pueden decidir seguir a la multitud, mientras que otras podrían esperar o buscar una salida diferente.

Un ejemplo sencillo es el de una evacuación de emergencia: si una persona tiene miedo y decide correr hacia la salida más cercana, otras personas que la ven correr podrían seguirla sin entender exactamente la razón de su acción. Esto muestra cómo el miedo y el pánico pueden influir en el comportamiento individual.

1.3.2. Pensamiento Colectivo

El pensamiento colectivo, por otro lado, ocurre cuando las decisiones de un individuo son influenciadas por las decisiones de los demás. En una multitud, el comportamiento de una sola

persona puede desencadenar una reacción en cadena. Por ejemplo, si una persona empieza a correr, otras personas pueden hacer lo mismo sin saber la razón, lo que podría generar una estampida.

En situaciones de pánico, como cuando las personas intentan escapar de un incendio, los individuos tienden a seguir el comportamiento de la multitud. Esto puede ser peligroso, ya que las personas actúan sin pensar en las consecuencias, lo que puede agravar la situación.

La simulación de multitudes trata de modelar tanto el pensamiento individual como el colectivo para predecir cómo las personas se moverán y tomarán decisiones en una emergencia. Otro objetivo importante es establecer reglas operativas, es decir, un conjunto de protocolos o directrices prácticas que guían el comportamiento y las acciones durante un simulacro, con el fin de optimizar la evacuación y garantizar la seguridad de los participantes.

1.4. Aplicaciones Reales de la Simulación de Evacuación

La simulación de evacuaciones no es solo una teoría; tiene aplicaciones prácticas que pueden salvar vidas. A continuación se describen algunas de estas aplicaciones:

1.4.1. Diseño de Espacios Seguros

Cuando se construyen edificios o estadios, los diseñadores utilizan simulaciones para asegurarse de que las personas puedan evacuar rápida y seguramente en caso de emergencia. Estas simulaciones permiten identificar las rutas de evacuación más eficientes y asegurarse de que las personas puedan salir del edificio sin problemas, incluso cuando la multitud está muy concentrada **(Ruiz, 2017)**.

1.4.2. Seguridad en Eventos Masivos y Respuesta a Emergencias

En eventos con gran concentración de personas, como conciertos o partidos deportivos, la simulación ayuda a prever posibles situaciones de riesgo, como la congestión de personas en pasillos o la posibilidad de una estampida. Gracias a estas simulaciones, los organizadores pueden tomar medidas preventivas para reducir estos riesgos **(Martínez, 2016)**.

En situaciones de desastre, como incendios o terremotos, los modelos de evacuación permiten predecir cómo se moverán las personas y cómo optimizar las rutas hacia las salidas. Esto puede ayudar a reducir los tiempos de evacuación y evitar accidentes **(Hernández, 2015)**.

1.4.3. Estudios de Caso

Para mostrar cómo la simulación de evacuación se aplica en la vida real en México, tenemos los siguientes casos:

1.4.3.1. Evacuación en el Sistema de Transporte Colectivo (STC) Metro de Ciudad de México

Uno de los estudios más relevantes en México fue realizado en el Sistema de Transporte Colectivo (STC) Metro de Ciudad de México, que es uno de los sistemas de transporte más grandes y concurridos del mundo. En este caso, se utilizaron simulaciones para analizar cómo los pasajeros reaccionarían ante emergencias, como fallos en el sistema eléctrico o la presencia de

un incendio en uno de los vagones. La simulación ayudó a identificar los puntos críticos donde se podrían generar embotellamientos o aglomeraciones peligrosas durante una evacuación. Como resultado, se implementaron mejoras en las rutas de salida, en las estaciones más concurridas, y se diseñaron procedimientos de evacuación más eficientes para reducir el tiempo de respuesta ante emergencias **(Gómez, 2020)**.

1.4.3.2. Evacuación en el Aeropuerto Internacional de la Ciudad de México (AICM)

El Aeropuerto Internacional de la Ciudad de México (AICM) ha utilizado simulaciones de multitudes y evacuación para mejorar la seguridad de los pasajeros en situaciones de emergencia. Este tipo de simulación es particularmente importante en aeropuertos internacionales, donde el tráfico de personas es intenso y las condiciones de evacuación deben estar perfectamente coordinadas. Los estudios realizados en el AICM permitieron optimizar la señalización de las rutas de evacuación y prever las zonas donde la densidad de personas podría volverse peligrosa durante un desastre natural, como un terremoto. Las simulaciones también ayudaron a mejorar los protocolos de seguridad y a entrenar al personal en la gestión de multitudes durante una crisis **(Morales, 2018)**.

1.4.3.3. Simulación de Evacuación en Centros Comerciales de Cancún

En Cancún, uno de los destinos turísticos más importantes de México, varios centros comerciales de gran concurrencia implementaron simulaciones para evaluar sus planes de evacuación en caso de un desastre natural, como un huracán o un sismo. A través de modelos de simulación discreta, se analizó cómo los turistas y los residentes locales se desplazarían hacia las salidas en diferentes escenarios de emergencia. Las simulaciones revelaron puntos vulnerables, como áreas con flujo congestionado y rutas secundarias que podrían ser mejoradas. Como resultado, se mejoraron las estrategias de evacuación, se establecieron procedimientos más claros y se optimizó el uso de las salidas de emergencia para evitar accidentes **(Rodríguez, 2019)**.

1.5. Fundamentos de la Simulación de Multitudes

1.5.1. Introducción a la Simulación de Multitudes

La **simulación de multitudes** es un `campo interdisciplinario` que emplea herramientas matemáticas y computacionales para modelar el comportamiento colectivo de personas en contextos específicos, tales como eventos masivos, evacuaciones o flujos en entornos urbanos. Este campo es esencial en áreas como la **seguridad pública**, el **diseño de espacios urbanos** y la **gestión de emergencias**, ya que permite prever patrones de movimiento, identificar posibles puntos críticos de aglomeración y evaluar rutas de evacuación, todo con el fin de mejorar la seguridad y la eficiencia en situaciones de alta concentración de personas.

Según **Hernández (2015)**, la simulación de multitudes ofrece un marco valioso para comprender cómo se comportan las personas en circunstancias particulares, especialmente durante situaciones de crisis, como **evacuaciones masivas** en estadios o edificios públicos. Estos modelos de simulación permiten estudiar fenómenos como los **cuellos de botella**, los bloqueos y las interacciones humanas que pueden ocurrir durante una evacuación, ayudando a planificar mejor los diseños de espacios para minimizar riesgos.

La capacidad para predecir la **dinámica de las multitudes** también tiene aplicaciones en eventos masivos como conciertos, protestas o manifestaciones, donde la predicción precisa de

los flujos de personas puede ser crucial para evitar desbordamientos o situaciones peligrosas **(Ruiz, 2017)**.

1.5.2. Teoría Matemática de la Simulación de Multitudes

Uno de los conceptos fundamentales que subyace en la simulación de multitudes es el **camino aleatorio** o **random walk**, un modelo estocástico que describe el movimiento aleatorio de partículas o individuos dentro de un sistema. Este modelo es esencial para representar el comportamiento errático de las personas dentro de un espacio determinado, ya que las personas no siempre siguen trayectorias predecibles, sino que interactúan con su entorno y con otras personas de forma impredecible **(Rycroft, 2005)**.

El **paseo aleatorio** puede verse como una forma de modelar cómo un individuo se desplaza de manera aleatoria dentro de un espacio, tomando decisiones basadas en su entorno inmediato. Según **Edelman (2004b)**, este tipo de modelos se pueden extender a situaciones más complejas, como la simulación de multitudes en lugares cerrados, donde la interacción entre los individuos y los obstáculos físicos, como las paredes o las puertas de salida, juega un papel crucial en la dinámica del flujo de personas.

Además, los modelos de **flujo de tráfico** y **aglomeración** también son aplicados en el contexto de la simulación de multitudes. Estos modelos, como los descritos por **Haberman (2012)**, ayudan a simular cómo los individuos se mueven a través de un entorno cerrado, como un edificio o una estación de metro. Cuando las multitudes se agrupan, pueden formarse cuellos de botella, lo que ralentiza o incluso detiene el proceso de evacuación. Este fenómeno puede modelarse mediante sistemas de **redes de tráfico**, donde cada individuo se ve como un nodo dentro de una red que interactúa con otros nodos **(Rodríguez, 2019)**.

1.5.3. Modelos de Comportamiento Humano en la Simulación de Multitudes

Uno de los aspectos más complejos de la simulación de multitudes es la **modelización del comportamiento humano**, ya que las personas no actúan de forma totalmente predecible. En situaciones de emergencia, factores como: el *pánico*, la *confusión* o la *imitación social* juegan un papel fundamental. Las personas tienden a seguir a otras cuando están inseguras de cómo actuar, lo que puede generar efectos de **masa** en los que la multitud se desplaza de forma desorganizada, con el potencial de crear bloqueos o situaciones de **pánico colectivo (Ruiz, 2017)**.

En el contexto de las **evacuaciones masivas**, el comportamiento humano puede ser modelado utilizando **modelos de agentes**. Estos modelos representan a cada individuo como una unidad autónoma que toma decisiones basadas en su entorno inmediato y en las interacciones con otros agentes **(Hernández, 2015)**. En situaciones de evacuación, las decisiones que toma una persona, como seguir a otros, buscar una salida más cercana o reagruparse con amigos, pueden influir en la dinámica general del flujo de personas.

Rodríguez (2019) discute cómo el **comportamiento social** en situaciones de crisis puede influir en las decisiones de las personas, especialmente cuando se observan señales claras de evacuación. Un factor importante que se destaca es la **identificación de puntos de salida seguros**, lo cual es esencial para diseñar simulaciones de evacuación efectivas. Este comporta-

miento colectivo puede influir en cómo las multitudes se organizan y se mueven en el espacio, un concepto que es fundamental para crear modelos precisos de evacuación.

1.5.4. Métodos de Simulación y Herramientas

Existen varias herramientas computacionales utilizadas para simular el comportamiento de multitudes. Algunas de las más comunes son: *AnyLogic*, *Simul8* y *Pathfinder*. Estas plataformas permiten modelar flujos de personas y simular situaciones de evacuación, desde emergencias con pánico hasta flujos más controlados en espacios públicos.

Estas simulaciones están basadas en **modelos matemáticos** como: *redes*, *paseos aleatorios* y *matrices de adyacencia*, que describen cómo las personas interactúan en un espacio. Una **matriz aleatoria** es una matriz cuyos elementos son variables que toman valores determinados por un proceso de azar, en lugar de ser fijos. **Edelman (2004)** explica cómo las matrices aleatorias se pueden usar para modelar sistemas donde las interacciones entre los agentes no son deterministas, sino que dependen del azar.

Por su parte, **Rycroft (2005)** y **Burch (2006)** analizan cómo los modelos matemáticos son clave para entender el comportamiento de las multitudes en espacios grandes, como estaciones de metro o estadios. Herramientas como *AnyLogic* permiten crear estos modelos de forma visual, ajustando parámetros como el **ancho de las puertas de salida**, las **señales de evacuación** o las condiciones de hacinamiento, y estudiar cómo las multitudes reaccionan ante estos factores.

1.5.5. Validación y Verificación de Modelos de Simulación

La *validación* y *verificación* de los modelos de simulación son fundamentales para garantizar que las simulaciones reflejen con precisión la realidad. *Verificación* asegura que el modelo esté implementado correctamente, mientras que *validación* evalúa si el modelo es adecuado para representar el sistema real. Este proceso puede incluir comparar los resultados de las simulaciones con *datos históricos de evacuaciones* o realizar *experimentos controlados* para recopilar datos en tiempo real sobre el comportamiento de las multitudes **(Gómez, 2020)**.

Hernández (2015) explica que la **validación de modelos de evacuación** se realiza comparando los datos obtenidos en evacuaciones anteriores o simulando diferentes escenarios para ver cómo el modelo responde a cambios en las condiciones. Este proceso es clave para asegurar que las simulaciones sean útiles y precisas en situaciones de emergencia.

Capítulo 2

Algoritmo A*: De la imagen al modelo computacional

En este capítulo se integran los procesos de construcción del entorno y de búsqueda de rutas, los cuales originalmente se abordaban por separado. La intención es mostrar de forma continua cómo una imagen digital puede convertirse en un modelo computacional que permite aplicar el algoritmo A* para la simulación de trayectorias de evacuación. Esta integración facilita comprender el flujo completo de la metodología, desde la representación espacial hasta la obtención de rutas óptimas.

La comprensión del comportamiento colectivo en situaciones de evacuación parte de la capacidad de representar, de forma matemática, las decisiones individuales y las interacciones entre personas dentro de un entorno físico. En el capítulo anterior se exploraron las bases teóricas que permiten construir ese tipo de modelos: desde la descripción continua del flujo de multitudes hasta los enfoques discretos basados en agentes, donde cada individuo responde a su entorno y a la influencia del grupo. Estos marcos conceptuales mostraron cómo la dinámica de una multitud puede entenderse como un sistema complejo, donde la organización global emerge de acciones locales guiadas por reglas simples.

A partir de esta visión, el siguiente paso es llevar esas ideas al terreno práctico de la simulación computacional. Aquí, el espacio físico deja de ser una abstracción y se convierte en un conjunto de datos que puede analizarse, manipularse y recorrer. En este punto, las **imágenes digitales** juegan un papel clave: permiten transformar planos o mapas en estructuras numéricas que representan el entorno de manera precisa.

De este modo, la simulación de multitudes pasa del modelo teórico al modelo visual, donde algoritmos como **A*** pueden emplearse para calcular rutas de evacuación y estudiar cómo las personas se desplazarían en escenarios reales. Este cambio marca el inicio de una etapa más aplicada del estudio, donde la matemática, la computación y la observación del entorno se integran en un solo proceso de modelación. El estudio y la gestión de las rutas de evacuación constituyen una de las áreas más críticas en la seguridad humana, tanto en espacios cerrados como abiertos.

La complejidad arquitectónica de los edificios, la dinámica de la movilidad humana, y las condiciones cambiantes durante una emergencia hacen que diseñar y analizar rutas de escape eficaces sea un desafío multidisciplinario que requiere herramientas matemáticas, computacionales y de representación espacial robustas y precisas **(González, 2018)**. En este contexto, el uso de imágenes digitales ha emergido como una solución innovadora y efectiva para la modela-

ción matemática de rutas de evacuación. Las imágenes digitales permiten capturar de manera fidedigna las características físicas del entorno y transformarlas en estructuras numéricas aptas para simulación y análisis matemático.

Este enfoque combina las ventajas del procesamiento de imágenes, la discretización espacial natural, y la posibilidad de integrar información dinámica para construir modelos de evacuación que sean tanto precisos como adaptativos **(Boyat, 2015)**. La presente investigación tiene como objetivo profundizar en la fundamentación matemática del uso de imágenes digitales, sus técnicas asociadas, ventajas comparativas, y aplicaciones prácticas en la simulación y optimización de rutas de evacuación. Se ofrece además una metodología detallada para la conversión de planos físicos en modelos computacionales, junto con una revisión crítica de métodos tradicionales y sus limitaciones.

Este trabajo se estructura en varios apartados que incluyen la teoría matemática, la transformación de imágenes en matrices funcionales, el análisis de ventajas, y el desarrollo de casos de uso concretos, culminando con una visión integral del potencial y los retos futuros de esta técnica.

2.1. Fundamentos matemáticos y computacionales de las imágenes digitales

2.1.1. Definición formal de imagen digital

Matemáticamente, una imagen digital se define como una función discreta:

$$f : \Omega \subset \mathbb{Z}^2 \to \mathcal{V}$$

donde Ω representa el conjunto finito de puntos discretos (píxeles) organizados en una malla bidimensional, y $\mathcal{V}$ es el conjunto de valores que pueden tomar los píxeles. En una imagen en escala de grises, $\mathcal{V} = \{0, 1, \ldots, 255\}$, correspondiente a niveles de intensidad lumínica, mientras que en imágenes en color se extiende a tuplas RGB o espacios más complejos como CMYK o HSV **(González, 2015)**.

- **RGB (Rojo, Verde, Azul)**: modelo de color basado en la luz, usado en pantallas.
- **CMYK (Cian, Magenta, Amarillo, Negro)**: modelo de color basado en tintas, usado en impresión.
- **HSV (Tono, Saturación, Valor)**: modelo que describe colores según su matiz, intensidad y brillo.

Este concepto discreto se diferencia de la representación continua tradicional en la geometría, pero ofrece ventajas para el procesamiento computacional y la modelación numérica, debido a que facilita la implementación de algoritmos iterativos y métodos numéricos con dominio claramente definido.

Ejemplo práctico de representación visual bidimensional

Para ilustrar estos conceptos, se presenta un ejemplo en Python donde se carga una imagen en escala de grises y se representa como una malla bidimensional. Cada celda de la malla corresponde a un píxel de la imagen, mostrando claramente las zonas transitables y los obstáculos.

El resultado del código del apéndice A, se muestra en la Figura 2.1. En ella, la imagen binaria se representa mediante una malla bidimensional superpuesta, donde cada celda corresponde a un píxel. Esta cuadrícula facilita la identificación de las áreas transitables, asociadas a los píxeles blancos, así como de los obstáculos, representados por los píxeles negros.

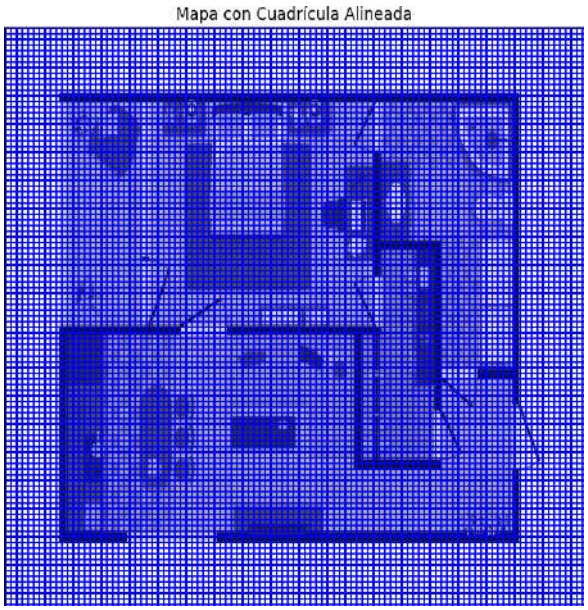


Figura 2.1: Visualización bidimensional de una imagen como malla, mostrando las celdas transitables y los obstáculos.

2.2. Discretización espacial y modelación matemática basada en imágenes

2.2.1. La discretización implícita en las imágenes digitales

La discretización del espacio es un paso fundamental para cualquier modelación matemática computacional. En el caso de imágenes digitales, esta discretización es inherente a la propia naturaleza del dato: la imagen está compuesta por una grilla regular de píxeles, lo que facilita la subdivisión del espacio en celdas manejables **(González, 2018)**.

Esta propiedad elimina la necesidad de diseñar mallas arbitrarias, que en métodos tradicionales suelen ser complejas y demandantes. Cada píxel representa un elemento elemental del entorno, con dimensiones físicas definidas a partir de la escala de la imagen.

2.2.2. Asignación de atributos a las celdas discretizadas

La matriz de píxeles puede enriquecerse asignando atributos adicionales a cada celda, como:

- Estado físico (transitable, obstáculo, salida, zona de peligro).

- Propiedades dinámicas (presencia de humo, densidad de personas).
- Costos o probabilidades asociadas al tránsito.

Representación en grafos y redes

A partir de la matriz de atributos se puede construir un grafo donde cada nodo corresponde a un píxel o conjunto de píxeles, y las aristas representan la posibilidad de tránsito. Esta representación es especialmente útil para aplicar algoritmos de búsqueda de rutas óptimas como Dijkstra o A* **(Hart, 1968)**. **Dijkstra**: algoritmo que encuentra la ruta más corta desde un punto de inicio hasta todos los demás puntos en un grafo con pesos no negativos.

2.3. Ventajas del uso de imágenes digitales como mapas de evacuación

El uso de imágenes digitales facilita la obtención de datos precisos, como planos, modelos CAD (diseño asistido por computadora, por sus siglas en inglés) o fotografías satelitales, y permite representar fielmente la geometría real de los espacios. También posibilita la integración de información dinámica en tiempo real, es escalable a diferentes tamaños de áreas y versátil para diversos escenarios, como edificios, eventos masivos o complejos industriales.

2.4. Metodología para la transformación de imágenes en modelos funcionales

La metodología propuesta comienza con la identificación de las imágenes que representarán el entorno de estudio, las cuales pueden provenir de diferentes fuentes: planos arquitectónicos en formato impreso o digital, fotografías aéreas obtenidas mediante drones o sistemas de información geográfica, y modelos tridimensionales elaborados mediante **CAD**.

En el caso particular de los modelos CAD, estos ofrecen una ventaja significativa al proporcionar geometrías precisas, capas organizadas de información y niveles de detalle que pueden ser directamente exportados como imágenes ortogonales o vistas en planta. Esta capacidad permite generar representaciones limpias del entorno, libres de elementos irrelevantes, lo que facilita enormemente el procesamiento posterior.

Una vez seleccionadas las imágenes, se lleva a cabo un preprocesamiento destinado a homogenizar la calidad visual y destacar las estructuras principales. Este proceso incluye la conversión a escala de grises, la reducción de ruido mediante filtros digitales y el ajuste de brillo o contraste para mejorar la separación entre espacios transitables y obstáculos.

El siguiente paso consiste en la segmentación del entorno, en la que se diferencian claramente las zonas accesibles de las áreas que actúan como barreras físicas. A partir de esta segmentación se genera un mapa binario, que posteriormente se transforma en una matriz donde cada celda adquiere atributos específicos de acuerdo con su función dentro del espacio simulado.

Esta matriz se convierte en el insumo principal para la construcción de modelos dinámicos capaces de reproducir el desplazamiento humano. Entre los métodos empleados se encuentran el algoritmo de búsqueda de rutas **A estrella** (A*), los `autómatas celulares`, que representan el espacio mediante celdas con reglas de transición definidas, y los modelos de `simulación`

`basada en agentes`, donde cada entidad actúa de acuerdo con su percepción y sus objetivos locales.

Finalmente, se ejecutan simulaciones que permiten verificar el desempeño del modelo y realizar ajustes necesarios para garantizar que la representación del entorno sea precisa y funcional. Esta etapa de validación es crucial para asegurar que el sistema pueda emplearse en análisis de seguridad, optimización de rutas o evaluación de escenarios de evacuación.

2.5. Aplicaciones y casos de estudio

El enfoque basado en imágenes digitales y modelos CAD ha sido aplicado en diversos contextos que van desde edificios públicos y centros comerciales hasta instalaciones industriales que requieren protocolos rigurosos de seguridad. También ha demostrado utilidad en la organización de eventos con gran afluencia de personas, en estudios de movilidad urbana y en plataformas de simulación en tiempo real que integran datos de sensores ambientales o de flujo peatonal.

En todos estos casos, la representación derivada de imágenes facilita la exploración de alternativas, la identificación de áreas críticas y la mejora de los procedimientos de evacuación bajo diferentes condiciones.

2.6. Esquema metodológico mediante diagrama de flujo

Para complementar la explicación de este capítulo, se incluye un diagrama de flujo que resume el proceso seguido para transformar una imagen digital en un modelo funcional de evacuación con aplicación del algoritmo A*. Este esquema permite visualizar de manera clara las etapas principales, desde la obtención de la imagen hasta la simulación computacional.

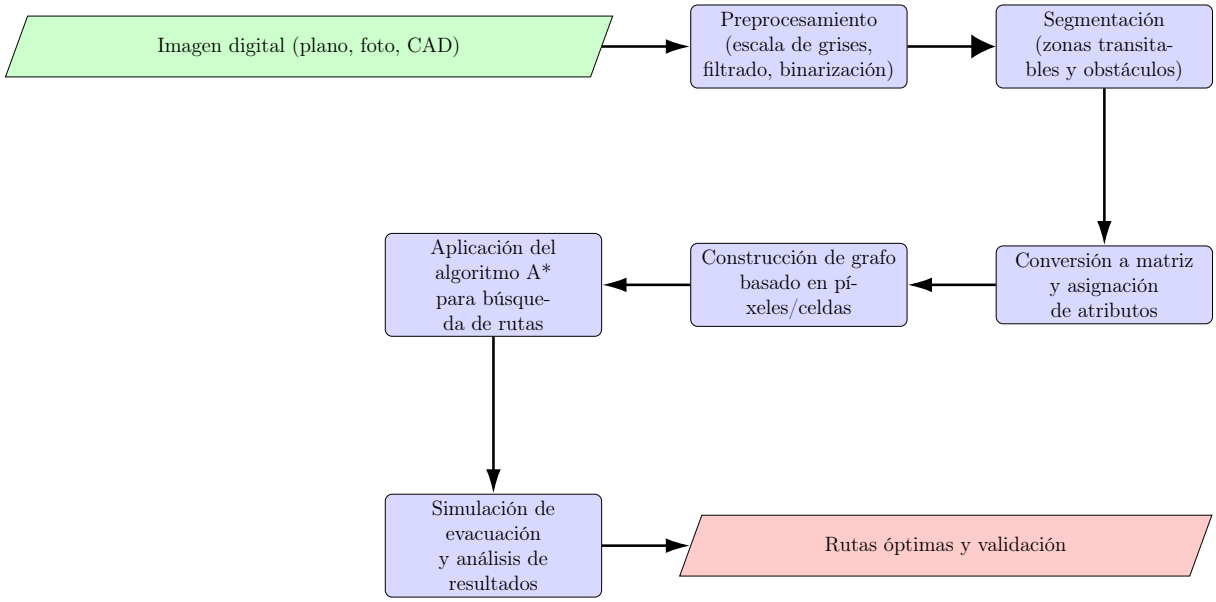


Figura 2.2: Diagrama de flujo en forma de serpiente del proceso de simulación de evacuaciones a partir de imágenes digitales y aplicación del algoritmo A*.

2.7. *Mapas Aleatorios*

La transformación de imágenes en escala de grises en mapas binarios suele emplearse como un recurso accesible para representar entornos dentro de sistemas de navegación. En este procedimiento, las zonas claras se interpretan como espacios transitables y las zonas oscuras como obstáculos, lo que permite traducir planos arquitectónicos, mapas urbanos o cualquier representación visual en una estructura discreta adecuada para métodos de búsqueda de rutas.

No obstante, cuando se pretende ampliar el análisis y explorar situaciones con distintos niveles de complejidad espacial, es común recurrir a la generación de mapas aleatorios. Este enfoque permite construir escenarios variados sin depender de un plano previo, lo cual resulta útil para examinar cómo se comporta el algoritmo **A estrella** (A*) en configuraciones con distribuciones diversas de obstáculos, pasajes más o menos definidos o regiones con distinta estructura geométrica.

Su principal ventaja radica en que posibilita ajustar parámetros de manera directa y observar el efecto de estos cambios en el proceso de búsqueda.

Si bien estos mapas no están diseñados para replicar la precisión de un entorno real, sí ofrecen un espacio de experimentación donde pueden identificarse patrones generales en las trayectorias calculadas. En algunas configuraciones aparecen zonas con obstáculos dispersos, corredores angostos o áreas más amplias; estos elementos no son planteados como dificultades para el algoritmo, sino como condiciones que permiten analizar cómo organiza la búsqueda según la disposición del entorno digital.

Este tipo de escenarios resulta útil cuando se comparan heurísticas, se prueban variaciones en la implementación o se busca comprender cómo influyen ciertas características espaciales en la estructura final de la ruta obtenida. En ese sentido, los mapas aleatorios no sustituyen a los derivados de imágenes reales, sino que complementan su uso al ofrecer un laboratorio controlado donde es posible reunir observaciones que posteriormente enriquecen el análisis en entornos físicos o arquitectónicos.

2.7.1. Representación matricial del entorno

Al igual que en el caso de los mapas derivados de imágenes, los mapas generados aleatoriamente se representan mediante una matriz binaria M de tamaño $m \times n$. Cada celda de la matriz corresponde a una posición del entorno, la cual puede estar libre o bloqueada por un obstáculo. Matemáticamente, esta representación se define como:

$$M_{ij} = \begin{cases} 0, & \text{si la celda } (i,j) \text{ es transitable,} \\ 1, & \text{si la celda } (i,j) \text{ contiene un obstáculo.} \end{cases}$$

Esta formulación es muy simple, pero constituye el corazón de los sistemas de planificación de rutas. Lo más importante es que esta misma estructura es compatible con diferentes formas de generación de mapas: tanto los que provienen de imágenes como los que se crean de forma aleatoria.

2.7.2. Generación de obstáculos aleatorios

La construcción del entorno artificial inicia con una matriz compuesta únicamente por ceros, lo que equivale a un espacio completamente libre para desplazarse. A partir de ahí se empieza

a introducir complejidad: se agregan obstáculos colocados en posiciones aleatorias dentro del mapa.

Cada obstáculo se representa como un bloque cuadrado cuyo tamaño puede variar, lo que permite generar configuraciones más parecidas a barreras reales y no solo puntos sueltos sin coherencia. En el **Código 2 apéndice** se muestra la implementación completa de este proceso.

El parámetro `cantidad` determina cuántos de estos bloques se insertan en el entorno, por lo que funciona como un control directo de la densidad del mapa. Este ajuste es importante, ya que a mayor densidad de obstáculos, más complicada se vuelve la búsqueda de rutas para el algoritmo.

2.7.3. Función heurística utilizada

El algoritmo A* necesita una función heurística para guiar su exploración hacia la meta. En este trabajo se utiliza la distancia de Manhattan, que es adecuada cuando el agente solo se puede mover en cuatro direcciones (arriba, abajo, izquierda y derecha). Dado una ruta entre el nodo inicial (x_0, y_0) y el nodo actual (x_n, y_n), la función heuristica se define como:

$$h(n) = |x_n - x_g| + |y_n - y_g|$$

donde:

- (x_n, y_n) son las coordenadas del nodo actual.
- (x_g, y_g) son las coordenadas de la meta u objetivo.

Esta medida es consistente y admisible, por lo que garantiza que el algoritmo A* encuentre una ruta óptima si existe. En otras palabras, nunca sobreestima el costo real.

2.7.4. Aplicación del algoritmo A*

El algoritmo A* combina el costo acumulado desde el inicio hasta el nodo actual $g(n)$ con la heurística $h(n)$. Esto se resume en la siguiente función de evaluación:

$$f(n) = g(n) + h(n)$$

Gracias a este equilibrio entre el costo real y la estimación, el algoritmo puede encontrar rutas eficientes evitando expandir nodos innecesarios. Esto resulta especialmente importante cuando el mapa contiene una gran cantidad de obstáculos, ya que la exploración ciega sería muy costosa.

En los mapas generados aleatoriamente, el proceso es exactamente el mismo que en los mapas derivados de imágenes: se define un nodo de inicio S, un nodo objetivo G, y A* se encarga de calcular la ruta más corta evitando los obstáculos.

2.7.5. Visualización de resultados

Una de las ventajas más notorias de este enfoque es la posibilidad de visualizar tanto el entorno generado como el proceso de exploración del algoritmo. En la Figura 2.3 se aprecia un ejemplo en el que los obstáculos se muestran en color negro, las celdas exploradas en azul, el camino encontrado en rojo, la posición inicial y final en verde.

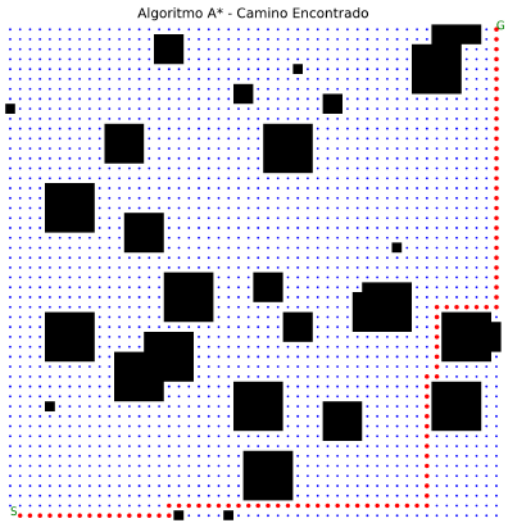


Figura 2.3: Mapa generado aleatoriamente con obstáculos y trayectoria calculada mediante A*.

Este tipo de visualización no solo ayuda a comprender mejor el funcionamiento del algoritmo, sino que también sirve como una herramienta didáctica para mostrar el proceso de exploración y la forma en que se evita atravesar obstáculos.

2.7.6. Comparación con mapas basados en imágenes

Es importante destacar que, aunque el procedimiento para aplicar A* es idéntico en ambos casos, existen diferencias fundamentales entre los mapas derivados de imágenes y los mapas generados aleatoriamente:

- Los mapas basados en imágenes representan entornos reales y permiten realizar simulaciones con mayor fidelidad al mundo físico **(Ruiz, 2018)**.
- Los mapas aleatorios, por su parte, son útiles para experimentar y poner a prueba el rendimiento del algoritmo en diferentes condiciones de densidad y distribución de obstáculos **(Castillo, 2019)**.

Por lo tanto, ambos enfoques son complementarios: primero se puede validar el algoritmo en entornos aleatorios y después trasladarlo a mapas reales para realizar simulaciones más cercanas a la práctica.

2.7.7. Diagrama de flujo del algoritmo A*

El algoritmo A* es uno de los métodos más utilizados para la planificación de rutas óptimas en entornos discretos, como mapas generados a partir de imágenes o matrices aleatorias. Su fundamento combina el costo real de llegar a un nodo con una estimación heurística de la distancia al objetivo, lo que permite encontrar caminos eficientes sin explorar nodos innecesarios **(Hart et al., 1968)**.

Para representar visualmente su funcionamiento, se puede elaborar un diagrama de flujo que sintetice las etapas principales: inicialización, exploración de nodos, evaluación de la heurística y construcción del camino final.

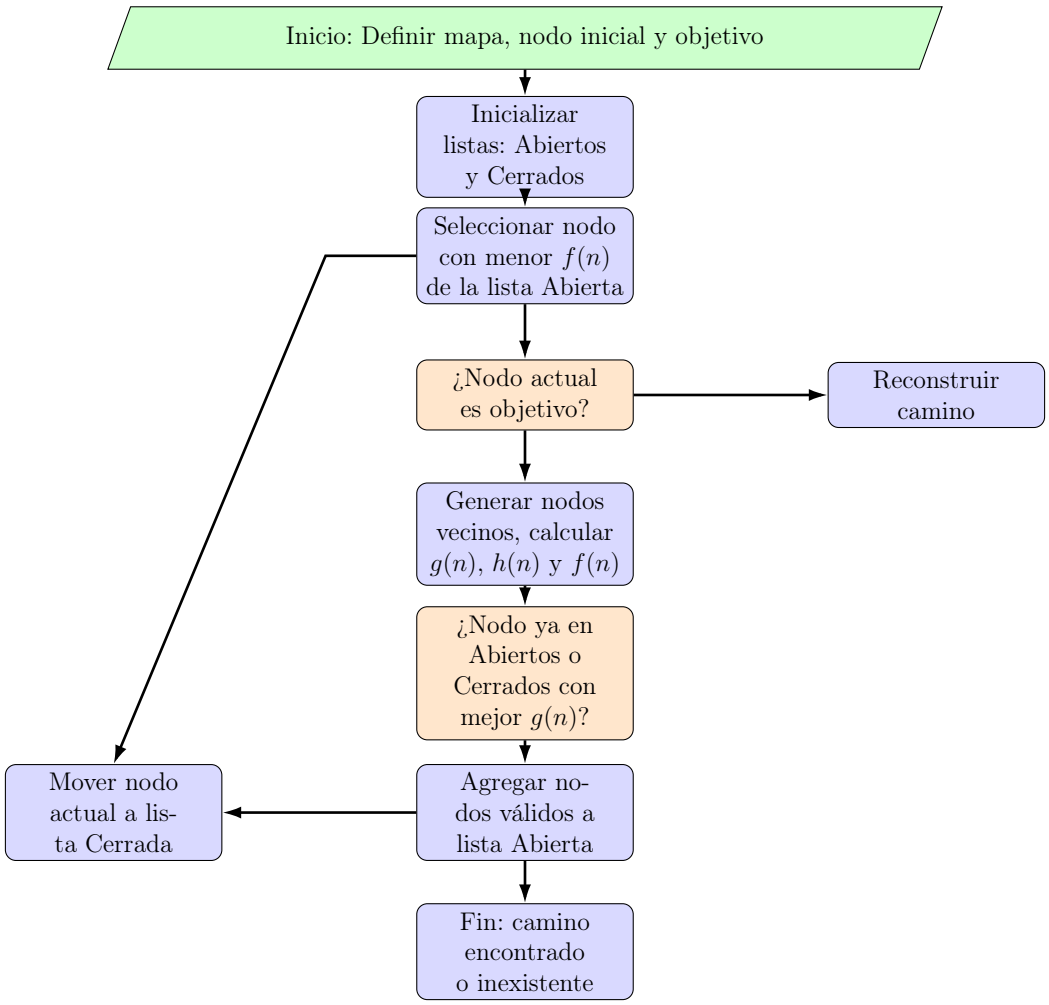


Figura 2.4: Diagrama de flujo del algoritmo A* con todos los nodos cuadrados.

Explicación del diagrama

El diagrama sintetiza los pasos fundamentales de A*:

- **Inicio e inicialización:** Se define el mapa, el nodo de inicio y el objetivo. Se crean las listas de nodos abiertos (para explorar) y cerrados (ya evaluados).
- **Selección del nodo:** De los nodos abiertos, se elige aquel con el menor valor de $f(n)$.
- **Verificación del objetivo:** Si el nodo seleccionado es el objetivo, se reconstruye el camino desde el nodo final hasta el inicio.
- **Expansión de nodos vecinos:** Se generan los nodos adyacentes, calculando los costos acumulados $g(n)$ y la heurística $h(n)$, sumando ambos para obtener $f(n)$.
- **Actualización de listas:** Los nodos válidos se agregan a la lista abierta y el nodo actual se mueve a la lista cerrada.
- **Repetición:** Se continúa el proceso hasta encontrar la ruta óptima o determinar que no existe.
- **Nodo abierto:** Son los nodos que ya fueron descubiertos o detectados como posibles candidatos, pero que aún no han sido procesados o expandidos por el algoritmo. Están en espera de ser evaluados.

- **Nodo cerrado:** Son los nodos que ya fueron expandidos, es decir, ya se exploraron sus vecinos y se calculó su costo. Una vez que un nodo pasa a la lista cerrada, el algoritmo no lo vuelve a revisar para evitar ciclos y mejorar la eficiencia.

2.7.8. Resultados posibles

Dependiendo de la configuración del mapa y la densidad de obstáculos, el algoritmo puede producir:

1. **Camino óptimo único:** Cuando existe un solo camino eficiente hacia la meta.
2. **Múltiples caminos equivalentes:** Cuando varias rutas tienen el mismo costo, A* puede elegir cualquiera según el orden de expansión.
3. **No hay camino:** En mapas muy densos o con obstáculos que bloquean completamente la ruta, el algoritmo detecta la imposibilidad de alcanzar la meta.

El método heurístico es mejor que una búsqueda exhaustiva porque esta última, al intentar explorar todas las rutas posibles, se enfrenta a un problema NP-completo, con complejidad exponencial. A* usa heurísticas, es decir, "pistas" o estimaciones de lo lejos que queda la meta, para guiar la búsqueda y así evita tener que revisarlo todo, que sería demasiado lento.

2.8. Aplicación del código de búsqueda de camino

A partir del procesamiento de las imágenes de origen se obtiene un mapa binario que distingue con claridad qué zonas pueden recorrerse y cuáles funcionan como obstáculos. Esta traducción de la información visual a una representación computacional facilita que los agentes se muevan de forma coherente con la estructura del entorno, y sirve como base para analizar y planificar sus trayectorias. De esta manera, el mapa resultante conserva la forma real del espacio y permite que los algoritmos trabajen con datos consistentes **(Hernández, 2015; Ruiz, 2017)**.

Con este mapa como referencia, el algoritmo A* se convierte en una herramienta práctica para determinar rutas eficientes, combinando el costo recorrido hasta el momento con la cercanía al objetivo. Esto permite generar trayectorias que esquivan obstáculos y que mantienen un comportamiento lógico para los agentes al desplazarse.

La integración entre un mapa bien definido y un método de búsqueda robusto abre la puerta a estudiar con mayor detalle cómo se comportan los agentes en entornos complejos y con distintas configuraciones.

2.8.1. Antecedentes

La necesidad de encontrar rutas dentro de un entorno aparece en muchas situaciones del día a día y también en distintos trabajos relacionados con computación. Por ejemplo, un robot que debe moverse dentro de una fábrica, o un personaje de un videojuego que necesita avanzar sin chocar con obstáculos. En todos estos casos, el objetivo es básicamente el mismo: decidir por dónde avanzar para llegar a un punto específico (meta).

Con el tiempo se han creado varios métodos para resolver este tipo de problemas. Entre ellos, el `algoritmo A*` se usa bastante porque combina la distancia recorrida con una estimación de

lo que falta por avanzar. Esto le permite revisar primero los caminos que parecen más convenientes, sin dejar de comparar el resto del entorno.

No se trata de que sea perfecto, sino que suele ofrecer un equilibrio razonable entre buscar bien y no gastar demasiado tiempo en hacerlo. Hay que tener claro que buscar la ruta óptima con el tiempo mínimo casi nunca se hace, porque llevaría muchísimo tiempo y el problema puede volverse exponencialmente complejo.

Cuando se trabaja con varios agentes al mismo tiempo, como en simulaciones de evacuación o desplazamiento dentro de un edificio, planificar rutas se vuelve todavía más importante. Cada agente necesita saber por dónde avanzar, evitar obstáculos y moverse sin complicarse con los demás. Por eso, además de ser un proceso técnico, también sirve para observar cómo se comporta un grupo cuando comparte un mismo espacio.

2.8.2. Origen del nombre del algoritmo A*

El algoritmo A* recibe este nombre porque, en sus primeras versiones, existían algoritmos llamados A1 y A2, desarrollados por **Peter E. Hart**, **Nils J. Nilsson** y **Bertram Raphael**, investigadores del `Stanford Research Institute`. Cuando se demostró que A2 era óptimo, se decidió llamarlo A* para indicar que era una versión especial o mejorada.

Más tarde, uno de sus creadores explicó que eligieron el asterisco porque en estadística y matemáticas este símbolo suele usarse para señalar una versión óptima o destacada. De esta forma, el nombre A* refleja que el algoritmo busca siempre la solución más eficiente **(Davis, 2020)**.

Esto es lo que los autores escribieron en su artículo original de 1968: .ᴬ Formal Basis for the Heuristic Determination of Minimum Cost Paths", publicado en IEEE Transactions on Systems Science and Cybernetics. En ese trabajo se presenta formalmente el algoritmo y se demuestra su optimalidad bajo ciertas condiciones.

2.8.3. Carga y procesamiento de la imagen

Una forma práctica de generar un entorno sin tener que dibujarlo a mano es usar una imagen como base. Cuando la imagen está en escala de grises, ya trae cierta información que puede aprovecharse: las zonas oscuras suelen interpretarse como muros o áreas bloqueadas, mientras que las zonas claras pueden entenderse como espacios por donde se puede avanzar **(Hernández, 2015; Ruiz, 2017)**.

El proceso empieza convirtiendo la imagen en una matriz. Cada píxel tiene un valor entre 0 y 255, donde 0 corresponde al negro y 255 al blanco. Esta escala facilita decidir qué partes del entorno serán obstáculos y cuáles se considerarán libres.

Para trabajar con la imagen se usa una biblioteca llamada `Python Imaging Library` (PIL), la cual permite abrir, visualizar y modificar imágenes en distintos formatos. Al usarla, la imagen se convierte en un arreglo bidimensional que el programa puede manipular directamente.

Una vez que se obtiene esta matriz, es importante que el modelo mantenga la misma estructura que la imagen original. Esto implica cuidar la orientación, la resolución y cualquier detalle visual que pueda cambiar la forma en que el entorno es interpretado dentro del código.

2.8.4. Generación del mapa binario

Una vez que se tiene la matriz con los valores de cada píxel, lo que sigue es transformarla en un mapa binario. La idea es dejar el entorno lo más simple posible: cada posición de la matriz solo debe indicar si se puede pasar (1) o si está bloqueada (0).

Para hacer esta conversión se usa un `umbral`. Por ejemplo, si se elige un valor de 120, entonces todo lo que esté por debajo de ese número se toma como obstáculo, y lo que esté por encima se considera zona libre. Este paso requiere un poco de cuidado, porque si el umbral no se elige bien, podría dejar cerrados caminos que sí existen o abrir rutas por donde realmente debería haber un muro. Se podría suponer que, con una calibración previa y varias pruebas de ensayo y error, es posible encontrar un umbral que funcione adecuadamente sin cometer errores graves.

Al final, lo importante es que el mapa binario conserve la forma general del lugar: pasillos delgados, esquinas, divisiones, intersecciones, etcétera. Si esta representación queda bien, entonces el algoritmo A* tiene una base sólida para explorar el espacio y buscar rutas que tengan sentido dentro del entorno.

2.8.5. Heurística utilizada

El algoritmo A* no solo necesita saber qué partes del mapa están libres, también debe tener alguna idea de qué tan lejos está del destino mientras avanza. Para eso usa algo llamado **"heurística"**, que básicamente es una forma rápida de calcular qué tan **''`prometedor`''** es un punto antes de explorarlo.

En este caso se usa **la distancia Manhattan**. Esta se obtiene comparando las coordenadas del punto actual con las del destino y sumando las diferencias absolutas de las coordenadas en un sistema referenciado. Es decir, se cuentan los pasos hacia arriba/abajo y derecha/izquierda, sin considerar diagonales. Este tipo de cálculo funciona bien cuando los movimientos están limitados a cuatro direcciones.

La ventaja de esta heurística es que es muy fácil de calcular y no requiere operaciones complicadas. Eso ayuda cuando el algoritmo debe revisar muchos puntos en poco tiempo. Además, su estimación suele estar por debajo del costo real del camino, lo que hace que A* pueda llegar a soluciones correctas sin perderse demasiado en el proceso.

2.8.6. Implementación del algoritmo A*

En la implementación de A*, se trabaja con tres funciones principales:

- El costo real desde el inicio hasta un nodo dado, conocido como $g(n)$.
- La estimación de costo restante desde el nodo actual hasta el destino, conocida como $h(n)$.
- La suma de ambas, llamada $f(n) = g(n) + h(n)$, que representa el costo total estimado.

Durante la ejecución, el algoritmo selecciona siempre el nodo con el menor valor de $f(n)$ y explora sus vecinos. Si encuentra que un vecino es más eficiente que los caminos anteriores, actualiza su información y continúa el proceso.

Esta lógica permite ir construyendo progresivamente el camino más corto desde el origen hasta el destino. Una vez alcanzado el destino, se retrocede a través de los nodos padres para reconstruir la ruta exacta.

2.8.7. Validación de coordenadas

Antes de poner en marcha la búsqueda, se debe verificar que tanto el punto de inicio como el punto de destino estén bien definidos. Esto significa que deben:

- Estar dentro de los límites de la matriz (x_1, y_1), donde:
  $-L_1 \leq x_1 \leq L_1$ y $-L_2 \leq y_1 \leq L_2$.
  Aquí, L_1 y L_2 son las mitades de las dimensiones de la matriz en horizontal y vertical respectivamente (por ejemplo, una matriz de tamaño $(2L_1 + 1) \times (2L_2 + 1)$).

- Corresponder a celdas transitables, es decir:
  $I(x_1, y_1) \leq 120$, donde I representa la intensidad o valor del píxel.

Si alguno de estos puntos no cumple estas condiciones, el algoritmo simplemente no podrá encontrar un camino válido, ya que partiría desde una zona bloqueada o intentaría llegar a un destino inalcanzable. Por eso, incluir esta validación al principio evita errores de ejecución y garantiza una simulación coherente.

2.8.8. Visualización y análisis

Uno de los aspectos más importantes del trabajo con algoritmos de búsqueda es poder visualizar los resultados. Esto no solo ayuda a entender cómo funciona el algoritmo, sino que también permite detectar errores, ajustar parámetros, o incluso mejorar la presentación del entorno.

En la visualización se representa:

- El entorno original, en blanco y negro.
- Los nodos que el algoritmo exploró, en un color suave (por ejemplo, azul claro).
- El camino encontrado, en un color destacado como rojo o verde.
- Los puntos de inicio y fin, con símbolos especiales (círculo, estrella, etc.).

La parte de la visualización consiste en juntar el mapa binarizado con la ruta que encontró el algoritmo y con los puntos que fue revisando durante la búsqueda. Todo se muestra con el mismo tamaño que tenía la imagen original, sin hacer ampliaciones ni reducciones. Esto ayuda a que el recorrido pueda interpretarse tal cual aparece en el entorno, sin distorsiones que confundan.

Ver el resultado de esta manera permite entender mejor cómo se movió un punto dentro del mapa: por dónde pasó, qué obstáculos evitó y en qué zonas tardó más en decidirse. También es una forma práctica de explicar el proceso a alguien que no esté familiarizado con el código, ya que la imagen final muestra de forma visual lo que pasó en cada paso del recorrido.

A continuación, se incluye una imagen de ejemplo generada en una simulación:

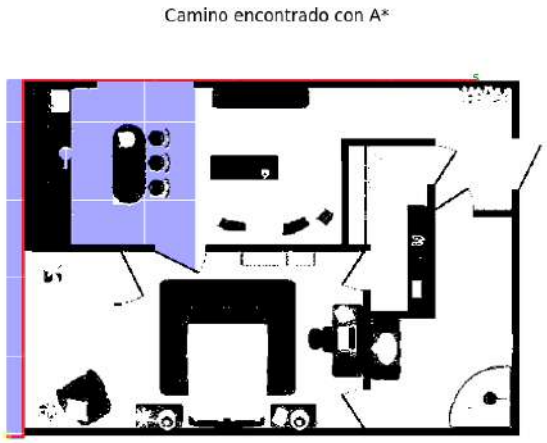


Figura 2.5: Visualización unificada del algoritmo A* sobre un entorno generado a partir de una imagen binarizada. El camino encontrado está marcado en rojo, los nodos explorados en azul claro, y los obstáculos en negro, todo presentado a tamaño real para facilitar la interpretación.

Los resultados mostrados corresponden al código presentado en el Apéndice C. Este código implementa el algoritmo A* y permitió cumplir con la misión de encontrar la ruta óptima en el entorno planteado.

2.8.9. Aplicación en simulación de multitudes

Además de usarse para encontrar el camino de un solo agente, estos algoritmos también se aplican mucho en simulaciones donde participan varias personas o elementos al mismo tiempo. En este tipo de escenarios se representan grupos grandes; `a veces decenas o incluso cientos de agentes`, que deben moverse dentro del mismo espacio sin chocar entre ellos, buscando rutas razonables y reaccionando a lo que ocurre alrededor, como zonas con fuego, salidas disponibles o áreas donde empieza a formarse una congestión.

Cada agente puede tener un destino distinto y calcular su propio camino usando A*. A esto se le pueden añadir modelos de comportamiento que permiten simular su intención de avanzar, cómo cambian su trayectoria si algo bloquea su paso o cómo se adaptan cuando el entorno se vuelve más complicado.

Por ejemplo, un modelo común es el de fuerza deseada, que se puede expresar como:

$$\vec{F}_{\text{deseo}} = m \cdot \frac{\vec{v}_0 - \vec{v}}{\tau}$$

donde:

- $\vec{v}_0$: la velocidad deseada del agente (en dirección a su objetivo),
- $\vec{v}$: la velocidad actual,
- τ: tiempo de adaptación del movimiento,
- m: masa del agente, que normalmente se toma como 1.

Este modelo permite simular cómo los agentes ajustan su movimiento hacia su meta, evitando choques, deteniéndose si es necesario, o acelerando en momentos clave. Al combinarse con el algoritmo A*, se obtiene una planificación realista del desplazamiento individual, complementada con comportamiento grupal.

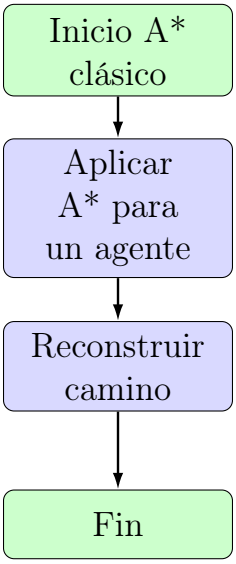


Diagrama de flujo del algoritmo A* clásico para un solo agente, en el cual se representa de manera gráfica el funcionamiento del algoritmo A* previamente explicado, tal como se muestra en la Figura 4.2.

2.9. Algoritmo A* cooperative: Rutas Cooperativas en Mapas Visuales

Continuamos el trabajo usado el algoritmo A* para mover solo a un agente. Pero en la mayoría de los lugares reales no se mueve una sola persona: en un hospital, un aeropuerto o una evacuación, hay muchas personas tratando de llegar a distintos puntos al mismo tiempo.

Lo mismo pasa en simulaciones con robots. Por eso aparece la pregunta: **¿cómo se planifican rutas para varios agentes sin que terminen colisionando entre ellos?**

La idea de la planificación cooperativa es justamente esa: que todos los agentes puedan avanzar desde su origen hasta su destino sin bloquearse, sin estorbarse y tratando de mantener tiempos razonables. Esto complica las cosas, porque ya no basta con evitar los muros del mapa, sino que también hay que considerar el movimiento de los demás agentes, que se convierten en obstáculos dinámicos.

2.9.1. Antecedentes y aplicaciones reales

La planificación para múltiples agentes ha sido investigada en varias áreas. Por ejemplo:

- **Silver (2005)** propuso el algoritmo Cooperative A* (CA*), que añade una dimensión temporal para evitar colisiones.
- **Standley (2010)** desarrolló el algoritmo CBS (Conflict-Based Search), que resuelve conflictos de manera más eficiente.

- **Amazon Robotics** usa este tipo de planificación para que cientos de robots trabajen juntos en almacenes.
- **Helbing y Molnár (1995)** crearon el modelo de fuerzas sociales, usado para simular evacuaciones de multitudes.

Todo esto demuestra que este tipo de planificación no es solo teórica, sino que tiene aplicaciones directas en la vida cotidiana y en la industria.

2.9.2. Uso de mapas visuales como entorno

En este trabajo se vuelve a usar el mismo método que ya había aparecido en capítulos anteriores: partir de una imagen en escala de grises y convertirla en un mapa binario. En este tipo de mapas, las zonas oscuras actúan como obstáculos que los agentes no pueden atravesar, mientras que las zonas claras marcan los caminos por donde sí pueden moverse.

Esta forma de construir el entorno resulta bastante práctica, porque permite tomar planos reales de edificios, escuelas u otros espacios y transformarlos en un mapa digital que después sirve para simular cómo se moverían los agentes dentro de ese lugar.

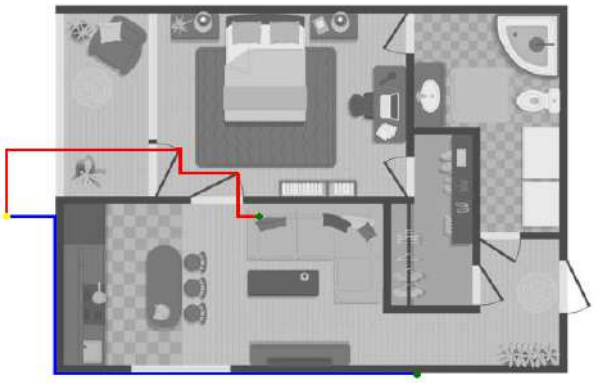


Figura 2.6: Mapa en escala de grises con dos trayectorias sobrepuestas, azul y roja, cada una con su inicio y objetivo, mostrando claramente zonas accesibles y obstáculos. Los resultados visualizados corresponden al código presentado en el Apéndice D.

2.9.3. Extensión temporal en el algoritmo

Cuando trabajamos con varios agentes, no basta con saber por dónde pasará cada uno, sino también cuándo lo hará. Por eso se agrega una tercera dimensión al espacio de búsqueda: el tiempo. Cada posición ahora es (x, y, t). Gracias a esto, cuando un agente planifica su ruta, puede reservar las celdas que usará en cada instante de tiempo. Así, los demás agentes evitarán pasar por esa celda en ese momento. Esto ayuda a evitar colisiones.

2.9.4. Manejo de conflictos

Incluso con reservas de tiempo, pueden aparecer conflictos. Para resolverlos hay diferentes estrategias:

- **Asignar prioridades:** algunos agentes tienen más "derecho"que otros de usar ciertas rutas.
- **Replanificación parcial:** si hay un choque, solo se replanifica parte del camino.
- **CBS:** se planifica cada camino por separado y sólo se corrige cuando se detecta un conflicto.

2.9.5. Ventajas de usar mapas visuales

Usar mapas derivados de imágenes tiene muchas ventajas. Podemos ver visualmente dónde están los agentes, cuáles son los caminos más usados o dónde se forma tráfico. Además, si queremos simular un entorno real, solo necesitamos una imagen del plano.

2.9.6. Comparación entre los algoritmos: A* y CA*

Criterio	**A***	**Cooperative A* (CA*)**
Tipo de algoritmo	Determinista, planea la ruta óptima de un solo agente.	Determinista con planificación cooperativa: considera múltiples agentes y el tiempo.
Función de evaluación	$f(n) = g(n) + h(n)$, donde $g(n)$ es el costo acumulado y $h(n)$ es la heurística hacia la meta.	Similar a A*, pero incorpora la dimensión temporal para evitar conflictos entre agentes.
Consideración de otros agentes	No toma en cuenta la posición ni la trayectoria de otros agentes.	Cada agente planifica considerando la ocupación de celdas en cada instante, evitando colisiones y bloqueos.
Uso de tiempo	No considera tiempo explícitamente.	Incluye el tiempo como una dimensión adicional para coordinar movimientos de múltiples agentes.
Ventaja principal	Planeación rápida y eficiente para un solo agente.	Reduce colisiones y bloqueos en entornos multiagente, permitiendo planificación cooperativa.
Limitación	No adecuado para múltiples agentes en simultáneo.	Mayor complejidad computacional debido a la coordinación temporal entre agentes.
Aplicación típica	Navegación de un solo robot o agente en un mapa estático.	Coordinación de múltiples robots o agentes en entornos dinámicos o congestionados.

Cuadro 2.1: Comparación entre A* y Cooperative A* (CA*) en planificación de rutas multiagente.

2.9.7. Unión de rutas para recorrer más del mapa

Cuando empezamos a usar el algoritmo, casi todas las pruebas que hacíamos eran muy básicas: un solo agente moviéndose de un punto A a un punto B y listo. Eso nos sirvió para entender la lógica del A* y ver cómo encontraba caminos, pero con el tiempo notamos que eso se quedaba corto.

En muchos casos reales, por ejemplo: `revisar distintas zonas de un edificio`, `hacer un recorrido de vigilancia` o `incluso mover a varias personas durante una evacuación`, no basta con que el agente llegue a un punto y se quede ahí. Lo que realmente hace falta es que pueda seguir avanzando y cubrir más áreas dentro del mismo mapa. Por eso surgió la idea de juntar varias rutas, de modo que el recorrido sea más largo, más continuo y más útil para este tipo de situaciones.

¿Cómo lo hicimos?

Lo que hicimos fue dividir el recorrido total en dos etapas. En la primera etapa, el agente se mueve desde su punto de inicio hasta un objetivo intermedio. Este punto intermedio suele ser una zona representativa del mapa, como su centro o algún punto importante.

Después, sin reiniciar el algoritmo ni crear un nuevo agente, se toma el último punto donde terminó la primera ruta y se utiliza como inicio para una segunda ruta que lleva al agente hacia otro destino, normalmente más alejado o en una esquina del mapa. Con esta estrategia, logramos que el agente recorra mucho más área en el mapa, como si estuviera haciendo dos misiones seguidas, pero de forma continua.

Esta idea se parece a lo que en robótica móvil se conoce como planificación jerárquica, donde una tarea más grande se divide en subtareas que se resuelven una tras otra **(Ruiz Velasco, 2014)**.

- **Primera ruta:** del punto inicial al centro (o punto intermedio).
- **Segunda ruta:** del final de la primera al destino final.

Esto también ayuda a evitar que el agente vuelva a pasar por los mismos lugares, ya que se guarda el camino recorrido y se evitan colisiones si hay otros agentes (usando reservas de tiempo, como vimos en el algoritmo CA*).

Cómo se ve en el código

Para lograr esto, modificamos el código del **apéndice E**. Lo importante aquí fue guardar la posición final del primer camino y usarla como punto de partida del segundo. Además, añadimos una validación extra para asegurarnos de que el nuevo objetivo esté en una zona libre (es decir, que no sea un obstáculo). Si el punto está bloqueado, el programa intenta ajustarlo automáticamente a la celda libre más cercana, para evitar errores durante la ejecución.

También hicimos que ambas rutas se dibujaran en la misma imagen de salida. La primera ruta se muestra en rojo y la segunda en azul. Además, se marcan el punto de inicio (círculo verde) y el punto final (círculo amarillo), para tener una mejor visualización.

Ejemplo de resultado

En la Figura 2.7, se puede ver un ejemplo real. El agente comienza en la parte superior izquierda del mapa. La línea roja muestra la primera parte del recorrido, que va hasta un punto

intermedio cerca del centro del mapa. Desde ahí, sin detenerse, el agente continúa con la segunda parte de la ruta (línea azul), hasta llegar a su destino final en la parte inferior derecha del mapa.

Este tipo de simulaciones es muy útil cuando queremos ver cómo un agente puede desplazarse de manera eficiente en entornos reales y complejos, como un plano de edificio, un hospital o una fábrica. También nos permitió observar el comportamiento del algoritmo cuando se le pide más que simplemente ir de A a B, lo cual refleja mejor los retos de la vida real.

Unir rutas nos abrió la puerta a experimentar con recorridos más largos, y nos hizo pensar en cómo planificar misiones más completas para un solo agente. Si bien esto puede parecer algo sencillo, tiene implicaciones importantes en el diseño de rutas en escenarios reales.

Además, como esta lógica también se puede aplicar a múltiples agentes, podríamos extender este trabajo en el futuro hacia un sistema donde varios agentes recorran rutas múltiples, dividan el trabajo entre ellos y se comuniquen para no chocar. En resumen, esta mejora fue un pequeño paso técnico, pero que mejoró mucho nuestras simulaciones y nos ayudó a ver posibilidades más amplias.

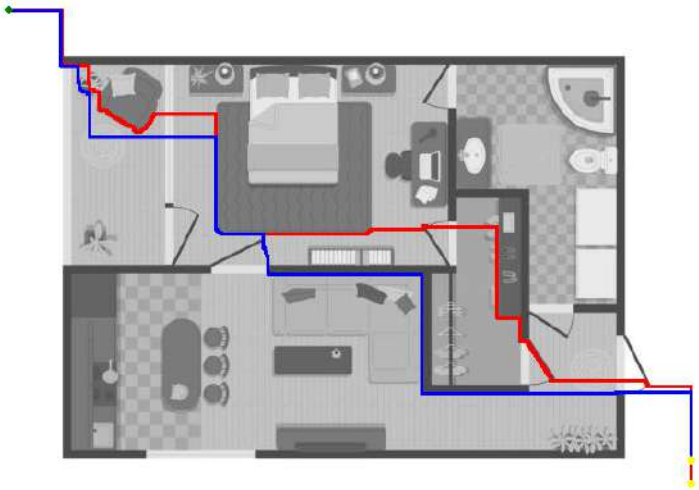


Figura 2.7: El agente recorre dos rutas seguidas: primero va del punto inicial (punto verde) al centro (rojo), y luego continúa hacia una esquina del mapa (azul).

Capítulo 3

Método de Monte Carlo en la planificación de rutas: Explorando la incertidumbre

El método de Monte Carlo es una herramienta fundamental en la modelación matemática para abordar problemas donde la incertidumbre y la variabilidad son importantes. A diferencia de algoritmos deterministas como A*, que buscan una solución única y óptima en un escenario fijo, Monte Carlo genera múltiples escenarios aleatorios para explorar la distribución de posibles resultados. Esto no solo implica generar obstáculos aleatorios en un mapa, sino también poner un grupo de agentes dispersos en un plano y que cada uno siga su propia ruta para encontrar la salida. **(Gómez, 2015)**.

Esta estrategia permite evaluar riesgos, estimar probabilidades y medir la robustez de sistemas complejos. La planificación de rutas usando Monte Carlo, resulta particularmente útil en entornos dinámicos o impredecibles, como evacuaciones, simulación de multitudes o navegación de robots, donde las condiciones pueden cambiar y no siempre es posible conocer todos los factores de manera exacta **(Gómez 2015, Valencia 2018)**.

3.1. Antecedentes históricos y Características del método

El método Monte Carlo surgió en la década de 1940 en el contexto del *Proyecto Manhattan*, cuando Estados Unidos buscaba desarrollar la bomba atómica. Los científicos enfrentaban problemas extremadamente complejos, como la difusión de neutrones en materiales fisionables, que eran prácticamente imposibles de resolver con cálculos deterministas.

`Stanislaw Ulam`, matemático de origen Polaco-Estadounidense, propuso utilizar simulaciones repetidas con números aleatorios para aproximar resultados probabilísticos, convirtiendo la aleatoriedad en una herramienta de modelación. Posteriormente, Ulam colaboró con John von Neumann y Nicholas Metropolis para formalizar la metodología y aplicarla a problemas de física nuclear.

El nombre "Monte Carlo" refleja la esencia del método: la utilización del azar para obtener resultados estadísticos. Hace referencia al famoso casino de Mónaco, símbolo de juegos de probabilidad y eventos impredecibles. Al igual que en la ruleta o los dados, Monte Carlo emplea números aleatorios para simular múltiples escenarios y estimar resultados probabilísticos, permitiendo analizar sistemas complejos donde la incertidumbre es clave.

Desde entonces, esta técnica se ha consolidado como fundamental en diversas disciplinas, permitiendo abordar problemas de alta complejidad e incertidumbre en áreas como física estadística, economía, inteligencia artificial y robótica.

El método Monte Carlo se distingue por varias características clave que explican su eficacia en la simulación de sistemas complejos bajo incertidumbre:

- **Estocástico:** utiliza números aleatorios para generar escenarios variados, lo que permite explorar de manera amplia todas las posibles configuraciones del sistema.
- **Repetitivo:** requiere múltiples simulaciones para que los resultados reflejen de manera confiable la distribución de posibles desenlaces. Cuantos más **ensambles** se realizan, más precisas serán las estimaciones. Un `ensamble` se puede definir como la ejecución de n agentes con *condiciones iniciales* diferentes.
- **Probabilístico:** no busca una única solución, sino que produce una distribución de resultados que permite calcular probabilidades, medias y varianzas, proporcionando una visión estadística del comportamiento del sistema.
- **Flexible:** se puede aplicar en problemas de diversa naturaleza, desde física y finanzas hasta inteligencia artificial y planificación de rutas, adaptándose a distintos tipos de incertidumbre y complejidad.
- **Aproximado:** no garantiza un resultado exacto, pero permite obtener estimaciones útiles y cada vez más precisas a medida que aumenta el número de simulaciones. Esto lo hace práctico cuando los métodos deterministas son inviables.

3.2. Fundamentos del método

El método Monte Carlo se basa en una serie de pasos que permiten simular sistemas complejos y evaluar su comportamiento bajo incertidumbre. A continuación se describen los fundamentos aplicados a la planificación de rutas, explicando la función de cada etapa:

1. **Definir el espacio del problema (Dominio):** se establece el entorno en el que se desarrollará la simulación, especificando mapas, obstáculos, puntos de inicio y destinos. Esta definición es clave, ya que determina las restricciones y posibilidades del movimiento de los agentes, y asegura que las simulaciones reflejen escenarios realistas.
2. **Generar configuraciones aleatorias:** se crean condiciones iniciales diversas de manera estocástica, incluyendo la posición de los agentes, la densidad de multitudes, la presencia de obstáculos móviles o variaciones en la velocidad. Este paso permite explorar múltiples escenarios posibles, capturando la variabilidad que puede presentarse en situaciones reales.

   La mayoría de los lenguajes incluyen generadores pseudoaleatorios con distribución regular en [0,1], aunque existen otros generadores menos usados pero igual de efectivos.
3. **Evaluar cada configuración:** para cada escenario generado, se aplica un algoritmo de búsqueda de rutas, como A* o CA*, que determina trayectorias viables desde los puntos de inicio hasta los destinos. Esta evaluación identifica cómo se comportarían los agentes bajo cada configuración específica y permite medir eficiencia, seguridad y factibilidad de las rutas.

4. **Recolectar métricas:** se registran indicadores relevantes, tales como el tiempo de recorrido, la longitud de las rutas, el número de colisiones evitadas y el nivel de congestión en distintas áreas del mapa. Estas métricas proporcionan datos cuantitativos que permiten comparar escenarios y evaluar el desempeño del sistema bajo distintas condiciones.

5. **Repetir el proceso:** la simulación se ejecuta numerosas veces, variando cada vez las condiciones iniciales y los escenarios generados. La repetición asegura que los resultados sean estadísticamente significativos y representativos de la variabilidad inherente al sistema.

6. **Analizar estadísticamente:** finalmente, los resultados acumulados se analizan para obtener distribuciones de probabilidad, promedios, varianzas y otras medidas estadísticas. Este análisis permite identificar patrones, estimar riesgos, evaluar la robustez de los algoritmos de planificación y tomar decisiones informadas basadas en evidencia cuantitativa.

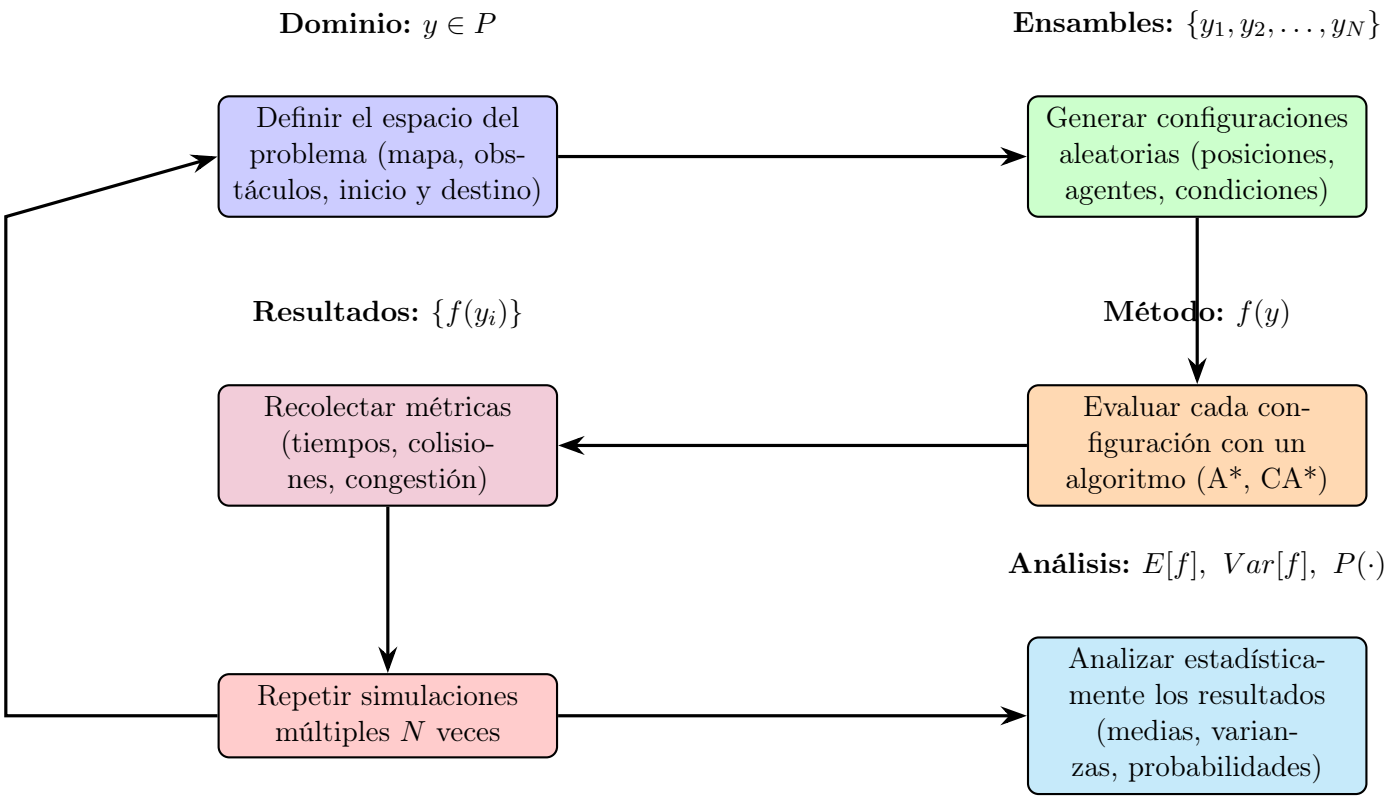


Figura 3.1: Diagrama de flujo del método Monte Carlo aplicado a planificación de rutas con interpretación probabilística.

3.3. Aplicaciones en planificación de rutas

El método Monte Carlo resulta especialmente valioso en la planificación de rutas para agentes móviles y multitudes, ya que permite evaluar escenarios inciertos y dinámicos que los algoritmos deterministas no pueden prever completamente. Algunas aplicaciones clave incluyen:

- **Evaluar probabilidades de éxito:** Monte Carlo permite estimar la probabilidad de que un grupo de agentes complete su recorrido sin colisiones, considerando variaciones en las

posiciones iniciales, velocidades y trayectorias de cada agente. Esto es útil para planificar rutas seguras y predecir riesgos en entornos congestionados.

- **Identificación de zonas críticas de congestión:** al simular múltiples escenarios, es posible detectar las áreas del mapa donde se concentra mayor densidad de agentes o donde ocurren cuellos de botella. Esta información ayuda a mejorar la distribución de rutas y a optimizar el flujo de tránsito de manera preventiva.
- **Evaluación de la robustez de algoritmos:** Monte Carlo permite analizar cómo algoritmos de planificación de rutas, como CA*, se comportan frente a cambios inesperados en las condiciones iniciales o en la configuración del entorno. Esto ayuda a determinar qué tan confiable y adaptable es un algoritmo antes de implementarlo en situaciones reales.

En general, Monte Carlo no sustituye a A* o CA*, sino que actúa como una capa de validación estadística. Su principal aporte es ofrecer información cuantitativa sobre la variabilidad y los posibles riesgos de los escenarios simulados, permitiendo tomar decisiones más informadas y seguras en la planificación de rutas complejas.

3.4. Comparación entre A* y Monte Carlo

Criterio	**A***	**Monte Carlo**
Tipo de enfoque	Determinista: busca un camino óptimo en un escenario definido.	Estocástico: genera múltiples escenarios para estimar comportamientos globales.
Entrada principal	Mapa, punto inicial y destino.	Mapa, número de simulaciones, distribución de condiciones iniciales.
Salida	Ruta única óptima (si existe).	Distribución de rutas, métricas estadísticas (probabilidades, medias, varianza).
Ventaja principal	Garantiza eficiencia en una instancia particular.	Evalúa robustez y confiabilidad frente a la incertidumbre.
Limitación	No maneja incertidumbre ni variabilidad en condiciones iniciales.	Requiere gran número de simulaciones y alto costo computacional.
Aplicación típica	Navegación de un agente en un mapa fijo.	Evaluación de escenarios de evacuación, multitudes o sistemas dinámicos.

Cuadro 3.1: Comparación entre A* y Monte Carlo en planificación de rutas.

3.5. Resultados y Observaciones del Método Monte Carlo

Para evaluar el desempeño del algoritmo Monte Carlo en la planificación de rutas, se implementó un programa en Python que simula el movimiento de un agente desde un punto de inicio hasta un objetivo en un mapa binario. El algoritmo genera múltiples rutas posibles de manera aleatoria, evaluando cada trayectoria según la distancia recorrida y la viabilidad (evitando obstáculos).

3.5.1. Visualización de Resultados

A continuación se presentan ejemplos de resultados obtenidos en un mapa de prueba:

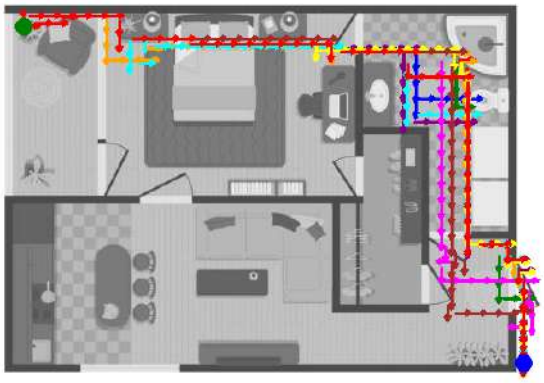


Figura 3.2: Rutas generadas por el algoritmo Monte Carlo sobre un mapa binario. Las líneas representan trayectorias posibles, y los puntos indican el inicio y la meta.

En la figura se observa que el algoritmo explora distintas rutas de manera aleatoria. Cada línea representa una trayectoria completa desde el inicio hasta el objetivo, evitando los obstáculos (zonas oscuras).

3.5.2. Observaciones

Del análisis de los resultados se destacan las siguientes observaciones:

- **Diversidad de rutas:** Monte Carlo permite explorar múltiples trayectorias posibles, lo que es útil en entornos con incertidumbre o dinámicos, donde no se conoce una única ruta óptima de antemano.
- **Variabilidad en la eficiencia:** Algunas rutas generadas son más cortas y directas, mientras que otras presentan desvíos innecesarios. Esto refleja la naturaleza estocástica del método.
- **Robustez frente a obstáculos:** El algoritmo evita de manera efectiva las zonas bloqueadas del mapa, mostrando que la generación aleatoria de trayectorias sigue cumpliendo con las restricciones del entorno.
- **Necesidad de múltiples iteraciones:** Para aumentar la probabilidad de encontrar rutas cercanas al óptimo, se requieren un número elevado de iteraciones o simulaciones, lo que puede aumentar el tiempo de cómputo.

3.5.3. Interpretación de Resultados

Los resultados confirman que Monte Carlo es especialmente útil cuando se desea explorar la incertidumbre en la planificación de rutas y obtener un conjunto de soluciones posibles en lugar de una única trayectoria óptima. Aunque no garantiza encontrar siempre la ruta más corta, permite evaluar distintas alternativas y tomar decisiones basadas en probabilidad y diversidad de caminos.

Capítulo 4

Simulación de Evacuaciones y Normas de Protección Civil

4.1. Introducción a la Evacuación y su Importancia

La simulación de evacuaciones permite predecir cómo se moverán las multitudes a través de un espacio determinado y puede ser utilizada para identificar puntos críticos donde podrían ocurrir accidentes o cuellos de botella. En este contexto, la normativa de protección civil juega un papel fundamental.

En el caso de la Ciudad de México, el Manual de evacuación en situaciones de emergencia **(Protección Civil de la Ciudad de México, 2025)** establece las directrices para realizar evacuaciones seguras, las cuales deben ser integradas en las simulaciones de evacuación. El manual cubre aspectos como las rutas de evacuación, la señalización y el control del flujo de personas en situaciones de pánico.

4.2. Factores que Afectan las Evacuaciones

Existen varios factores que influyen en la efectividad de una evacuación, y estos deben ser considerados en las simulaciones. La arquitectura de un edificio o la disposición de un espacio público son aspectos clave para determinar las rutas de escape más eficientes. Además, el comportamiento humano en situaciones de pánico puede afectar la rapidez y seguridad de la evacuación. Las simulaciones de multitudes deben modelar estos comportamientos, ya que las personas pueden no seguir las rutas más cortas o más seguras debido al estrés o la confusión.

4.3. Modelos de Simulación de Evacuación

En este capítulo, se exploran los diferentes **modelos de simulación** que se utilizan para estudiar las evacuaciones en situaciones de emergencia. Estos modelos permiten simular el comportamiento de las multitudes y evaluar la efectividad de los planes de evacuación sin necesidad de realizar un simulacro físico, lo que resulta muy útil para mejorar la seguridad antes de que ocurra un evento real.

Uno de los enfoques más comunes es el modelo basado en agentes. En este tipo de simulación, se representa a cada persona de la multitud como un ”agente” autónomo que se mueve y toma decisiones dentro del espacio simulado. Cada agente puede tener características y comportamientos únicos, como la velocidad al caminar, la capacidad de tomar decisiones bajo presión

o incluso la tendencia a seguir a otras personas. Los agentes interactúan entre sí y con su entorno, como las paredes, las puertas o las señales de evacuación. Esto permite simular cómo las personas se moverían de forma individual y en grupo en una situación de emergencia.

Este modelo es útil porque refleja de manera realista cómo las multitudes reaccionan ante diferentes circunstancias, como la presencia de obstáculos, el pánico o la falta de visibilidad. Los modelos basados en dinámica de fluidos son otro enfoque que se usa para simular evacuaciones. Básicamente, tratan a la multitud como si fuera un líquido que fluye por el espacio.

La razón por la que no se usa este enfoque en el trabajo es porque, al ver a la gente como un fluido, se pierde la individualidad de cada persona. Es decir, no puedes modelar que alguien decida frenar, cambiar de ruta o reaccionar ante otro agente. Pero ojo, esto no quiere decir que sea malo; de hecho, sería muy útil si se hubiera usado para analizar el flujo general en espacios enormes o cuando hay tanta gente que lo individual ya no importa tanto. Ahí la dinámica de fluidos ayuda a ver el panorama global sin perderse en los detalles de cada persona.

En lugar de simular a cada persona de forma individual, este modelo calcula el flujo general de la multitud, analizando cómo las personas se mueven a través de las rutas disponibles y cómo la densidad de la multitud afecta el movimiento. Este enfoque es útil para simular grandes multitudes y prever posibles puntos de congestión, como cuellos de botella en las salidas de emergencia o pasillos estrechos. Si bien no modela el comportamiento individual de cada persona, ayuda a entender cómo se comportan las multitudes en su conjunto y cómo se pueden optimizar las rutas de evacuación.

4.4. Otros enfoques utilizando matrices aleatorias

Además de los enfoques basados en agentes, MIT OpenCourseWare curso ***Infinite Random Matrix Theory***, **[2004]** menciona que los modelos basados en matrices aleatorias son otra herramienta útil para simular el comportamiento colectivo y el flujo de personas en situaciones de emergencia. Las matrices aleatorias permiten modelar el comportamiento impredecible de las multitudes, ya que se basan en probabilidades para predecir cómo se moverán las personas en función de diferentes factores, como la densidad de la multitud, la presencia de obstáculos o las decisiones de otras personas.

Al utilizar estas matrices, es posible realizar simulaciones precisas y detalladas que ayudan a entender cómo interactúan los individuos y cómo se pueden mejorar los planes de evacuación. Este enfoque es especialmente valioso para predecir cómo las personas se comportarán en situaciones complejas y cómo las decisiones individuales afectan al flujo general de la multitud.

4.5. Normas de Protección Civil y su Relación con las Simulaciones

Las normas de protección civil son reglas y procedimientos establecidos para asegurar que las personas estén protegidas en situaciones de emergencia, como incendios, terremotos o cualquier otra amenaza. Estas normas son fundamentales para que las evacuaciones se realicen de manera ordenada, rápida y segura, minimizando el riesgo de accidentes o pánico.

El Manual de evacuación de la Ciudad de México **(Protección Civil de la Ciudad de México, 2025)** establece varias pautas que deben seguirse durante una evacuación, tales como la identificación clara de las salidas de emergencia, la capacitación del personal encargado de la seguridad y la planificación de rutas alternativas en caso de que algunas salidas queden bloqueadas o sean inaccesibles.

El Manual de evacuación también enfatiza la importancia de la señalización adecuada, de modo que las personas sepan siempre hacia dónde deben dirigirse en caso de evacuación. Además, incluye directrices sobre cómo organizar a las personas de manera eficiente para evitar aglomeraciones y garantizar que todos tengan un camino libre hacia la salida. Este tipo de normativas no solo cubre los aspectos físicos de las rutas de evacuación, sino también la gestión del comportamiento de las personas durante la emergencia, ya que el pánico o la confusión pueden obstaculizar el proceso.

Por otro lado, las simulaciones de evacuación juegan un papel crucial para comprobar si los planes y las normativas realmente funcionan como se espera. Las simulaciones permiten crear un escenario virtual del edificio o espacio donde se desarrollará la evacuación, con los detalles sobre las salidas, las rutas, la disposición de las personas y otros factores importantes. Usando este tipo de herramientas, se puede verificar si las rutas de evacuación propuestas cumplen con los requisitos de seguridad establecidos por las normas de protección civil, como el espacio suficiente para que las personas pasen rápidamente y sin obstáculos.

Por ejemplo, a través de una simulación, se puede analizar si una puerta de emergencia está ubicada en el lugar adecuado o si es lo suficientemente ancha como para permitir que varias personas salgan al mismo tiempo. También se pueden probar diferentes escenarios de evacuación, como un incendio, y verificar si las personas pueden evacuar sin problemas y sin que se produzcan cuellos de botella en las salidas o pasillos. De esta manera, las simulaciones permiten comprobar si las rutas planificadas en los planes de evacuación son realmente efectivas para diferentes situaciones, como un mayor número de personas de lo previsto o condiciones de visibilidad reducida.

Además, las simulaciones también permiten evaluar cómo se comportaría la gente en diferentes situaciones. Las personas suelen seguir ciertas pautas de conductas cuando están bajo presión, como la tendencia a moverse hacia donde otras personas están yendo o la confusión por no saber qué hacer. A través de la simulación, se puede ver si estas reacciones podrían afectar el flujo de personas y si las normativas actuales contemplan estrategias para guiar de manera efectiva a las multitudes hacia las salidas.

Capítulo 5

Análisis Comparativo de las Simulaciones

En este capítulo se presentan los resultados obtenidos a partir de ocho simulaciones estratégicas del modelo multiagente, con algoritmo (Monte Carlo + CA*). En cada caso se modificó la cantidad de agentes, considerando los siguientes escenarios:
Dos simulaciones con imagen ya existente, con: 1, 10, 20, 40, 80 agentes.
Dos simulaciones con imagen generada, con 1, 50, 100, 200 agentes.

La intención es observar cómo cambia el comportamiento del sistema cuando aumenta la cantidad de agentes dentro del entorno. Para cada simulación se analizan cuatro aspectos principales: la distancia Manhattan recorrida, el número de giros realizados, la evolución de las coordenadas en los ejes x y y, y el mapa completo de rutas generadas.

Para mantener claridad en el análisis, todas las simulaciones siguen exactamente la misma estructura gráfica.

5.1. Modelo Teórico de Saturación del Entorno

Además del análisis visual y estadístico de las simulaciones, es importante incorporar un criterio teórico que permita estimar la capacidad máxima del entorno antes de entrar en un estado de congestión severa. Para ello se propone la siguiente expresión de saturación:

$$\text{Capacidad máxima} \approx \frac{\text{Área total del entorno}}{\text{Área promedio por agente}} \tag{5.1}$$

Esta expresión se basa en el concepto de densidad peatonal propuesto por **Fruin (1971)**, donde el número de personas en un espacio depende directamente del área disponible y del espacio que requiere cada individuo para moverse.

- **Área total del entorno**: corresponde al tamaño del mapa (ancho × alto), es decir, el número total de celdas transitables.

- **Área promedio por agente**: Representa el espacio que necesita un agente para desplazarse sin generar congestión. Este valor no solo incluye su tamaño físico, sino también una zona libre mínima alrededor. Por ejemplo, se puede considerar como primera aproximación que una persona o agente ocupa el área de una circunferencia de radio r, donde r es mucho menor que el ancho y el alto del espacio disponible.

- En simulaciones, este espacio puede relacionarse con la distancia que recorren los agentes, por ejemplo:
$$d_M = |x_f - x_i| + |y_f - y_i|$$
donde (x_i, y_i) es el inicio y (x_f, y_f) el destino.

La distancia Manhattan se utiliza con mayor frecuencia en este tipo de simulaciones porque el entorno está representado como una cuadrícula discreta, donde los agentes solo pueden moverse en direcciones ortogonales (arriba, abajo, izquierda y derecha). En este contexto, la distancia Manhattan refleja de manera más realista el costo del desplazamiento, ya que coincide con las restricciones del movimiento permitido.

En cambio, la distancia euclidiana asume trayectorias continuas en línea recta, lo cual no siempre es posible en mapas basados en celdas. Por ello, el uso de la distancia Manhattan simplifica los cálculos y mantiene coherencia con la estructura del entorno, siendo especialmente adecuada para algoritmos como A* en espacios discretos.

En términos simples, la fórmula se interpreta como:

"Cuántos agentes caben en el área disponible considerando el espacio que cada uno necesita para moverse".

Este modelo no define un límite exacto, sino una estimación teórica. Cuando la densidad aumenta demasiado, comienzan a aparecer fenómenos como congestión, reducción de velocidad y formación de cuellos de botella.

5.2. Clasificación de Niveles de Saturación para el analisis de una imagen generada

En el caso de un solo agente, el movimiento es totalmente independiente, lo que permite que el algoritmo A* genere trayectorias óptimas sin ningún tipo de interferencia. Este escenario representa una condición ideal donde el entorno no limita el desplazamiento.

Para 50 agentes, el sistema se mantiene dentro de un nivel moderado de saturación. Aunque las rutas siguen siendo eficientes, comienzan a aparecer ligeras desviaciones y ajustes en las trayectorias debido a la interacción entre agentes. En este punto, ya se pueden identificar caminos preferentes o **corredores de tránsito**.

Cuando se incrementa a 100 agentes, la interacción se vuelve más constante, lo que provoca una mayor variabilidad en las rutas. La saturación es más evidente, especialmente en zonas críticas del mapa, donde los agentes tienden a concentrarse. Esto genera una disminución en la fluidez del movimiento y en la eficiencia global del sistema.

Finalmente, con 200 agentes, el sistema se acerca a un estado de alta saturación. Se observan acumulaciones más marcadas y la formación de cuellos de botella en puntos específicos. Aunque el algoritmo A* continúa guiando a los agentes, la densidad limita su desempeño, provocando trayectorias menos óptimas y un incremento en el tiempo de evacuación.

En conjunto, estos resultados muestran que la saturación no ocurre de manera abrupta, sino que surge de forma gradual conforme aumenta la densidad de agentes. Esto permite entender cómo el sistema evoluciona desde condiciones ideales hasta escenarios reales de evacuación, donde la

interacción entre individuos es fundamental. Cabe señalar que, con ligeros retrasos de tiempo, las trayectorias superpuestas no son malas porque los agentes no se enciman; la superposición espacial solo es problemática cuando ocurre simultáneamente en el mismo punto.

5.2.1. Primera simulación con imagen generada

Simulación con un agente

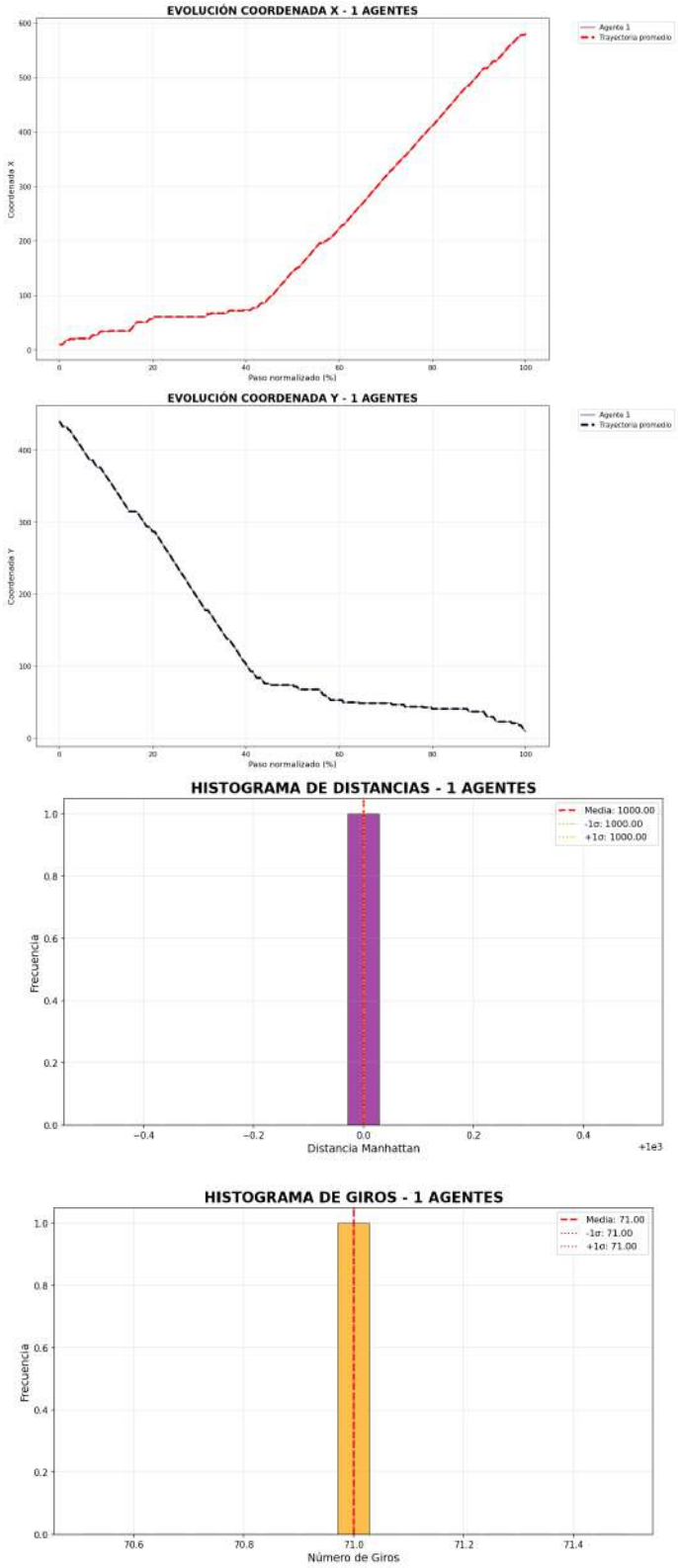
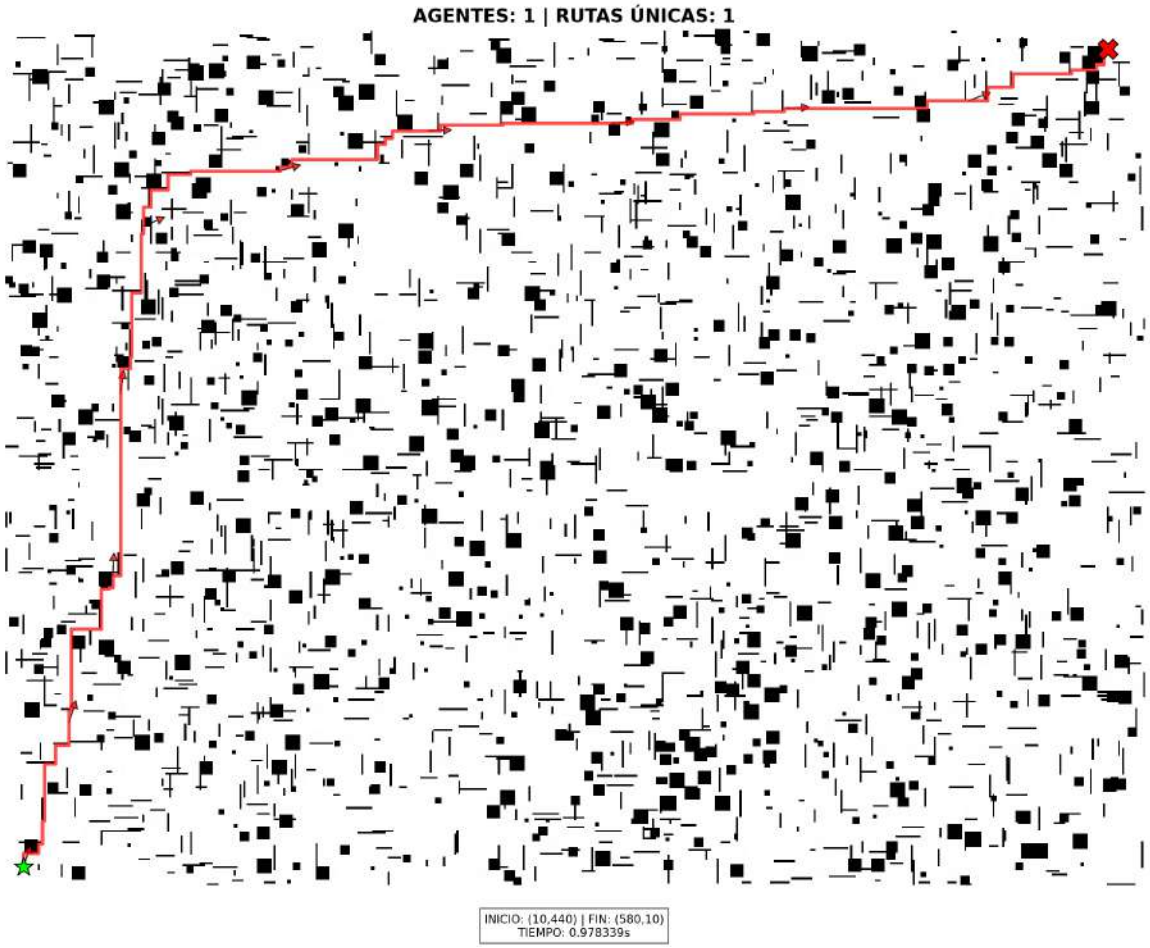


Figura 5.1: Análisis del recorrido: series temporales de coordenadas en eje X y eje Y, distribución de costes Manhattan y estadística de orientación.

Trayectoria óptima y directa. Las gráficas de evolución en X/Y muestran un desplazamiento suave y monótono. Los histogramas concentran una única distancia Manhattan (1000) y 71 giros, sin dispersión alguna.

Mapa general de rutas únicas encontradas



El recorrido se adapta bien a los obstáculos y sigue el camino más lógico. Se nota que el algoritmo encuentra una solución ordenada y eficiente.

Simulación con 50 agentes

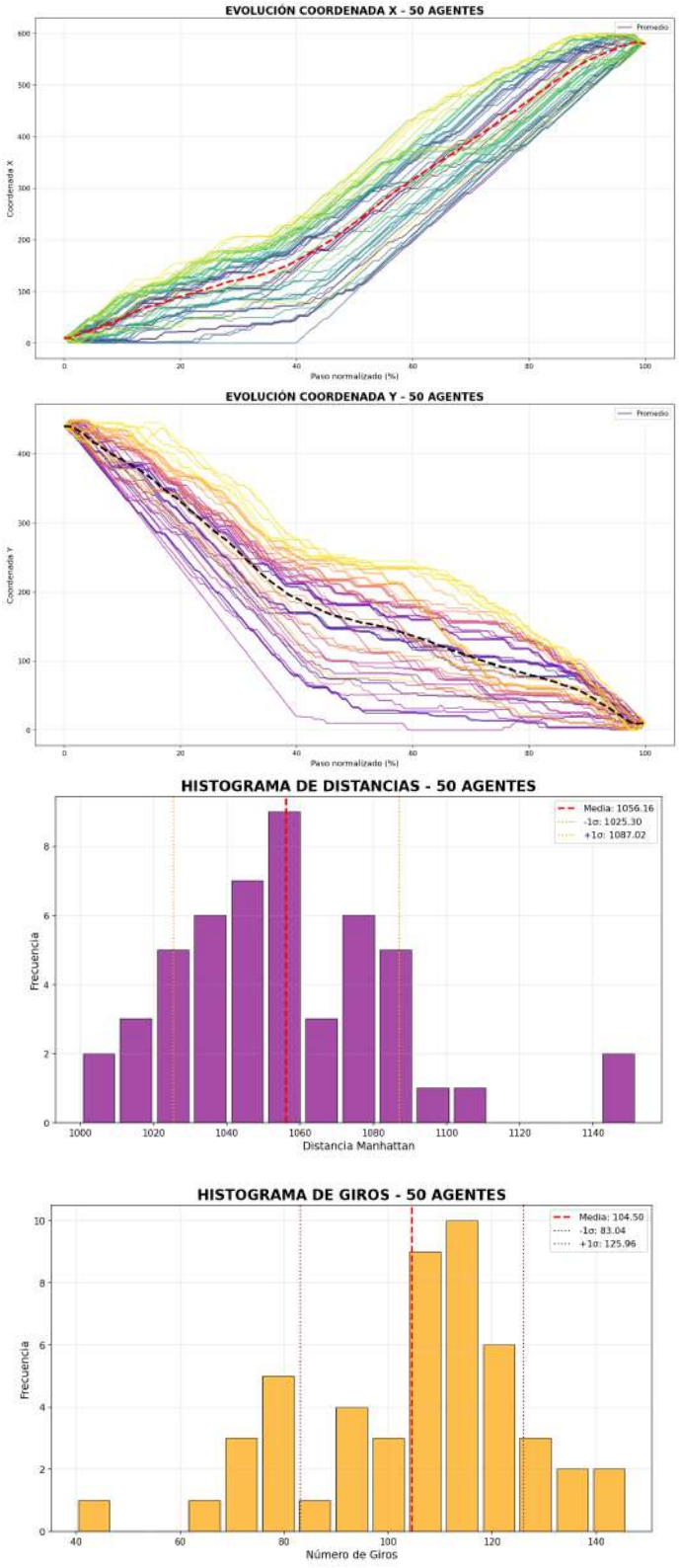
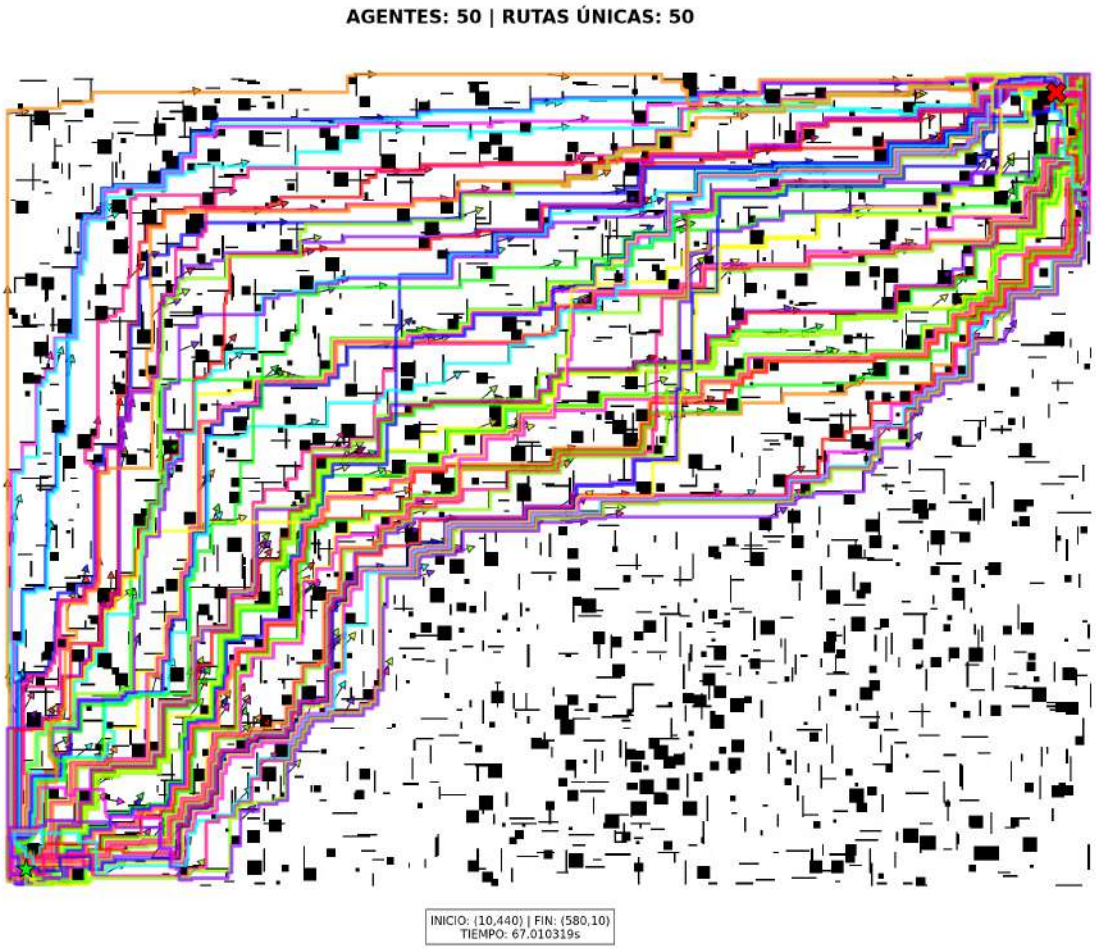


Figura 5.2: Análisis del recorrido: series temporales de coordenadas en eje X y eje Y, distribución de costes Manhattan y estadística de orientación.

Las coordenadas X/Y empiezan a mostrar pequeñas bifurcaciones. El histograma de distancias presenta una campana estrecha: distancia promedio de 1056, con poca dispersión entre agentes (desviación de solo 30.86). Los giros se dispersan hasta 146, reflejando interacciones

moderadas.

Mapa general de rutas únicas encontradas



El recorrido todavía se adapta bien a los obstáculos, pero ya no es tan directo como en casos simples. Se empiezan a notar pequeños ajustes en las trayectorias debido a la presencia de más agentes. El algoritmo sigue encontrando rutas eficientes, aunque ahora hay ligeras desviaciones para evitar coincidencias. La saturación comienza a aparecer en zonas específicas, pero no afecta gravemente el flujo general.

Simulación con 100 agentes

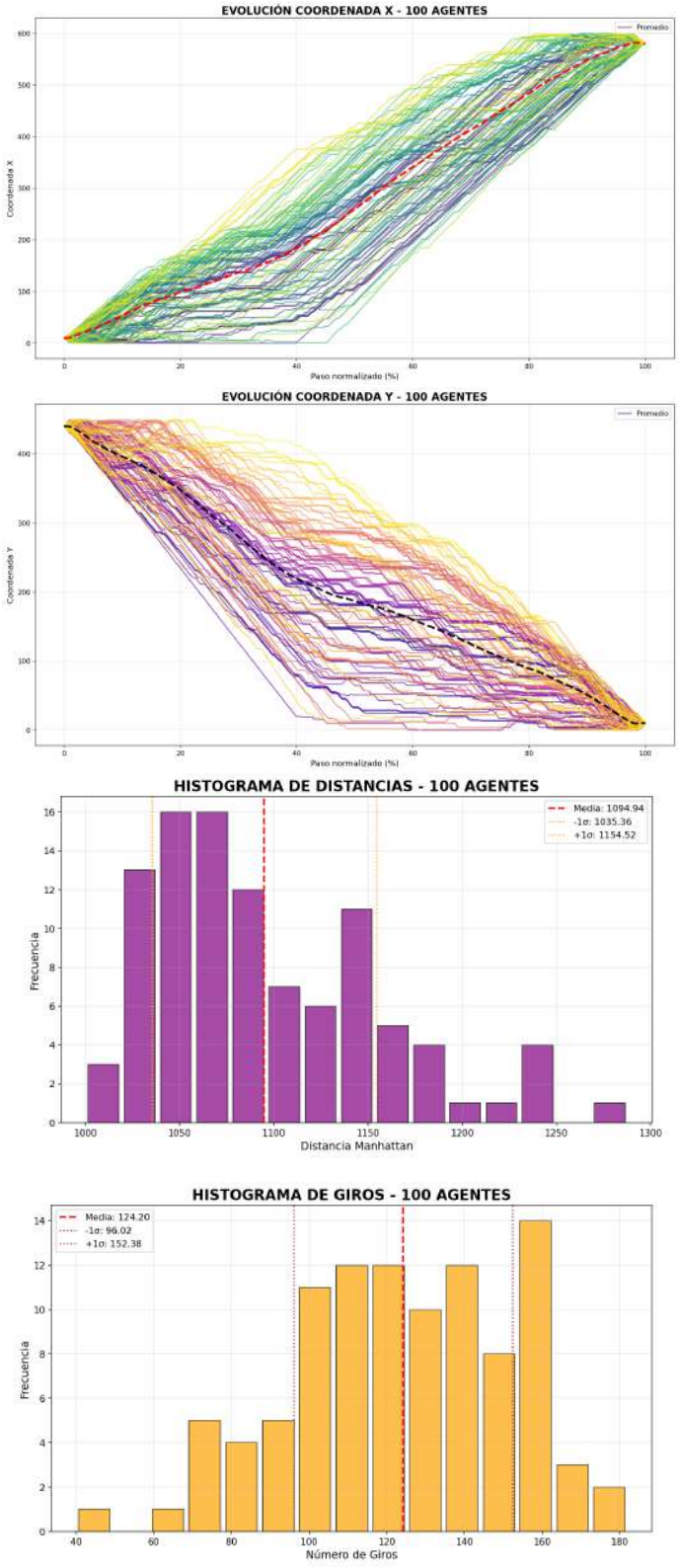
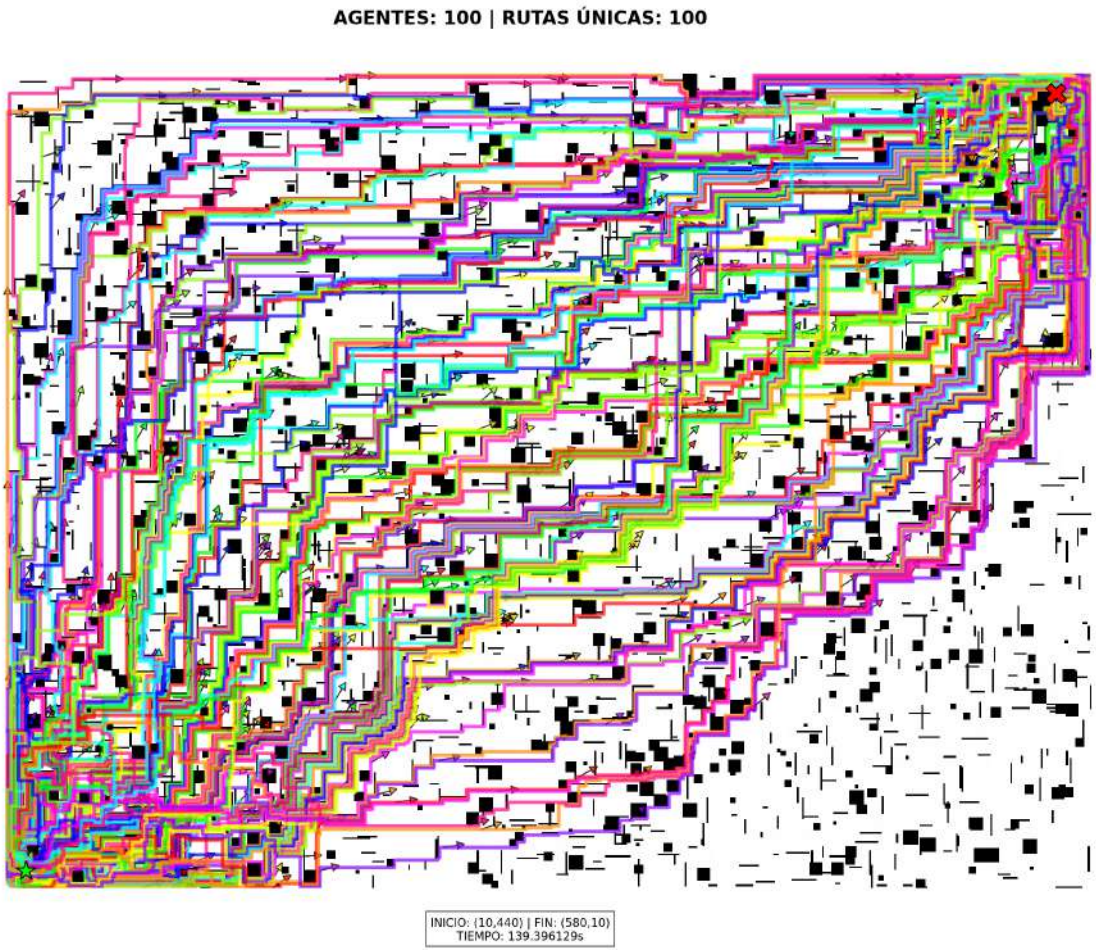


Figura 5.3: Análisis del recorrido: series temporales de coordenadas en eje X y eje Y, distribución de costes Manhattan y estadística de orientación.

Mayor dispersión en evolución de coordenadas. Las distancias Manhattan se ensanchan (promedio 1094, con una desviación de 59.58, indicando que las rutas comienzan a diferenciarse más entre sí). Giro promedio de 124 con máximo de 182. Saturación evidente en zonas críticas.

Mapa general de rutas únicas encontradas



El recorrido se vuelve más variable y menos uniforme, ya que los agentes interactúan constantemente entre sí. Se observan cambios más notorios en las trayectorias, especialmente en zonas de paso común. El algoritmo sigue siendo funcional, pero las rutas dejan de ser completamente óptimas debido a la densidad. La saturación es evidente en varios puntos, generando acumulaciones y reduciendo la fluidez del movimiento.

Simulación con 200 agentes

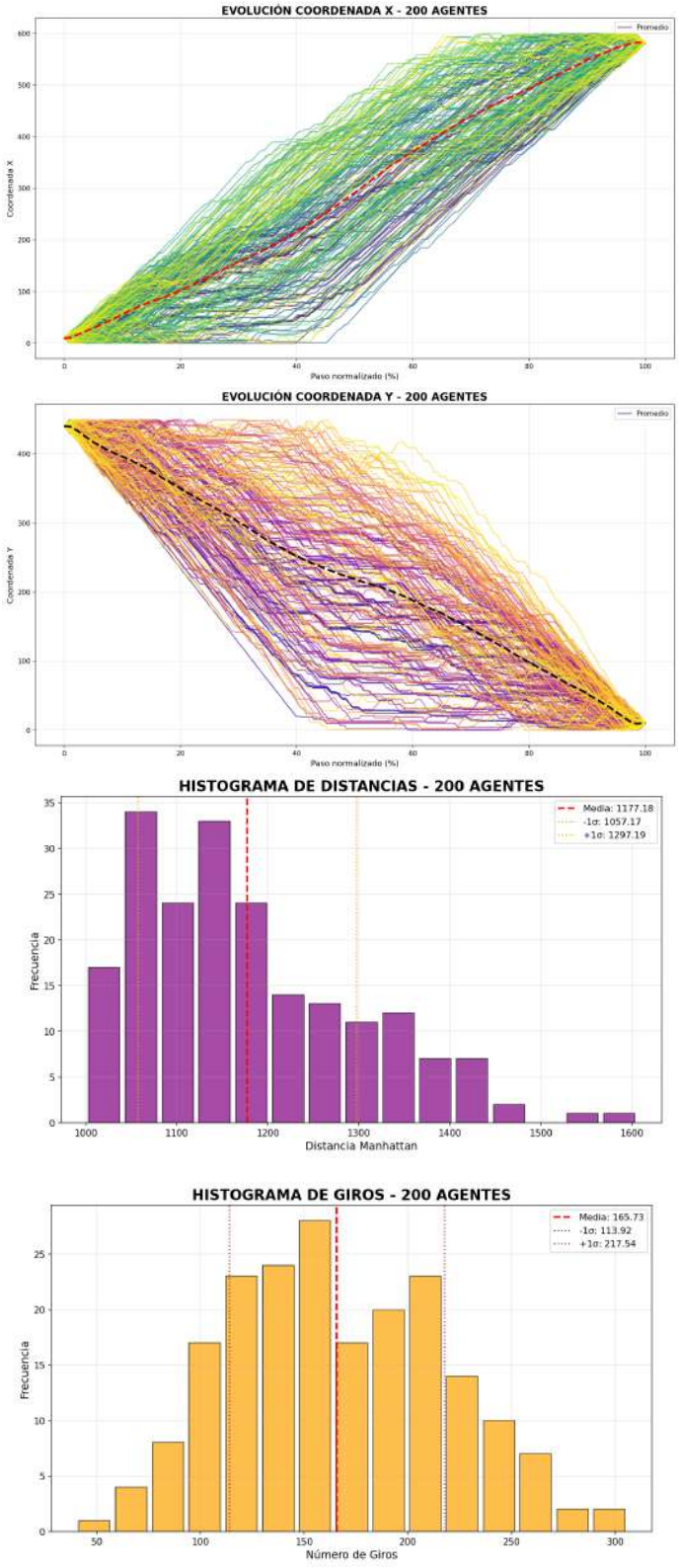
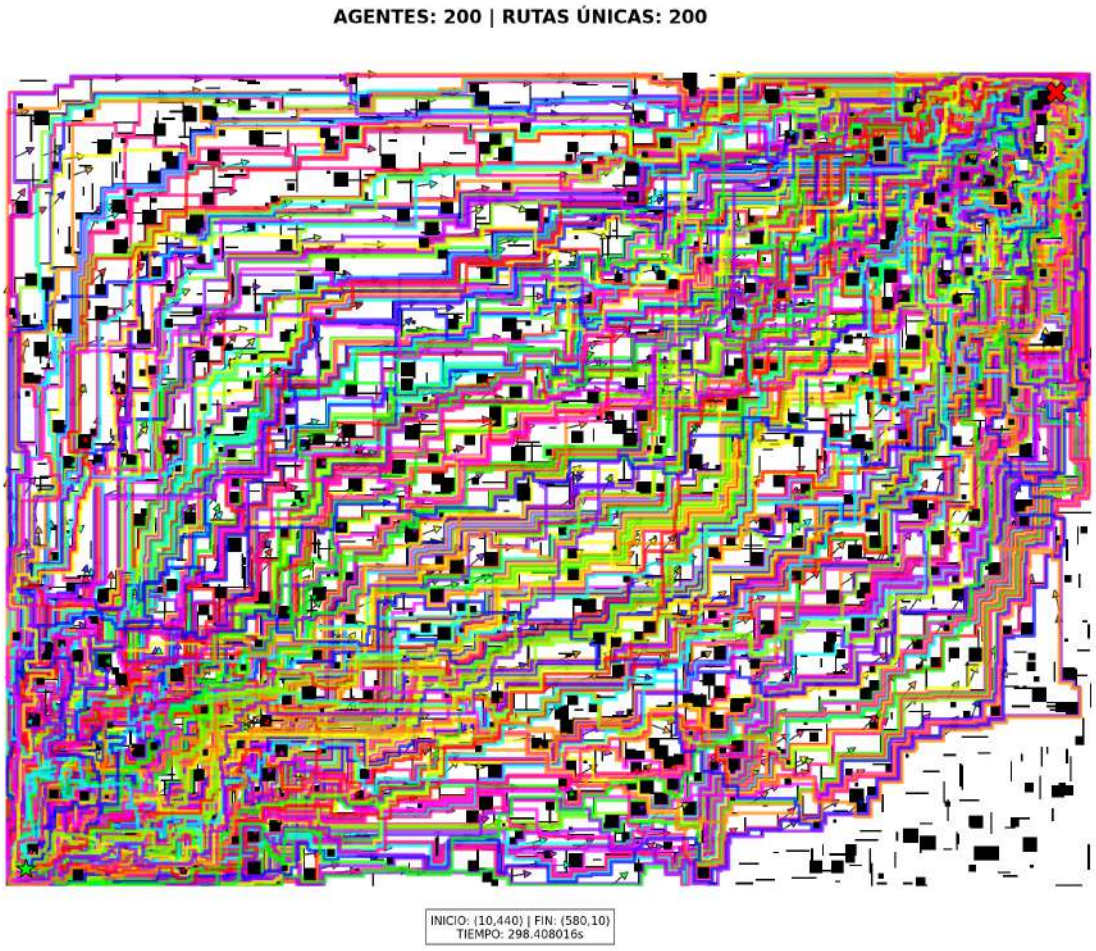


Figura 5.4: Análisis del recorrido: series temporales de coordenadas en eje X y eje Y, distribución de costes Manhattan y estadística de orientación.

Evolución de coordenadas muy dispersa y congestionada. Las distancias alcanzan promedio de 1177 con alta variabilidad (desviación de 120, señal de que algunos agentes hacen rutas muy largas mientras otros mantienen caminos más cortos). Giros promedio de 165.7, máximos de

306. Saturación alta con cuellos de botella.

Mapa general de rutas únicas encontradas



El recorrido presenta múltiples desviaciones y pierde en gran medida la forma óptima inicial. Los agentes tienden a agruparse en ciertas zonas, lo que provoca congestión y movimientos más lentos. El algoritmo continúa guiando a los agentes, pero la alta densidad limita su eficiencia. La saturación es alta, con presencia clara de cuellos de botella que afectan significativamente el flujo y el tiempo de evacuación.

5.2.2. Tabla comparativa general

Para organizar los resultados de este análisis con la imagen generada en el primer escenario, se construyó la siguiente tabla comparativa.

Cuadro 5.1: Registro comparativo de las simulaciones

AGENTES	RUTAS TERMINADAS	TIEMPO (s)	CORR PROM	CORR MIN	CORR MAX	DIF MAX-MIN	MEDIA DIST	DESV DIST	MEDIA GIROS	MAX GIROS
1	1	0.978339	N/A	N/A	N/A	N/A	1000.00	0.00	71.00	71
50	50	67.010319	0.976289	0.873367	0.999775	0.126408	1056.16	30.86	104.50	146
100	100	139.396129	0.954997	0.742461	0.999775	0.257314	1094.94	59.58	124.20	182
200	200	298.408016	0.932836	0.605207	0.999775	0.394568	1177.18	120.01	165.73	306

5.2.3. Segunda simulación con imagen generada, con difentes puntos de partida y llegada

Simulación con un agente

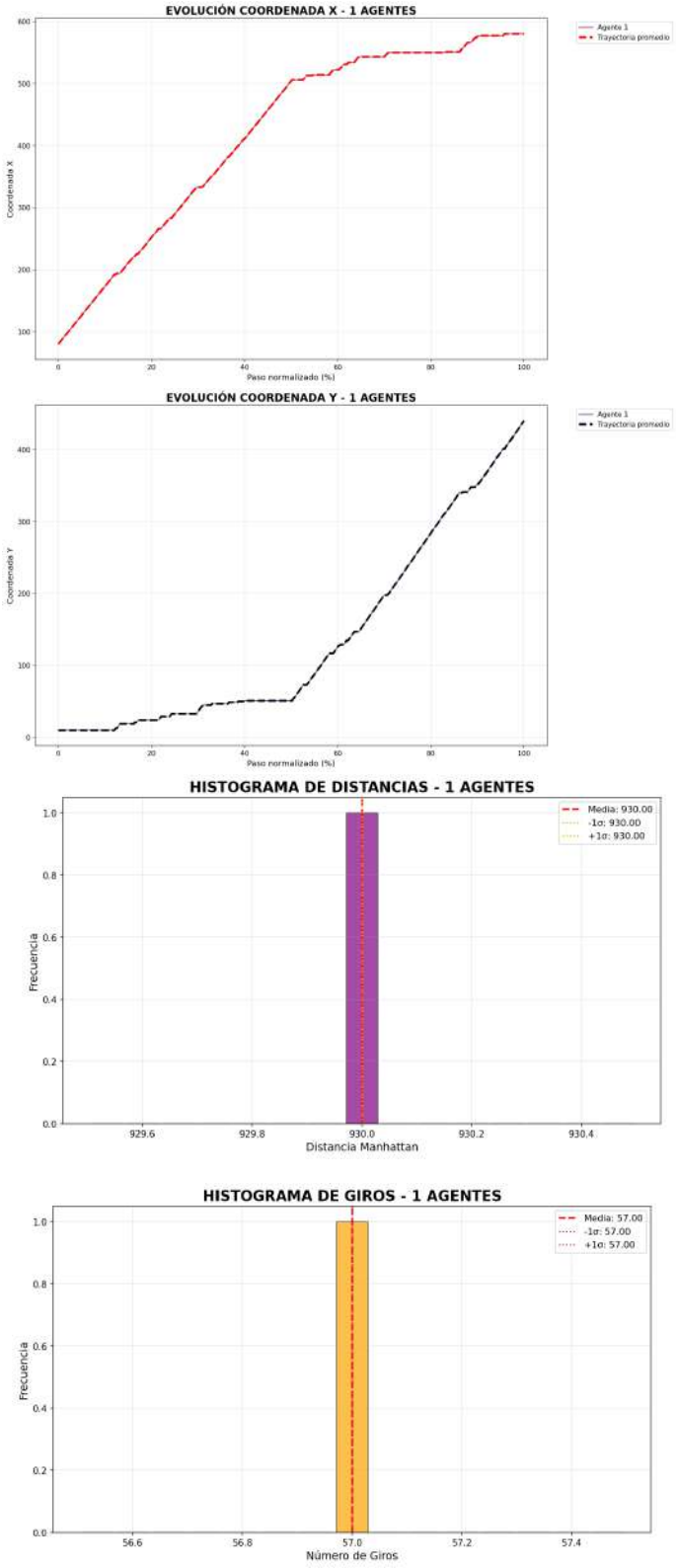
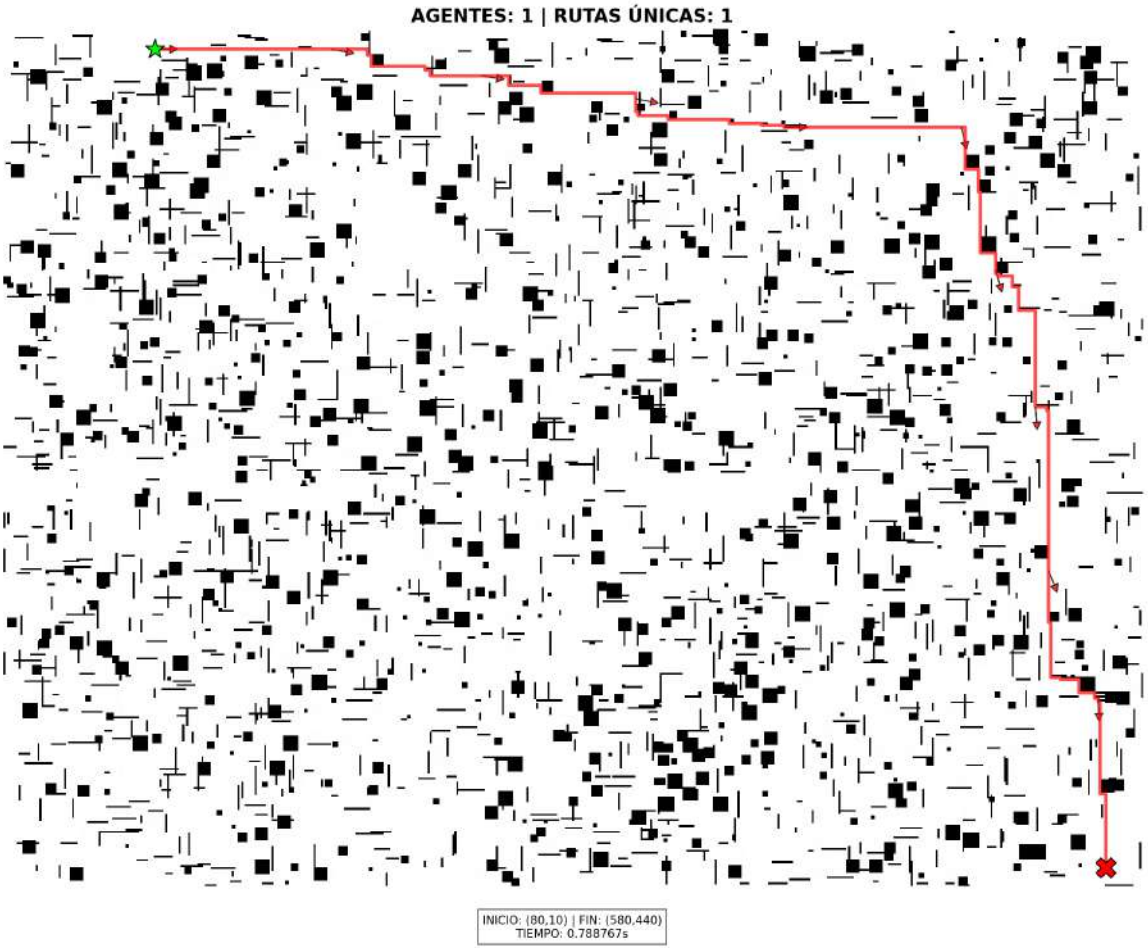


Figura 5.5: Análisis del recorrido: series temporales de coordenadas en eje X y eje Y, distribución de costes Manhattan y estadística de orientación.

Trayectoria óptima con distancia Manhattan de 930 y 57 giros. Evolución limpia y ordenada. Saturación nula.

Mapa general de rutas únicas encontradas



La trayectoria responde adecuadamente a los obstáculos, manteniendo una dirección coherente a lo largo del recorrido. Se observa que el algoritmo genera una solución clara, organizada y eficiente.

Simulación con 50 agentes

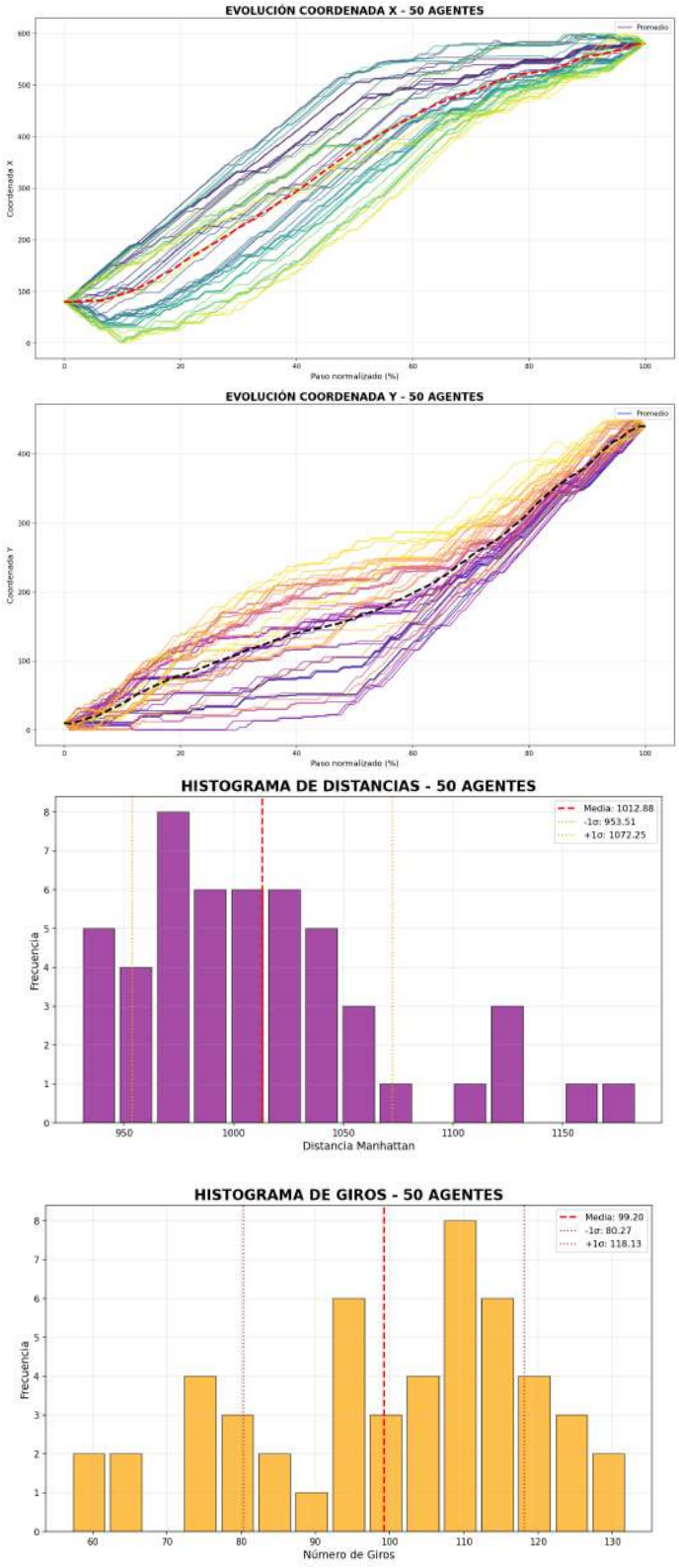
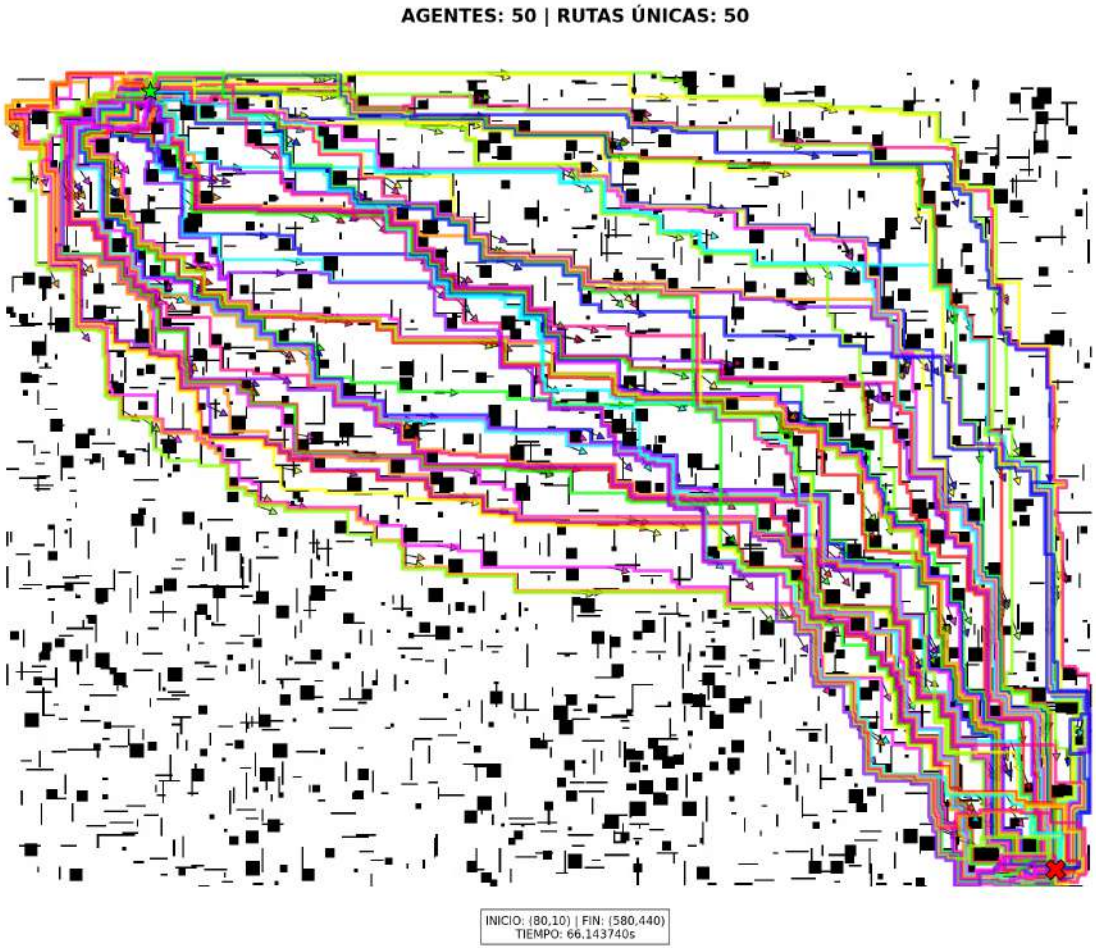


Figura 5.6: Análisis del recorrido: series temporales de coordenadas en eje X y eje Y, distribución de costes Manhattan y estadística de orientación.

Coordenadas con dispersión moderada. Distancia promedio de 1012 (desviación de 59.37, mostrando diferencias notables entre rutas). Giros promedio de 99.2 con máximo de 132. Saturación moderada incipiente.

Mapa general de rutas únicas encontradas



El recorrido mantiene su capacidad de adaptación ante obstáculos, aunque pierde algo de directividad (es decir, la ruta se vuelve menos rectilínea y con más pequeños rodeos) debido a la interacción con otros agentes, generando ligeros ajustes en las trayectorias. El algoritmo aún produce rutas eficientes, pero comienzan a surgir zonas con saturación moderada que no comprometen significativamente el flujo general.

Simulación con 100 agentes

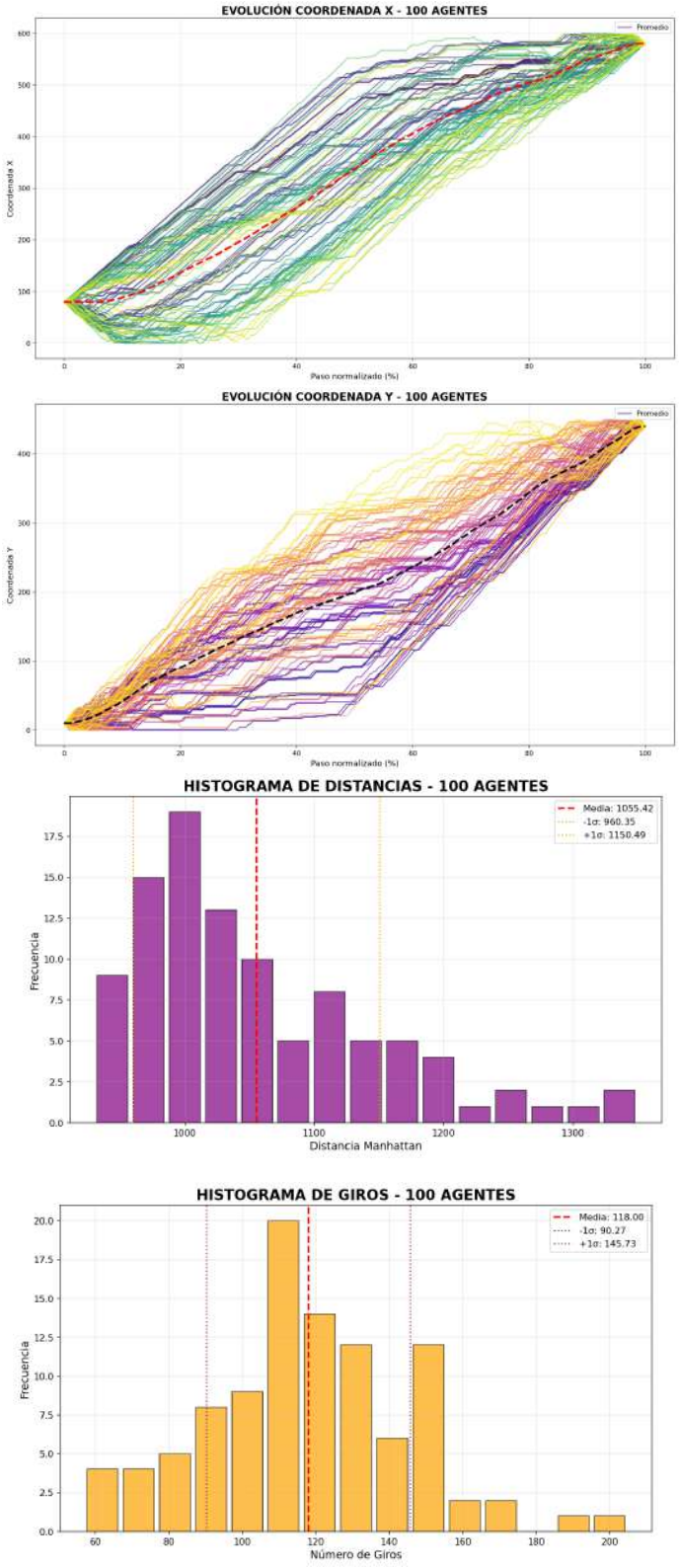
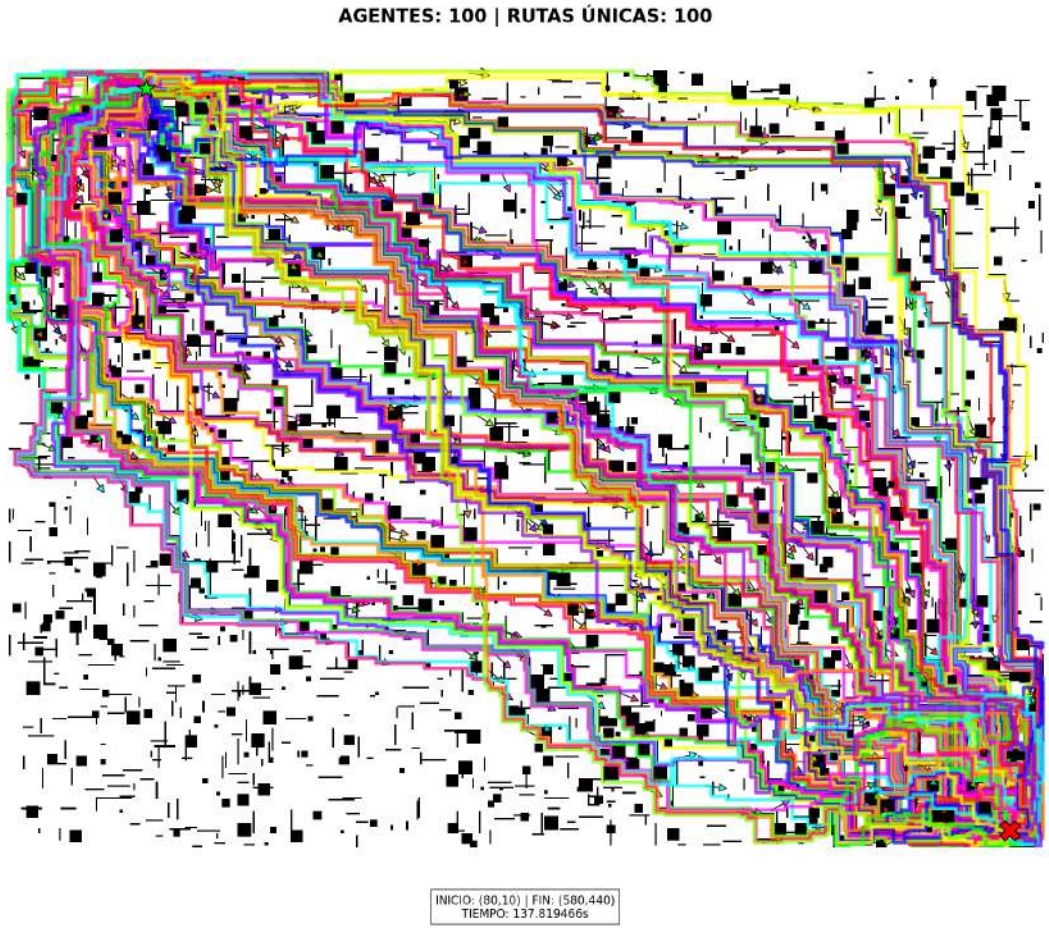


Figura 5.7: Análisis del recorrido: series temporales de coordenadas en eje X y eje Y, distribución de costes Manhattan y estadística de orientación.

Evolución irregular en ambos ejes. Distancia promedio de 1055 (desviación de 95.07, variabilidad alta). Giros promedio de 118, máximo 205. Congestión en zonas de paso común.

Mapa general de rutas únicas encontradas



El desplazamiento se vuelve más irregular y dinámico debido a la interacción constante entre los agentes, generando variaciones notables en zonas de tránsito frecuente y afectando la optimalidad de las rutas. La saturación se manifiesta en distintos puntos del entorno, provocando acumulaciones que disminuyen la fluidez y eficiencia del movimiento general.

Simulación con 200 agentes

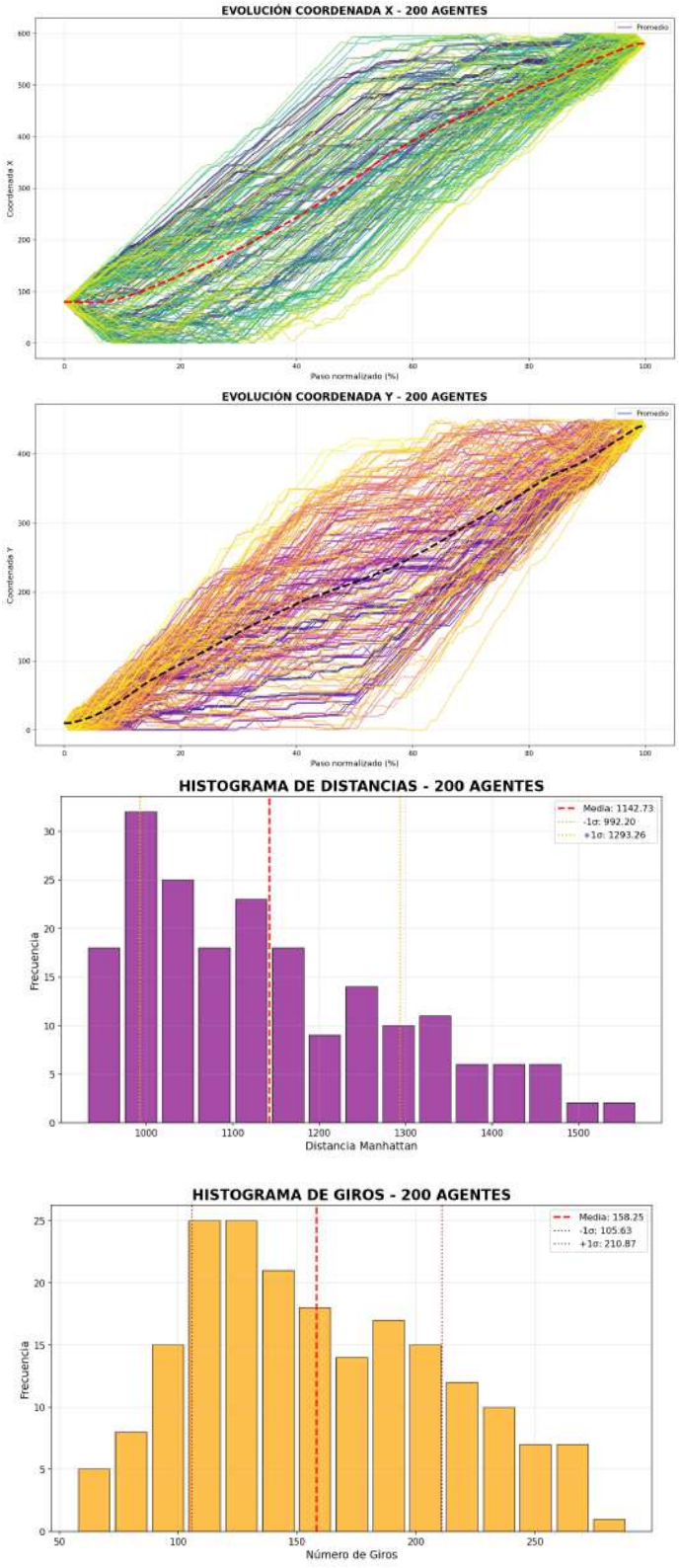
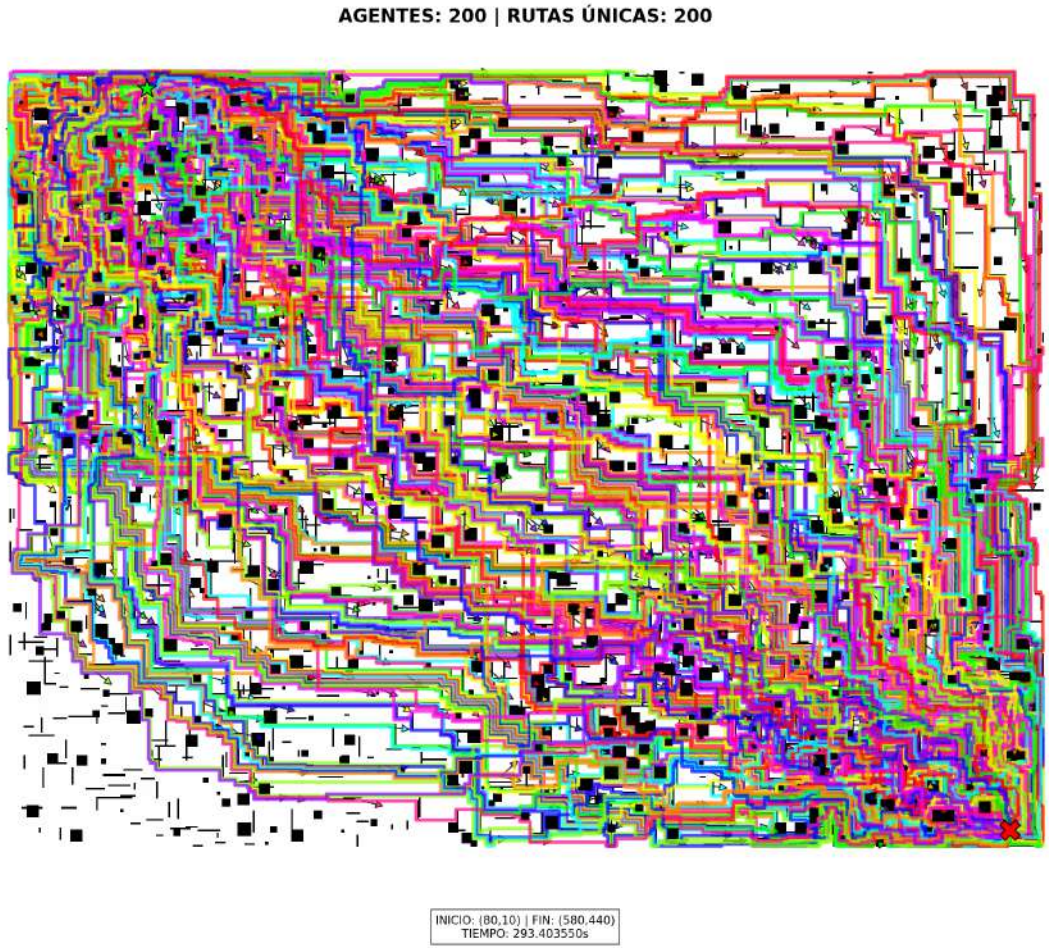


Figura 5.8: Análisis del recorrido: series temporales de coordenadas en eje X y eje Y, distribución de costes Manhattan y estadística de orientación.

Alta dispersión y acumulación de trayectorias. Distancia promedio de 1142 (desviación muy alta de 150.53, indicando comportamientos muy diversos entre agentes). Giros promedio de 158.25, máximo 289. Saturación elevada con cuellos de botella severos.

Mapa general de rutas únicas encontradas



La trayectoria muestra diversas desviaciones, alejándose considerablemente de la ruta óptima inicial. Los agentes se concentran en puntos específicos, generando acumulaciones que reducen la velocidad de desplazamiento. El algoritmo sigue orientando el movimiento, pero la elevada densidad disminuye su desempeño.

El nivel de saturación es elevado, evidenciándose cuellos de botella que afectan de forma importante el flujo general. Esto repercute directamente en un incremento del tiempo requerido para la evacuación.

5.2.4. Tabla comparativa general

Para organizar los resultados de este análisis con la imagen generada en el segundo escenario, se construyó la siguiente tabla comparativa.

Cuadro 5.2: Registro comparativo de las simulaciones

AGENTES	RUTAS TERMINADAS	TIEMPO (s)	CORR PROM	CORR MIN	CORR MAX	DIF MAX-MIN	MEDIA DIST	DESV DIST	MEDIA GIROS	MAX GIROS
1	1	0.788767	N/A	N/A	N/A	N/A	930.00	0.00	57.00	57
50	50	66.143740	0.970479	0.853510	0.999858	0.146349	1012.88	59.37	99.20	132
100	100	137.819466	0.954689	0.739154	0.999858	0.260704	1055.42	95.07	118.00	205
200	200	293.403550	0.929451	0.513478	0.999858	0.486381	1142.73	150.53	158.25	289

5.3. Clasificación de Niveles de Saturación para el análisis de una imagen ya existente

En el caso de un solo agente, el movimiento es completamente independiente, lo que permite que el algoritmo A* genere una trayectoria óptima sin ningún tipo de interferencia. El recorrido es directo y ordenado, representando un escenario ideal donde el entorno no limita el desplazamiento y la saturación es nula.

Para 10 agentes, el sistema se mantiene en un nivel bajo de saturación. Las trayectorias siguen siendo eficientes, aunque comienzan a presentarse ligeras variaciones entre los recorridos. El algoritmo continúa encontrando soluciones cercanas a las óptimas, y el flujo general se mantiene estable sin presencia de congestión significativa.

Cuando se incrementa a 20 agentes, la interacción entre agentes se vuelve más evidente. Se observan pequeñas desviaciones en las rutas y una mayor diversidad en los recorridos. Aunque el sistema sigue siendo funcional y eficiente, empiezan a identificarse zonas donde los agentes coinciden con mayor frecuencia. La saturación comienza a aparecer de forma localizada.

Para 40 agentes, el sistema entra en un nivel moderado de saturación. Se forman caminos preferentes y se incrementa la concentración de trayectorias en ciertas zonas del entorno. Los agentes tienden a agruparse, lo que provoca ajustes constantes en las rutas. El algoritmo sigue operando correctamente, pero las trayectorias ya no son completamente directas y la fluidez comienza a verse afectada.

Finalmente, con 80 agentes, el sistema presenta un nivel alto de saturación. Se observan múltiples trayectorias superpuestas, especialmente en zonas cercanas a la salida, donde se generan acumulaciones claras. El movimiento se vuelve más denso y menos uniforme, lo que reduce la eficiencia del sistema y aumenta considerablemente el tiempo de desplazamiento.

En conjunto, estos resultados muestran que la saturación no ocurre de manera abrupta, sino que surge de forma gradual conforme aumenta la densidad de agentes. Este comportamiento permite entender cómo el sistema evoluciona desde condiciones ideales hasta escenarios con alta interacción, donde la coincidencia espacial y la congestión comienzan a afectar de manera directa el desempeño del algoritmo y la fluidez del movimiento.

5.3.1. Primera simulación con imagen ya existente

Simulación con un agente

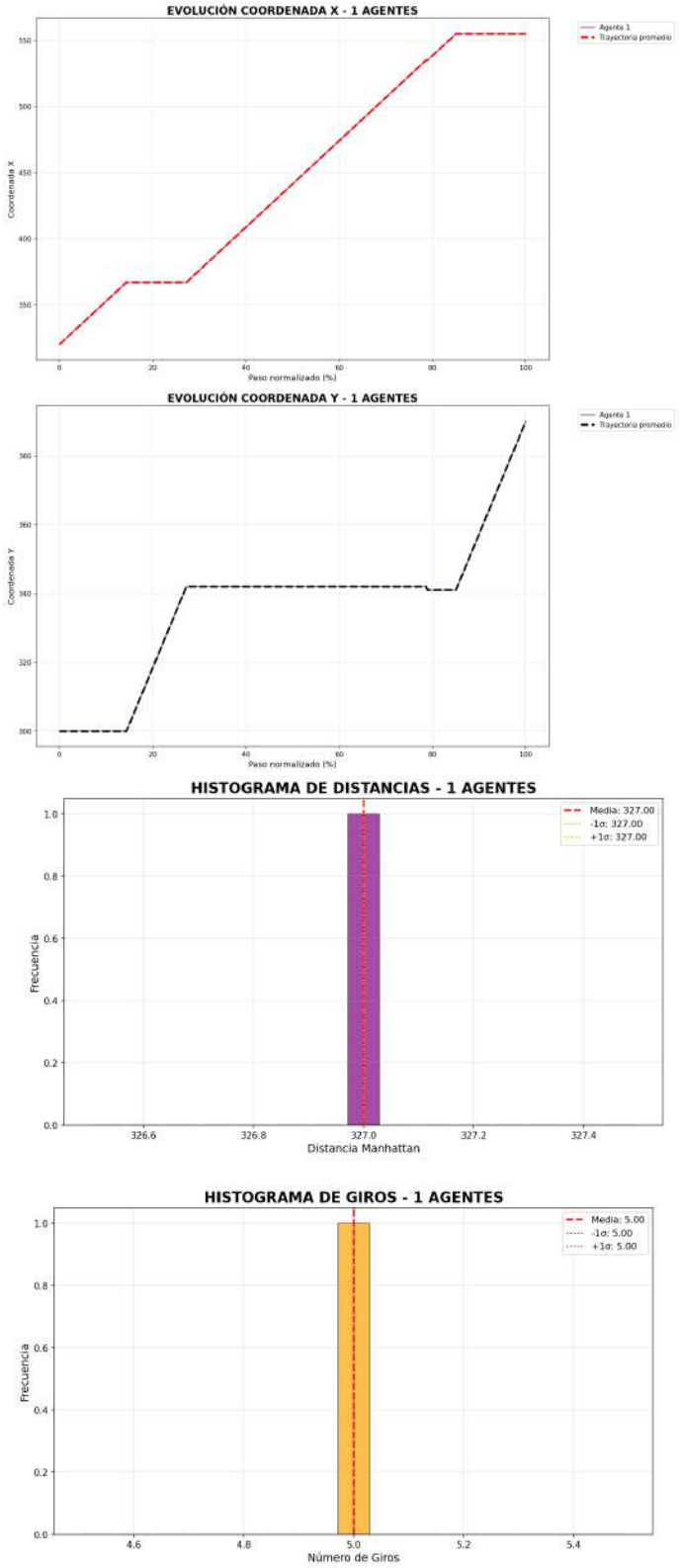
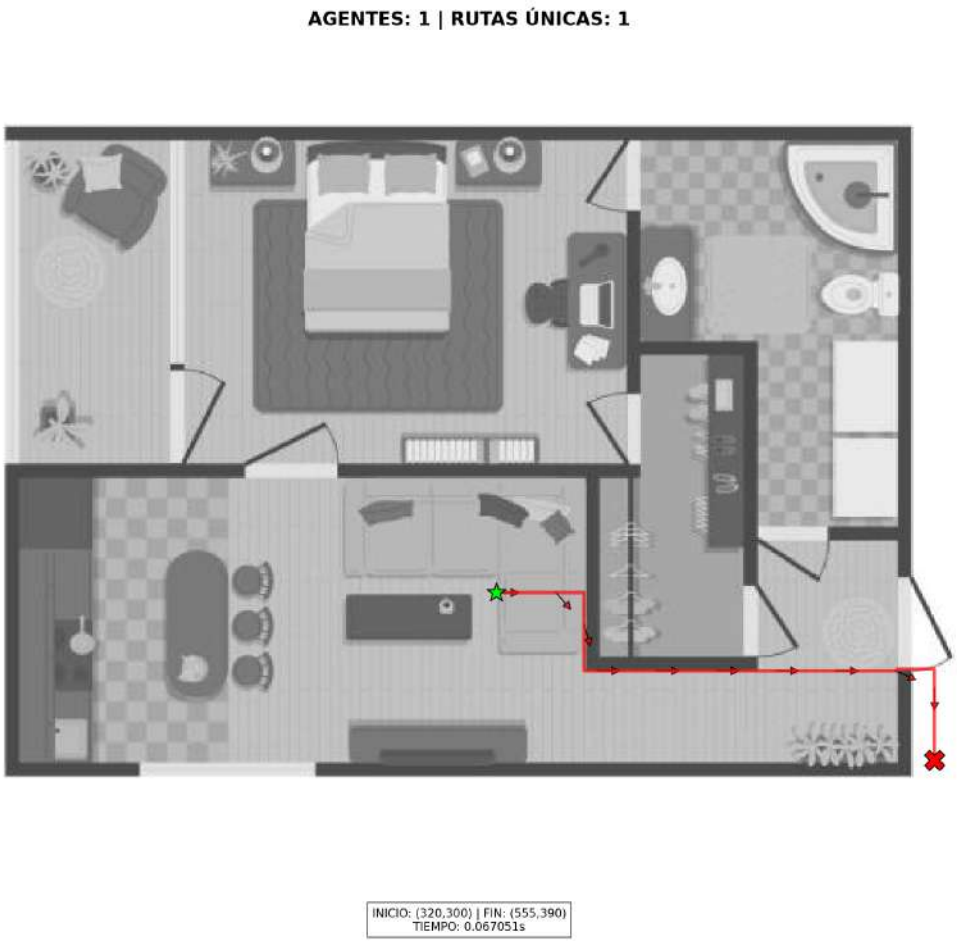


Figura 5.9: Desplazamiento del agente: evolución coordenada en eje X y en eje Y, espectro de distancias Manhattan e incidencia de giros.

Trayectoria limpia y óptima. Distancia Manhattan 327, solo 5 giros. Evolución de coordenadas suave y directa. Saturación nula.

Mapa general de rutas únicas encontradas



El recorrido se adapta completamente a los obstáculos y sigue el camino más directo posible. No existen desviaciones ni interferencias, por lo que la trayectoria es limpia y ordenada. El algoritmo encuentra una solución óptima sin necesidad de ajustes. La saturación es nula y el flujo es totalmente libre.

Simulación con 10 agentes

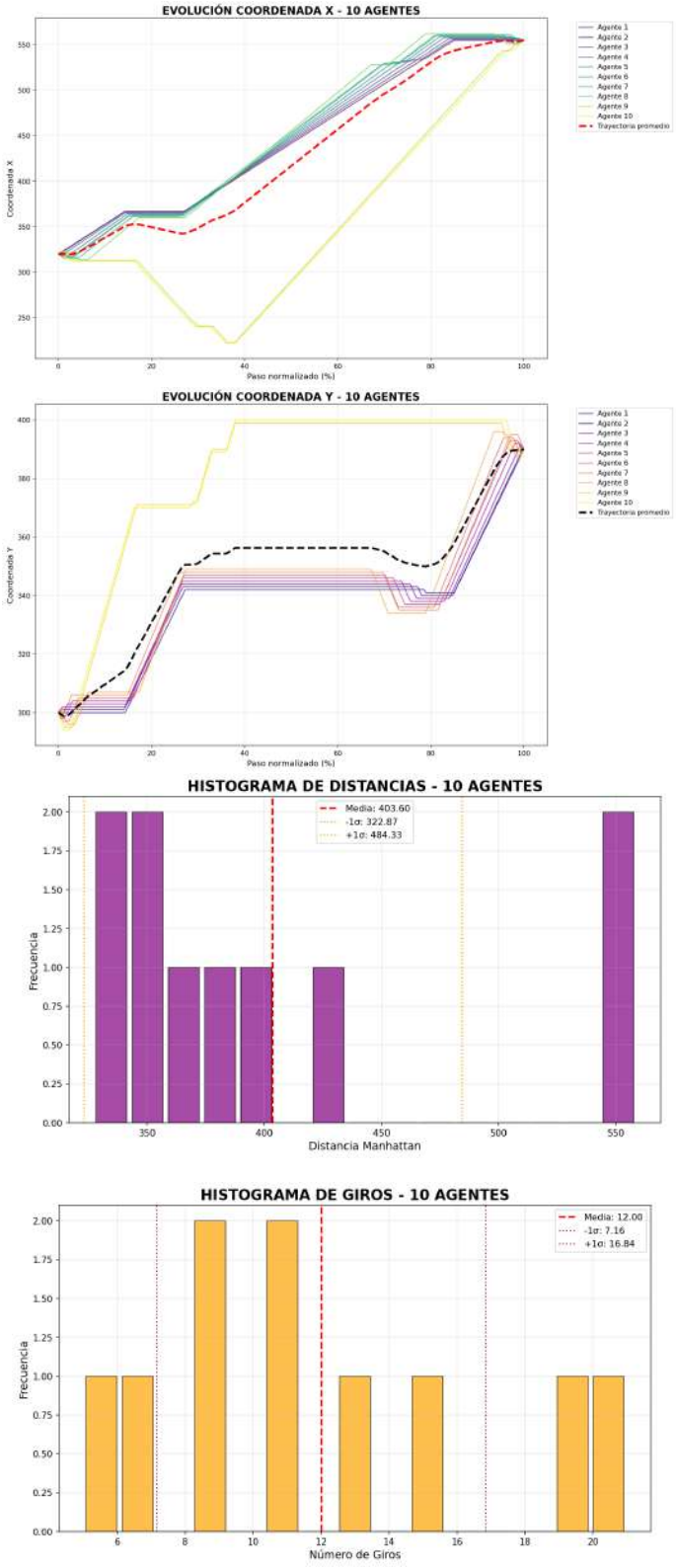
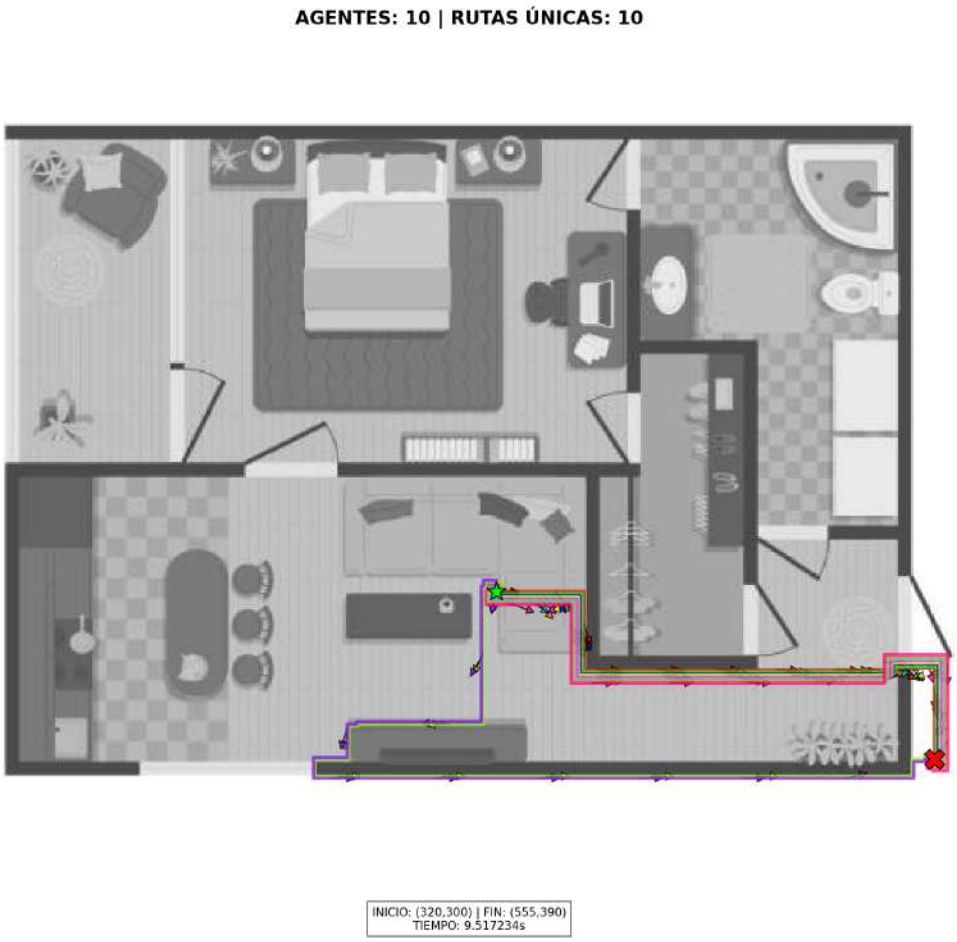


Figura 5.10: Desplazamiento del agente: evolución coordenada en eje X y en eje Y, espectro de distancias Manhattan e incidencia de giros.

Coordenadas con ligeras variaciones. Las distancias se concentran en promedio 403.6, con una desviación mínima de solo 0.73, lo que indica que todos los agentes hacen prácticamente la misma ruta. Giros promedio de 12, máximo 20. Saturación baja.

Mapa general de rutas únicas encontradas



El recorrido sigue siendo bastante eficiente, aunque ya no es completamente uniforme. Se comienzan a notar pequeñas variaciones entre trayectorias, pero sin afectar el camino general. El algoritmo mantiene soluciones cercanas a las óptimas. La saturación es baja y el flujo se conserva estable.

Simulación con 20 agentes

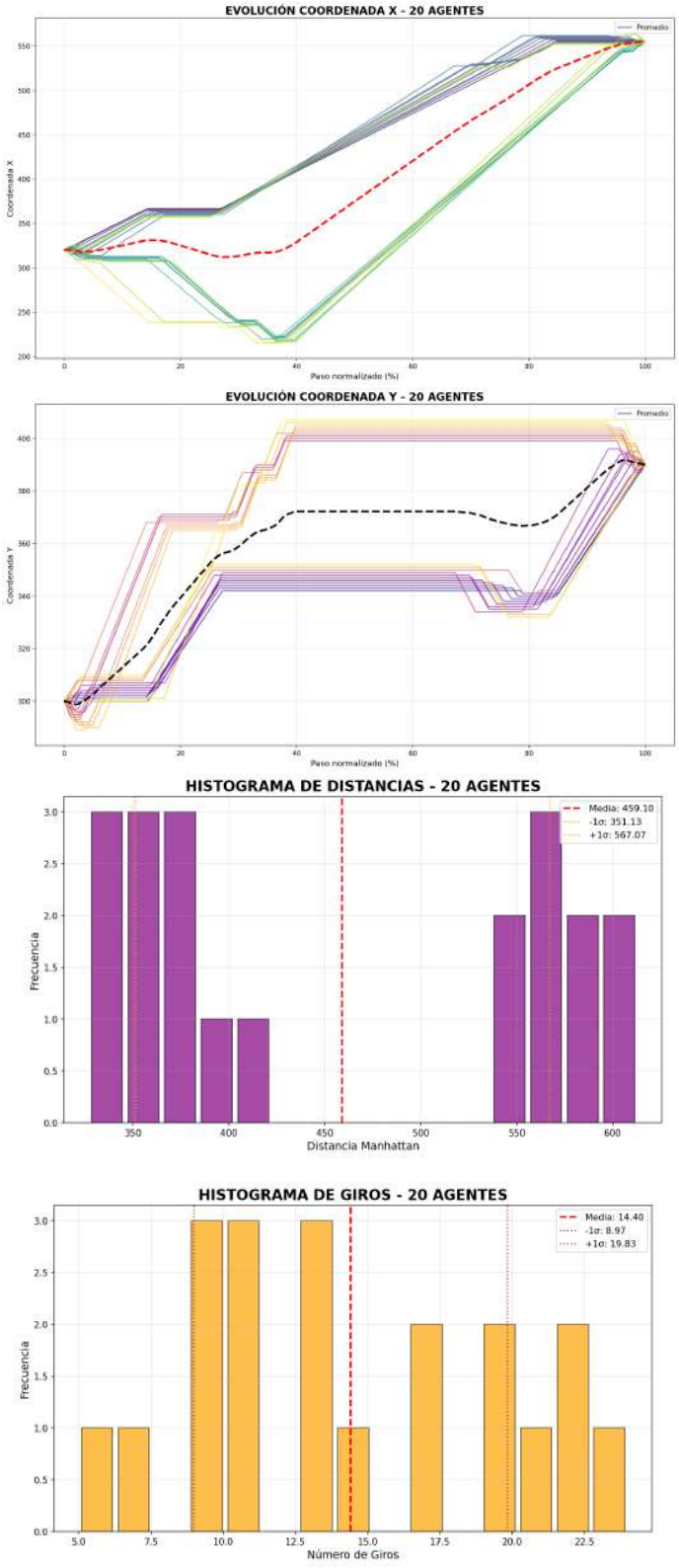
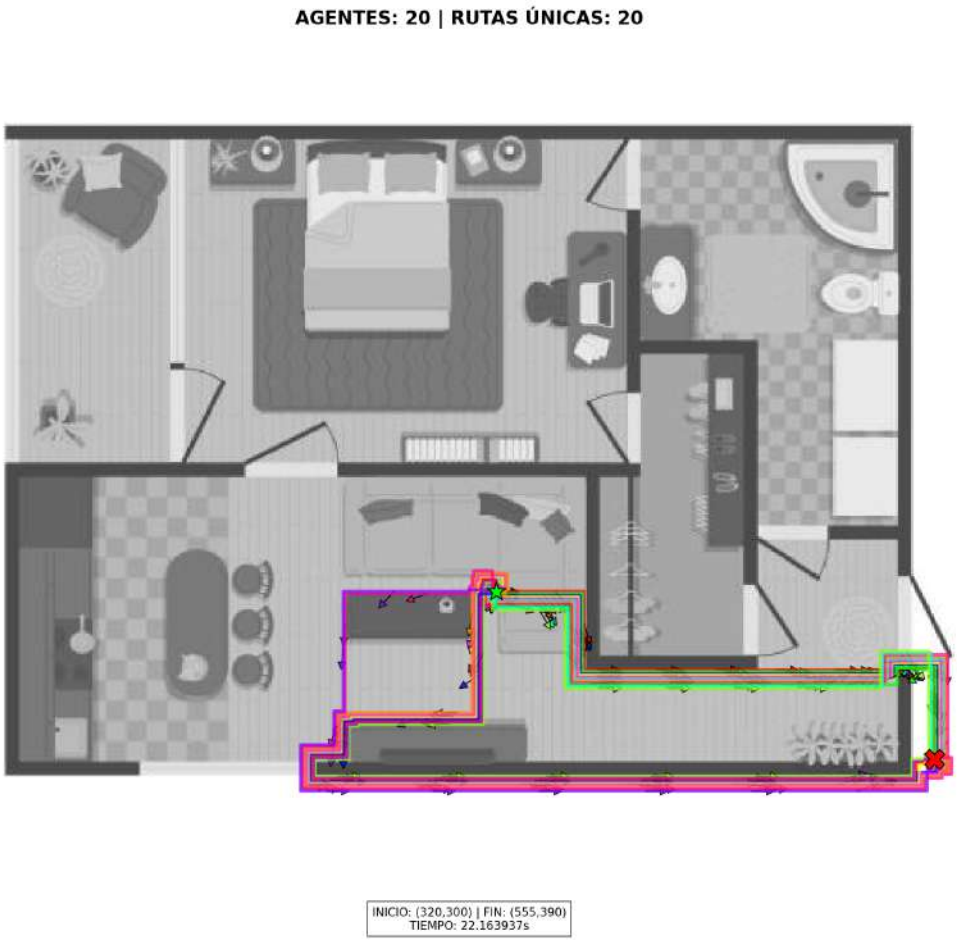


Figura 5.11: Desplazamiento del agente: evolución coordenada en eje X y en eje Y, espectro de distancias Manhattan e incidencia de giros.

Mayor diversidad de trayectorias. Distancia promedio de 459.1, pero con una desviación de 107.97, revelando que algunos agentes se desvían significativamente de la ruta óptima. Giros promedio de 14.4, máximo 40. Saturación localizada emergente.

Mapa general de rutas únicas encontradas



El recorrido presenta ligeros cambios más visibles, con algunas rutas que empiezan a separarse o rodear zonas comunes. Los agentes ya no siguen exactamente el mismo camino, lo que genera una mayor diversidad de trayectorias. El algoritmo sigue siendo eficiente, pero con más ajustes. La saturación comienza a notarse en puntos específicos.

Simulación con 40 agentes

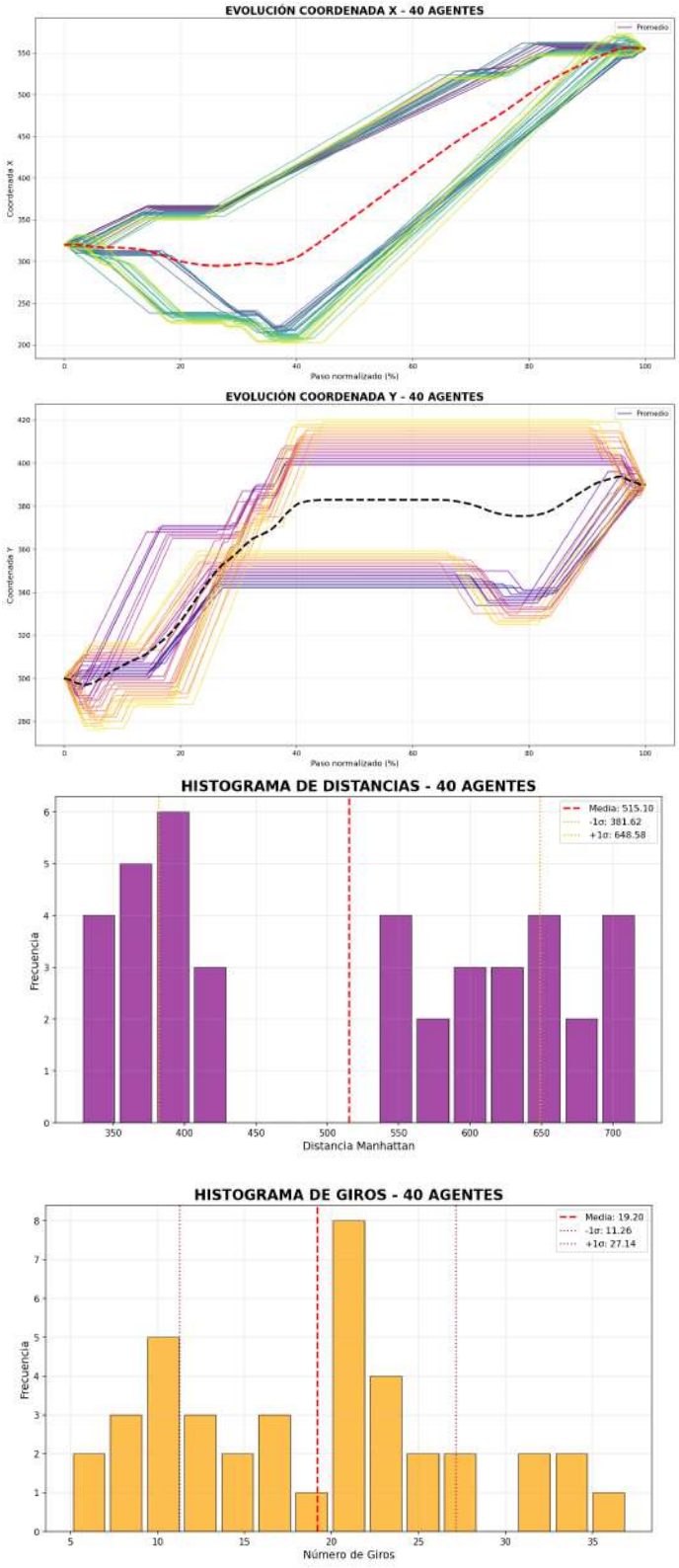
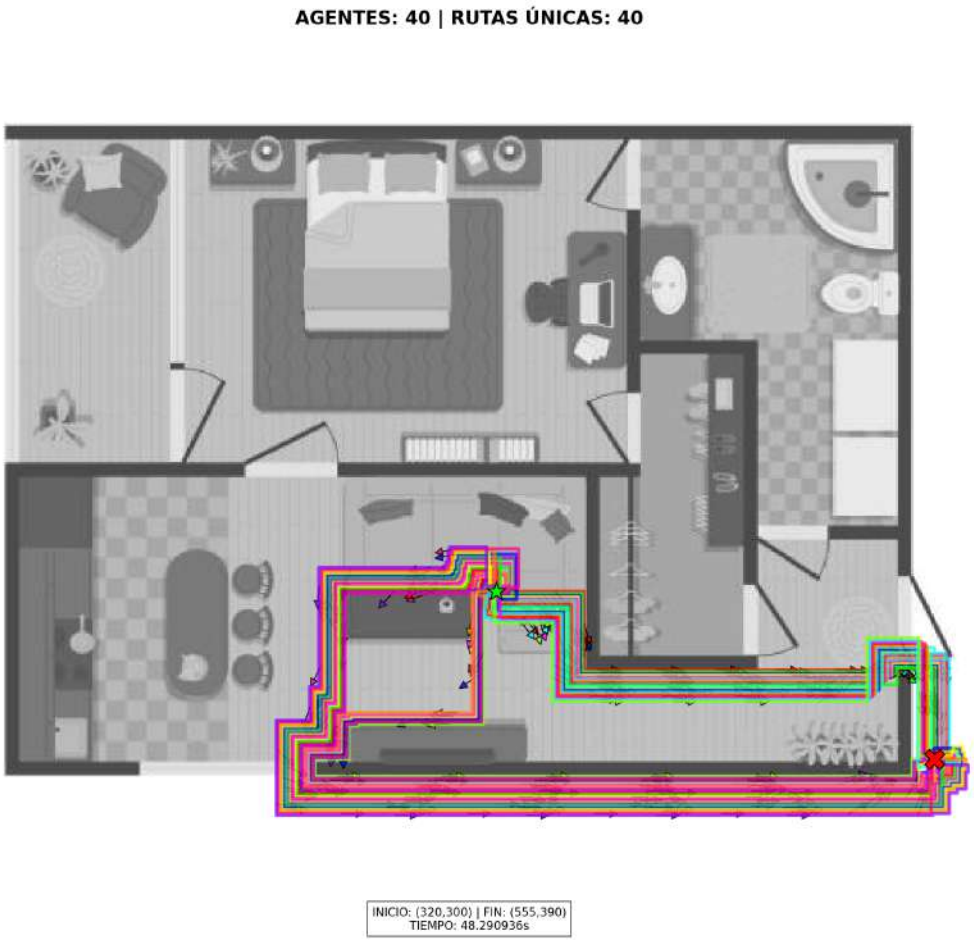


Figura 5.12: Desplazamiento del agente: evolución coordenada en eje X y en eje Y, espectro de distancias Manhattan e incidencia de giros.

Coordenadas más dispersas. Distancia promedio de 515.1 (desviación de 133.48, variabilidad creciente). Giros promedio de 19.2, máximo 78. Saturación moderada con caminos preferentes.

Mapa general de rutas únicas encontradas



El recorrido muestra una mayor concentración en ciertas zonas, donde varias trayectorias coinciden. Se forman caminos preferentes y los agentes empiezan a agruparse. El algoritmo sigue encontrando rutas funcionales, pero ya no siempre son las más directas. La saturación es moderada y afecta la fluidez en algunas áreas.

Simulación con 80 agentes

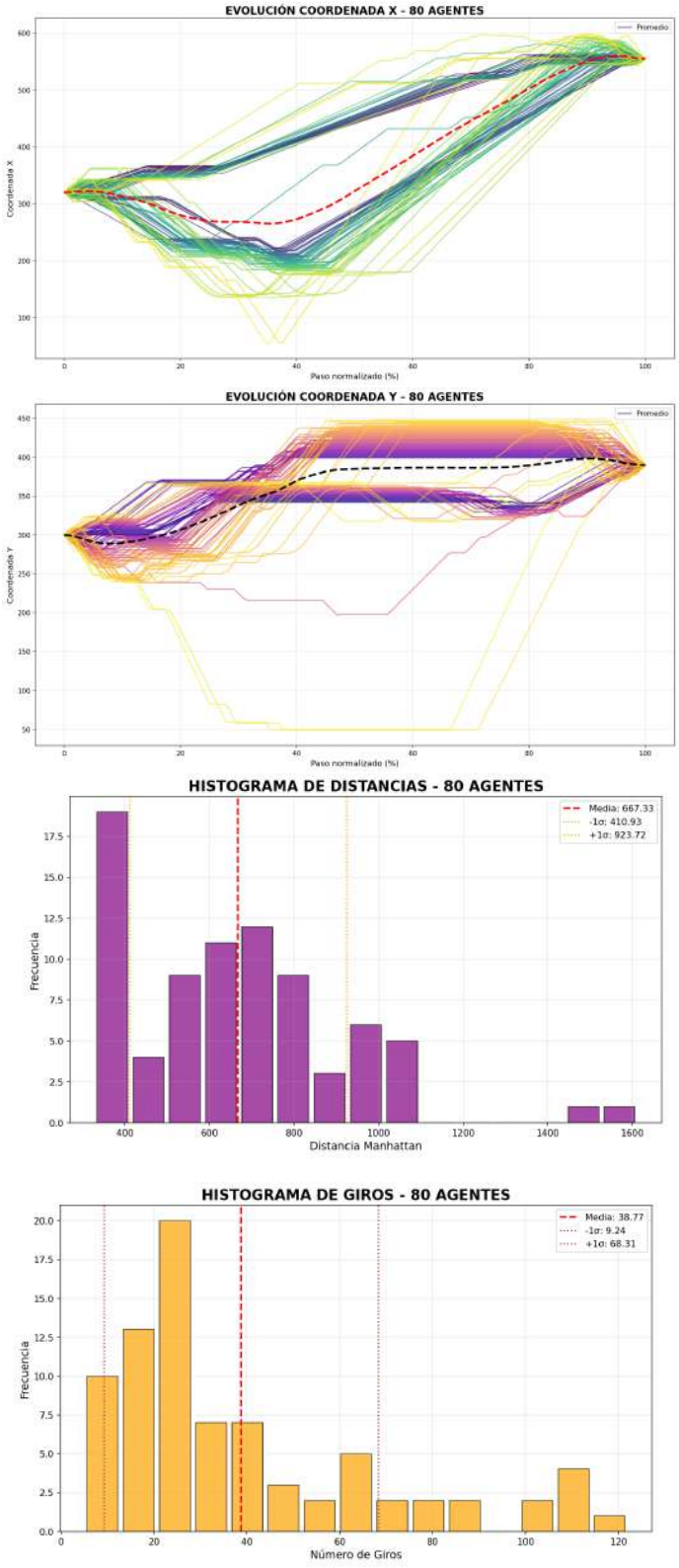
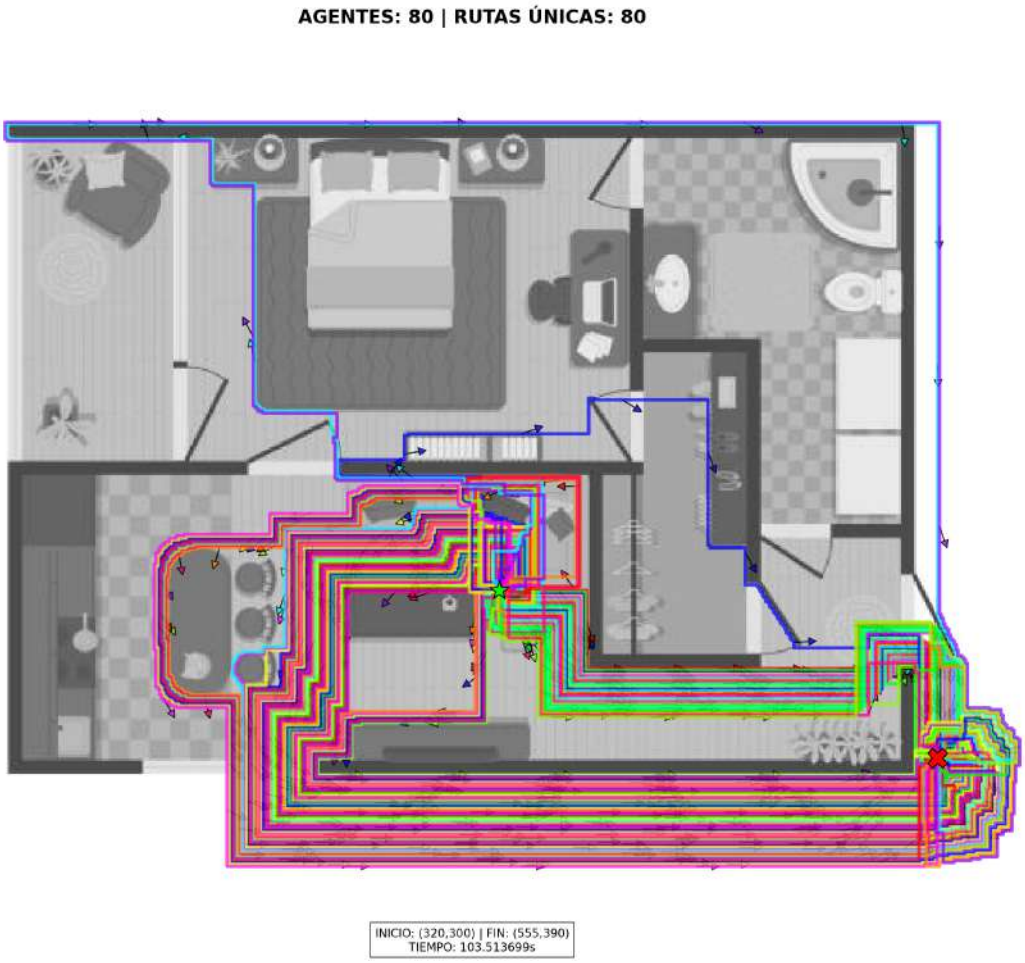


Figura 5.13: Desplazamiento del agente: evolución coordenada en eje X y en eje Y, espectro de distancias Manhattan e incidencia de giros.

Alta superposición de trayectorias. Distancia promedio de 667.33 con una desviación muy amplia de 256.39, lo que demuestra que algunos agentes encuentran rutas mucho más largas que otros debido a la congestión. Giros promedio de 38.77, máximo 122. Saturación alta con

congestión visible.

Mapa general de rutas únicas encontradas



El recorrido se vuelve más denso y menos uniforme, con muchas trayectorias superpuestas, especialmente cerca de la salida. Se observan acumulaciones claras y patrones repetitivos de movimiento. El algoritmo continúa generando rutas, pero con mayor número de desviaciones. La saturación es alta y reduce considerablemente la eficiencia del flujo.

5.3.2. Tabla comparativa general

Para organizar los resultados de este análisis con la imagen ya existente en el primer escenario, se construyó la siguiente tabla comparativa.

Cuadro 5.3: Registro comparativo de las simulaciones

AGENTES	RUTAS TERMINADAS	TIEMPO (s)	CORR PROM	CORR MIN	CORR MAX	DIF MAX-MIN	MEDIA DIST	DESV DIST	MEDIA GIROS	MAX GIROS
1	1	0.067051	N/A	N/A	N/A	N/A	327.00	0.00	5.00	5
10	10	9.517234	0.919305	0.759465	0.999635	0.240170	403.60	80.73	12.00	21
20	20	22.163937	0.888739	0.759465	0.999816	0.240351	459.10	107.97	14.40	24
40	40	48.290936	0.869151	0.598308	0.999816	0.401508	515.10	133.48	19.20	37
80	80	103.513699	0.781598	0.087588	0.999926	0.912338	667.33	256.39	38.77	122

5.3.3. Segunda simulación con imagen ya existente, con otras coordenadas de inicio y fin en el recorrido

Simulación con un agente

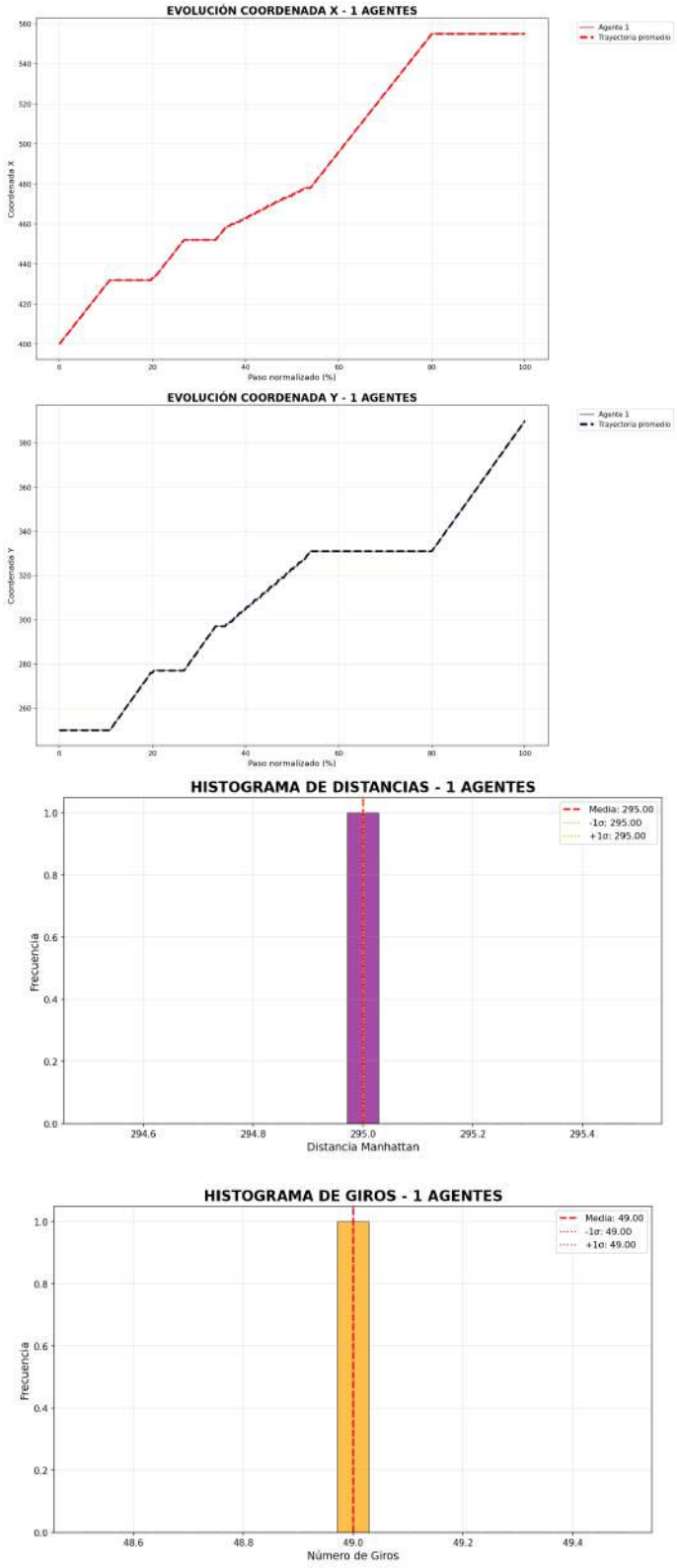
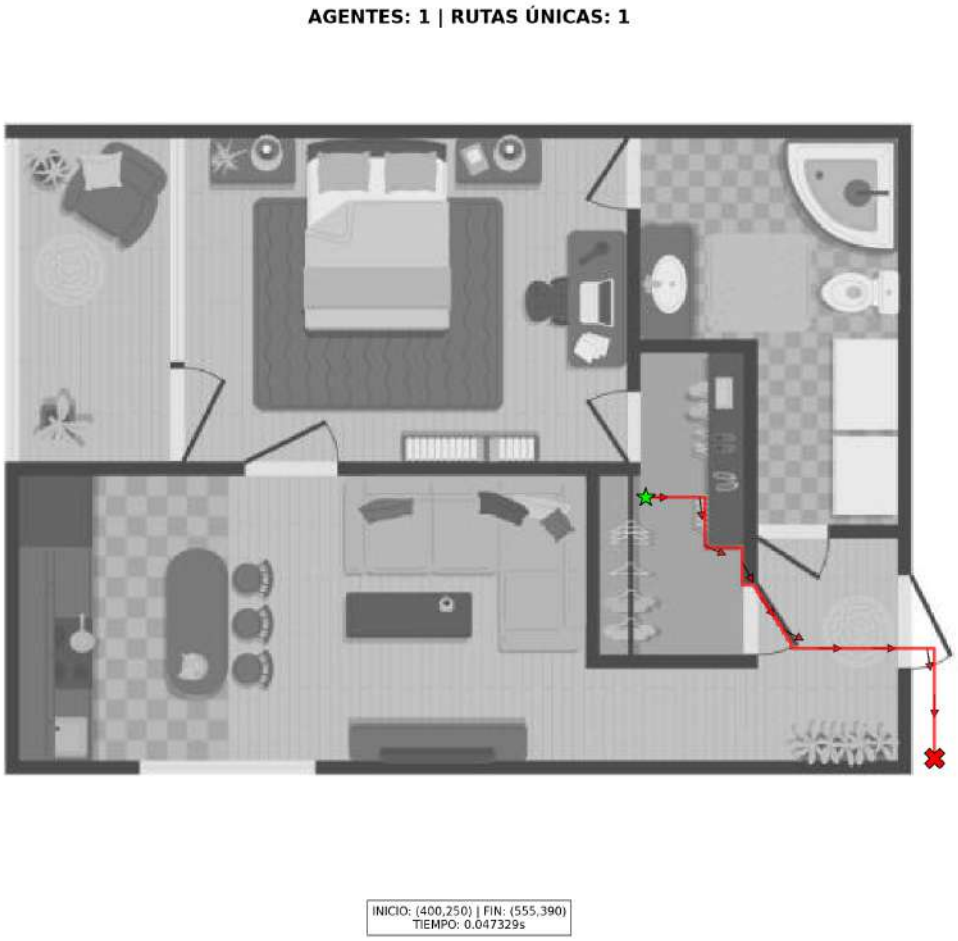


Figura 5.14: Desplazamiento del agente: evolución coordenada en eje X y en eje Y, espectro de distancias Manhattan e incidencia de giros.

Recorrido óptimo. Distancia 295, 49 giros. Evolución limpia. Saturación nula.

Mapa general de rutas únicas encontradas



El recorrido se ajusta plenamente a los obstáculos, manteniendo una trayectoria directa, limpia y sin desviaciones ni interferencias. El algoritmo obtiene una solución óptima sin requerir ajustes, y la ausencia de saturación permite un flujo completamente libre.

Simulación con 10 agentes

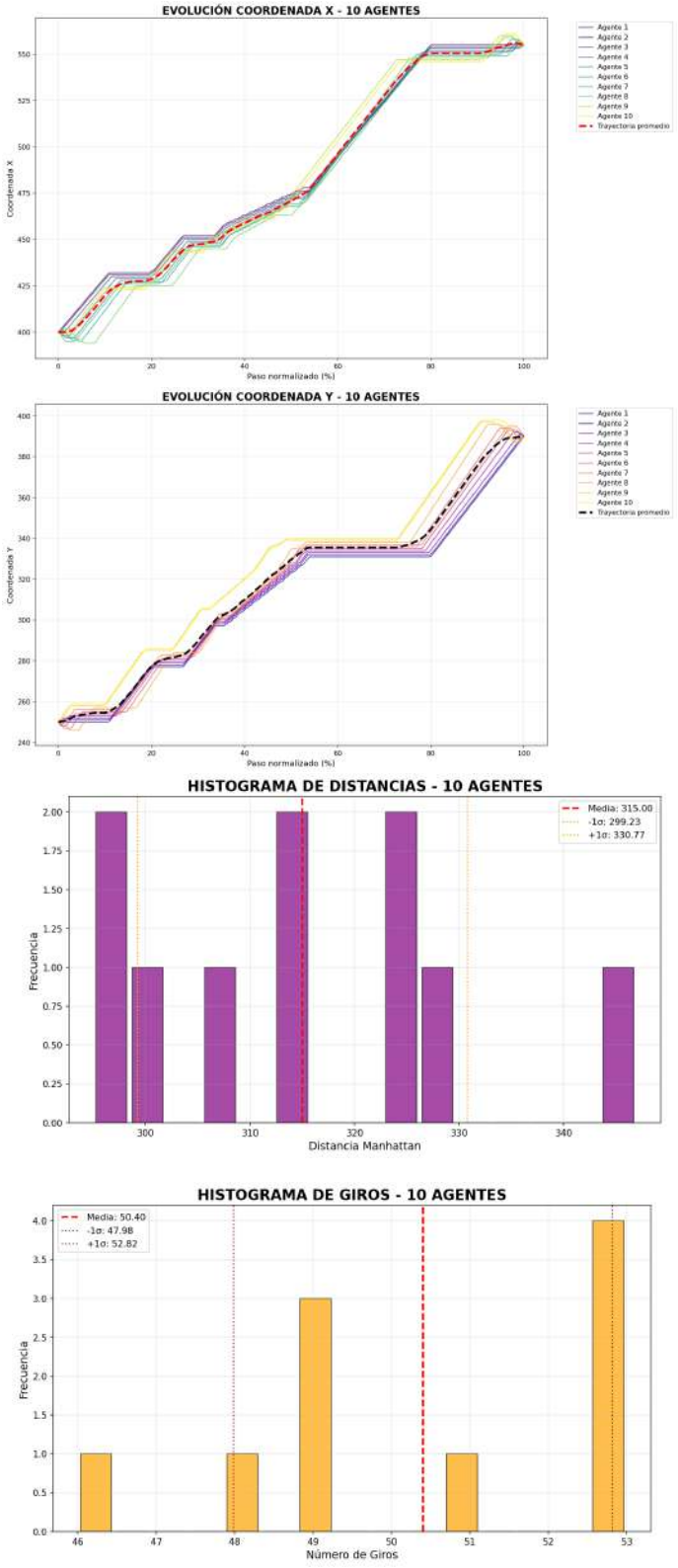
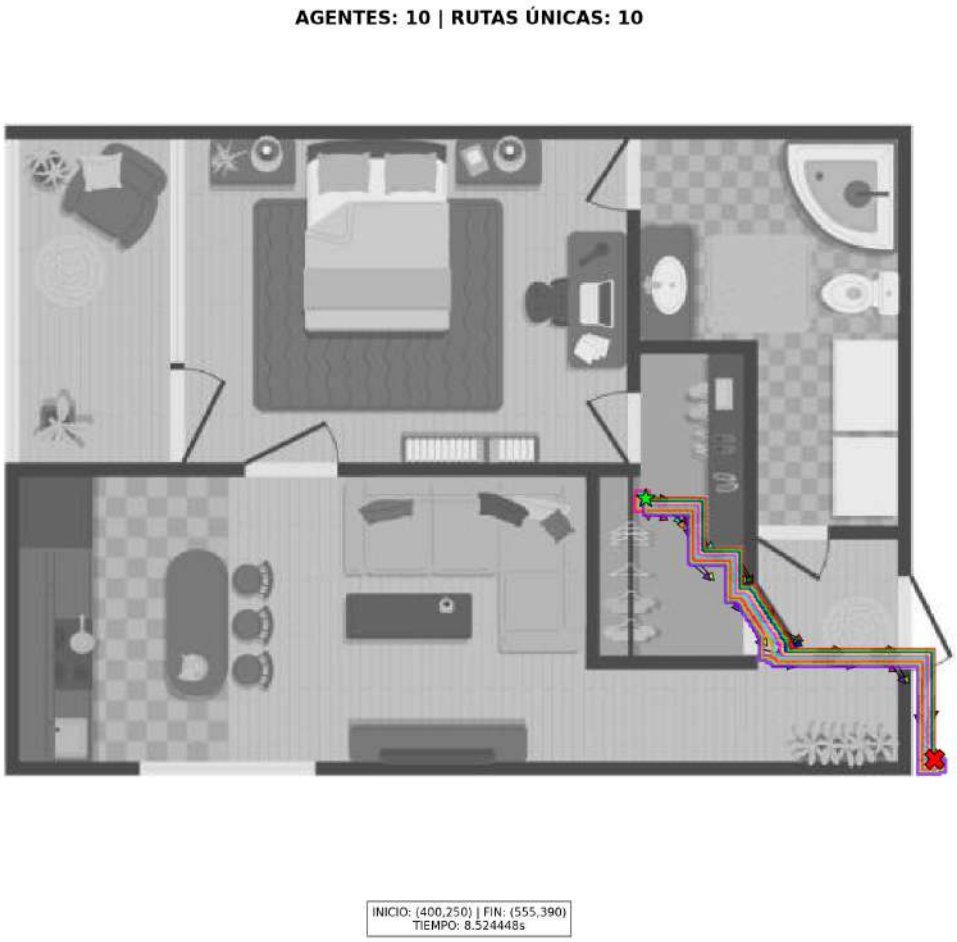


Figura 5.15: Desplazamiento del agente: evolución coordenada en eje X y en eje Y, espectro de distancias Manhattan e incidencia de giros.

Variaciones leves. Distancia promedio de 315, con desviación de solo 15.77 (comportamiento bastante uniforme). Giros promedio de 50.4, máximo 53. Saturación baja.

Mapa general de rutas únicas encontradas



El recorrido mantiene una buena eficiencia, aunque pierde cierta uniformidad al aparecer ligeras variaciones entre las trayectorias sin alterar el desplazamiento global. El algoritmo conserva soluciones cercanas a las óptimas y la baja saturación permite que el flujo se mantenga estable.

Simulación con 20 agentes

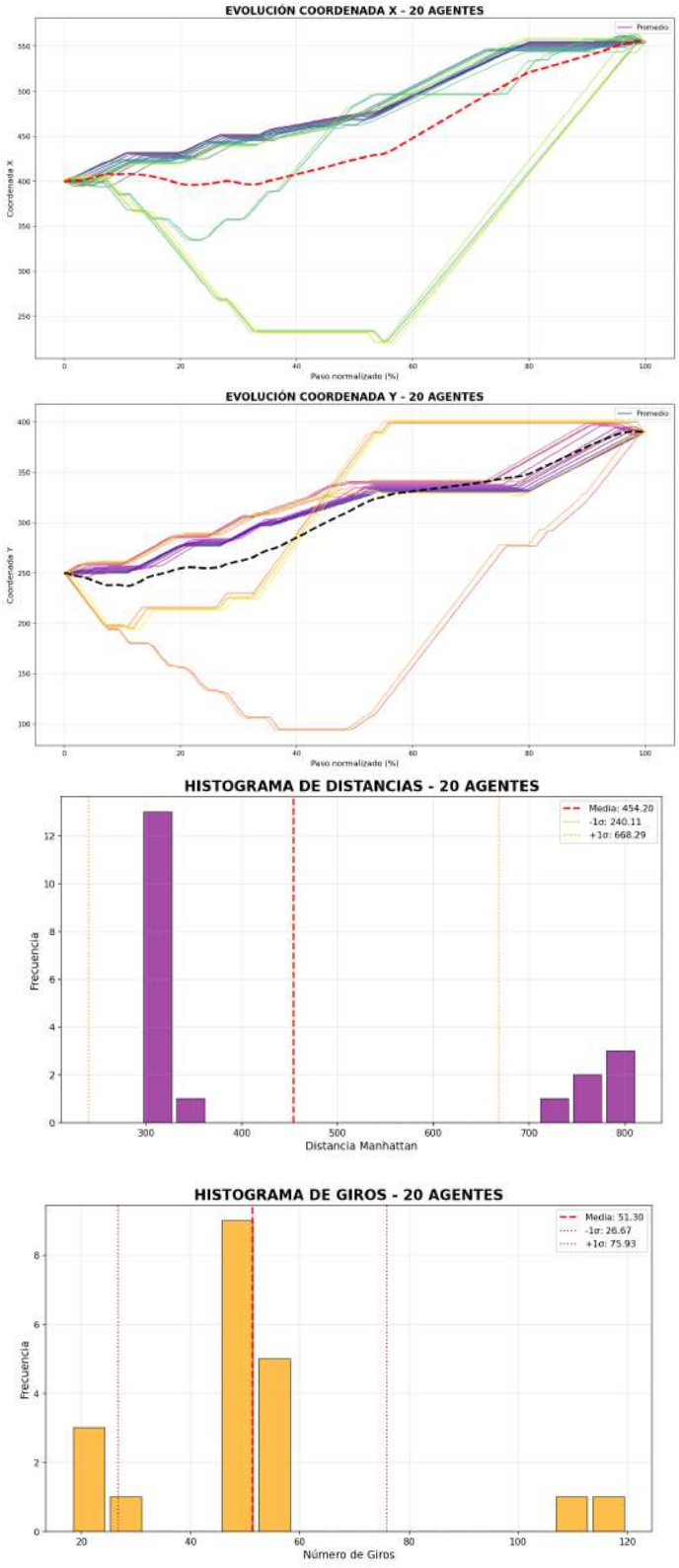
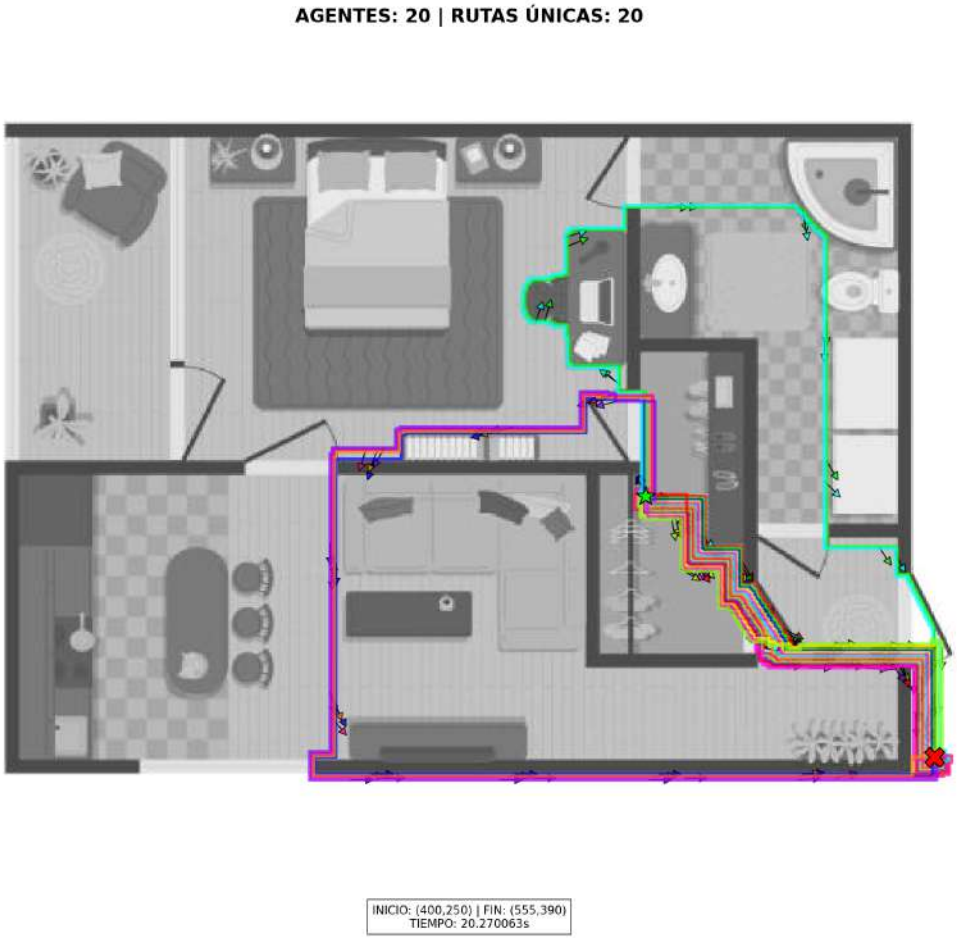


Figura 5.16: Desplazamiento del agente: evolución coordenada en eje X y en eje Y, espectro de distancias Manhattan e incidencia de giros.

Trayectorias más diversas. Distancia promedio de 454.2, pero con una desviación alta de 214.09, lo que indica una gran diferencia entre el agente que mejor ruta hace y el que peor. Giros promedio de 51.3, máximo 120. Saturación moderada.

Mapa general de rutas únicas encontradas



El recorrido muestra variaciones más perceptibles, con trayectorias que comienzan a desviarse o bordear zonas compartidas, generando mayor diversidad en los desplazamientos. El algoritmo se mantiene eficiente aunque requiere más ajustes, y la saturación empieza a hacerse visible en puntos específicos del entorno.

Simulación con 40 agentes

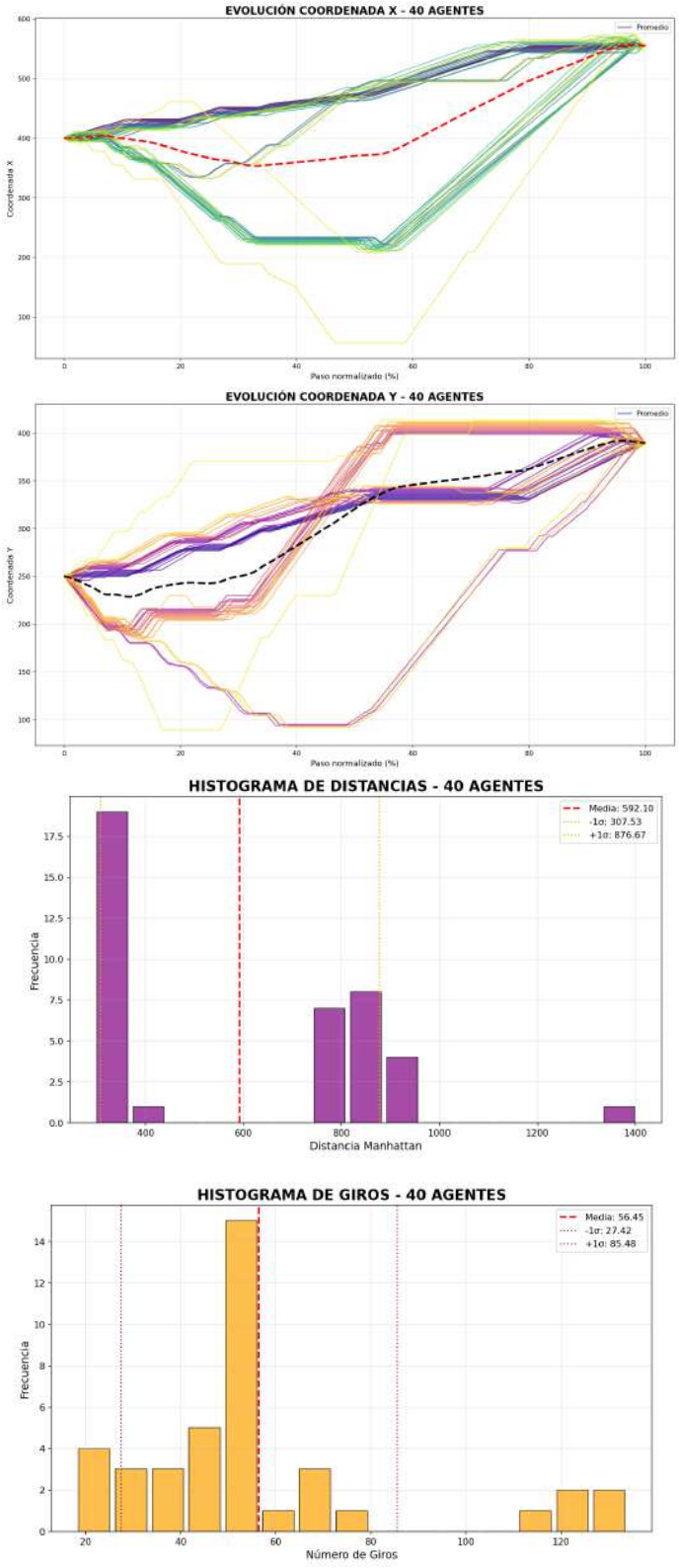


Figura 5.17: Desplazamiento del agente: evolución coordenada en eje X y en eje Y, espectro de distancias Manhattan e incidencia de giros.

Concentración en zonas específicas. Distancia promedio de 592.1 (desviación de 284.57, variabilidad muy alta). Giros promedio de 56.45, máximo 134. Saturación moderada-alta.

Mapa general de rutas únicas encontradas

El recorrido presenta concentraciones en zonas específicas donde las trayectorias convergen, generando caminos preferentes y agrupamientos de agentes. Aunque el algoritmo continúa ofreciendo rutas funcionales, estas pierden directividad y la saturación moderada reduce la fluidez en ciertas áreas.

Simulación con 80 agentes

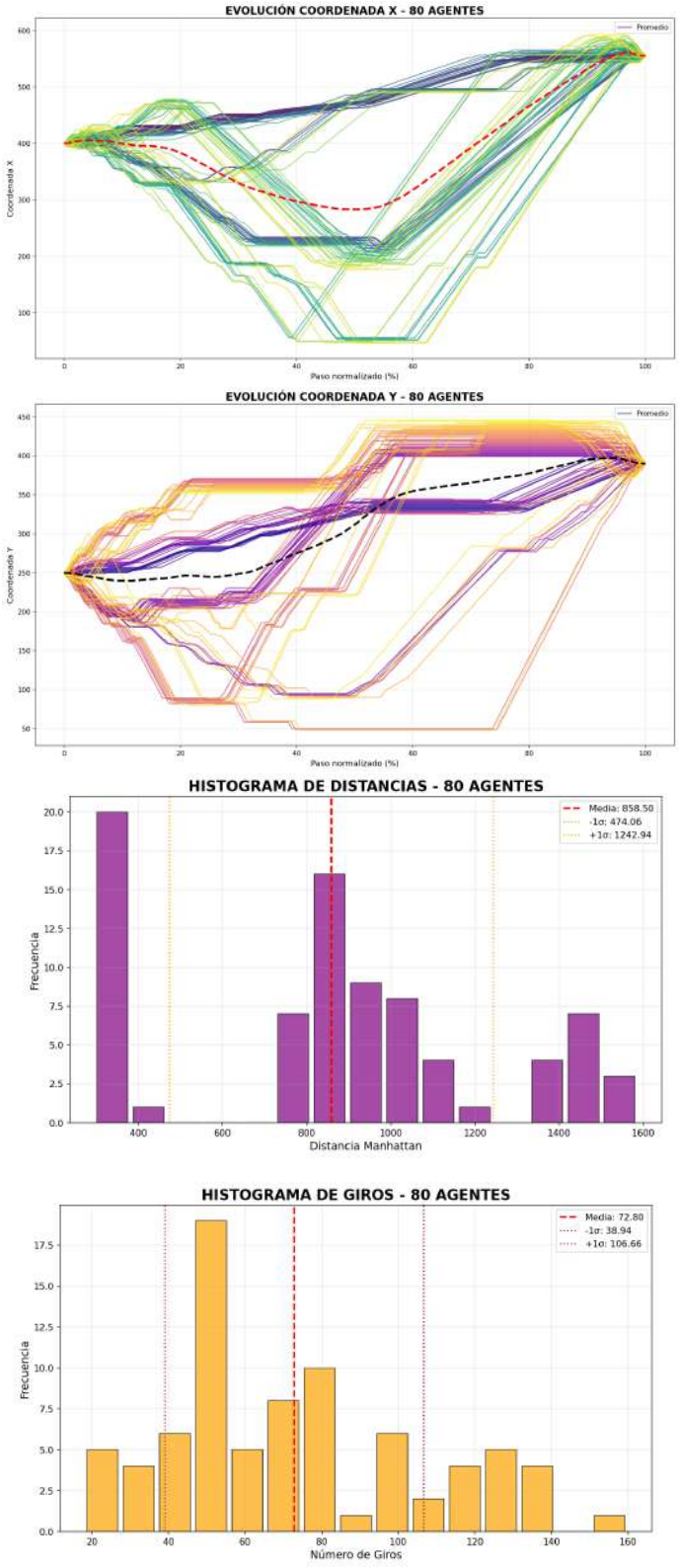
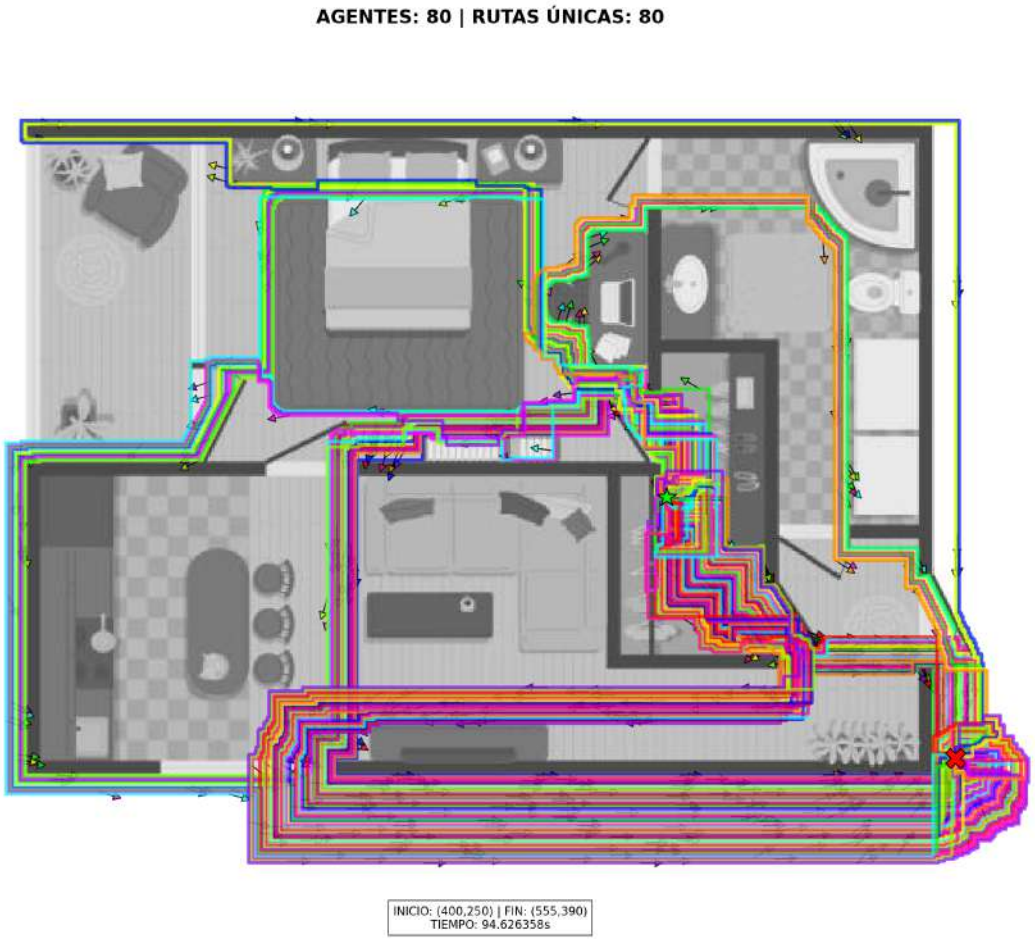


Figura 5.18: Desplazamiento del agente: evolución coordenada en eje X y en eje Y, espectro de distancias Manhattan e incidencia de giros.

Alta superposición, especialmente cerca de salida. Distancia promedio de 858.5 con una desviación enorme de 384.44, reflejando que unos pocos agentes logran rutas cortas mientras la mayoría sufre grandes rodeos por la congestión. Giros promedio de 72.8, máximo 160. Satura-

ción alta con reducción significativa de eficiencia.

Mapa general de rutas únicas encontradas



El desplazamiento se torna más concentrado y menos homogéneo, con múltiples trayectorias que se superponen, sobre todo en las cercanías de la salida, generando acumulaciones visibles. Aunque el algoritmo sigue produciendo rutas, estas presentan más desviaciones y la alta saturación disminuye de forma notable la eficiencia del flujo.

Tabla comparativa general

Para organizar los resultados de este análisis con la imagen ya existente en el primer escenario, se construyó la siguiente tabla comparativa.

Cuadro 5.4: Registro comparativo de las simulaciones

AGENTES	RUTAS TERMINADAS	TIEMPO (s)	CORR PROM	CORR MIN	CORR MAX	DIF MAX-MIN	MEDIA DIST	DESV DIST	MEDIA GIROS	MAX GIROS
1	1	0.047329	N/A	N/A	N/A	N/A	295.00	0.00	49.00	49
10	10	8.524448	0.994892	0.986442	0.999994	0.013553	315.00	15.77	50.40	53
20	20	20.270063	0.843106	0.476037	0.999994	0.523957	454.20	214.09	51.30	120
40	40	44.371167	0.781579	0.169604	0.999994	0.830390	592.10	284.57	56.45	134
80	80	94.626358	0.695716	0.128449	0.999994	0.871545	858.50	384.44	72.80	160

5.4. Comparación entre simulaciones con imagen generada e imagen existente

Al comparar las simulaciones realizadas con imágenes generadas y aquellas basadas en imágenes existentes, se pueden notar diferencias importantes en la forma en que los agentes se desplazan y en cómo responde el algoritmo A*. En el caso de las imágenes generadas, el entorno es más ordenado, lo que hace que las rutas sean más claras y directas.

Esto permite que el algoritmo funcione de manera más eficiente, ya que no hay tantas irregularidades que compliquen el movimiento. Además, la saturación aparece de forma más gradual, lo que facilita observar cómo cambia el comportamiento conforme aumenta el número de agentes. Por otro lado, cuando se trabaja con imágenes existentes, el entorno se vuelve más complejo.

Los obstáculos no siguen una estructura uniforme y esto provoca que las trayectorias sean menos predecibles. En estos casos, los agentes tienden a adaptarse constantemente, generando más desviaciones y acumulaciones, especialmente en zonas de paso reducido.

También es importante notar que el comportamiento colectivo cambia entre ambos escenarios. En las imágenes generadas, el movimiento tiende a ser más organizado, mientras que en las imágenes reales se vuelve más irregular, con zonas donde el flujo se detiene o se vuelve más lento.

En general, el algoritmo A* sigue funcionando en ambos casos, pero su desempeño se ve más afectado cuando el entorno es más complejo. Por esta razón, las imágenes generadas son útiles para entender el funcionamiento básico del modelo, mientras que las imágenes existentes permiten analizar situaciones más cercanas a la realidad.

5.4.1. Tabla comparativa

Criterio	**Imagen generada**	**Imagen existente**
Estructura del entorno	Más ordenada y uniforme	Más irregular y compleja
Trayectorias	Más directas y claras	Más variables y con desviaciones
Saturación	Aparece de forma gradual	Aparece más rápido en zonas específicas
Cuellos de botella	Menos comunes	Más frecuentes
Flujo de agentes	Más estable y organizado	Más irregular y lento en algunas zonas
Desempeño de A*	Más eficiente	Se ve más afectado por el entorno
Aplicación	Escenarios controlados	Escenarios más reales

Cuadro 5.5: Comparación entre simulaciones con imagen generada e imagen existente

5.5. Uso de script en Bash para conexión remota y optimización de simulaciones

Con el objetivo de mejorar la eficiencia en la ejecución de las simulaciones, se implementó un script en lenguaje Bash (`.sh`) que permite establecer una conexión entre la máquina local y una máquina remota. Esta estrategia surge como una necesidad práctica, ya que al trabajar únicamente en un equipo local, el tiempo de cómputo se incrementa considerablemente cuando se desea realizar un gran número de simulaciones.

El uso de este script facilita la automatización del proceso de conexión y ejecución remota, permitiendo aprovechar los recursos computacionales de otro equipo. De esta manera, las simulaciones pueden distribuirse o ejecutarse en un entorno con mayor capacidad de procesamiento, lo que reduce significativamente los tiempos de espera y mejora el flujo de trabajo.

Además, este enfoque no solo optimiza el tiempo, sino que también permite realizar un mayor número de pruebas en menor tiempo, lo cual es fundamental para el análisis y validación de los resultados obtenidos en este trabajo. En particular, resulta útil cuando se requiere explorar diferentes configuraciones o parámetros dentro del modelo de simulación.

El script desarrollado, junto con una descripción más detallada de su funcionamiento, se presenta en el **Apéndice G**, donde se incluye su estructura y los comandos utilizados para llevar a cabo la conexión y ejecución remota.

Capítulo 6

Conclusiones generales

En este trabajo se desarrolló e implementó una metodología completa para la simulación de evacuaciones de multitudes, integrando imágenes digitales como representación del entorno, el algoritmo A* y su variante CA* para la planificación de rutas, una extensión cooperativa para múltiples agentes y métodos estocásticos para el análisis de incertidumbre.

Los resultados obtenidos permiten observar de manera clara cómo el sistema evoluciona desde condiciones de baja densidad, donde las trayectorias son casi óptimas, hasta escenarios de alta saturación, donde aparecen cuellos de botella y el tiempo de evacuación se incrementa significativamente.

6.1. Alcances del trabajo

Se logró transformar imágenes digitales (planos, fotografías o mapas) en modelos computacionales discretos, representados como matrices binarias donde se distingue entre zonas transitables y obstáculos. Sobre esta representación, el algoritmo A* demostró ser una herramienta eficaz para aproximar rutas óptimas en entornos estáticos, incluso cuando la densidad de agentes aumenta.

La implementación de A* cooperativo, con su dimensión temporal, permitió coordinar múltiples agentes y reducir colisiones, mostrando que es posible planificar trayectorias para decenas de individuos sin que se bloqueen entre sí. Además, el método de Monte Carlo aportó un enfoque probabilístico útil para explorar rutas alternas y evaluar la variabilidad del sistema.

Las simulaciones realizadas con diferentes cantidades de agentes (desde 1 hasta 200 en mapas generados, y hasta 80 en mapas reales) permitieron clasificar niveles de saturación y observar que la congestión no aparece de forma abrupta, sino gradual. Esta gradualidad es un hallazgo importante, pues sugiere que el sistema tiene una capacidad de respuesta que se degrada de manera predecible al aumentar la densidad.

La cantidad de tiempo en explorar las rutas de escape dependen tanto de la geometría como del número de agentes en la simulación y puede ser un problema de complejidad computacional exponencial, para bajar los tiempo de calculo se usan algoritmos heurística y cómputo distribuido para generar diversas rutas en menor tiempo.

6.2. Limitaciones identificadas como oportunidades de mejora

Ningún modelo es completo, y este trabajo no es la excepción. Sin embargo, las limitaciones detectadas no invalidan los resultados, sino que abren líneas concretas para trabajos futuros.

1. **Condiciones ideales:** El modelo asume entornos estáticos, agentes con información completa y ausencia de obstáculos variables. Esta simplificación fue necesaria para establecer una base funcional, pero en trabajos futuros se pueden incorporar dinámicas como humo, puertas que se cierran o cambios en la iluminación o inclusive fronteras variables.

2. **Ausencia de variables físicas:** No se incluyeron velocidad, aceleración, fuerzas de interacción o inercia. Los agentes se mueven paso a paso en una cuadrícula sin considerar que las personas pueden correr, frenar o empujarse.

3. **Validación empírica pendiente:** Los resultados no se contrastaron con datos reales de evacuaciones. Esta es una de las limitaciones temporales, pero también una oportunidad porque en futuras investigaciones pueden comparar estas simulaciones con experimentos controlados o con registros de emergencias reales. Esto no está aún en los simulacros implementados en las simulaciones reales de evacuación.

6.3. Trabajos futuros

A partir de lo construido y de las limitaciones identificadas, se proponen las siguientes líneas de desarrollo:

1. Incorporar dinámicas físicas mediante modelos de fuerzas sociales o autómatas celulares con velocidad y aceleración.

2. Realizar validación empírica comparando simulaciones con datos reales de evacuaciones documentadas (mediciones de dependencias y protección civil).

3. Implementar entornos dinámicos con obstáculos variables y agentes capaces de replanificar rutas en tiempo real.

4. Escalar el modelo a entornos más grandes (centros comerciales, estadios, zonas urbanas) utilizando computación distribuida/paralela.

6.4. Reflexión final

En conjunto, este trabajo presenta una aproximación funcional a la simulación de evacuaciones utilizando herramientas accesibles y fundamentos sólidos. Se implementaron algoritmos de búsqueda de rutas, se procesaron imágenes reales, se coordinaron múltiples agentes y se exploró la incertidumbre con métodos estocásticos.

Las limitaciones identificadas no invalidan los resultados, sino que abren líneas claras para futuros desarrollos. El modelo desarrollado puede considerarse una base preliminar sobre la cual podrían construirse versiones más realistas y robustas en trabajos futuros.

Apéndice A

Representación Visual Bidimensional

```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from google.colab import files

# Subir el archivo de imagen
uploaded = files.upload()

# Obtener la ruta del archivo subido
ruta_imagen = list(uploaded.keys())[0]

# Funcin para cargar y procesar la imagen en escala de grises
def cargar_mapa_en_escala_de_grises(ruta_imagen):
    imagen = Image.open(ruta_imagen).convert("L")
    imagen_redimensionada = imagen.resize((512, 512), resample=Image.BICUBIC)
    datos = np.array(imagen_redimensionada)

    plt.figure(figsize=(6, 6), dpi=100)
    plt.imshow(imagen_redimensionada, cmap='gray', interpolation='nearest')
    plt.title("Imagen en Escala de Grises (512x512)")
    plt.axis('off')
    plt.show()

    print("Datos de la imagen (matriz 512x512 en escala de grises):")
    print(datos)

    return imagen_redimensionada, datos

# Funcin para mostrar la cuadrcula sobre la imagen y guardarla como archivo
def mostrar_mapa_cuadricula(imagen, nombre_archivo="imagen_con_cuadricula.png"):
    ancho, alto = imagen.size
    fig, ax = plt.subplots(figsize=(8, 8), dpi=100)
    ax.imshow(imagen, cmap="gray", interpolation="nearest")

    # Coordenadas de pxeles
    ax.set_xticks(np.arange(-0.5, ancho, 1), minor=True)
    ax.set_yticks(np.arange(-0.5, alto, 1), minor=True)

    # Cuadrcula alineada con los pxeles
    ax.grid(which="minor", color="red", linestyle='-', linewidth=0.5)
```

```
    ax.tick_params(which="both", bottom=False, left=False, labelbottom=False,
    labelleft=False)

    plt.title("Mapa con Cuadrcula Alineada")

    # Guardar la figura como imagen
    fig.savefig(nombre_archivo, bbox_inches='tight', pad_inches=0)
    plt.show()

    # Descargar la imagen
    files.download(nombre_archivo)

# Ejecutar
imagen, datos = cargar_mapa_en_escala_de_grises(ruta_imagen)
mostrar_mapa_cuadricula(imagen)
```

Apéndice B

Generación de mapas aleatorios con implementación de A*

```
import numpy as np
import matplotlib.pyplot as plt
import random
import heapq
from google.colab import files

# Dimensiones del mapa
ALTO, ANCHO = 50, 50

# Generar mapa vaco
mapa = np.zeros((ALTO, ANCHO))

# Funcin para colocar obstculos de tamao aleatorio
def colocar_obstaculos(cantidad):
    for _ in range(cantidad):
        x, y = random.randint(0, ALTO-1), random.randint(0, ANCHO-1)
        tamano = random.randint(1, 5)
        for i in range(tamano):
            for j in range(tamano):
                if 0 <= x+i < ALTO and 0 <= y+j < ANCHO:
                    mapa[x+i][y+j] = 1

# Colocar 30 obstculos
colocar_obstaculos(30)

# Definir posiciones de inicio y objetivo
inicio = (0, 0)
objetivo = (ALTO-1, ANCHO-1)

# Funcin heurstica (distancia de Manhattan)
def heuristica(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

# Algoritmo A*
def a_star(mapa, inicio, objetivo):
    abierto = []
    heapq.heappush(abierto, (0 + heuristica(inicio, objetivo), 0, inicio))
```

```
    came_from = {}
    g_score = {inicio: 0}
    f_score = {inicio: heuristica(inicio, objetivo)}
    explorados = []

    while abierto:
        _, current_g, current = heapq.heappop(abierto)

        if current == objetivo:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.reverse()
            return path, explorados

        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            vecino = (current[0] + dx, current[1] + dy)
            if 0 <= vecino[0] < ALTO and 0 <= vecino[1] < ANCHO and mapa[vecino[0], vecino[1]] == 0:
                tentative_g_score = current_g + 1
                if vecino not in g_score or tentative_g_score < g_score[vecino]:
                    came_from[vecino] = current
                    g_score[vecino] = tentative_g_score
                    f_score[vecino] = tentative_g_score + heuristica(vecino, objetivo)
                    heapq.heappush(abierto, (f_score[vecino], tentative_g_score, vecino))
                    explorados.append(vecino)
    return [], explorados

# Visualizar el mapa y el recorrido, y guardar como imagen descargable
def visualizar(mapa, camino, explorados, nombre_archivo="camino_a_estrella.png"):
    plt.figure(figsize=(8, 8))
    plt.imshow(mapa, cmap='gray_r')
    for (i, j) in explorados:
        plt.text(j, i, '.', ha='center', va='center', color='blue', fontsize=8)
    for (i, j) in camino:
        plt.text(j, i, 'o', ha='center', va='center', color='red', fontsize=8)
    plt.text(inicio[1], inicio[0], 'S', ha='center', va='center', color='green', fontsize=12)
    plt.text(objetivo[1], objetivo[0], 'G', ha='center', va='center', color='yellow', fontsize=12)
    plt.gca().invert_yaxis()
    plt.axis('off')
    plt.title("Algoritmo A* - Camino Encontrado")

# Ejecutar A*
camino, explorados = a_star(mapa, inicio, objetivo)

# Visualizar imagen
visualizar(mapa, camino, explorados)
```

Apéndice C

Caminos Planificados con A*

```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import heapq
from google.colab import files

# Subir archivo
uploaded = files.upload()
ruta_imagen = list(uploaded.keys())[0]

# Cargar imagen y convertir a escala de grises
def cargar_mapa_en_escala_de_grises(ruta_imagen):
    imagen = Image.open(ruta_imagen).convert("L")
    datos = np.array(imagen)

    plt.figure(figsize=(6, 6), dpi=100)
    plt.imshow(imagen, cmap='gray', interpolation='nearest')
    plt.title(f"Imagen en Escala de Grises ({datos.shape[0]}x{datos.shape[1]})")
    plt.axis('off')
    plt.show()

    print(f"Dimensiones de la imagen: {datos.shape}")
    return imagen, datos

# Generar mapa binario desde imagen
def generar_mapa_desde_imagen(datos, umbral=128):
    return np.where(datos < umbral, 1, 0)  # 1 = obstculo, 0 = camino libre

# Heurstica de Manhattan
def heuristica(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

# Algoritmo A*
def a_star(mapa, inicio, objetivo):
    abierto = []
    heapq.heappush(abierto, (heuristica(inicio, objetivo), 0, inicio))
    came_from = {}
    g_score = {inicio: 0}
    f_score = {inicio: heuristica(inicio, objetivo)}
    explorados = []
```

```
41
42      while abierto:
43          _, current_g, current = heapq.heappop(abierto)
44
45          if current == objetivo:
46              path = []
47              while current in came_from:
48                  path.append(current)
49                  current = came_from[current]
50              path.reverse()
51              return path, explorados
52
53          for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
54              vecino = (current[0] + dx, current[1] + dy)
55              if 0 <= vecino[0] < mapa.shape[0] and 0 <= vecino[1] < mapa.shape[1] and
    mapa[vecino[0], vecino[1]] == 0:
56                  tentative_g_score = current_g + 1
57                  if vecino not in g_score or tentative_g_score < g_score[vecino]:
58                      came_from[vecino] = current
59                      g_score[vecino] = tentative_g_score
60                      f_score[vecino] = tentative_g_score + heuristica(vecino, objetivo
    )
61                      heapq.heappush(abierto, (f_score[vecino], tentative_g_score,
    vecino))
62                      explorados.append(vecino)
63      return [], explorados
64
65  # Ingresar coordenadas de inicio y objetivo, verificando que estn en camino libre
66  def ingresar_coordenadas(mapa):
67      filas, columnas = mapa.shape
68
69      while True:
70          print("Ingrese coordenadas de entrada (inicio):")
71          x = int(input(f"X (0 a {filas-1}): "))
72          y = int(input(f"Y (0 a {columnas-1}): "))
73          if 0 <= x < filas and 0 <= y < columnas and mapa[x, y] == 0:
74              inicio = (x, y)
75              break
76          print(" Coordenada no vlida o est en un obstculo.")
77
78      while True:
79          print("Ingrese coordenadas de salida (objetivo):")
80          x = int(input(f"X (0 a {filas-1}): "))
81          y = int(input(f"Y (0 a {columnas-1}): "))
82          if 0 <= x < filas and 0 <= y < columnas and mapa[x, y] == 0:
83              objetivo = (x, y)
84              break
85          print(" Coordenada no vlida o est en un obstculo.")
86
87      return inicio, objetivo
88
89  # Visualizacin del camino
90  def visualizar(mapa, camino, explorados, inicio, objetivo):
```

```
    plt.figure(figsize=(8, 8))
    plt.imshow(mapa, cmap='gray_r')

    for (i, j) in explorados:
        plt.text(j, i, '.', ha='center', va='center', color='blue', fontsize=4)
    for (i, j) in camino:
        plt.text(j, i, 'o', ha='center', va='center', color='red', fontsize=4)

    plt.text(inicio[1], inicio[0], 'S', ha='center', va='center', color='green',
    fontsize=8)
    plt.text(objetivo[1], objetivo[0], 'G', ha='center', va='center', color='yellow',
     fontsize=8)

    plt.gca().invert_yaxis()
    plt.axis('off')
    plt.title("Camino encontrado con A*")
    plt.show()

# Ejecutar flujo principal
imagen, datos = cargar_mapa_en_escala_de_grises(ruta_imagen)
mapa = generar_mapa_desde_imagen(datos)
inicio, objetivo = ingresar_coordenadas(mapa)
camino, explorados = a_star(mapa, inicio, objetivo)
visualizar(mapa, camino, explorados, inicio, objetivo)
```

Apéndice D

Trayectorias Óptimas en Entornos Compartidos

```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw
import heapq
from google.colab import files

# Subir imagen
uploaded = files.upload()
ruta_imagen = list(uploaded.keys())[0]

# Cargar imagen en escala de grises
def cargar_mapa_en_escala_de_grises(ruta_imagen):
    imagen = Image.open(ruta_imagen).convert("L")
    datos = np.array(imagen)
    print(f"Dimensiones de la imagen: {datos.shape}")
    return imagen, datos

# Convertir imagen a mapa binario
def generar_mapa_desde_imagen(datos, umbral=128):
    return np.where(datos < umbral, 1, 0)  # 1 = obstculo, 0 = libre

# Heurstica Manhattan
def heuristica(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

# Reservaciones espacio-tiempo
class Reservaciones:
    def __init__(self):
        self.reservado = {}

    def reservar(self, posicion, tiempo):
        self.reservado[(posicion, tiempo)] = True

    def esta_reservado(self, posicion, tiempo):
        return self.reservado.get((posicion, tiempo), False)

    def reservar_camino(self, camino):
```

```
        for t, pos in enumerate(camino):
            self.reservar(pos, t)

# A* cooperativo
def a_star_cooperativo(mapa, inicio, objetivo, reservaciones):
    abierto = []
    heapq.heappush(abierto, (heuristica(inicio, objetivo), 0, inicio, 0))
    came_from = {}
    g_score = {(inicio, 0): 0}
    f_score = {(inicio, 0): heuristica(inicio, objetivo)}
    explorados = []

    while abierto:
        _, g, actual, tiempo = heapq.heappop(abierto)

        if actual == objetivo:
            path = []
            while (actual, tiempo) in came_from:
                path.append(actual)
                actual, tiempo = came_from[(actual, tiempo)]
            path.append(inicio)
            path.reverse()
            return path, explorados

        for dx, dy in [(-1,0), (1,0), (0,-1), (0,1), (0,0)]:
            vecino = (actual[0] + dx, actual[1] + dy)
            nuevo_tiempo = tiempo + 1

            if 0 <= vecino[0] < mapa.shape[0] and 0 <= vecino[1] < mapa.shape[1] and
    mapa[vecino[0], vecino[1]] == 0:
                if reservaciones.esta_reservado(vecino, nuevo_tiempo):
                    continue

                if ((vecino, nuevo_tiempo) not in g_score) or (g + 1 < g_score[(
    vecino, nuevo_tiempo)]):
                    came_from[(vecino, nuevo_tiempo)] = (actual, tiempo)
                    g_score[(vecino, nuevo_tiempo)] = g + 1
                    f_score[(vecino, nuevo_tiempo)] = g + 1 + heuristica(vecino,
    objetivo)
                    heapq.heappush(abierto, (f_score[(vecino, nuevo_tiempo)], g + 1,
    vecino, nuevo_tiempo))
                    explorados.append(vecino)

    return [], explorados

# Ingreso de coordenadas mltiples
def ingresar_coordenadas_multiples(mapa, n):
    filas, columnas = mapa.shape
    agentes = []

    for i in range(n):
        print(f"\n Agente #{i+1}")
        while True:
```

```
            print("Ingrese coordenadas de entrada:")
            x = int(input(f"X (0 a {filas-1}): "))
            y = int(input(f"Y (0 a {columnas-1}): "))
            if 0 <= x < filas and 0 <= y < columnas and mapa[x, y] == 0:
                inicio = (x, y)
                break
            print(" Coordenada invlida o bloqueada.")

        while True:
            print("Ingrese coordenadas de objetivo:")
            x = int(input(f"X (0 a {filas-1}): "))
            y = int(input(f"Y (0 a {columnas-1}): "))
            if 0 <= x < filas and 0 <= y < columnas and mapa[x, y] == 0:
                objetivo = (x, y)
                break
            print(" Coordenada invlida o bloqueada.")

        agentes.append((inicio, objetivo))

    return agentes

# Dibuja trayectorias con bloques visibles (sin descargar)
def dibujar_trayectorias_en_imagen(imagen_original, caminos, agentes, nombre_archivo="trayectorias_agentes.png"):
    imagen_color = imagen_original.convert("RGB")
    draw = ImageDraw.Draw(imagen_color)
    colores = ['red', 'blue', 'green', 'orange', 'purple', 'cyan', 'magenta', 'lime', 'turquoise', 'brown']

    for idx, camino in enumerate(caminos):
        color = colores[idx % len(colores)]

        # Dibujar trayectoria como bloques (visibles)
        for punto in camino:
            x, y = punto[1], punto[0]  # PIL usa (x, y) = (columna, fila)
            draw.rectangle([x-1, y-1, x+1, y+1], fill=color)

        # Dibujar inicio y objetivo
        inicio, objetivo = agentes[idx]
        draw.ellipse([(inicio[1]-3, inicio[0]-3), (inicio[1]+3, inicio[0]+3)], fill='green')
        draw.ellipse([(objetivo[1]-3, objetivo[0]-3), (objetivo[1]+3, objetivo[0]+3)], fill='yellow')

    # Mostrar resultado
    plt.figure(figsize=(6, 6))
    plt.imshow(imagen_color)
    plt.title("Trayectorias de Mltiples Agentes (A* Cooperativo)")
    plt.axis('off')
    plt.show()

    # Guardar la imagen (opcional, se puede comentar esta lnea si no quieres guardar nada)
```

```
    # imagen_color.save(nombre_archivo)
    # print(f" Imagen guardada como '{nombre_archivo}'")

#  Flujo principal
imagen, datos = cargar_mapa_en_escala_de_grises(ruta_imagen)
mapa = generar_mapa_desde_imagen(datos)

num_agentes = int(input("Cuntos agentes deseas planificar? "))
agentes = ingresar_coordenadas_multiples(mapa, num_agentes)

reservaciones = Reservaciones()
caminos = []
explorados_list = []

for inicio, objetivo in agentes:
    camino, explorados = a_star_cooperativo(mapa, inicio, objetivo, reservaciones)
    if not camino:
        print(f" No se encontr camino para el agente de {inicio} a {objetivo}")
    else:
        print(f" Camino encontrado para el agente de {inicio} a {objetivo} ({len(
    camino)} pasos)")
        reservaciones.reservar_camino(camino)
        caminos.append(camino)
        explorados_list.append(explorados)

# Mostrar imagen con trayectorias
dibujar_trayectorias_en_imagen(imagen, caminos, agentes)
```

Apéndice E

Ruteo Cooperativo por Etapas en Mapas

```python
import numpy as np
from PIL import Image, ImageDraw
import heapq

def cargar_mapa_en_escala_de_grises(ruta_imagen):
    imagen = Image.open(ruta_imagen).convert("L")
    datos = np.array(imagen)
    print(f" Dimensiones de la imagen: {datos.shape}")
    return imagen, datos

def generar_mapa_desde_imagen(datos, umbral=128):
    return np.where(datos < umbral, 1, 0)  # 1 = obstculo, 0 = libre

def heuristica(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

class Reservaciones:
    def __init__(self):
        self.reservado = {}

    def reservar(self, posicion, tiempo):
        self.reservado[(posicion, tiempo)] = True

    def esta_reservado(self, posicion, tiempo):
        return self.reservado.get((posicion, tiempo), False)

    def reservar_camino(self, camino):
        for t, pos in enumerate(camino):
            self.reservar(pos, t)

def a_star_cooperativo(mapa, inicio, objetivo, reservaciones, tiempo_inicio=0):
    abierto = []
    heapq.heappush(abierto, (heuristica(inicio, objetivo), 0, inicio, tiempo_inicio))
    came_from = {}
    g_score = {(inicio, tiempo_inicio): 0}

    while abierto:
        _, g, actual, tiempo = heapq.heappop(abierto)

        if actual == objetivo:
```

```
            path = []
            key = (actual, tiempo)
            while key in came_from:
                path.append(key[0])
                key = came_from[key]
            path.append(inicio)
            path.reverse()
            return path

        for dx, dy in [(-1,0), (1,0), (0,-1), (0,1), (0,0)]:
            vecino = (actual[0] + dx, actual[1] + dy)
            nuevo_tiempo = tiempo + 1

            if 0 <= vecino[0] < mapa.shape[0] and 0 <= vecino[1] < mapa.shape[1] and
    mapa[vecino[0], vecino[1]] == 0:
                if reservaciones.esta_reservado(vecino, nuevo_tiempo):
                    continue

                if ((vecino, nuevo_tiempo) not in g_score) or (g + 1 < g_score[(
    vecino, nuevo_tiempo)]):
                    came_from[(vecino, nuevo_tiempo)] = (actual, tiempo)
                    g_score[(vecino, nuevo_tiempo)] = g + 1
                    f = g + 1 + heuristica(vecino, objetivo)
                    heapq.heappush(abierto, (f, g + 1, vecino, nuevo_tiempo))

    return []

def buscar_punto_libre_cercano(punto, mapa, max_radio=10):
    x, y = punto
    for r in range(max_radio + 1):
        for dx in range(-r, r + 1):
            for dy in range(-r, r + 1):
                nx, ny = x + dx, y + dy
                if 0 <= nx < mapa.shape[0] and 0 <= ny < mapa.shape[1]:
                    if mapa[nx, ny] == 0:
                        return (nx, ny)
    return punto

def calcular_ruta_con_fallback(mapa, inicio, objetivo, reservaciones, tiempo_inicio
    =0):
    ruta = a_star_cooperativo(mapa, inicio, objetivo, reservaciones, tiempo_inicio)
    if ruta:
        return ruta

    print(f" Ruta directa {inicio}  {objetivo} no encontrada. Intentando con punto
    intermedio...")
    alto, ancho = mapa.shape
    punto_medio = (alto // 2, ancho // 2)

    if mapa[punto_medio[0], punto_medio[1]] != 0:
        punto_medio = buscar_punto_libre_cercano(punto_medio, mapa)
        print(f" Punto intermedio ajustado a {punto_medio}")

```

```
    ruta1 = a_star_cooperativo(mapa, inicio, punto_medio, reservaciones,
    tiempo_inicio)
    if not ruta1:
        print(" No se pudo conectar al punto intermedio.")
        return []

    # Reservar ruta1 temporalmente para evitar colisiones
    for t, pos in enumerate(ruta1):
        reservaciones.reservar(pos, tiempo_inicio + t)

    ruta2 = a_star_cooperativo(mapa, punto_medio, objetivo, reservaciones,
    tiempo_inicio + len(ruta1) - 1)
    if not ruta2:
        print(" Solo se logr la mitad del camino.")
        return ruta1

    return ruta1 + ruta2[1:]

def dibujar_trayectorias_en_imagen(imagen, rutas, agentes, nombre_archivo="
    ruta_multiple_tramos.png"):
    imagen_color = imagen.convert("RGB")
    draw = ImageDraw.Draw(imagen_color)
    colores = ['red', 'blue', 'green', 'orange', 'purple', 'cyan', 'magenta', 'yellow
    ', 'brown', 'pink']

    for i, ruta in enumerate(rutas):
        color = colores[i % len(colores)]
        for p in ruta:
            x, y = p[1], p[0]
            draw.rectangle([x-1, y-1, x+1, y+1], fill=color)

    for inicio, fin in agentes:
        draw.ellipse([(inicio[1]-3, inicio[0]-3), (inicio[1]+3, inicio[0]+3)], fill='
    green')
        draw.ellipse([(fin[1]-3, fin[0]-3), (fin[1]+3, fin[0]+3)], fill='yellow')

    imagen_color.save(nombre_archivo)
    print(f" Imagen guardada como: {nombre_archivo}")

def main():
    print("=== Planificador de rutas cooperativas con n tramos por agente ===")

    ruta_imagen = input(" Ruta del mapa (imagen JPG o PNG): ").strip()

    try:
        n_agentes = int(input(" Nmero de agentes a planificar: "))
        if n_agentes <= 0:
            print(" Debe ser mayor que 0.")
            return
    except ValueError:
        print(" Entrada invlida.")
        return

```

```
    imagen, datos = cargar_mapa_en_escala_de_grises(ruta_imagen)
    mapa = generar_mapa_desde_imagen(datos)

    rutas_completas = []
    agentes = []
    reserv = Reservaciones()

    for i in range(n_agentes):
        print(f"\n Agente {i+1}:")
        try:
            x_inicio = int(input(f"   Coordenada X punto inicial: "))
            y_inicio = int(input(f"   Coordenada Y punto inicial: "))
            n_tramos = int(input(f"   Nmero de tramos (destinos) para este agente: ")
    )
            if n_tramos <= 0:
                print(" Nmero de tramos debe ser > 0.")
                return
        except ValueError:
            print(" Entrada invlida.")
            return

        inicio = (y_inicio, x_inicio)
        if mapa[inicio[0], inicio[1]] != 0:
            inicio = buscar_punto_libre_cercano(inicio, mapa)
            print(f" Punto inicial ajustado a {inicio}")

        destinos = []
        for t in range(n_tramos):
            try:
                x_d = int(input(f"     Coordenada X destino {t+1}: "))
                y_d = int(input(f"     Coordenada Y destino {t+1}: "))
                destino = (y_d, x_d)
                if mapa[destino[0], destino[1]] != 0:
                    destino = buscar_punto_libre_cercano(destino, mapa)
                    print(f"     Destino ajustado a {destino}")
                destinos.append(destino)
            except ValueError:
                print(" Entrada invlida.")
                return

        ruta_agente = []
        tiempo_actual = 0
        punto_origen = inicio

        for destino in destinos:
            tramo = calcular_ruta_con_fallback(mapa, punto_origen, destino, reserv,
    tiempo_actual)
            if not tramo:
                print(f" No se pudo calcular tramo {punto_origen}  {destino} para
    agente {i+1}.")
                return
            # Ajustamos tiempos y reservamos posiciones
            for t, pos in enumerate(tramo):
```

```
                reserv.reservar(pos, tiempo_actual + t)
            # Concatenamos ruta sin repetir el punto origen en siguientes tramos
            if ruta_agente:
                ruta_agente += tramo[1:]
            else:
                ruta_agente += tramo
            tiempo_actual += len(tramo) - 1
            punto_origen = destino

        rutas_completas.append(ruta_agente)
        agentes.append((inicio, destinos[-1]))

    nombre_salida = input("\n Nombre del archivo de salida (enter para 'ruta_multiple_tramos.png'): ").strip()
    if not nombre_salida:
        nombre_salida = "ruta_multiple_tramos.png"

    dibujar_trayectorias_en_imagen(imagen, rutas_completas, agentes, nombre_salida)

if __name__ == "__main__":
    main()
```

Apéndice F

metodo montecarlo

```python
import numpy as np
import random
from PIL import Image, ImageDraw
import heapq

def cargar_mapa(ruta_imagen, umbral=128):
    img = Image.open(ruta_imagen).convert("L")
    arr = np.array(img)
    mapa = np.where(arr < umbral, 1, 0)  # 1 = obstculo, 0 = libre
    return img, mapa

def heuristica(a, b):
    # Distancia Manhattan
    return abs(a[0]-b[0]) + abs(a[1]-b[1])

def a_star(mapa, inicio, fin):
    """Ruta ptima (clsica) A* sin consideraciones temporales ni Monte Carlo."""
    abierto = []
    heapq.heappush(abierto, (heuristica(inicio, fin), 0, inicio))
    came_from = {}
    g_score = {inicio: 0}
    visited = set()

    while abierto:
        f, g, actual = heapq.heappop(abierto)
        if actual == fin:
            # reconstruir ruta
            ruta = []
            node = fin
            while node in came_from:
                ruta.append(node)
                node = came_from[node]
            ruta.append(inicio)
            ruta.reverse()
            return ruta

        visited.add(actual)

        for dx, dy in [(-1,0),(1,0),(0,-1),(0,1)]:
            vecino = (actual[0] + dx, actual[1] + dy)
```

```
            if vecino[0] < 0 or vecino[0] >= mapa.shape[0] or vecino[1] < 0 or vecino
    [1] >= mapa.shape[1]:
                continue
            if mapa[vecino[0], vecino[1]] == 1:
                continue  # obstculo
            tentative_g = g + 1
            if vecino in visited and tentative_g >= g_score.get(vecino, float('inf'))
    :
                continue
            if tentative_g < g_score.get(vecino, float('inf')):
                came_from[vecino] = actual
                g_score[vecino] = tentative_g
                f2 = tentative_g + heuristica(vecino, fin)
                heapq.heappush(abierto, (f2, tentative_g, vecino))

    return None  # sin ruta

def costo_ruta(mapa, ruta):
    """Costo de ruta: longitud ms penalizacin por proximidad a obstculos."""
    if ruta is None:
        return float('inf')
    costo = 0
    for (x, y) in ruta:
        costo += 1
        # Penalizacin si est cerca de obstculos
        # verificar vecinos en un radio pequeo
        radio = 2
        for dx in range(-radio, radio+1):
            for dy in range(-radio, radio+1):
                nx, ny = x+dx, y+dy
                if 0 <= nx < mapa.shape[0] and 0 <= ny < mapa.shape[1]:
                    if mapa[nx, ny] == 1:
                        costo += 0.5  # penalizacin
    return costo

def generar_ruta_monte_carlo(mapa, inicio, fin, num_muestras=50):
    mejores_ruta = None
    mejor_costo = float('inf')

    for i in range(num_muestras):
        # Generar un punto intermedio aleatorio (o varios)
        hx = random.randint(0, mapa.shape[0]-1)
        hy = random.randint(0, mapa.shape[1]-1)
        punto_intermedio = (hx, hy)

        # Asegurarse de que el punto intermedio est libre
        if mapa[hx, hy] == 1:
            continue

        # Ruta parte 1
        ruta1 = a_star(mapa, inicio, punto_intermedio)
        # Ruta parte 2
        ruta2 = a_star(mapa, punto_intermedio, fin)
```

```
92          if ruta1 is None or ruta2 is None:
93              continue
94
95          # Concatenar (evitando duplicar el punto intermedio)
96          ruta_total = ruta1 + ruta2[1:]
97
98          # Calcular costo
99          c = costo_ruta(mapa, ruta_total)
100         if c < mejor_costo:
101             mejor_costo = c
102             mejores_ruta = ruta_total
103
104     # Tambin comparar la ruta directa A*
105     ruta_directa = a_star(mapa, inicio, fin)
106     cdir = costo_ruta(mapa, ruta_directa)
107     if cdir < mejor_costo:
108         mejores_ruta = ruta_directa
109         mejor_costo = cdir
110
111     return mejores_ruta
112
113 def dibujar_ruta(imagen, ruta, inicio, fin, nombre_salida="ruta_evacuacion.png"):
114     img = imagen.convert("RGB")
115     draw = ImageDraw.Draw(img)
116     # Dibujar ruta
117     if ruta:
118         for (x, y) in ruta:
119             draw.rectangle([y-1, x-1, y+1, x+1], fill="red")
120     # Dibujar inicio y fin
121     draw.ellipse([(inicio[1]-3, inicio[0]-3), (inicio[1]+3, inicio[0]+3)], fill="
    green")
122     draw.ellipse([(fin[1]-3, fin[0]-3), (fin[1]+3, fin[0]+3)], fill="blue")
123     img.save(nombre_salida)
124     print(f"Ruta dibujada guardada como: {nombre_salida}")
125
126 def main():
127     ruta_img = input("Ruta imagen mapa: ")
128     img, mapa = cargar_mapa(ruta_img)
129
130     try:
131         x0 = int(input("X inicio: "))
132         y0 = int(input("Y inicio: "))
133         xf = int(input("X fin: "))
134         yf = int(input("Y fin: "))
135     except ValueError:
136         print("Coordenadas invlidas.")
137         return
138
139     inicio = (y0, x0)
140     fin = (yf, xf)
141
142     ruta = generar_ruta_monte_carlo(mapa, inicio, fin, num_muestras=100)
143     if ruta is None:
```

```
144         print("No se encontr ruta de evacuacin.")
145     else:
146         print("Ruta encontrada de longitud:", len(ruta))
147         dibujar_ruta(img, ruta, inicio, fin)
148 
149 if __name__ == "__main__":
150     main()
```

Apéndice G

Script utilizado para la conexión remota

```
#!/bin/bash

# Definir las mquinas y TODOS los parmetros del script Python
# Formato: "usuario@ip"|"contrasea"|"imagen"|"valores"|"param4"|"param5"|"param6"|"param7"|"opciones_adicionales"
configuraciones=(
"USUARIO@172.17.133.14|USUARIO|mapa.jpg|1,10,20,40,80|400|250|555|390|--no-anim --auto-zip"
)

# Funcin para ejecutar comandos en una mquina remota
run_simulation() {
    local machine=$1
    local password=$2
    local imagen=$3
    local valores=$4
    local param4=$5
    local param5=$6
    local param6=$7
    local param7=$8
    local opciones=$9

    echo "========================================="
    echo "Ejecutando en: $machine"
    echo "Comando: python3 evacuacion_remota.py $imagen '$valores' $param4 $param5 $param6 $param7 $opciones"
    echo "========================================="

    # Comandos a ejecutar en la mquina remota
    sshpass -p "$password" ssh -o StrictHostKeyChecking=no "$machine" "
        cd Descargas/progra 2>/dev/null || (cd Descargas && mkdir -p progra && cd progra)
        python3 evacuacion_remota.py $imagen '$valores' $param4 $param5 $param6 $param7 $opciones
        exit
    "

    # Verificar si el comando anterior fue exitoso
    if [ $? -eq 0 ]; then
        echo " Simulacin completada en $machine"
```

```
36
37          # Copiar los resultados
38          echo "Copiando archivos zip desde $machine..."
39          sshpass -p "$password" scp -o StrictHostKeyChecking=no "$machine:/home/${
    machine%@*}/Descargas/progra/resultados_evacuacion_*.zip" ./pruebas/ 2>/dev/null
40
41          if [ $? -eq 0 ]; then
42              echo " Archivos copiados exitosamente desde $machine"
43          else
44              echo " No se encontraron archivos zip para copiar desde $machine"
45          fi
46      else
47          echo " Error en la simulacin de $machine"
48      fi
49
50      echo ""
51  }
52
53  # Crear directorio de pruebas si no existe
54  mkdir -p pruebas
55
56  # Verificar que sshpass est instalado
57  if ! command -v sshpass &> /dev/null; then
58      echo "Error: sshpass no est instalado."
59      echo "Instlalo con: sudo apt-get install sshpass (Ubuntu/Debian)"
60      echo "o: sudo yum install sshpass (CentOS/RHEL)"
61      exit 1
62  fi
63
64  # Ejecutar simulaciones en todas las mquinas
65  for config in "${configuraciones[@]}"; do
66      IFS='|' read -r machine password imagen valores param4 param5 param6 param7
    opciones <<< "$config"
67      run_simulation "$machine" "$password" "$imagen" "$valores" "$param4" "$param5" "
    $param6" "$param7" "$opciones"
68  done
69
70  echo "========================================="
71  echo "Todas las simulaciones han finalizado"
72  echo "Los resultados estn en el directorio: pruebas/"
73  echo "========================================="
74
75  # Listar archivos copiados
76  ls -la pruebas/resultados_evacuacion_*.zip 2>/dev/null || echo "No se encontraron
    archivos zip en el directorio pruebas/"
```

Referencias

Referencias generales

Hernández, A. (2015). *Simulación de multitudes y evacuaciones en espacios públicos*. Editorial Universitaria.

Ruiz, M. (2017). Modelos y simulaciones en la seguridad pública. *Revista de Ciencias de la Seguridad*.

Martínez, L. (2016). Evacuación y simulación de multitudes: Aplicaciones y estudios de caso. *J. de Arquitectura y Seguridad*.

Gómez, J. (2020). Estudio de simulación de evacuación en el STC Metro de la Ciudad de México. *Revista Mexicana de Transporte y Seguridad*.

Morales, R. (2018). Simulación de multitudes en aeropuertos: El caso del Aeropuerto Internacional de la Ciudad de México. *J. de Investigación en Seguridad Aérea*.

Rodríguez, M. (2019). Simulación de evacuación en centros comerciales: Caso Cancún. *Revista de Seguridad y Emergencias*.

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences, 99*(3), 7280-7287. https://www.pnas.org/doi/10.1073/pnas.082080899

Epstein, J. M. (2005). Modeling civil violence: An agent-based computational approach. *Proceedings of the National Academy of Sciences, 99*(suppl 3), 7243-7250. https://www.pnas.org/doi/10.1073/pnas.092080199

Farmer, J. D. (2005). Market forces and agent-based modeling. In Handbook of Computational Economics, Vol. 2, pp. 1039-1077. Elsevier.

Helbing, D. (2000). Simulating crowd dynamics. In J.-S. Yang (Ed.), *Proceedings of the 5th International Symposium on Social Behavioural Research* (pp. 1-10). World Scientific Publishing.

Helbing, D. (2012). *Social self-organization: Agent-based simulations and experiments to study emergent social behavior*. Springer.

Nagel, K., & Schreckenberg, M. (1992). A cellular automaton model for freeway traffic. *Journal of Physics I, 2*(12), 2221-2229. https://doi.org/10.1051/jp1:1992222

Gonzalez, R. C., & Woods, R. E. (1992). *Digital Image Processing* (3rd ed.). Prentice Hall. Disponible en https://sde.uoc.ac.in/sites/default/files/sde_videos/Digital%20Image%20Processing%203rd%20ed.%20-%20R.%20Gonzalez%2C%20R.%20Woods-ilovepdf-compressed.pdf

Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th ed., Global Edition). Pearson. Disponible en https://www.cl72.org/090imagePLib/books/Gonzales%2CWoods-Digital.Image.Processing.4th.Edition.pdf

Boyat, A. K., & Joshi, B. K. (2015). A review paper: Noise models in digital image processing. *arXiv preprint arXiv:1505.03489*. Disponible en https://arxiv.org/abs/1505.03489

Semboloni, F. (2007). The Social Force Model in Pedestrian Simulation. *Mathematical and Computer Modelling*, 45(5–6), 617–626. Versión gratuita disponible en ResearchGate: https://www.researchgate.net/publication/222382637_The_Social_Force_Model_in_Pedestrian_Simulation.

Helbing, D., & Molnár, P. (1995). Social force model for pedestrian dynamics. *Physical Review E*, 51(5), 4282. Accesible en: https://journals.aps.org/pre/abstract/10.1103/PhysRevE.51.4282.

Wikipedia contributors. (2025). Manhattan distance. Disponible en: https://en.wikipedia.org/wiki/Taxicab_geometry.

Fruin, J. J. (1971). *Pedestrian Planning and Design*. Metropolitan Association of Urban Designers and Environmental Planners.

Referencias de los cursos MIT:

Edelman, A. (2004, September 14). *Las matemáticas de matrices aleatorias infinitas: Histograma* [Video]. MIT OpenCourseWare. Recuperado de https://ocw.mit.edu.

Edelman, A. (2004, September 14). *Las matemáticas de matrices aleatorias infinitas: Experimentos con los conjuntos clásicos* [Video]. MIT OpenCourseWare. Recuperado de https://ocw.mit.edu.

Sutton, B. (2004, September 23). *Las matemáticas de matrices aleatorias infinitas: Matrices tridiagonales, polinomios ortogonales y lo clásico* [Video]. MIT OpenCourseWare. Recuperado de https://ocw.mit.edu.

Edelman, A. (2004, September 30). *Las matemáticas de matrices aleatorias infinitas: Ideas para proyectos* [Video]. MIT OpenCourseWare. Recuperado de https://ocw.mit.edu.

Rycroft, C. H. (2005, February 1). *Introducción a los paseos aleatorios y la difusión* [Video]. MIT OpenCourseWare. Recuperado de https://ocw.mit.edu.

Bird, J. (2003, February 8). *Teorema del límite central* [Video]. Harvard Division of Engineering and Applied Sciences. Recuperado de https://www.seas.harvard.edu.

Allen, E. (2005, February 10). *Asintóticas en la Región Central* [Video]. MIT OpenCourseWare. Recuperado de https://ocw.mit.edu.

Kilie, M. S., & Bazant, M. Z. (2005, February 15). *Asintóticas con colas gordas* [Video]. MIT Department of Mathematics. Recuperado de https://ocw.mit.edu.

Burch, D. (2006, September 26-28). *Asintóticas de colas gordas* [Video]. MIT OpenCourseWare. Recuperado de https://ocw.mit.edu.

MIT Department of Mathematics. (2005, February 24). *Asintóticas del paseo aleatorio de Bernoulli* [Video]. MIT OpenCourseWare. Recuperado de https://ocw.mit.edu.

Referencias de los programas utilizados

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100–107.

Ruiz Velasco, E. (2014). *Introducción a la robótica móvil*. UNAM. Disponible en: http://www.mech.northwestern.edu/robotics/recursos/robotica_movil_ruiz_velasco.pdf

Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall. Versión accesible en: https://ia800301.us.archive.org/18/items/ArtificialIntelligenceAModernApproach3rdEdition/Artificial%20Intelligence-A%20Modern%20Approach%20%283rd%20Edition%29.pdf.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100–107. Disponible en: https://en.wikipedia.org/wiki/A*_search_algorithm.

Ruiz, J., & Sánchez, M. (2018). Simulación de evacuaciones de edificios utilizando algoritmos de búsqueda. *Revista Iberoamericana de Sistemas, Cibernética e Informática*, 15(2), 45–52.

Castillo, L. (2019). Inteligencia Artificial: fundamentos, práctica y aplicaciones. Editorial UNED, Madrid.

Morales, A. (2020). Algoritmos de búsqueda informada en inteligencia artificial. *Revista Digital Universitaria*, 21(4), 1–12.

`Python Imaging Library` (PIL) https://pillow.readthedocs.io/en/stable/

Imagen de referencia https://www.google.com/imgres?imgurl=https://media.istockphoto.com/id/1297818885/es/vector/plano-para-apartamento-o-piso-con-muebles-ilustraci%C3%B3n-vectorial-aislada.jpg?s%3D612x612%26w%3D0%26k%3D20%26c%3DXKEPVTDP661GpRL1QkNvneALJ_DKQc1bgYQB1uY6xFk%3D&tbnid=tIkYg5SuS8dyYM&vet=1&imgrefurl=https://www.istockphoto.com/es/fotos/mapa-de-dormitorio&docid=nGcMA85Qr-jw3M&w=612&h=449&hl=es-MX&source=sh/x/

im/m1/0&kgs=a18c887889259f46&utm_source=sh/x/im/m1/0