Universidad de Lima

Facultad de Ingeniería

Carrera de Ingeniería de Sistemas

IMPLEMENTACIÓN DE UN SISTEMA DE BÚSQUEDA DE RUTAS DE EVACUACIÓN EFICIENTES ANTE LA PRESENCIA DE SISMOS EN UN CENTRO COMERCIAL USANDO EL ALGORITMO D ESTRELLA

Tesis para optar el Título Profesional de Ingeniero de Sistemas

Walter Steven Pariona Sanchez

Código 20140989

Asesor

Nadia Katherine Rodriguez Rodriguez

Lima – Perú

Setiembre de 2023

UNIVERSIDAD DE LIMA
MCMLXV
SCIENTIA ET PRAXIS

IMPLEMENTATION OF A SYSTEM TO SEARCH EFFICIENT EVACUATION ROUTES IN THE PRESENCE OF EARTHQUAKES IN A SHOPPING CENTER USING THE D STAR ALGORITHM

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Alrededor del mundo diferentes autores han demostrado gran interés en los desastres naturales como los temblores o terremotos generando soluciones tecnológicas en torno a sistemas de evacuación. En la presente investigación se explica por qué es importante la implementación de un sistema de evacuación inteligente que tenga la capacidad de reconocer el camino más corto en un movimiento sísmico real y se hace una revisión de los diferentes sistemas de evacuación inteligentes empleados en la actualidad. Seguido de esto, al finalizar las etapas de análisis, se logró construir una prueba de concepto para hallar la ruta más eficiente en el primer nivel de un centro comercial empleando el algoritmo de búsqueda D estrella. Además, se consideró la revisión del algoritmo A estrella y D estrella para comparar la eficiencia de estos y se encontró que el A estrella emplea 3 milisegundos en encontrar la ruta más corta pero este no contempla escenarios que requieren de caminos dinámicos. Así, se encontró que el algoritmo D estrella es capaz de reconocer caminos con obstáculos dinámicos y encontrar la ruta más corta en solo 24 milisegundos, convirtiéndolo en el más adecuado para entornos de evacuaciones reales.

Por último, para evaluar el desempeño de la prueba de concepto elaborada, se midió el tiempo experimental empleado por el algoritmo D estrella para encontrar la ruta de evacuación más eficiente en ambientes con las siguientes variaciones: diferentes tamaños de planos con escala 5, 10 y 15, diferentes cantidades de obstáculos existentes en el plano y generados en tiempo real. Los resultados obtenidos en el sistema fueron favorecedores al demostrar que la ruta más corta es encontrada en 22 milisegundos y una ruta alternativa es encontrada en 3 milisegundos para escenarios que presentan nuevos obstáculos en el camino.

Palabras clave:

Sistemas de evacuación, algoritmos de búsqueda, algoritmo A estrella, algoritmo D estrella, caminos más cortos, obstáculos dinámicos

ABSTRACT

Around the world there are different authors that have shown big interest related to natural disasters such as earthquakes. This have generated different technological solutions around modern evacuation systems. This research explains why it is important to implement an intelligent evacuation system that can recognize the shortest path in a real earthquake. The research also reviews the different modern evacuation systems being used nowadays. In this way, after many phases of analysis, a proof of concept was built to find the shortest path in the first level of a shopping mall using the D star search algorithm. Furthermore, a comparison between the A star algorithm and D star algorithm was made to find out the performance of these. It was found that the A star algorithm finds the shortest path in 3 milliseconds, but it doesn't support scenarios with dynamic paths. Thus, it was found that D start algorithm can support paths with dynamic obstacles and find the shortest path in 24 milliseconds, making it the most accurate algorithm for real evacuations scenarios.

Ultimately, in order to evaluate the performance of the system, the experimental time was measured when the system calculates the shortest path in different environments such as: maps with scales of 5, 10 and 15, different number of existing obstacles and different number of obstacles generated in real time. The results of this research were favorable since the system was able to find an efficient shortest path in 22 milliseconds. Also, the simulator found an alternative path in 3 milliseconds for scenarios in which new obstacles were introduced in the path.

Keywords:

Evacuation System, Path Search Algorithms, A star Algorithm, D star Algorithm, shortest paths, dynamic obstacles

INTRODUCCIÓN

En el Perú, existen diferentes casos de eventos relacionados a los temblores o terremotos. Estos pueden ocurrir frecuentemente y en cualquier momento ya que es parte del conjunto de países que están ubicados sobre el área del cinturón del fuego la cual origina el 90% de los movimientos telúricos que ocurren en el mundo (Rosenberg, 2018). Como consecuencia a esto, el Instituto Geofísico del Perú publicó en su base de datos sísmicos que en el Perú han ocurrido 9085 eventos telúricos en los últimos 10 años (desde el 2011 al 2021) entre sismos clasificados como superficiales, intermedios y profundos en relación a la profundidad de sus focos. También se observó que en el año 2014 se detectaron 1025 sismos, lo cual representa el año con mayor actividad sísmica en el territorio peruano. Asimismo, los reportes encontraron que en el año 2019 se presentó un sismo de magnitud de 8 grados siendo el sismo de mayor escala en los últimos 10 años (CENSIS - Instituto Geofísico del Perú, s. f.). Acorde con estos datos observados, se logró tener una referencia clara de que se necesita establecer planes de evacuación en todo el territorio peruano debido a que presenta una alta actividad geofísica.

Además de lo mencionado, también se observó que hubo un mayor incremento de personas afectadas y damnificadas en los últimos años en el Perú. En el Reporte de Emergencias del portal del Instituto Nacional de Defensa Civil (INDECI) se observó que en el 2019 hubieron 6150 personas afectadas y 4806 personas damnificadas, mientras que en el año 2021 se reportaron 18704 personas afectadas y 6219 personas damnificadas, lo cual evidencia un gran incremento en el número de incidencias como consecuencias de los sismos (INDECI, n.d.). Así, se observa que hay una falta de preocupación por parte de las empresas públicas y privadas y poca concientización para difundir información sobre cómo llevar a cabo una evacuación segura y reducir a la población afectada.

Asimismo, incluso cuando existen construcciones modernas que cuentan con planes de evacuación regularizados y autorizados por INDECI, estos planes de escape no siempre son eficaces debido a que solo involucran el posicionamiento de señaléticas estáticas las cuales no garantizan un plan de evacuación óptimo al realizar un simulacro o en caso de un siniestro real. Por consiguiente, al no contar con un plan de evacuación

exacto y de conocimiento general por todos los individuos que están bajo una misma área, estos tienden a dirigirse a la primera salida que encuentren al desplazarse por el espacio afectado (Ying et al., 2017). Además, también es recomendable considerar la probabilidad de que una persona puede imitar el comportamiento de un conglomerado. Esta probabilidad es mayor en los casos en que un individuo no consigue visualizar un indicador de salida y a la vez observa un gran flujo de personas dirigiéndose hacia cualquier dirección. Esta tendencia se debe a que las personas tienden a imitar las decisiones de otros evacuadores que sí logran ver dichas salidas (Haghani & Sarvi, 2016).

Por otro lado, también se realizó una revisión de los sistemas de evacuación existentes en el Perú y se encontró que en su mayoría los establecimientos como edificios o centros comerciales solo cuentan con sistemas de detección y alarmas de incendios (DACI) los cuales están compuestos por sensores de humo, sensores de temperatura y sirenas con luz. Existen casos en los que también se implementan sistemas de audio de evacuación y perifoneo que cuentan con paneles por donde se provee instrucciones e información por voz al recinto, con la finalidad de realizar una evacuación ordenada (Securitech, s.f.). Igualmente, para complementar estos sistemas también se encontró una plataforma cloud que se encarga de monitorear los sistemas de protección contra incendios en tiempo real, permite gestionar de forma remota el sistema contra incendios desde el celular y a la vez brinda la posibilidad de enviar notificaciones a todos los usuarios (Firecity, s.f.). Sin embargo, estos sistemas encontrados en el Perú solo funcionan en base a planos estáticos del área y la identificación inicial de los caminos que estén debidamente señalizados, careciendo de cualquier funcionalidad para recomendar la ruta más corta desde cualquier punto y tampoco toman en cuenta el estado o condición de las rutas en tiempo real.

Es así que en base a lo expuesto anteriormente se presenta el principal problema. En la actualidad los planes o guías de evacuación solo constan de guías estáticas y de rutas confiables hacia las salidas de escape para realizar una evacuación en un establecimiento moderno; no obstante, estos planes no son considerados eficientes u óptimos ya que no brindan la menor distancia dinámica y no proponen rutas o recorridos alternos en caso de que se presenten obstáculos en el trayecto. Además, al momento, en la realidad peruana, solo se cuenta con el sistema de Alerta Sísmica del Perú (SASPe) el cual es un sistema de monitoreo y difusión que cuenta con sensores sísmicos (acelerómetros) para alertar a la población con segundos de anticipación cuando ocurre

un sismo en el país (Sistema de Alerta Sísmica del Perú (SASPe), 2022). Por otra parte, en cuanto a acciones a tomar durante casos de emergencias y desastres, el país solo cuenta con guías para elaborar un Plan Familiar de Emergencias, Plan Comunitario de Emergencias, planes de Preparación de Contingencia y pasos para organizar y ejecutar simulacros (Recibir información u orientación sobre preparación ante emergencias y desastres, 2022). A partir de esto se puede observar que en el país no se cuenta con un sistema automatizado que implemente y/o solucione los puntos observados anteriormente. De esta forma, en base a lo explicado, se evidencia que es importante la implementación de un sistema inteligente capaz de reconocer el camino más corto en una evacuación real ocasionada por un movimiento sísmico en el Perú que brinde mayor seguridad a los evacuados y de esta forma solucionar el problema identificado.

La presente solución propuesta está basada en gran medida en el trabajo presentado por el autor Walter Pariona en el Segundo Congreso Internacional de Ingeniería de Sistemas (Pariona, 2019). El objetivo de esta investigación es construir un sistema de evacuación inteligente a través de la implementación de un simulador capaz de calcular la ruta de escape más segura y eficiente a una superficie libre de riesgos en un centro comercial usando un algoritmo de caminos más cortos. Entre ellos se evaluará el uso del algoritmo de búsqueda A* o A estrella y el algoritmo de búsqueda D* o D estrella.

A continuación, el presente trabajo de investigación será desarrollado en diferentes partes divididas por capítulos. En el primer capítulo hace referencia al estado del arte donde se revisarán las literaturas desarrolladas por otros autores. En el capítulo 2 se revisarán los antecedentes a tomar en cuenta para el desarrollado del trabajo. En el capítulo 3 del presente trabajo se encuentra la metodología en la que se hablará de la propuesta de solución así como todas las etapas que se llevaron a cabo para la implementación del simulador. En el cuarto y quinto capítulo se presentarán los resultados y la discusión de estos, respectivamente. Por último, en el capítulo 6 se expondrán las conclusiones en base a lo encontrado y seguido de esto se mencionarán los trabajos futuros.

CAPÍTULO I: ESTADO DEL ARTE

El problema anteriormente expuesto ha generado interés en diferentes autores alrededor del mundo. Estos presentan una preocupación sobre qué se está haciendo para evitar o reducir las pérdidas y daños lamentables, generalmente humanas, frente a sucesos espontáneos como son los sismos o movimientos telúricos. Esta inquietud ha motivado a que se construyan soluciones para mitigar los problemas relacionados a los planes de evacuación tradicionales.

1.1 Investigaciones sobre sistemas de evacuación inteligentes en diferentes ambientes

En la investigación realizada por Gu et al. (2018), se encontró que en el año 2017 ocurrieron hasta 219,000 incendios en China evidenciando que la población necesitaba usar un método de evacuación más sofisticado. Debido a esto, los autores crearon un sistema de realidad virtual (VR) para simular evacuaciones usando un modelo de evacuación híbrido. Este modelo fue desarrollo con dos componentes: un componente para analizar el desplazamiento global de la población usando como algoritmo un modelo de redes dinámico y un componente con un modelo basado en agentes para analizar y calcular la ruta local de cada agente. Además, también desarrollaron un mecanismo de emergencia para generar incendios en diferentes periodos de la simulación VR. El modelo de evacuación se implementó en el lenguaje C# y para su ejecución se usó el visor de VR HTC Vive. Este modelo soporta algunas customizaciones como ajustar la velocidad de los peatones para que estos puedan caminar, correr, esperar o competir con otros durante la evacuación. Durante la evacuación los agentes son guiados por la ruta de evacuación más corta utilizando el algoritmo Floyd. Para la etapa de resultados se modeló virtualmente el tren subterráneo de Shanghai y se realizó un experimento con 1102 agentes virtuales y tomando en cuenta una densidad de 2.5 personas por metro cuadrado. Así, en la simulación se obtuvo que todos los agentes pudieron evacuar el establecimiento en 245 segundos lo cual es mucho menor al tiempo registrado cuando se realizó una evacuación sin un sistema de evacuación inteligente (6 minutos). Además, con este resultado los autores descubrieron que una de las salidas de emergencia del tren

subterráneo no era muy útil ya que la salida número 3 no era capaz de compartir el flujo de personas con otras salidas. Esto también demuestra que los sistemas de evacuación inteligentes son útiles como una etapa de pre planeamiento al diseñar establecimientos donde las salidas tienen que estar conectadas.

En otra investigación realizada por Hridi et al. (2016) en base a los desastres ocurridos en Bangladesh, los autores propusieron un sistema inteligente que conduce en el menor tiempo posible a los usuarios a un espacio seguro usando sus ubicaciones actuales y considerando su comportamiento frente a desastres naturales como sismos, inundaciones, lluvias torrenciales y ciclones. El sistema propuesto va dirigido a usuarios que están conectados y usuarios que no se encuentran en línea. Los usuarios conectados al sistema reciben sugerencias basadas en datos de otros usuarios recolectados en tiempo real. Según los autores, para estos usuarios el comportamiento de la aplicación varía de acuerdo a sus posiciones: alejados de zonas de peligro, a salvo, dentro de áreas de peligro. La parte más innovadora de la investigación se observa en el caso de usuarios de la aplicación que no están en línea. De acuerdo a los autores, la aplicación almacena los siguientes datos para esos usuarios:

- Información recolectada sobre el patrón de movimiento del usuario en los últimos 14 días.
- Información relacionada a la calidad de las autopistas próximas al área.

En base a la información recolectada en los últimos 7 días, el sistema propuesto construye un modelo del desplazamiento del usuario y predice las diferentes rutas que son propensas a ser más usadas durante un siniestro. Basado en estas rutas, la aplicación construye un grafo y asigna valores a las aristas dependiendo de cuán conocida sea la ruta y la probabilidad de estar congestionada. Seguidamente, la aplicación procesa múltiples veces un algoritmo de búsqueda de enrutamiento adaptativo dinámico y en base a sus resultados encuentra las posibles rutas para realizar una evacuación frente a los siniestros mencionados. Por otra parte, en relación a los usuarios que no están en línea, el servidor toma en cuenta el movimiento histórico e implementa un modelo de comportamiento del usuario para determinar la probabilidad de congestión de las rutas y así conocer cuáles son las más propensas a ser usadas durante un escenario de desastre. De esta manera, el sistema logra reducir la congestión sugiriendo a los usuarios que están en línea caminos alternativos que probablemente no vayan a ser utilizados por usuarios que no están en

línea y, por consiguiente, permitirá que se realice una evacuación más dinámica para los dos tipos de usuarios (Hridi et al., 2016).

Por otro lado, en una investigación realizada por Vokřínek et al. (2010), se llevó a cabo la implementación de un sistema de navegación inteligente pero esta vez enfocado a encontrar la ruta más corta para una escolta de vehículos tripulados que desean evacuar en áreas urbanas con la ayuda de vehículos terrestres no tripulados o vehículos autónomos. La escolta de vehículos tripulados comienza su desplazamiento sobre la ciudad usando el algoritmo D estrella y este incorpora la información obtenida por el grupo de vehículos autónomos que analizan la ruta del área urbana tomando en cuenta obstáculos desconocidos y zanjas sobre las calles.

Para realizar la implementación de este sistema y encontrar la ruta más corta se emplearon 3 técnicas. La primera técnica fue el uso del algoritmo D estrella para encontrar la ruta más corta a nivel global en toda el área y tomar en cuenta la presencia de nuevos obstáculos, además este también se uso para identificar los límites o fronteras de contingencia sobre el plano que puedan aparecer en el proceso. Seguido de esto, se utilizó una técnica de enrutamiento de vehículos para asignar estas fronteras a los vehículos autónomos y así estos puedan minimizar la distancia de recorrido. Esta técnica opera en la fase en que se genera el gráfico de navegación lo cual mejora el desempeño del simulador (Vokřínek et al., 2010).

Por último, después de obtener la ruta compuesta por una secuencia de objetivos en el plano, se utiliza un algoritmo de planificación de maniobras para refinar el camino encontrado ya que los vehículos necesitan realizar pequeñas maniobras para sobrepasar los obstáculos y zanjas presentes en las calles (Vokřínek et al., 2010).

Para la fase de resultados se diseño un área urbana en 3D y se ubicó a la escolta de vehículos tripulados en la esquina inferior izquierda del área y el punto de evacuación se ubicó en la esquina superior derecha. La ejecución del simulador encontró que al utilizar solo el algoritmo D estrella la ruta más corta es encontrada en 300 segundos siendo este caso más óptimo que al emplear un algoritmo de enrutamiento estándar. En una segunda ejecución se utilizó el algoritmo D estrella y un vehículo autónomo para analizar el área, así se encontró que el tiempo empleado para calcular la ruta más corta se redujo en 100 segundos aunque explorando una mayor cantidad de nodos en el plano. En consecuencia, los autores concluyeron que cuando se emplea un mayor número de vehículos autónomos

el número de nodos explorados aumenta y el tiempo de desplazamiento se reduce. Además, también se encontró que solo sería óptimo emplear 5 vehículos autónomos como máximo para explorar la ruta, ya que de 5 a más la distancia a recorrer converge a ser siempre la misma (Vokřínek et al., 2010).

1.2 Investigaciones que implementan algoritmos conocidos de caminos más cortos

Según Pelechano y Badler (2006), los procesos de evacuación de multitudes en edificaciones grandes y complejas usualmente son obstaculizadas por individuos que desconocen sobre la conectividad interna. Además, los autores demostraron que los ocupantes tienden a utilizar salidas con las que están familiarizadas causando que las salidas de emergencia sean ignoradas. Así, los autores propusieron el desarrollo de MACES (Sistema de comunicación multi-agente para la simulación de evacuaciones), un sistema multi-agente distribuido que emplea el algoritmo de búsqueda DFS en el que cada agente tiene su propio comportamiento basado en 3 diferentes rasgos de personalidades que dependen de habilidades como el liderazgo y atributos de entrenamiento. La primera personalidad de un agente es ser un líder entrenado el cual tiene completo conocimiento de las conexiones internas de un edificio y ayudaría a otras personas. La segunda personalidad que puede tener un agente es ser un líder no entrenado y corresponde a personas que pueden controlar mejor el estrés y tienden a ayudar a otras personas a buscar rutas de evacuación. Por último, la tercera personalidad corresponde a un seguidor no entrenado el cual representa a personas que dependen de otras y que podrían entrar en pánico durante una situación de emergencia. Por otro lado, el movimiento de los agentes es calculado en dos niveles. El nivel alto corresponde al proceso de encontrar un camino de evacuación, mientras que el nivel bajo corresponde al movimiento local dentro de un espacio (habitación, sala, cuarto, etc.). Los datos de entrada recibidos por el sistema MACES son características del entorno como dimensiones, número de salidas, el plano del edificio, cantidad de agentes y porcentaje de agentes entrenados. A partir de estos datos, por cada celda del plano el algoritmo del sistema genera el camino más corto a cada salida automáticamente. Por otro lado, cada agente virtual se puede comunicar con otros, intercambiando información de las ubicaciones de los obstáculos que bloquean el recorrido y las partes del edificio que no

tienen salidas. Luego de realizar un análisis del entorno, el algoritmo procede mediante 3 pasos principales:

- Los líderes entrenados que se encuentran en una habitación comparten la información del ambiente con los otros agentes.
- Los agentes verifican que el camino más corto se encuentra fuera de peligro, avanzan y añaden la siguiente celda al mapa mental.
- Dependiendo del tipo, un agente líder entrenado se dirige al camino más corto sin inconvenientes, un agente líder no entrenado revisa el edificio empleando el algoritmo de búsqueda de profundidad (DFS), un agente seguidor no entrenado no comprende qué hacer y por lo tanto solo imita las decisiones tomadas por otro agente.

Finalmente, para obtener resultados se usaron 3 planos con 100 habitaciones y 8 de ellas bloqueadas por algún obstáculo. En la primera ejecución de pruebas para comparar métodos de búsqueda, se obtuvo que el algoritmo de búsqueda DFS conlleva a que los agentes busquen habitaciones de una manera más estructurada a comparación de un algoritmo de búsqueda aleatoria en el cual los individuos solo se encargan de explorar espacios adyacentes de manera incierta e intentan no retroceder a menos de que se encuentren atrapados. El resultado muestra que la aplicabilidad del algoritmo DFS fue 15 veces más rápida y que el comportamiento de los agentes fue más similar al comportamiento esperado de una multitud real en relación a como se deberían desplazar a través de la ruta de evacuación. En la segunda prueba para comparar el uso de la comunicación y no comunicación, se obtuvo que la simulación con presencia de comunicación finalizó en la mitad del tiempo que le tomó a la prueba sin comunicación (Pelechano & Badler, 2006).

En la investigación de otros autores, se diseñó e implementó un sistema de guía basado en celulares inteligentes (smartphones) llamado EasyGO que usa el algoritmo Dijkstra para encontrar el camino más corto. El objetivo de esta investigación es ofrecer una solución para evacuaciones en ambientes cerrados usando dispositivos IoT, para ser específicos, dispositivos iBeacon. Estos dispositivos IoT toman un rol importante ya que son usados para enviar constantemente señales a los celulares inteligentes de los usuarios y así detectar la posición actualizada de cada uno de ellos. La implementación de este sistema se llevó a cabo mediante el uso de dispositivos Android y los dispositivos

iBeacon ubicados en espacios interiores. Es así que el sistema es capaz de calcular un camino de evacuación con el menor tiempo de desplazamiento a un punto de destino tomando en cuenta la densidad de cada área o pasadizo. Antes de comenzar el planeamiento de las rutas, se genera un modelado espacial con las salidas, puertas y puntos de intersección para construir el grafo. Además, para cada parte del grafo se consideró que los pesos deben ser calculados midiendo la capacidad del pasadizo, es decir, el número de personas que pueden transitar por cada segundo en áreas como puertas, salidas y puntos de intersección en el área. Luego, cuando el usuario lo requiera, el dispositivo móvil manda una señal con la ubicación actual al servidor central. Este servidor central planea y calcula la ruta más rápida usando el algoritmo de caminos más corto Dijkstra tomando en cuenta la densidad actualizada de cada pasadizo. Una vez que el cálculo finaliza, el servidor central envía a cada usuario la ruta de escape adecuada. Por último, el sistema EasyGO también es capaz de ejecutarse más de una vez para prevenir a las personas de ingresar a nuevas áreas peligrosas (L. W. Chen & Liu, 2018).

En la siguiente investigación recopilada, los autores reconocen que es importante diseñar un plan de evacuación eficiente basado en la distribución dinámica de multitudes. El método propuesto es capaz de predecir la distribución de la multitud en el futuro cercano para evitar una potencial congestión en las intersecciones de la ruta. Además, esta investigación también reconoce que es importante que el algoritmo se ejecute repetidas veces para tener una ruta actualizada y así no necesitar una predicción para un futuro lejano. Para calcular la ruta más corta de desplazamiento la investigación emplea el algoritmo de caminos más cortos A*. Los autores mencionan que el uso de este algoritmo es ideal ya que incurre en el menor costo de tiempo y toma en cuenta grafos con peso. Por otro lado, para tomar en cuenta la etapa de replaneamiento, la investigación propone que el algoritmo sea ejecutado cada vez que un individuo llega al próximo nodo en su recorrido. De esta forma, si se tiene un recorrido del punto A al punto B, cada vez que el agente se desplace un nodo, el algoritmo propuesto verifica si la ruta calculada aún es válida. En caso la ruta presente obstáculos o haya mayor congestión de personas, el algoritmo se vuelve a recalcular y proponer una nueva ruta. En la etapa de resultados, los autores decidieron comparar el algoritmo junto con el algoritmo de caminos más corto Dijkstra y con un algoritmo de evacuación no dinámico. Además, para una verificación más exhaustiva de los resultados, la literatura plantea la ejecución del algoritmo en diferentes escenarios. Para los escenarios de ejecución se consideró planos con una sola

salida, planos con múltiples salidas y planos complejos con múltiples intersecciones. Como resultados, se obtuvo que el algoritmo propuesto se ejecutó en 1075 milisegundos tomando en cuenta 390 agentes. Además, en todos los escenarios el método desarrollado aplicando el algoritmo A* es capaz de ahorrar aproximadamente un 9.5% del tiempo en comparación con los otros métodos. Así, se comprobó que es beneficioso que un sistema de evacuación tome en cuenta la actualización constante de la ruta de escape (X. Chen et al., 2019).

1.3 Investigaciones cuyos sistemas consideran el comportamiento de las personas

En la investigación realizada por Koo et al. (2013) se evidenció que la mayor parte de estudios solo toman en cuenta poblaciones homogéneas, es decir, personas que tienen las mismas capacidades y no presentan ninguna limitación, dejando de lado a individuos con discapacidades sensoriales y motrices que requieren de ayuda para dirigirse por caminos difíciles de negociar. Además, es posible que estas personas bloqueen la evacuación de otros involucrados debido a el espacio requerido y a su pausada velocidad. Es así que proponen el sistema de simulación BUMMPEE el cual es capaz de guiar distintos tipos de agentes virtuales de evacuación como agentes con impedimento visual, con sillas de ruedas, agentes sin discapacidad, entre otros. Para realizar un estudio se empleó un edificio de 24 pisos construido con ascensores apropiados para realizar evacuaciones. Considerando que en evacuaciones en entornos reales es probable que un individuo en silla de ruedas se movilice por espacios con la asistencia de otros ocupantes, el simulador BUMMPEE contempla una funcionalidad de "sistema de amigos" en el cual otros ocupantes pueden brindar ayuda a agentes discapacitados. Asimismo, los agentes en sillas de ruedas también tienen la posibilidad de usar como medio de evacuación los ascensores. Basado en esta propuesta se llevaron a cabo experimentos con dos tipos de estrategias: evacuaciones controladas basadas en agentes y evacuaciones que emplean ascensores. En la primera prueba consideraron un tiempo de espera de 60 segundos para agentes en sillas de ruedas. El resultado arrojó que los agentes que no empleaban sillas de ruedas fueron capaces de evacuar más rápido respecto a una evacuación con ejecución simultánea. En la segunda prueba el uso de ascensores para evacuar estaba permitido para los agentes con discapacidades. En consecuencia, se obtuvo que estos agentes redujeron su tiempo de evacuación en 21.5% comparado al de una evacuación que solo emplea escaleras y como consecuencia se logró evacuar a todas las personas en 977.2 segundos.

En la siguiente literatura a revisar se implementó un sistema cuyo ámbito de funcionamiento reside en los dispositivos móviles de los evacuados. Este sistema no necesita contar con un servidor central y realiza cálculos distribuidos usando un framework llamado DCOP (Distributed Constraint Optimization Problem). El modelo de asistente de evacuación se encarga de intercambiar información acerca de la seguridad de las rutas, planificar la ruta de evacuación, negociación y manifestación de la guía de evacuación. Para comenzar con la fase de negociación del tiempo y emplear el framework DCOP se necesita realizar una etapa de formalización. Para esto, el sistema considera como un recurso a una ruta de evacuación y los evacuados con un dispositivo móvil son considerados agentes (Iizuka & Iizuka, 2015).

El sistema logra que los agentes encuentren la ruta más corta intercambiando información a través de la transmisión de mensajes con otros agentes mediante sus dispositivos móviles. Cada agente almacena en una variable la ubicación a la cual deben desplazarse en el tiempo $t + 1$ ejecutando el framework DCOP. Para la fase de resultados se utilizó un simulador capaz de manejar agentes múltiples y se tomó como referencia los edificios del campus de una universidad para realizar las pruebas de simulación. Además, en el experimento no se contempla el factor psicológico de los evacuadores. En el primer experimento, se estableció un edificio con dos pisos con seis habitaciones, una sola ruta de evacuación y el número de agentes fue 400. En el experimento se obtuvo como resultado que al realizar una evacuación usando el sistema propuesto, el tiempo total de la evacuación se reduce en 10% comparado a una evacuación en el que no se usa un sistema guiado. En el segundo experimento se estableció un mapa con dos rutas de evacuación, un número de agentes de 700 y 4 habitaciones por piso. Como resultado se mostró que el tiempo de finalización de la evacuación disminuyó en 30% usando un sistema guiado de rutas de evacuación y negociación del tiempo (Iizuka & Iizuka, 2015).

CAPÍTULO II: MARCO TEÓRICO

En el presente capítulo se abordarán diferentes temas que son relevantes para la presente investigación y para entender el contexto bajo el que se desarrollará la propuesta de solución. Es así que primero se explicará la teoría de los sismos para entender por qué se originan estos y cómo se miden, luego se revisará el concepto de grafos, los tipos de grafos, las estructuras de datos más conocidas para implementar estos y cómo serán empleados para representar un plano virtual.

Además, se revisarán los algoritmos más importantes que existen para poder encontrar el camino más corto sobre un área. Por último, se explicará a detalle el funcionamiento del algoritmo A estrella y el algoritmo D estrella que serán fundamentales para la implementación de la solución planteada.

2.1 Teoría de sismos

2.1.1 Definición de los sismos

Según Chancay (2018), los movimientos telúricos, temblores, sismos o terremotos son fenómenos de vibración o perturbaciones de la corteza terrestre los cuales se originan por las fuerzas de fricción que ocurren en las placas tectónicas y por liberación de energía acumulada que luego es propagada por el medio terrestre como ondas sísmicas desde el lugar en que se origina (epicentro) hacia todas las direcciones.

2.1.2 Magnitud de los sismos

De acuerdo con Espindola y Perez (2018), la magnitud representa la cantidad de energía liberada durante un sismo y es usada para medir y cuantificar los movimientos telúricos. Esta magnitud fue propuesta originalmente por Charles Ritcher en 1935 al establecer una ecuación logarítmica; sin embargo, existen muchos tipos de magnitudes estimadas en base a diferentes criterios. Actualmente, la magnitud más usada es la de momento sísmico (M_w), la cual fue propuesta por Hanks y Kanamori en 1979 para determinar la magnitud de un sismo en base a la medición de la energía liberada en el área de ruptura y

deslizamiento donde ocurre la falla. En la tabla 2.1 se puede observar la clasificación de los movimientos telúricos según esta magnitud.

No obstante, también existe otra magnitud conocida para clasificar los sismos a partir de sus consecuencias o efectos. El físico Giuseppe Mercalli propuso en 1902 una escala que consiste en 12 niveles clasificados en números romanos. El nivel I hace referencia a que el sismo solo fue percibido por algunas personas y el nivel más alto XII indica un sismo que ocasionará destrucción total. A pesar de que la información brindada por esta escala es muy útil, actualmente no es ampliamente usada ya que ahora contamos con registros sísmicos que brindan mayor información sobre las variaciones del suelo (Espindola & Perez, 2018).

Tabla 2.1

Tabla de clasificación de los movimientos telúricos en base a su magnitud

Magnitud (M_w)	Efectos del movimiento telúrico
< 2.5	Usualmente no es percibido pero puede ser detectado por un sismógrafo.
2.5 a 5.4	A menudo se siente, pero solo causa daños menores.
5.5 a 6.0	Causa daños leves en edificios y otras estructuras.
6.1 a 6.9	Puede causar mucho daño en áreas muy pobladas.
7.0 a 7.9	Terremoto de gran escala. Daño grave.
> 8.0	Gran terremoto. Puede destruir totalmente las comunidades cercanas al epicentro.

Nota. De Earthquake Magnitude Scale, por Michigan Tech, 2022 (https://www.mtu.edu/geo/community/seismology/learn/earthquake-measure/magnitude/).

2.2 Teoría de grafos

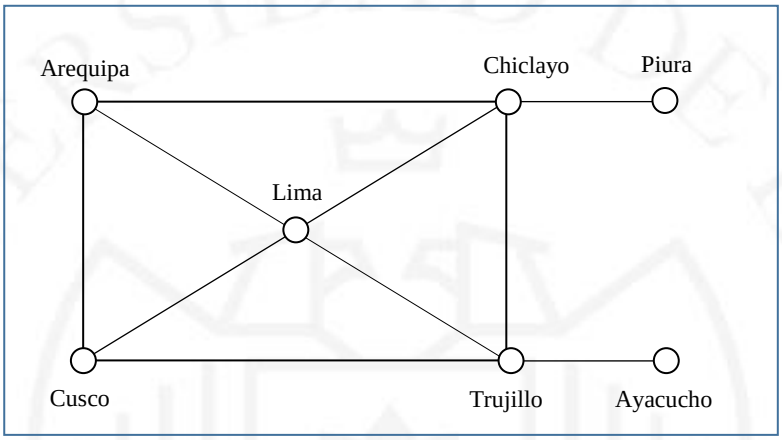
Debido a que la construcción del sistema involucra la transformación de un establecimiento a una red de grafos para poder generar la simulación, se necesita tener claro el concepto de un grafo.

Los grafos son importantes debido a que pueden ser usados para representar esencialmente cualquier tipo de relación. Por ejemplo, un grafo puede modelar una red

de caminos donde las ciudades son vértices y los caminos entre las ciudades son lados, así como se muestra en la figura 2.1. También son usados para modelar circuitos electrónicos con uniones como vértices y componentes como lados. Es así que el concepto de un grafo consiste en un conjunto de vértices *V* junto con un conjunto de lados *E* (Skiena, 2008).

Figura 2.1

Ejemplo de modelado de algunas ciudades del Perú como un grafo



El primer paso para decidir qué tipo de estructura de datos usar para representar a un grafo es determinar qué propiedades debe tener el grafo, es así que existen diferentes tipos de ellos. A continuación, se explicarán los tipos de grafos más importantes.

2.2.1 Grafos dirigidos y no dirigidos

Los grafos dirigidos o dígrafos son grafos cuyos lados *(x, y)* tienen un solo sentido, es decir, el lado se puede recorrer en una sola dirección, mientras que, los grafos no dirigidos son grafos cuyos lados pueden ser recorridos en las dos direcciones *(x,y)* y *(y,x).* Por ejemplo, las autopistas entre ciudades generalmente son representadas como grafos no dirigidos ya que tienen dos sentidos. Sin embargo, la ejecución de un programa sería representada como un grafo dirigido ya que cada línea de código se ejecuta una tras de otra (Skiena, 2008).

2.2.2 Grafos ponderados y no ponderados

En un grafo ponderado sus lados tienen asignado un valor numérico o un peso. Por otro lado, los grafos no ponderados no tienen un costo de distinción entre los diferentes lados y vértices. Por ejemplo, los lados de un grafo que representa una autopista pueden tener pesos que representen su distancia, tiempo de manejo, etc. La diferencia entre estos dos grafos toma relevancia cuando se desea encontrar el camino más corto entre dos puntos (Skiena, 2008).

2.2.3 Grafos simples y no simples

Existen grafos llamados de tipo simple porque cada par de vértices *(x,y)* está relacionado por máximo un lado. Sin embargo, a veces se necesita permitir que diferentes lados puedan relacionarse con el mismo par de vértices, a estos grafos se les denomina grafos no simples o multígrafos (Kocay & Kreher, n.d.).

Además, también existen conceptos generales de grafos donde, por ejemplo, un grafo con ningún lado es considerado un grafo vacío y un grafo con ningún vértice es llamado grafo nulo. Por último, si en un grafo dos de sus lados tienen un vértice final en común, entonces estos lados son considerados como adyacentes (Thulasiraman & Swamy, 2011).

2.3 Estructura de datos para grafos

Después de haber entendido el concepto de los grafos y sus diferentes tipos, es importante saber cuál es la mejor estructura de datos que se debe usar para implementar un grafo ya que elegir la estructura de datos correcta puede tener un gran impacto en el desempeño de la aplicación.

Es así, que para la implementación de los grafos tenemos dos opciones que se explicarán a continuación.

2.3.1 Matriz de adyacencia

Podemos representar un grafo *G* usando una matriz *M* de tamaño *n x n*, donde el elemento *M[i,j]* es igual a 1 si *(i,j)* es un lado del grafo o es 0 si no lo es. Esto nos permite saber rápidamente si dos vértices pertenecen al grafo. Sin embargo, esta estructura de datos podría usar un excesivo espacio en memoria para grafos que

tienen muchos vértices y lados. Por ejemplo, si quisiéramos representar en un grafo el mapa de Manhattan en la ciudad de Nueva York tendríamos que tomar en cuenta 15 avenidas y cada una de ellas cruzando 200 calles aproximadamente. Esto resultaría en un aproximado de 3000 vértices y 6000 lados lo cual podría almacenarse en memoria sin problemas, pero la matriz de adyacencia estaría almacenando cerca de 9 millones de celdas vacías sin ninguna información. Es por esto que muchas veces es mejor usar la siguiente estructura de datos (Skiena, 2008).

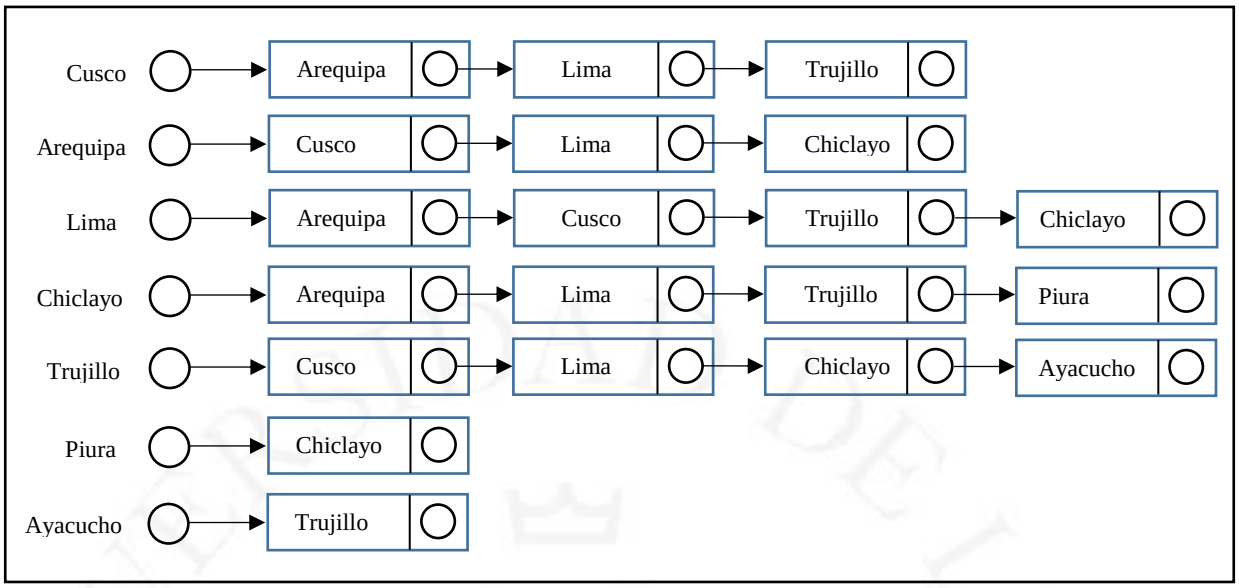
2.3.2 Listas adyacentes

Si se trata de eficiencia y desempeño, las listas adyacentes son una mejor opción para representar grafos usando listas enlazadas y así almacenar a los vecinos de cada vértice del grafo (Skiena, 2008).

Es importante mencionar que al almacenar información en una lista enlazada es posible realizar una búsqueda por cada vecino uno tras otro, lo cual es un clásico escenario en grafos. El principio de esta estructura de datos es que cada elemento de la lista está asociado con la dirección del siguiente elemento, una dirección que es llamada *puntero*. Así, cada elemento de la lista es representado por un rectángulo con dos partes: en la primera se especifica cuál es el valor o nombre del vecino y en la segunda se agrega el puntero al siguiente elemento (Fournier, 2009). A continuación, en la figura 2.2 se puede observar la representación de una lista enlazada del grafo mostrado anteriormente en la figura 2.1.

Figura 2.2

Ejemplo de lista enlazada para representar los nodos vecinos del grafo de la figura 2.1



2.4 Revisión general de los algoritmos de caminos más cortos

2.4.1 Algoritmo de Bellman-Ford

El algoritmo de Bellman-Ford resuelve el problema del camino más corto en base a un punto de origen y considera el caso en el que las aristas o recorridos de un grafo pueden poseer pesos negativos siempre y cuando el grafo no tenga ciclos negativos. Además, la complejidad algorítmica del algoritmo de Bellman-Ford es exponencial $O(n^2)$ ya que el tiempo de ejecución computacional incrementa exponencialmente con el aumento del número de nodos (Sathyaraj et al., 2008).

Por otro lado, al plantear una comparación entre el algoritmo de Bellman-Ford y el algoritmo de Dijkstra, se confirmó que ambos estiman los caminos más cortos desde un solo punto de origen en un grafo con pesos hacia un nodo final. No obstante, también se encontró que el algoritmo de Bellman-Ford es mucho más lento en tiempo de ejecución (Gutenschwager et al., 2012).

2.4.2 Algoritmo de Dijkstra

Desarrollado por el holandés Edsger Dijkstra en 1956, es un algoritmo de búsqueda de grafos que resuelve el problema del camino más corto desde un origen único en un grafo que tiene costos de aristas no negativos. Como resultado

del proceso se obtiene un árbol de rutas más cortas. Por ejemplo, si los vértices de un grafo representaran ciudades y el costo del recorrido de las aristas fueran distancias de manejo entre diferentes ciudades conectadas, el presente algoritmo podría ser usado para encontrar la ruta más corta entre una ciudad y el resto de todas las ciudades (IORDAN, 2012).

Este proceso ocurre puesto que a partir de un vértice inicial, el algoritmo examina el camino más corto desde ese punto a todos los otros vértices en el grafo, incluyendo el destino deseado. Al ejecutar el algoritmo, en cada iteración se añade un vértice al árbol de vértices con el valor del costo del recorrido desde el punto inicial. Para añadir el vértice, se rastrea la menor distancia analizada hasta el momento en el grafo y luego se inserta en el árbol. Además, debido a que no es una búsqueda de amplitud o de profundidad, el principal interés no está en el número de trayectorias en el árbol de grafos, sino en la suma de los costos o pesos (Sathyaraj et al., 2008). Por último, al analizar la implementación del algoritmo Dijkstra, se encontró que su complejidad algorítmica es exponencial $O(n^2)$ (Skiena, 2008).

2.4.3 Algoritmo de Floyd-Warshall

El algoritmo de Floyd-Warshall es un algoritmo que resuelve el problema del camino más corto. Sin embargo, a diferencia del algoritmo de Dijkstra, no calcula un recorrido desde un solo punto de origen hacia los demás nodos sino que calcula todos los caminos más cortos existentes en una sola iteración, es decir, desde cada nodo hacia los otros nodos finales. Como resultado se obtiene una matriz llamada *Dist*, donde *Dist[i,j]* denota la distancia del nodo *i* al nodo *j*. Además, también se calcula una matriz llamada *Next*, donde *Next[i,j]* representa al sucesor del nodo *i* en el camino más corto desde el nodo *i* al nodo *j* (Gutenschwager et al., 2012).

De manera similar, el algoritmo de Dijkstra podría resolver el camino más corto entre todos los pares de nodos iterando por cada posible vértice inicial. No obstante, usar el algoritmo de Floyd-Warshall es una manera más astuta y eficiente de construir una matriz de distancia *n x n* en base a una red de grafos con pesos (Skiena, 2008).

Acerca de la complejidad algorítmica, el algoritmo de Floyd-Warshall se ejecuta $O(n^3)$ veces, la cual no es asintóticamente mejor que *n* iteraciones del

algoritmo de Dijkstra. Sin embargo, en ciertos escenarios, el algoritmo Floyd-Warshall se ejecuta mejor en la práctica cuando las iteraciones son reducidas y eficientes (Skiena, 2008).

2.4.4 Algoritmo A* o A estrella para la búsqueda del camino más corto

Debido a que el problema de la planificación de rutas está relacionado con el problema de encontrar el camino más corto, se ha elegido el algoritmo de búsqueda A estrella para la implementación del sistema planteado ya que es el más común y eficiente al momento de encontrar las rutas más cortas en un área determinada (Masudur Rahman Al-Arif et al., 2012). A continuación, se explicará la lógica del algoritmo A estrella que luego será usado para implementar una primera versión del sistema propuesto.

El algoritmo A estrella cuenta con dos arreglos de listas para almacenar los nodos:

i) Lista Cerrada

ii) Lista Abierta

La lista cerrada almacena las celdas que deben guardarse conforme se avance en el recorrido. Por otro lado, la lista abierta se encarga de almacenar las celdas vecinas del plano que son posibles de atravesar con el objetivo de encontrar la celda de destino. Los elementos de este arreglo son revisados para saber si alguno de ellos es la celda objetivo o no. Además, antes de que sean almacenadas en la lista abierta, estas celdas se revisan de acuerdo a sus costos, pero a comparación de otros algoritmos que se aplican para la búsqueda del camino más corto, en este existen dos funciones de costos. La función *G*, la cual es el costo de pasar de la celda de partida a la celda actual y la función *H*, la cual representa el costo de pasar de la celda actual a la celda objetivo. De esta manera, la fórmula para calcular el costo en cualquier punto *n*, también conocido como el criterio de comparación para todas las celdas, se denota de la siguiente manera: $F(n) = G(n) + H(n)$ (Masudur Rahman Al-Arif et al., 2012).

Para hallar el valor de la función de evaluación, se necesita calcular las funciones *G* y *H*. El costo de la función *G* puede ser calculado; sin embargo, el costo de la función *H* solo puede ser estimado. Es por eso que a esta función de costo se le denomina función de costo heurístico (Masudur Rahman Al-Arif et al., 2012). Además, es importante resaltar que una restricción clave de la función *H* es que sea un heurístico admisible; es decir, que es fundamental no sobreestimar la distancia desde el nodo actual al nodo objetivo. Para estos escenarios de enrutamiento, la función *H* puede ser representada

como la distancia en línea recta desde la posición del agente a la posición final (Muntean, 2016).

En el proceso de ejecución, la lista abierta debe ser ordenada y los padres de las celdas vecinas son ordenadas en un arreglo de padres. Después de esto, si la celda en la que ahora uno se encuentra ubicado existe en la lista abierta su nuevo costo debe ser comparado con el costo de la celda anterior. Así, si este costo es menor, la celda se convierte en el padre y los costos de *G* y *F* deben ser calculados nuevamente (Masudur Rahman Al-Arif et al., 2012).

Por otro lado, la ejecución del algoritmo y la simulación finalizan cuando la celda objetivo se añade a la lista abierta y el camino más corto se consigue al seguir el camino trazado por las celdas padres (Masudur Rahman Al-Arif et al., 2012).

Por último, se tiene que resaltar que para el escenario descrito que contiene mapas con diferentes costos y con un solo punto de partida y un punto de llegada, el algoritmo A estrella es el mejor en tiempo computacional y tamaño de memoria. Por otro lado, si se requiere obtener el camino más corto y el menor costo se tendrá que elegir adecuadamente la función heurística *H* (Masudur Rahman Al-Arif et al., 2012). Otro punto clave a resaltar es que el algoritmo A estrella difiere de otros algoritmos de búsqueda parecidos (ej.: *best-first search*) ya que este toma en cuenta la distancia que se ha recorrido hasta el momento (Muntean, 2016). El pseudocódigo de este algoritmo será descrito líneas más abajo.

Un ejemplo clave en el que se empleó el Algoritmo A estrella es en el enrutamiento de las redes inalámbricas. En una literatura encontrada se hizo uso del algoritmo para optimizar el número de saltos en la red haciendo pruebas en un plano con 25, 50, 75 y 100 nodos activos. Los autores encontraron que dependiendo de la implementación de la función heurística la cantidad de nodos recorridos disminuyeron. Por ejemplo, en uno de los escenarios, el número de saltos se redujo de 18 a 3 en un plano con 100 nodos activos y, por lo tanto, se consiguió descubrir el camino más corto en la red. Además, se comprobó que el número de nodos recorridos es menor al usar más de una función heurística a la vez (Septiana et al., 2016).

ALGORITMO 1. A estrella

0. Inicializamos la lista ABIERTA
1. Inicializamos la lista CERRADA
2. Inicializamos el nodo objetivo T
3. Inicializamos el nodo origen O
4. Mientras lista ABIERTA no esté vacía:
    a. Obtener el nodo n de la lista ABIERTA con el menor f(n)
    b. Agregar n a la lista CERRADA
    c. Si n is igual a el nodo T entonces
        i. Terminar la ejecución y retornar la solución (el recorrido de la ruta más corta)
    d. Generar cada nodo sucesor n' de n
    e. Por cada nodo sucesor n':
        i. Actualizar el nodo n como el nodo padre de n'
        ii. Calcular la función h(n')
        iii. Calcular la función g(n') = g(n) + costo de ir desde n a n'
        iv. Calcular f(n') = g(n') + h(n')
        v. Si n' está contenida en lista ABIERTA y el nodo actual actual es más óptimo que n'
            1. Entonces descartar n' y continuar
        vi. Si n' está contenida en lista CERRADA y el nodo actual es más óptimo que n'
            1. Entonces descartar n' y continuar
        vii. Remover todas las ocurrencias de n' de lista ABIERTA y lista CERRADA
        viii. Agregar n' a lista ABIERTA
5. Si es que se realizó la búsqueda por todos los nodos y no se encontró una solución
    a. Retonar error

Fuente: (Muntean, 2016)

2.4.5 Algoritmo D* o D estrella para la búsqueda dinámica del camino más corto

Para brindar la solución planteada ante el problema de la planificación de rutas también es importante tener en cuenta el estado del terreno o plano sobre el cual se va a ejecutar. De esta manera, es necesario comprobar en tiempo de ejecución si el ambiente presenta nuevos obstáculos como caminos obstruidos, personas accidentadas, paredes derrumbadas, etc. En un plano virtual, estos obstáculos son representados por el costo óptimo de un nodo y es necesario monitorearlo para conocer las variaciones de su costo en un tiempo *t* y así deducir si el nodo aún es óptimo para seguir tomándolo en cuenta en la trayectoria del camino más corto.

Es por ello que en esta investigación se analizará la lógica e implementación del algoritmo de búsqueda D estrella. La implementación del algoritmo D estrella está basada en la lógica del algoritmo A estrella con la particularidad de que el primero tiene mayor complejidad, ya que logra convertirse en un algoritmo de búsqueda dinámico debido a que los valores del costo del arco pueden variar a través del proceso de resolución del problema.

Al inicio de la simulación, el agente virtual comienza en un estado particular y se mueve a través de los arcos a otros estados hasta que alcanza un punto *G* el cual representa la ubicación final. Una de las diferencias importantes del algoritmo D estrella con el A estrella es que cada nodo *X*, excepto la ubicación final *G*, posee un puntero posterior a un siguiente estado *Y* (también llamado backpointer) y este trayecto es denotado por $b(X) = Y$. El algoritmo D* usa los punteros posteriores para representar los caminos hacia el nodo objetivo. Por otro lado, el algoritmo D* emplea la función $c(X, Y)$ para calcular el costo de arco, es decir, el costo requerido para atravesar un arco o trayecto desde un estado *Y* al siguiente estado *X*. Además, si el nodo *Y* no posee un arco hacia el nodo *X*, entonces el resultado de $c(X, Y)$ es indefinido. Así, dos estados *X* e *Y* son vecinos en el espacio si $c(X, Y)$ o $c(Y, X)$ se encuentran definidos (Stentz, 1994).

Asimismo, es importante mencionar que el algoritmo D* también tiene una lista abierta de estados como el algoritmo A*. La lista abierta es usada para propagar información sobre los diferentes valores del costo del arco y para calcular los costos de atravesar la ruta. Una característica adicional de este algoritmo es el uso de tags o etiquetas donde cada nodo *X* tiene un tag $t(X)$ y este puede ser NUEVO, ABIERTO o CERRADO. El primer valor de la etiqueta es asignado si el nodo nunca ha estado en la

lista abierta, el segundo si se encuentra en la lista abierta y el tercero si ya no se encuentra en la lista abierta (Stentz, 1994).

Otra característica a recalcar del algoritmo D* es que emplea dos funciones principales en su ejecución: *PROCESS-STATE* y *MODIFY-COST*. La primera función es empleada para obtener los costos de caminos más cortos hacia el nodo objetivo y la segunda se usa para modificar la función de costo de arco de cada trayecto y así almacenar los nuevos estados en la lista abierta (incluyendo la presencia de obstáculos) (Stentz, 1994).

Inicialmente, cada nodo *X* tiene asignado como etiqueta o tag el valor NUEVO, el costo de atravesar la ruta es cero y se agrega el nodo destino a la lista abierta. Seguidamente, la función *PROCESS-STATE* es ejecutada repetitivamente hasta que el nodo actual del agente virtual sea removido de la lista abierta o la función retorne -1, lo cual significa que ya se encontró la ruta de evacuación más corta o no existe alguna respectivamente. Seguidamente, el agente virtual procede a seguir la ruta trazada por los punteros posteriores hasta que alcance el nodo objetivo o descubra la existencia de un error en la función de costo del arco, esto debido a la detección de un obstáculo. Después de esto, la función *MODIFY-COST* es referenciada inmediatamente para corregir la función de costo del arco y ubicar en la lista abierta a los estados afectados. De esta forma, si se encuentra un error en el nodo *Y* del agente, nuevamente se ejecuta la función *PROCESS-STATE* hasta que retorne valores correctos para los nodos, en esta llamada a la función los cambios en los costos son propagados al estado *Y*. Así, una nueva secuencia de nodos *Y* es construida y el agente virtual recorre los punteros posteriores con el objetivo de llegar a su destino final (Stentz, 1994).

Al implementar este algoritmo en un modelo de planeamiento de rutas primero se tiene que asignar la posición de la persona virtual, robot o agente. Al agente se le asigna como posición inicial la celda de inicio y se establece el nodo destino o punto *G* que equivale a la celda objetivo. Luego, el algoritmo siempre obtiene el camino más corto desde el nodo actual, denotado por nodo *X*, hasta la celda objetivo bajo la suposición de que las celdas pueden ser recorridas siempre y cuando no estén señaladas como bloqueadas, es aquí donde el algoritmo comienza a emplear las función PROCESS-STATE, anteriormente explicada, para obtener el costo del camino más corto. Después de obtener la ruta más corta, el algoritmo recorre el camino hasta llegar a la celda objetivo o punto de evacuación y es ahí cuando termina su ejecución. Por otro lado, si en el trayecto el algoritmo reconoce que una de las celdas del camino no es atravesable,

este invoca a la función MODIFY-COST para evaluar nuevamente los nodos, actualizar sus valores en la lista abierta y así recalcular el nuevo camino más corto desde la celda actual donde se encuentra el agente hasta la celda objetivo y así sucesivamente hasta encontrar la ruta ideal (Koenig & Likhachev, 2002).

Un claro ejemplo de la ejecución del algoritmo D* se puede encontrar en un sistema implementado para la navegación de rutas con vehículos autónomos militares. Estos vehículos ya contaban con un sistema de navegación local pero no tenían implementado un algoritmo que encuentre la ruta más corta. Es así, que se implementó el uso del algoritmo D* para encontrar el camino más corto y enviarlo al sistema de navegación GPS existente. Luego, la información del algoritmo y del sistema de navegación es consolidada y enviada a un tercer sistema para emitir comandos de manejos a un controlador del vehículo. Antes de comenzar la simulación, primero se definió un ambiente de simulación de 256 x 256 celdas y con posibles obstáculos del tamaño de una celda, luego se ubicó al vehículo sobre el mapa y también el destino objetivo. Por último, el simulador fue ejecutado con el algoritmo de búsqueda D* y se obtuvo que el vehículo pudo encontrar el camino más corto en solo 6 minutos evitando 80 obstáculos encontrados en tiempo real sobre una superficie de 200 metros (Yahja et al., 1998).

Finalmente, al realizar comparaciones del algoritmo de búsqueda D* con otros algoritmos, se logró observar que sus resultados encontrados son más óptimos debido a que, ante la presencia de obstáculos, este recalcula un nuevo trayecto desde la última posición en la que estuvo el agente mientras que otros algoritmos replantean nuevamente toda la trayectoria como una nueva ruta global.

ALGORITMO 2. D estrella

0. Por cada estado X en el grafo:
    a. T(x) = NUEVO
1. D_{actual} = 0; R_{actual} = S
2. insert(G, 0)
3. val = <0,0>
4. Mientra t(s) no esté en lista CERRADA:
    a. val = PROCESS-STATE()
5. Si t(s) = NUEVO entonces
    a. Retornar que no se encontró un camino
6. R = S
7. Mientras R sea diferente a G:
    a. Si s(X, Y) es diferente a c(X,Y) entonces
        i. Si R_{actual} es diferente a R entonces
            1. D_{actual} = D_{actual} + GVAL(R, R_{actual})
            2. R_{actual} = R
        ii. Por cada S(X, Y) diferente a C(X,Y):
            1. val = MODIFY-COST(X,Y, S(X,Y))
        iii. Mientras LESS(val, COST(R)):
            1. val = PROCESS-STATE()
    b. R = b(R)
8. Retornar DESTINO FINAL ENCONTRADO

Fuente: (Stentz, 1994)

ALGORITMO 3. D estrella - PROCESS-STATE()

```
0. X = MIN_STATE()
1. Si X = NULL entonces retornar valor 0
2. Val = <f(x), k(x)>; k_val = k(x); BORRAR(X)
3. Si K_val < h(x) entonces
     a. Por cada vecino Y de X:
          i. Si t(Y) = NUEVO o
             (b(Y) = X y h(Y) <> h(X)  + c(X,Y)) o
             (b(Y) <> X y h(Y) > h(X) + c(X,Y)) entonces
               1. b(Y) = X; INSERT(Y, h(X) + c(X, Y))
4. Sino
     a. Por cada vecino Y de X:
          i. Si t(Y) = NUEVO o
             (b(Y) = X y h(Y) <> h(X) + c(X,Y)) entonces
               1. b(Y) = X; INSERT(Y, h(X) + c(X,Y))
         ii. Sino
               1. Si b(Y) <> X y h(Y) > h(X) + c(X,Y) y
                  t(X) = CERRADO entonces
                    a. INSERT(X, h(X))
               2. Sino
                    a. Si b(Y) <> X y h(X) > h(Y) +
                       c(Y,X) y t(Y) = CERRADO y
                       MENOR(val, COST(Y)) entonces
                         i. INSERT(Y, h(Y))
5. Retornar el mínimo valor encontrado
```

Fuente: (Stentz, 1994)

ALGORITMO 4. D estrella - MODIFY-COST(X, Y, c_{val})

```
0. C(X,Y) = C_val
1. Si t(X) = CERRADO entonces
     • INSERT(X, h(X))
2. Retornar el mínimo valor encontrado
```

Fuente: (Stentz, 1994)

CAPÍTULO III: METODOLOGÍA

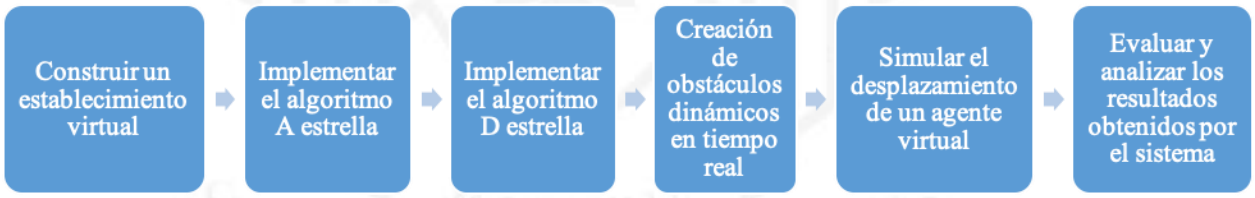
Como se mencionó en los capítulos anteriores, esta investigación tiene como objetivo construir un sistema de evacuación inteligente el cual debe contar con un simulador que implemente un algoritmo de búsqueda de caminos más cortos. De esta manera, en base a la literatura anteriormente revisada, se podrá generar un sistema para calcular las rutas de evacuación más eficientes y seguras hacia un área que no represente un peligro para las personas en casos de siniestros considerando la posición actual de un individuo.

3.1 Diseño Metodológico

Para lograr construir la propuesta de solución se llevaron a cabo 6 etapas que se pueden apreciar en la figura 3.1. Estas etapas serán de mucha importancia debido a que nos permitirán elaborar una prueba de concepto, realizar el análisis y diseño de la solución así como la implementación y la ejecución del simulador. Por último, después de realizar todas las etapas, se analizarán los resultados obtenidos.

Figura 3.1

Diagrama de las etapas requeridas en la investigación



En la primera etapa se realizó el trabajo de construir un establecimiento virtual para modelar el espacio inicial donde se simularía la evacuación. Seguido de esto, en la segunda y tercera etapa se llevaron a cabo las implementaciones del algoritmo A estrella y D estrella para posteriormente realizar sus ejecuciones, encontrar la ruta más corta y obtener resultados. Luego, en una cuarta etapa se implementó un mecanismo para la generación de obstáculos en tiempo real que serán insertados en el plano virtual dinámicamente. En la etapa número 5 se realizó la simulación del desplazamiento de un agente virtual ubicando el punto de origen o nodo inicial y el punto de destino o de extracción. Por último, en la etapa número 6 se obtuvieron los resultados arrojados por el

sistema para posteriormente analizarlos y llegar a una conclusión. Estas etapas serán explicadas a mayor profundidad en el capítulo de experimentación.

Por otro lado, para la implementación del sistema también fue necesario definir los requerimientos que deberán ser tomados en cuenta por el simulador. A continuación, se revisarán los requerimientos y el diagrama de clases realizados para llevar a cabo la implementación del simulador.

3.2 Requerimientos

3.2.1 Requerimientos funcionales:

- El sistema debe ser capaz de construir un entorno de evacuación virtual
- En el sistema se deben poder agregar objetos en 3d para ser usados como agentes virtuales, tiendas, habitaciones.
- El sistema tiene que tener la capacidad de asignar un comportamiento o lógica a cada objeto.
- El sistema deberá poder encontrar la ruta más corta en el establecimiento virtual.
- El sistema deberá tener la capacidad de generar obstáculos dinámicos sobre el plano.

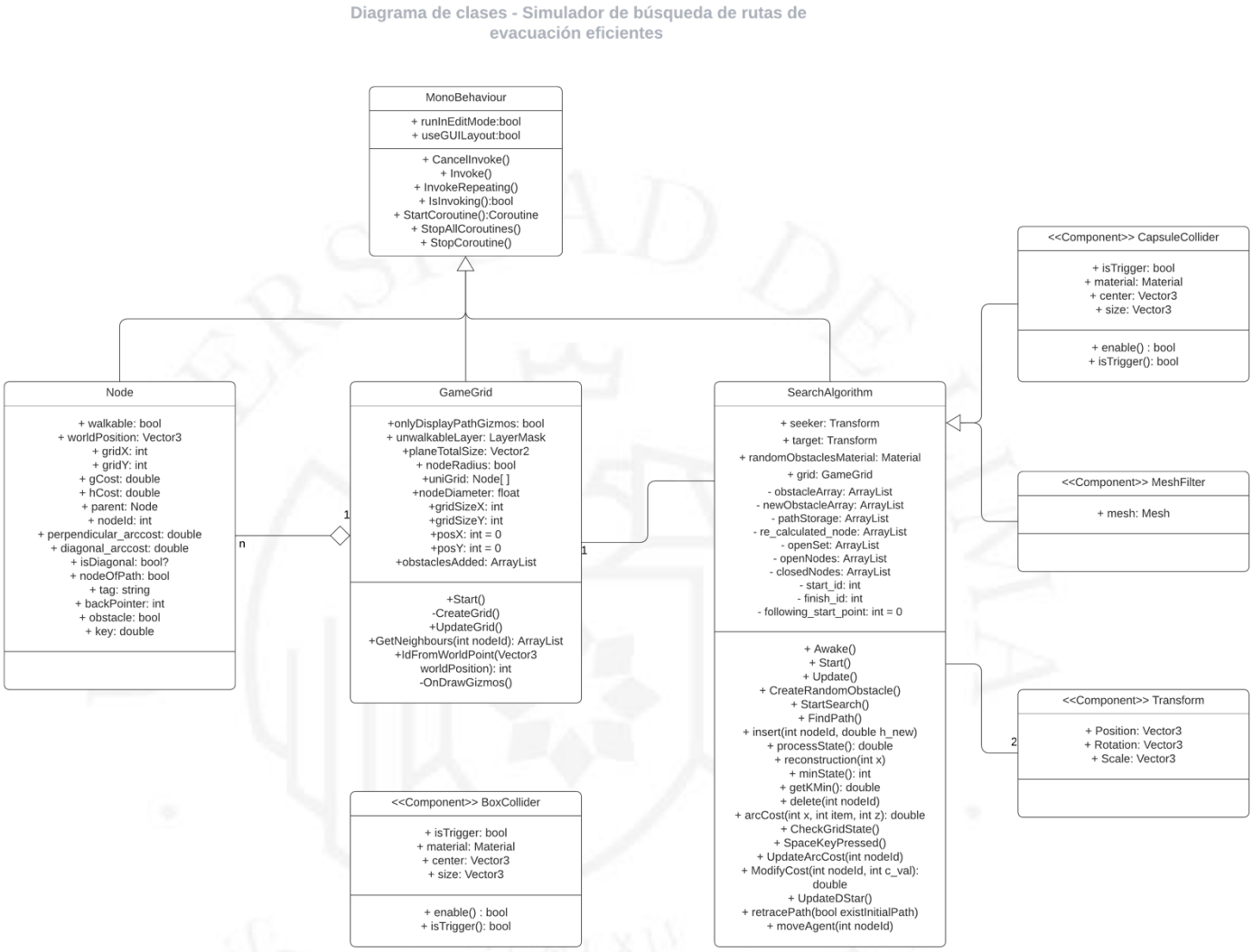
3.2.2 Requerimientos no funcionales:

- El sistema debe ser capaz de enlazar el entorno de evacuación virtual con código de programación que implemente un algoritmo para encontrar la ruta más corta.
- El sistema deberá ejecutar el algoritmo de caminos más corto de la forma más eficiente.
- El sistema tiene que tener la capacidad de medir y registrar los tiempos de ejecución desde el inicio de la simulación hasta que esta finalice.

3.3 Diagrama de clases del sistema

Figura 3.2

Diagrama de clases del simulador de búsqueda de rutas de evacuación eficientes



Como se puede observar en el diagrama de clases, la implementación del sistema se llevará acabo usando 3 clases que extienden de la clase MonoBehavior, la cual es la clase base del motor de desarrollo Unity. La utilidad de estas 3 clases serán explicadas en la siguiente sección. Además, también se emplearán componentes nativos de Unity como Transform, MeshFilter y Colliders para la representación del agente virtual, la salida de escape y de los obstáculos en el establecimiento.

3.4 Ambiente de la simulación

Para representar la ejecución del algoritmo planteado se decidió hacer una simulación sobre un plano representando el primer nivel de un centro comercial que cuenta con un conjunto de tiendas. Además, con el objetivo de contar con múltiples ambientes de experimentación, la investigación tomó en cuenta diferentes variaciones del escenario inicial para así comprobar la funcionalidad del algoritmo D* y a la vez conocer el desempeño de este en diferentes escenarios. Dichas variaciones están relacionadas a la presencia de diferentes obstáculos creados manual y aleatoriamente sobre el plano y también se añadieron variaciones relacionadas a la escala del plano.

Debido a que el propósito de la investigación se basa en encontrar la ruta más corta en el menor tiempo posible, es importante conocer la complejidad en tiempo del algoritmo D estrella ya que este también nos indicará cuán óptima es su implementación en código. Para ello se usará la notación **Big-O** la cual representa la complejidad de los algoritmos en el peor de los escenarios. Con esta notación, es fácil comparar diferentes algoritmos ya que nos indica claramente cómo se comporta el algoritmo cuando el conjunto de datos de entrada incrementa (Dineshpathak, 2022). En base a la implementación del algoritmo A estrella y D estrella, se pudo observar que la complejidad de estos algoritmos en el peor de los casos es una complejidad cuadrática o $\boldsymbol{O(n^2)}$, esto debido a que dichos algoritmos verifican la trayectoria y los posibles caminos de cada nodo, y eso lo realiza por cada nodo encontrado en el plano de evacuación, es así que el algoritmo como mínimo tiene que iterar 2 veces el plano representado.

Sin embargo, esta complejidad en tiempo generalmente es más óptima debido a que estos algoritmos también implementan una función heurística la cual se encarga de recolectar más información en cada tramo del grafo y así reducir el tiempo de ejecución.

CAPÍTULO IV: EXPERIMENTACIÓN

En este capítulo de experimentación se expondrá a mayor profundidad cada etapa mostrada en la figura 3.1 del capítulo 3 de metodología. Además, cada una de estas etapas están alineadas a los requerimientos funcionales y no funcionales mencionados anteriormente y al diagrama de clases diseñado para la solución propuesta.

Para la implementación del sistema propuesto es necesario utilizar como variable de entrada un grupo de datos que serán empleados al ejecutar la simulación. Así, esta investigación considerará como variables de entrada al costo de arco o los pesos entre los nodos, la cantidad de nodos en un plano y la distancia entre estos. En relación a los datos de salida del sistema, se considerarán los tiempos de ejecución experimentales obtenidos de la ejecución de los algoritmos A estrella y D estrella para calcular la ruta más eficiente tomando en cuenta distintos escenarios.

Por último, la ejecución del sistema para obtener el camino más eficiente se llevó a cabo en una estación de trabajo con las siguientes especificaciones: procesador Intel Core i7 6-core 2.6Ghz, 16 GB DDR4 de memoria RAM, tarjeta gráfica Radeon Pro 560X 4GB y sistema operativo MacOS Big Sur.

4.1 Proceso de construir un establecimiento virtual

La primera etapa requerida para generar la solución planteada es comenzar a construir un establecimiento bajo un entorno virtual. En esta etapa se realizará la configuración base o inicial del ambiente. Para esto, se utilizará Unity 3d la cual es una herramienta multiplataforma que trabaja en diferentes sistemas operativos. Con esta solución se podrá crear contenido interactivo y dinámico en 3 dimensiones aunque también es posible generar contenido en 2d. Además, cabe mencionar que esta aplicación soporta lenguajes de programación como UnityScript y C#.

Es importante mencionar que el uso del software Unity será empleado a lo largo del proceso de construcción del sistema de evacuación inteligente debido a que nos da la capacidad de crear entornos visuales junto con lógica de programación. De esta forma, con la ayuda de la herramienta se podrá modelar de forma virtual el primer nivel de un centro comercial incluyendo figuras solidas sobre el plano que representan obstáculos

como tiendas por departamento, restaurantes, paredes, zonas restringidas, pasadizos, etc. Además, estos obstáculos podrán ser creados inicialmente o dinámicamente en la ejecución del simulador.

Por otra parte, para la construcción del plano virtual, se emplearán variables de tipo Vector2 para guardar las dimensiones y el área del lugar (largo y ancho). Además, también se usará este tipo de variables para almacenar la ubicación de cada objeto creado. Seguido de esto, para que la ubicación del agente pueda ser representado en el espacio, el sistema va a dividir el plano virtual en diferentes nodos o también llamado plano en forma de grillas. Cada nodo generado tendrá una dimensión de un metro cuadrado (diámetro de cada nodo).

4.2 Proceso de implementar el algoritmo A estrella

En esta etapa de la investigación se llevó a cabo la implementación del primer algoritmo del simulador para encontrar rutas más cortas sin tomar en cuenta obstáculos dinámicos en tiempo real sobre el plano. Este algoritmo hará uso de 3 clases diferentes: GameGrid, Node y Pathfinding.

El script Node es una clase que representa un nodo en memoria e inicializa cada nodo con variables fundamentales como su identificador único, su posición en el espacio 3d, las diferentes posiciones en el plano, el valor del puntero posterior, el valor del campo tag, el valor de su función de costo y un valor verdadero o falso que indicará si dicho nodo es un obstáculo.

El script GameGrid se encargará de dividir el plano en nodos en base a las dimensiones establecidas del plano y del tamaño del nodo o también llamado nodeRadius. Además, esta clase implementará la lógica para representar el plano virtual. Es decir, su función será generar el plano con sus dimensiones en memoria y encontrar los nodos disponibles para el recorrido y los nodos considerados como obstáculos iniciales que fueron añadidos a través del lado gráfico del simulador. Por otro lado, el script también se encargará de mantener el plano actualizado con el fin de guardar los nuevos obstáculos que fueron insertados o generados en tiempo de ejecución del simulador. Asimismo, el script tendrá la capacidad de obtener todos los vecinos de la posición actual del nodo a través de un método getNeighbours y también se encargará de retornar el identificador único de cada nodo basado en su posición. Por último, esta clase será responsable de

procesar todo lo anteriormente mencionado y representarlo gráficamente en el plano virtual.

Por último, el tercer script Pathfinding es el script principal para poder ejecutar este algoritmo y enlazarlos con el motor gráfico Unity. Para iniciar, se guardan en dos nodos el nodo de inicio y el nodo objetivo o de destino. Luego, se crean e inicializan las listas ABIERTA y CERRADA y se agrega el nodo de inicio a la lista ABIERTA. Seguidamente de esto, se declara un ciclo iterativo hasta que la lista ABIERTA tenga al menos un elemento, se obtiene el primer elemento de esta lista y este se inserta en la lista CERRADA. Luego, si el elemento es igual al nodo objetivo el proceso se detiene y se traza la ruta encontrada. Si es que la condición no se cumple, se obtienen todos los nodos vecinos del nodo actual y se declara otro ciclo iterativo. En este nuevo ciclo, se evalúa si es que el vecino es transitable, es decir, que no sea un obstáculo existente o que el vecino esté en la lista CERRADA. Si esta condición es cierta, se omite el nodo vecino y se continúa con el siguiente. En caso contrario, se obtiene el costo para moverse desde el nodo actual hasta el nodo vecino y seguido de esto se evalúa si es que este costo es menor al costo G del vecino o que el vecino no se encuentre en la lista CERRADA. Si esta condición es cierta, se actualizan los costos del vecino y se establece el nodo actual como padre del nodo vecino. Por último, si el vecino no se encuentra en la lista ABIERTA, se agregará el nodo a esta lista.

4.3 Proceso de implementar el algoritmo D estrella

Esta etapa es la de mayor importancia en el proceso de desarrollo del sistema de evacuación debido a que se implementará el algoritmo de búsqueda D estrella y posteriormente su lógica se enlazará con el plano virtual generado en Unity. Para lograr lo mencionado, se reutilizarán las clases GameGrid y Node implementadas anteriormente en el algoritmo A estrella y además se implementará el script SearchAlgorithm.

El script SearchAlgorithm es la clase principal para calcular y obtener la ruta más óptima. Este será responsable de implementar la lógica del algoritmo D estrella. En la primera etapa, en el lado gráfico del simulador se crearán dos GameObjects que representarán la posición inicial del agente virtual y el punto de evacuación seguro más cercano. Seguidamente, se inicializará un arreglo ABIERTO y un arreglo CERRADO con cero elementos. El primer arreglo se encargará de almacenar el conjunto de nodos a

ser evaluados en la simulación y el segundo tiene como función guardar los nodos que ya fueron evaluados a través de la ejecución del algoritmo.

Por último, este script tendrá una función con la capacidad de encontrar la ruta más corta y eficiente desde la posición inicial del individuo hasta el punto de evacuación seguro. En el proceso de encontrar la ruta más óptima se van a ejecutar funciones importantes para el funcionamiento del simulador, las dos más relevantes son *process-state()* y *modify-cost()*. Además, estas funciones cuentan con funciones auxiliares que se encargarán de borrar, insertar, obtener el nodo con menor costo y recalcular la búsqueda del algoritmo en caso se presenten nuevos obstáculos. También es importante mencionar que estas funciones importantes tienen la capacidad de gestionar planos dinámicos que presenten obstáculos en tiempo real.

4.4 Proceso para crear obstáculos dinámicos en tiempo real

Como tercera fase de la metodología y con el objetivo de simular un entorno semejante al de una evacuación real llevada a cabo en el primer nivel de un centro comercial, se consideró una etapa específica para la implementación de lógica que genere aleatoria y dinámicamente obstáculos sobre el ambiente.

Así, además de implementar el algoritmo D estrella, se creó una función llamada *createRandomObstacles()*. Esta función será responsable de generar GameObjects 2D de tipo Cubo, definir las posiciones y las dimensiones de los objetos en base a valores aleatorios y establecerlos en el plano virtual.

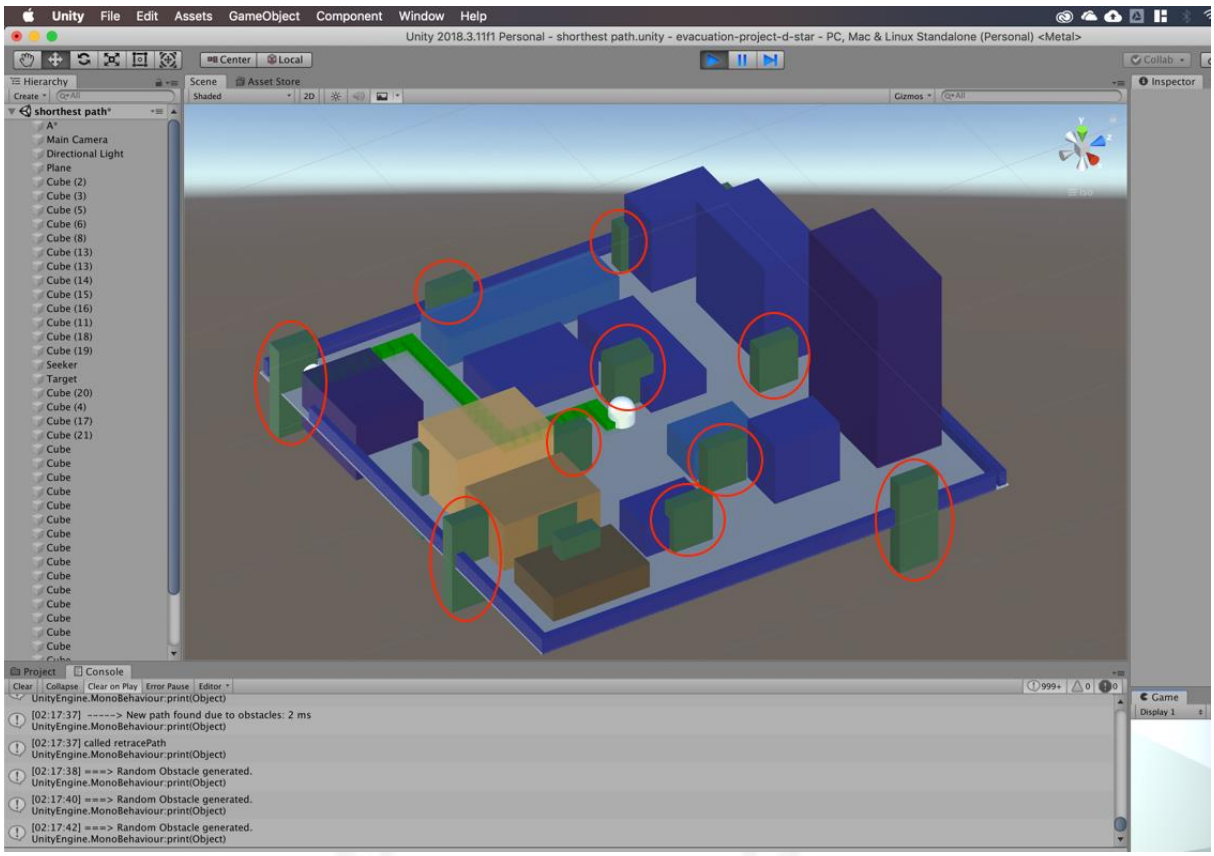
Finalmente, es necesario ejecutar la función *createRandomObstacles()* múltiples veces para generar diferentes obstáculos dinámicos en la simulación. Para esto, se empleará la función nativa de Unity llamada *InvokeRepeating()*. Esta función recibe 3 parámetros: el primero será la función a ejecutar, en este caso será la función: *createRandomObstacles()*. Con el segundo parámetro indicaremos que esta se ejecute cada tres segundos y, por último, con el tercer parámetro especificaremos que el proceso comience cinco segundos después de iniciar la simulación.

Debido a la nueva inserción de esta lógica en el simulador, se tuvo que contemplar que el algoritmo de búsqueda D estrella se continúe ejecutando en paralelo a la generación de los obstáculos aleatorios, es decir, que el replaneamiento de rutas alternas tome en cuenta estos nuevos obstáculos generados en tiempo real.

A continuación, en la figura 4.1 se muestra el estado del plano virtual después de haber iniciado la ejecución del simulador y al generar obstáculos dinámicos, los cuales están representados de color verde.

Figura 4.1

Estado del plano virtual con obstáculos aleatorios generados en tiempo real



4.5 Simular el desplazamiento de una persona o agente virtual

En esta cuarta etapa se lleva a cabo el proceso de ejecución del simulador. Durante este proceso, se le asignará al agente virtual un patrón de desplazamiento o comportamiento en base a los resultados del algoritmo A estrella y D estrella implementados y así se encontrará la ruta más óptima de forma rápida.

Para comprender mejor la ejecución del simulador, este se dividirá en dos fases. Primero se establecerá en el plano virtual la posición inicial del individuo y la ubicación del punto de extracción seguro. Seguido de esto, el simulador podrá comenzar con el

procesamiento del plano y así obtener la ruta más corta a recorrer desde la posición inicial hasta la zona de evacuación.

En las figuras 4.2 y 4.3 se puede observar la fase inicial del simulador y el recorrido calculado después de ejecutar el algoritmo respectivamente.

Figura 4.2

Estado inicial del plano virtual antes de ejecutar el simulador

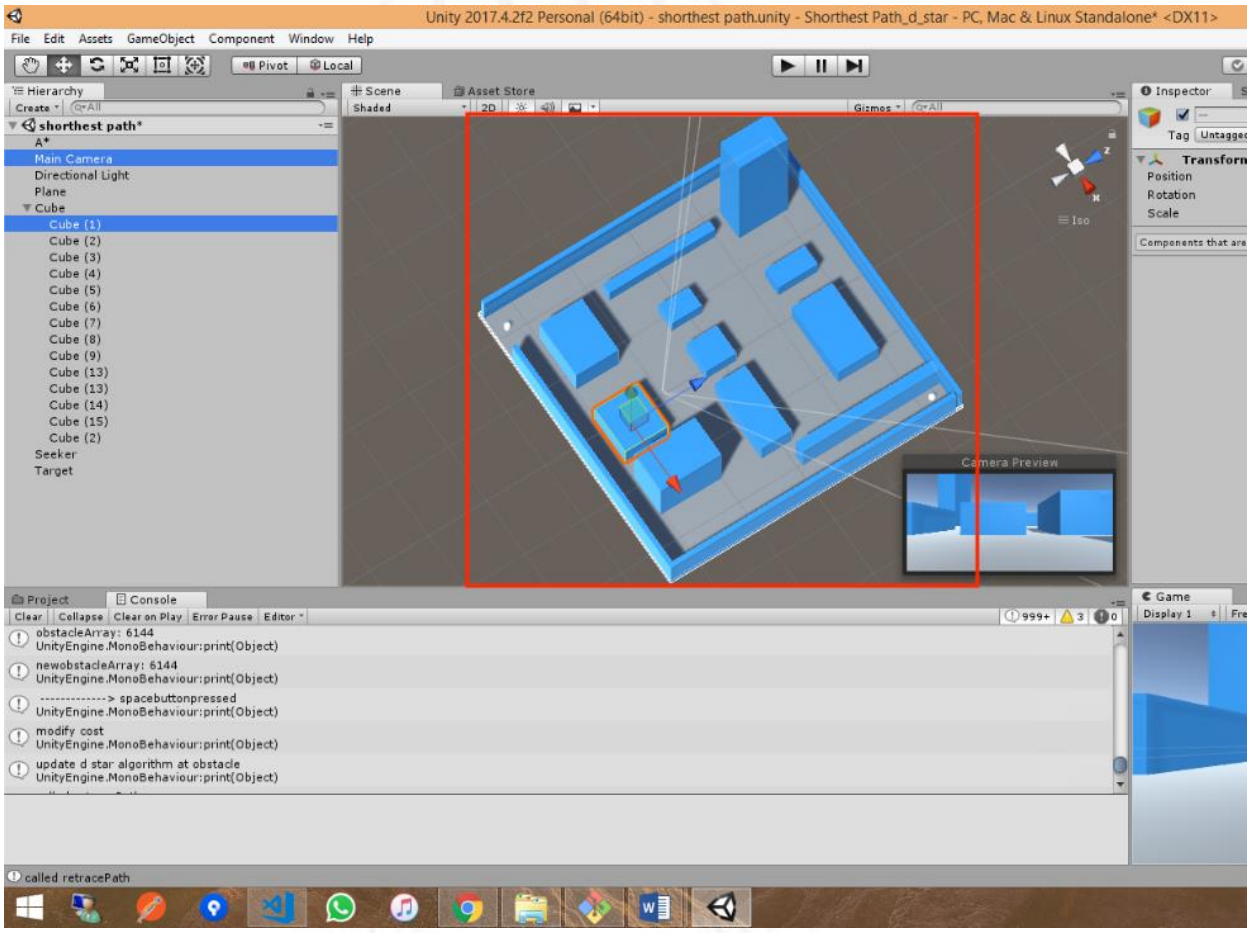
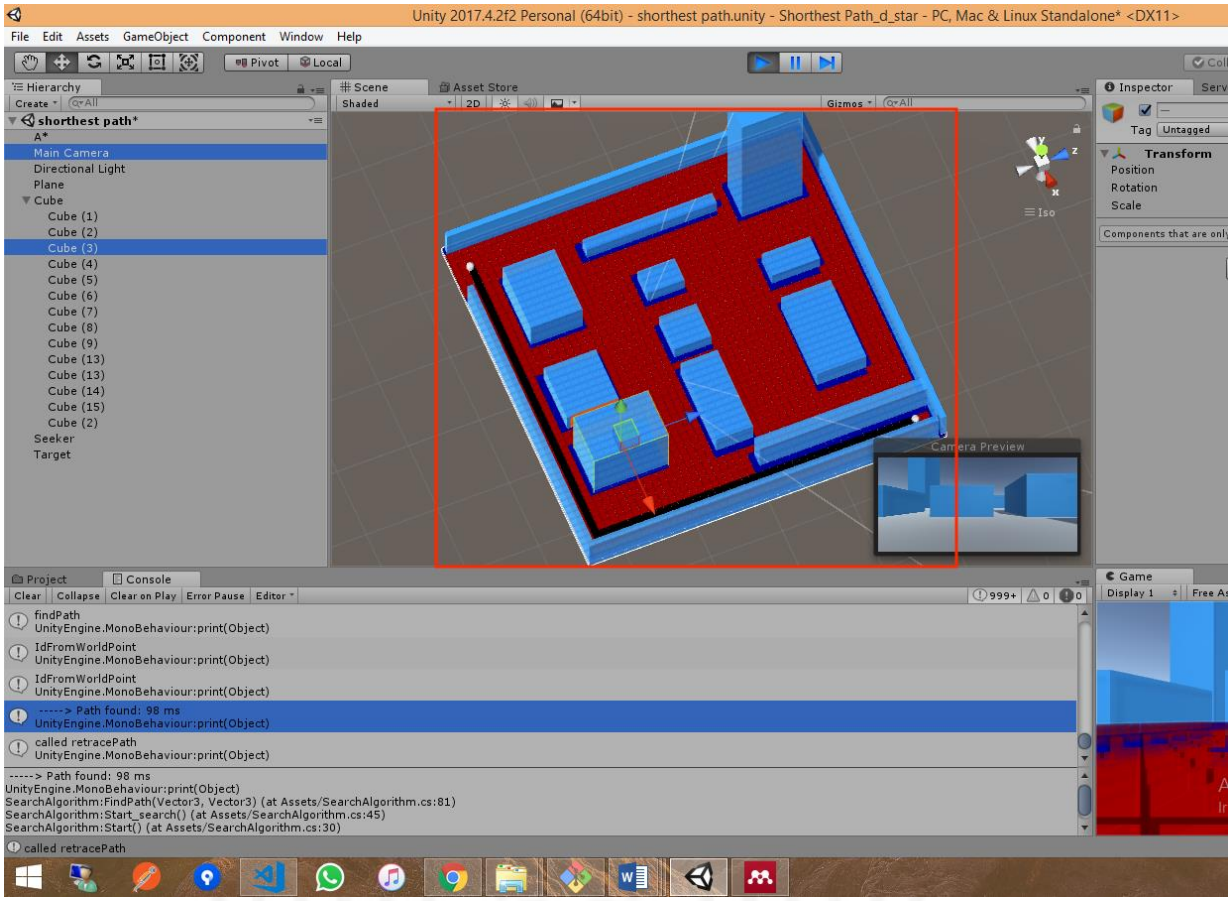


Figura 4.3

Plano virtual después de ejecutar la simulación y el camino más corto encontrado



En la segunda etapa de la ejecución del simulador se insertarán manualmente y en tiempo de ejecución nuevos obstáculos en el plano virtual con el objetivo de recalcular un nuevo recorrido en caso alguno de estos interfieran con la posición actual del individuo o con el camino encontrado inicialmente.

A continuación se puede observar que en la figura 4.4 se insertan obstáculos en el recorrido inicial y en la figura 4.5 se observa el replaneamiento de la ruta obtenida por el algoritmo D estrella.

Figura 4.4

Inserción de obstáculo sobre la ruta más corta encontrada inicialmente

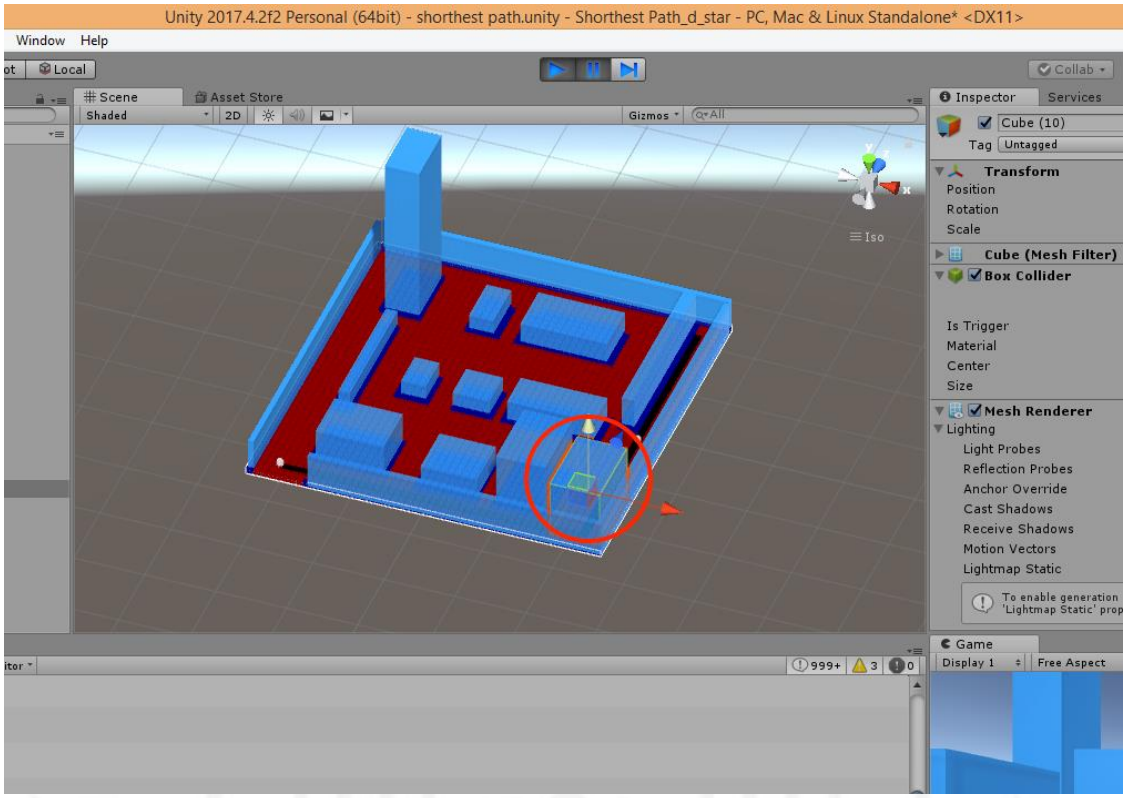
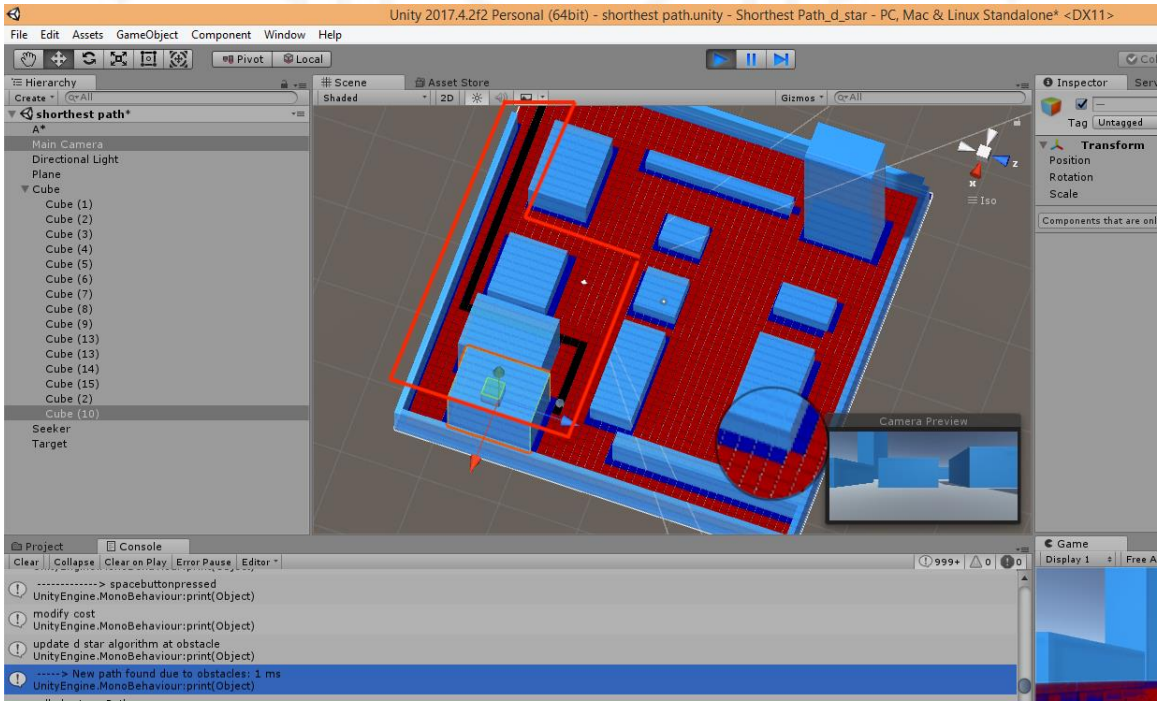


Figura 4.5

Replaneamiento de la ruta de evacuación más corta ante la presencia de un obstáculo en el camino inicial



4.6 Evaluar y analizar los resultados obtenidos por el sistema

En la última fase del proceso de implementación del sistema se evaluarán los resultados obtenidos y serán analizados. Los resultados a conocer serán los tiempos de ejecución experimentales calculados por el sistema al usar los algoritmos A estrella y D estrella para encontrar el camino más corto. Por otra parte, también se validarán y analizarán los resultados obtenidos al ejecutar el sistema con el algoritmo D estrella en escenarios con distintos números de obstáculos iniciales y generados en tiempo real, y diferentes tamaños del plano. Así, se logrará demostrar el correcto comportamiento del algoritmo y su funcionamiento.

Adicionalmente, también se obtendrá como resultado el tiempo de ejecución requerido al realizar un replaneamiento del camino ante la aparición de nuevos obstáculos que interfieran con la ruta más corta encontrada inicialmente.

CAPÍTULO V: RESULTADOS

Luego de la ejecución del simulador, para poder obtener los resultados se calcularon y almacenaron los tiempos de ejecución experimentales empleados por el sistema para encontrar el camino más corto hacia el punto de evacuación seguro.

Estas mediciones se llevaron a cabo con el paquete *System.Diagnostics* que "proporciona clases que permiten interactuar con procesos del sistema, registros de eventos y contadores de rendimiento" (Microsoft, 2016). Así, ciertos métodos del paquete fueron usados para calcular el tiempo empleado por el algoritmo desde el comienzo hasta el final de su ejecución.

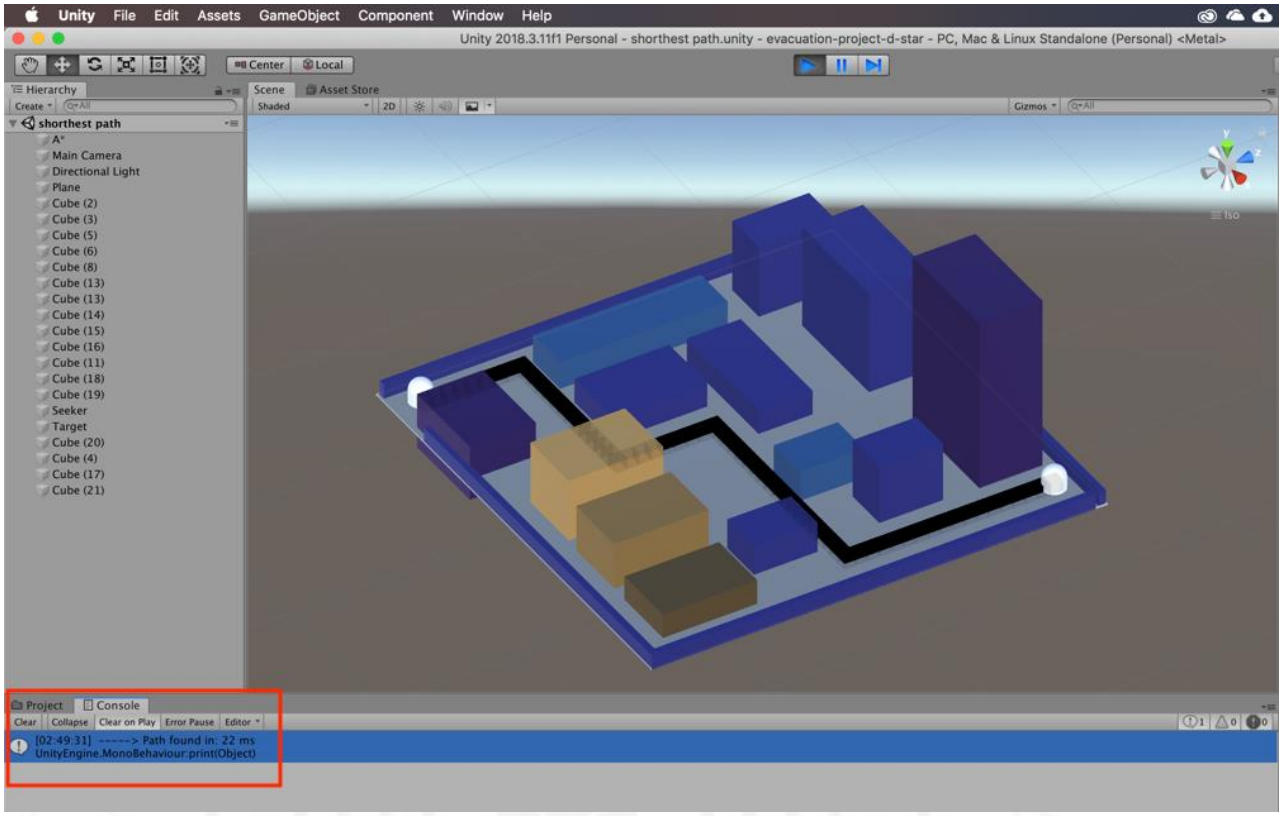
Los resultados serán divididos en tres etapas: resultados de la ejecución con el algoritmo D estrella, resultados de una comparación entre el algoritmo A estrella y D estrella y por último la validación de los resultados ejecutando el algoritmo D estrella bajo diferentes escenarios. Además, la primera etapa de los resultados constará de tres partes.

En la primera parte se realizó una simulación para ejemplificar la evacuación de una persona o agente en el primer nivel de un centro comercial usando el camino más corto y considerando solo su posición inicial y el punto de evacuación seguro más cercano. Asimismo, este criterio será usado más adelante para la validación de los resultados en base a diferentes ejecuciones del algoritmo en diferentes entornos.

Como consecuencia, en la figura 5.1 se visualiza la ejecución del algoritmo y el tiempo que empleó este para obtener la ruta más corta. En esta primera parte, el camino más corto fue encontrado en 22 milisegundos, es decir, 0.022 segundos.

Figura 5.1

Ejecución del sistema y el tiempo empleado para encontrar la ruta más corta

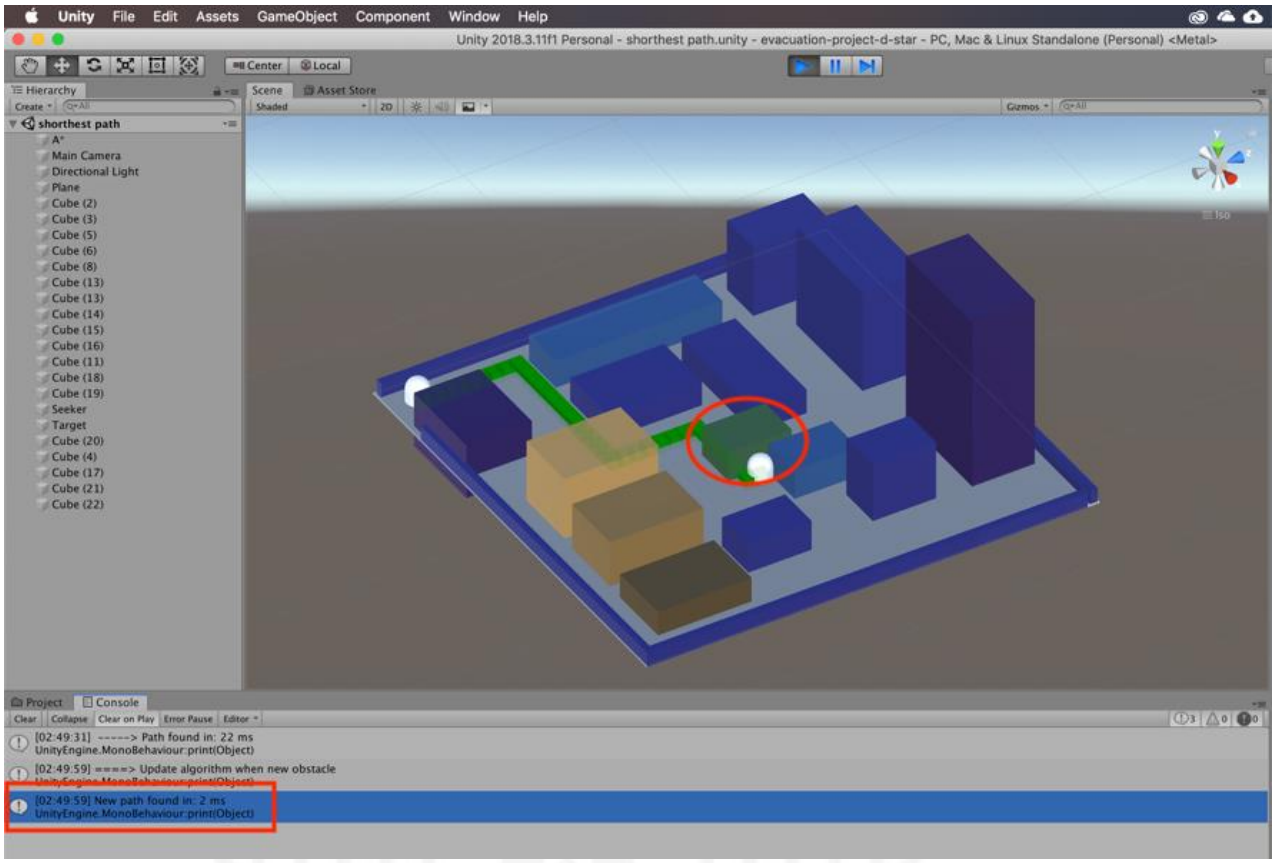


Para la segunda parte de resultados se realizó una simulación para ilustrar el caso en el que un agente necesita evacuar el primer nivel de un centro comercial pero su ruta de evacuación encontrada previamente es bloqueada por la presencia de un solo obstáculo. En relación a la presencia de este obstáculo, el algoritmo realiza un replaneamiento de la ruta a medida que el agente se desplaza por el camino inicial y reconoce que uno de los nodos está obstruido por el obstáculo. Es en base a este problema encontrado que el algoritmo comienza a buscar la segunda mejor ruta.

De esta forma, en la figura 5.2 se logra observar el tiempo requerido para encontrar la nueva ruta más corta debido a un obstáculo que interfirió con el desplazamiento del agente. Así, se obtuvo que el algoritmo empleó 2 milisegundos, o 0.002 segundos, en recalcular el nuevo camino más corto.

Figura 5.2

Ejecución del sistema y el camino más corto encontrado en base la detección de un obstáculo

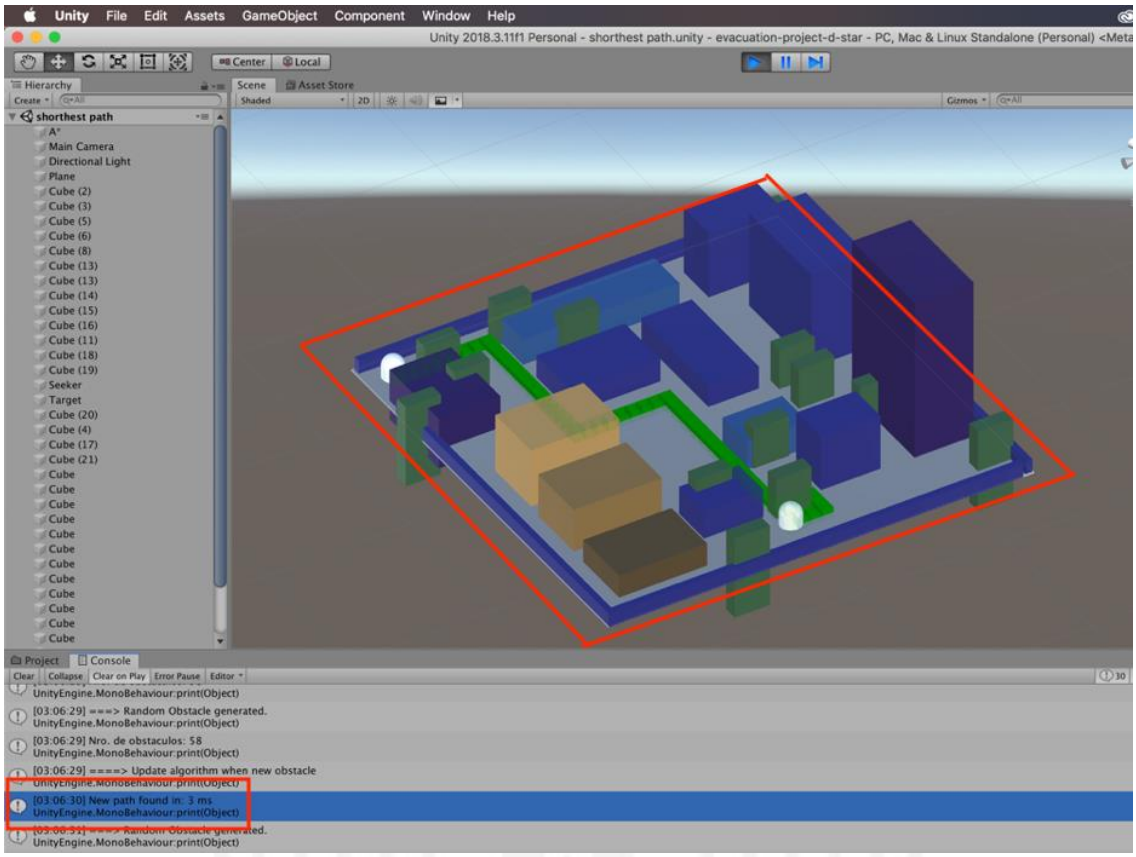


Para finalizar la primera etapa de resultados, también se tomó en cuenta un tercer escenario en el que se generaron obstáculos dinámicos en tiempo real. Esta simulación demuestra una evacuación más frecuente donde una persona o agente necesita evacuar el primer nivel de un centro comercial pero el camino en el que se encuentra es bloqueado por múltiples obstáculos como tiendas derrumbadas, pasadizos obstruidos, salidas cerradas, etc. Así, se realizó la simulación para obtener el tiempo de ejecución experimental en milisegundos que emplea el algoritmo para encontrar la nueva ruta más corta ante la presencia de múltiples obstáculos encontrados en tiempo real que interceptan con la ruta inicial.

Como consecuencia, en la figura 5.3 se puede observar el tiempo empleado para recalcular el nuevo camino más corto cuando el sistema encontró 58 obstáculos dinámicos sobre el plano creados en posiciones aleatorias y alguno interfirió con el desplazamiento inicial del agente. Para este caso, el algoritmo requirió 3 milisegundos, o 0.003 segundos, para encontrar la nueva ruta más corta en el plano.

Figura 5.3

Ejecución del sistema y la nueva ruta más corta encontrada ante la aparición de obstáculos dinámicos



A continuación, a manera de ilustración, en la tabla 5.1 se presenta una comparativa de los resultados obtenidos en los diferentes escenarios.

Tabla 5.1

Tabla de tiempos de ejecución obtenidos como resultado de la simulación

Algoritmo	Tiempo empleado para encontrar la ruta más corta sin obstáculos	Tiempo de ejecución para recalcular el camino con 1 obstáculo añadido manualmente	Tiempo de ejecución para recalcular la ruta con obstáculos dinámicos
Algoritmo D estrella	22 milisegundos o 0.022 segundos	2 milisegundos o 0.002 segundos	3 milisegundos o 0.003 segundos

Para la segunda etapa de los resultados, se planteó hacer una comparación de los tiempos de ejecución requeridos por los algoritmos A estrella y D estrella con el objetivo de conocer la diferencia de tiempo en milisegundos que le toma a cada algoritmo al encontrar la ruta más corta en el plano.

Sin embargo, debido a que el algoritmo de búsqueda A estrella no es un algoritmo de enrutamiento dinámico, no es recomendable emplearlo para recalcular caminos en escenarios que presentan obstáculos dinámicos. Así, la comparación planteada se realizó en un escenario en el que se tiene que encontrar el camino más corto a través de obstáculos existentes sobre el plano.

Esta comparativa ilustra la situación en la que una persona o agente requiere evacuar el primer nivel de un centro comercial donde se presentan diferentes obstáculos iniciales. Además, se desea conocer cuál sería el algoritmo ideal a ser empleado por el agente en dicho contexto. Para esto, se recrearon dos escenarios iguales bajo el motor de Unity, con la diferencia de que un escenario tiene asignado como algoritmo de búsqueda al A estrella y el otro escenario tiene asignado al D estrella. En las figuras 5.4 y 5.5 se puede apreciar de forma gráfica estos escenarios respectivamente y además los resultados arrojados en la ejecución de cada escenario.

Figura 5.4

Plano virtual en Unity que tiene asignado el algoritmo de búsqueda A estrella

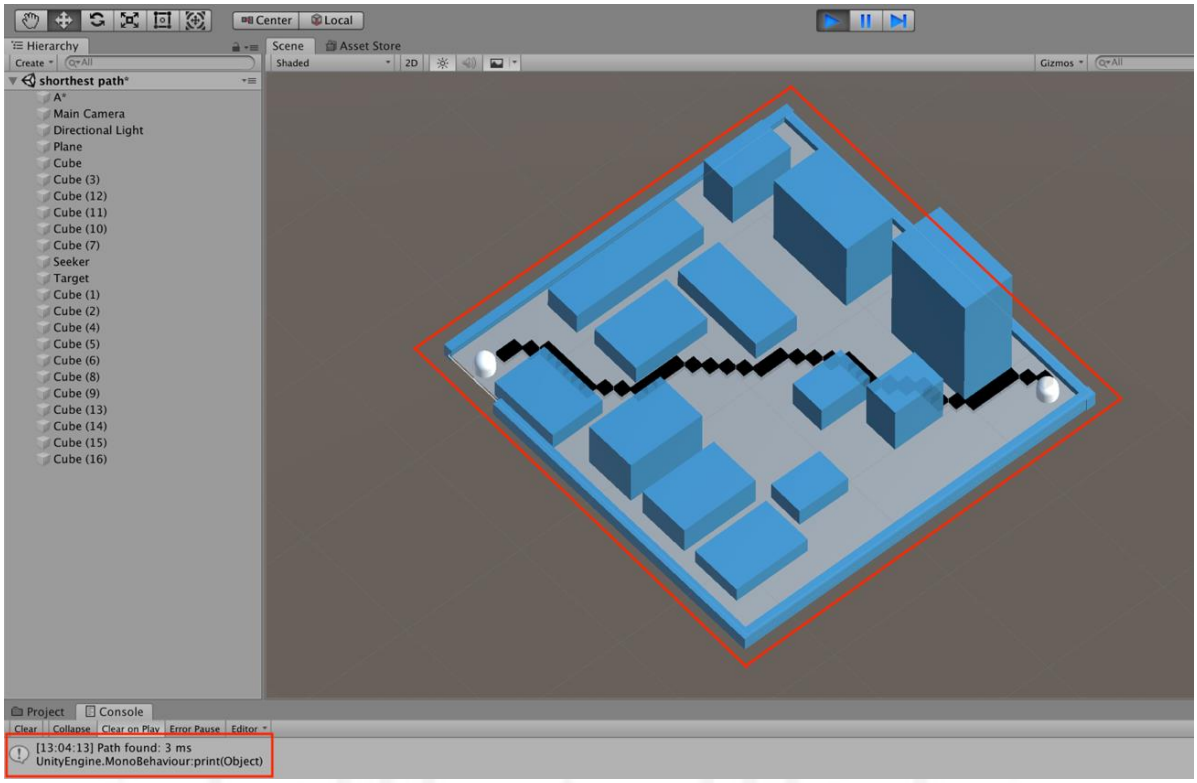
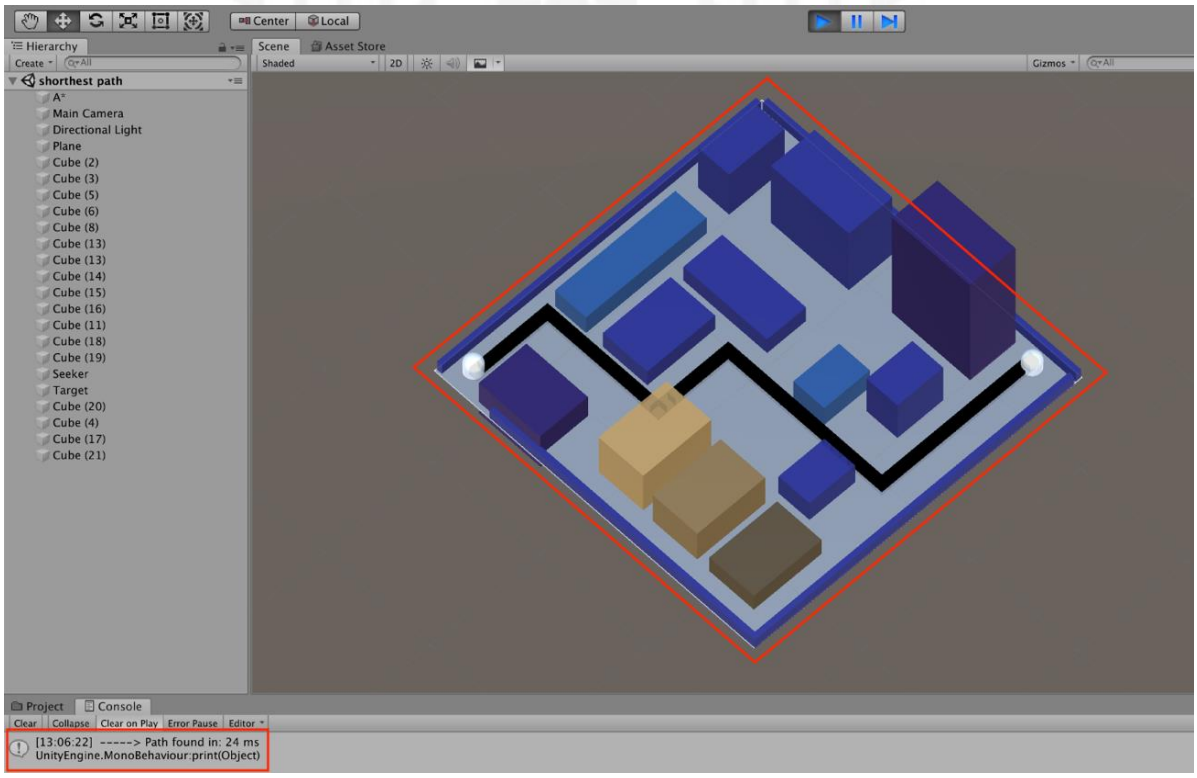


Figura 5.5

Plano virtual en Unity que tiene asignado el algoritmo de búsqueda D estrella



Como podemos observar, los resultados en consola muestran el tiempo de ejecución experimental que le tomó a cada algoritmo encontrar el camino más corto hacia el punto de evacuación.

De esta forma, para resumir los resultados obtenidos se presenta la tabla comparativa número 5.2.

Tabla 5.2

Tabla comparativo de los resultados del algoritmo A estrella y D estrella

Escenario	Algoritmo de búsqueda A estrella (A*)	Algoritmo de búsqueda D estrella (D*)
Tiempo empleado para calcular la ruta más corta con obstáculos existentes	3 milisegundos o 0.003 segundos	24 milisegundos o 0.024 segundos

En base a la tabla comparativa 5.2 de esta segunda etapa de los resultados, podemos concluir que el algoritmo de búsqueda A* es mucho más rápido y eficiente para escenarios en el que un agente necesita evacuar y en su alrededor no se presenten obstáculos dinámicos que interfieran con su ruta. Esto debido a que presenta una diferencia de 21 milisegundos al ser ejecutado respecto al algoritmo de búsqueda D*. Sin embargo, como se observa en la tabla anterior número 5.1, el uso del algoritmo de búsqueda D* es mucho más conveniente y necesario para escenarios donde un agente requiere realizar una evacuación en una situación donde exista la presencia de obstáculos dinámicos ya que el algoritmo recalcula la ruta automáticamente en el menor tiempo posible.

Para culminar con la fase de resultados, se realizó una etapa de validación del funcionamiento del algoritmo. Para esta etapa se empleó una técnica utilizada en las literaturas revisadas que implementaron algoritmos de caminos más cortos. Esta técnica se basa en la ejecución del algoritmo en diferentes ambientes relacionados a la variación del tamaño del plano virtual y el número de obstáculos existentes sobre este. Asimismo, se escogió uno de los escenarios para ejecutarlo en distintas oportunidades y así verificar la consistencia del algoritmo y el tiempo experimental encontrado en cada ejecución.

A continuación, se explicarán los 3 escenarios empleados para llevar a cabo la etapa de validación de los resultados.

5.1 Primer escenario

Como primer escenario se consideró la variación de las dimensiones del plano. Esto debido a que la complejidad del algoritmo es cuadrática $O(n^2)$ y se necesita analizar cómo el tamaño del plano afecta al desempeño del algoritmo. Es así que a mayor escala se tendrá un mayor número de nodos sobre el plano y, por consiguiente, una mayor área a ser calculada por el algoritmo, mayor tiempo de cálculo y mayor tiempo de ejecución.

En base a ello, se llevaron a cabo ejecuciones con planos de escala 5, 10 y 15. Además, se consideró de forma inicial 10 obstáculos para todas las ejecuciones y así poder realizar una comparación de los tiempos encontrados. Así, se presenta la tabla comparativa número 5.3 con los resultados obtenidos en cada variación del tamaño del plano.

Tabla 5.3

Tabla comparativa de los resultados obtenidos de las variaciones del tamaño del plano

Escala del plano virtual	Tiempo de ejecución experimental encontrado (ms)
Escala 5	101 milisegundos
Escala 10	543 milisegundos
Escala 15	1474 milisegundos

Al analizar la tabla comparativa presentada, se observa que el tiempo de ejecución experimental es directamente proporcional al aumento de la escala del plano. Es decir, el simulador empleará más tiempo en encontrar la ruta óptima cuando el área de evacuación sea más grande.

5.2 Segundo escenario

Para este segundo escenario se tomó en cuenta la cantidad de obstáculos iniciales ubicados en el plano virtual. Como configuración inicial, se estableció un plano de escala 5 y las simulaciones se llevaron a cabo con 10, 15 y 20 obstáculos sobre el establecimiento. Este escenario tiene como objetivo analizar los diferentes tiempos de ejecución respecto a una mayor presencia de obstáculos.

Seguidamente, en la tabla comparativa número 5.4 se presentan los resultados encontrados al ejecutar el algoritmo con una cantidad de 10, 15 y 20 obstáculos.

Tabla 5.4

Tabla comparativa de los resultados obtenidos de las variaciones de obstáculos

Número de obstáculos en el plano virtual	Tiempo de ejecución experimental encontrado (ms)
10	99 milisegundos
15	103 milisegundos
20	104 milisegundos

Al analizar los resultados de la tabla 5.4 se observa que el tiempo de ejecución no varía de manera sustancial en comparación al escenario anterior. De esta forma, se puede entender que, con el algoritmo D estrella, el número de obstáculos iniciales afecta al tiempo de ejecución en menor medida comparado a una alteración de la escala del plano. Por otro lado, la variación del tiempo de ejecución experimental está sujeta a la presencia de obstáculos en la ruta de desplazamiento del agente hacia el punto de extracción seguro.

5.3 Tercer escenario

Para el tercer y último escenario, se consideró tomar como referencia a Stentz (1994) para realizar la validación de resultados experimentales y así obtener conclusiones sobre la implementación del simulador y el algoritmo D estrella. Además, es importante mencionar que en las literaturas anteriormente revisadas también se usa el patrón de Stentz para analizar y validar el uso de otros algoritmos o técnicas.

De esta forma, se definió un nuevo escenario en base a los seis escenarios observados en los puntos anteriores. Sobre este nuevo escenario se realizaron 5 ejecuciones sobre un área de escala 10 con 15 obstáculos definidos inicialmente. Finalmente, se compararon los resultados para verificar los tiempos de ejecución de las simulaciones realizadas.

A manera de resumen, se presenta la tabla comparativa número 5.5 que contiene los tiempos de ejecución obtenidos al simular múltiples veces el escenario propuesto.

Tabla 5.5

Tabla comparativa de los tiempos de ejecución para validar la consistencia del algoritmo cuando se ejecuta el simulador múltiples veces

Número de simulación realizada	Tiempo de ejecución experimental encontrados (ms)
1	612 milisegundos
2	607 milisegundos
3	621 milisegundos
4	612 milisegundos
5	605 milisegundos

Al observar los resultados de la tabla 5.5, se aprecia que los tiempos de ejecución experimental son semejantes en las 5 simulaciones realizadas sobre el mismo escenario y que la diferencia entre estos no excede los 16ms. En conclusión, se infiere que el algoritmo de búsqueda D estrella es consistente ya que en promedio el tiempo de búsqueda de la ruta más corta siempre es el mismo, lo cual le asegura al usuario que el sistema cuenta con un algoritmo confiable y estable. Además, es importante resaltar que los resultados de las simulaciones son consistentes debido a que su complejidad algorítmica cuadrática o $\boldsymbol{O(n^2)}$ siempre será la misma bajo cualquier escenario.

Por otro lado, también cabe resaltar que la variación de los tiempos obtenidos en los diferentes escenarios están sujetos al rendimiento del procesador y especificaciones del equipo en el momento de ejecutar la simulación.

CAPÍTULO VI: DISCUSIÓN

A continuación, en esta sección discutiremos y analizaremos los resultados más relevantes expuestos en la sección anterior.

Como se ha podido observar, el algoritmo A estrella es 21 milisegundos más rápido en comparación al algoritmo D estrella. No obstante, también se evidenció que la utilidad del primer algoritmo solo es beneficiosa en escenarios que no presentan obstáculos, lo cual no es beneficioso para un ambiente real de evacuación. Es así que, para implementar un verdadero sistema de evacuación, se debe contemplar un algoritmo de enrutamiento dinámico como el D estrella.

Esto también se evidencia en múltiples investigaciones que compararon una versión inicial de un algoritmo de camino más corto y una versión mejorada que actualiza constantemente las rutas. Como se observa en la tabla 6.1, en la investigación de X. Chen et al. (2019), se realizó una simulación con el algoritmo de búsqueda A estrella y se logró encontrar la ruta más corta en 1153 milisegundos. Por otro lado, al realizar una simulación con una versión mejorada del mismo algoritmo, que toma en cuenta obstáculos sobre la ruta, se obtuvo que el camino más corto es encontrado en 1075 milisegundos, demostrando que el desempeño del algoritmo es superior al emplear esta mejora.

En una segunda investigación los autores Iizuka y Iizuka (2015) también implementaron dos versiones del algoritmo DCOP. Con la implementación de este algoritmo se encontró el camino más corto en 1804 milisegundos. Sin embargo, al implementar el mismo algoritmo con enrutamiento dinámico se obtuvo que el tiempo de ejecución disminuyó a 1397 milisegundos. Así, nuevamente se comprobó que un algoritmo que toma en cuenta la presencia de obstáculos tiene un mejor desempeño y en consecuencia un menor tiempo de ejecución.

Por último, en una tercera investigación donde los autores Gu et al. (2018) implementaron el algoritmo Floyd, se tomaron en cuenta dos versiones diferentes del algoritmo para demostrar que una simulación que soporta rutas dinámicas es más rápida. De esta manera los autores realizaron ejecuciones tomando en cuenta 1102 agentes sobre el establecimiento virtual y se encontró que el método más eficiente fue la versión del algoritmo Floyd que implementa enrutamiento dinámico ya que la versión normal del

algoritmo encontró la ruta más corta en 280000 milisegundos mientras que la versión que contempla obstáculos la encontró en 245000 milisegundos.

En consecuencia, se demuestra que para la presente investigación el uso del algoritmo D estrella es el más eficiente para escenarios donde se presentan obstáculos dinámicos y es de mayor utilidad para la implementación real de la solución propuesta.

Tabla 6.1

Tabla comparativa de resultados obtenidos en las investigaciones de diferentes autores

Articulo de investigación	Número de agentes virtuales	Algoritmo de caminos más cortos	Tiempo de ejecución del algoritmo	Algoritmo dinámico de caminos más cortos	Tiempo de ejecución de algoritmo con enrutamiento dinámico
Artículo 1 [a]	390 [a]	A estrella [a]	1153 ms [a]	A estrella dinámico [a]	1075 ms [a]
Artículo 2 [b]	700 [b]	DCOP [b]	1804 ms [b]	DCOP dinámico [b]	1397 ms [b]
Artículo 3 [c]	1102 [b]	Floyd [b]	280000 ms [b]	Floyd dinámico [b]	245000 ms [b]

[a](X. Chen et al., 2019), [b](Iizuka & Iizuka, 2015), [c](Gu, Wang, & He, 2018).

Por otro lado, al observar los resultados obtenidos en la Tabla 5.1 y la Tabla 5.2, se evidenció que tanto el algoritmo A estrella como el algoritmo D estrella encontraron la ruta más corta en 3 milisegundos. Sin embargo, se encontró que no es recomendable usar el algoritmo A estrella ya que este no es dinámico y, en caso se presenten obstáculos sobre la ruta, el algoritmo tendría que ejecutarse en cada tiempo t lo cual es ineficiente en consumo de tiempo y espacio. Por otro lado, el algoritmo D estrella demostró ser más eficiente ya que contempla la presencia de obstáculos en el espacio en cualquier tiempo t y recalcula la ruta de evacuación automáticamente por lo que siempre se obtendría un tiempo de 3 milisegundos o similar al encontrar diferentes obstáculos en el plano de evacuación. Asimismo, haciendo una analogía con los estudios de los autores mencionados en la tabla 6.1, se puede corroborar esta afirmación debido a que en la simulación de X. Chen et al. (2019) se obtuvo una mejora de 78 milisegundos al usar la versión dinámica de su algoritmo. Por otro lado, en el sistema propuesto por Iizuka y Iizuka (2015), al hacer pruebas con su algoritmo con enrutamiento dinámico se obtuvo una mejora de 407 milisegundos al encontrar la ruta. Por último, en la simulación realizada por Gu et al. (2018), al usar el algoritmo Floyd con enrutamiento dinámico, se

obtuvo una mejora en el desempeño del simulador reduciendo el tiempo de búsqueda en 35000 milisegundos.

En relación a la fase de validación de resultados se pudieron encontrar resultados relevantes. Para el escenario donde se usó diferentes dimensiones del plano, se observó que el tiempo de ejecución siempre se incrementa conforme el tamaño del plano aumente. Esto debido a que el algoritmo emplea mayor tiempo en recorrer más nodos sobre la superficie. En el segundo escenario nuevamente se obtuvo que el tiempo de ejecución es variable en base a la cantidad de obstáculos iniciales sobre la superficie. Sin embargo, esta variación de tiempo es mucho menor comparado al escenario donde se evaluaron diferentes dimensiones del plano. De esta manera se demuestra que en una evacuación real, donde el tamaño del plano siempre es constante, el tiempo de ejecución del simulador no será afectado ya que el incremento de obstáculos no disminuye en gran medida el desempeño del algoritmo.

Por último, en el tercer escenario se analizó la consistencia del algoritmo en diferentes ejecuciones del simulador. Se obtuvo que el tiempo de ejecución experimental fue semejante durante 5 ejecuciones diferentes y la variación entre estas no excedió los 16 milisegundos. En consecuencia, se demostró que la ejecución del algoritmo D estrella es estable y siempre logra encontrar el camino más corto en el menor tiempo posible. Esto es un buen indicador ya que para una implementación real se necesita hacer uso de técnicas que generen resultados consistentes y confiables para siempre obtener las rutas de evacuación más eficientes y seguras.

Por consiguiente, en base a los resultados obtenidos en esta investigación y a la literatura de los autores mencionados se demuestra que un algoritmo con enrutamiento dinámico, como lo es el algoritmo D estrella, sigue siendo más eficiente comparado a otros algoritmos como el de Dijkstra, Floyd o como el algoritmo A estrella propuesto inicialmente en esta investigación. Además, en base a los experimentos realizados, también se evidenció que el algoritmo D estrella continúa demostrando un buen desempeño incluso bajo condiciones donde el tamaño del área a evacuar o el número de obstáculos es distinto y también cuando existen múltiples agentes o personas sobre el establecimiento.

En conclusión, en base a los resultados obtenidos en el capítulo 5, al usar la implementación del algoritmo D estrella en un escenario donde se requiera evacuar el

primer nivel de un centro comercial o un espacio similar, las personas que necesiten evacuar el área pueden encontrar el camino más corto y seguro en solo 22 milisegundos. Esto indica que con este sistema es posible que una persona encuentre de forma precisa el camino más corto desde su punto inicial hasta un punto de extracción seguro en menos de 1 segundo. Finalmente, ante la presencia de obstáculos inesperados en el camino trazado, el simulador es capaz de recalcular y encontrar automáticamente una segunda ruta eficiente en solo 3 milisegundos. Como resultado, esto evidencia que ante la presencia de diferentes obstrucciones que se interpongan en el camino inicial, es posible que una persona encuentre en menos de 0.001 segundos una ruta alternativa hacia la zona segura más cercana.

CAPÍTULO VII: CONCLUSIONES

Como se ha podido observar, los resultados obtenidos en diferentes escenarios y ejecuciones han arrojado resultados que indican que el tiempo de ejecución experimental del algoritmo D estrella en diferentes simulaciones solo toma milisegundos. Estos resultados nos sirven como sustento para comprobar que la propuesta de la investigación cumple con sus objetivos y la implementación de un sistema de evacuación inteligente que implemente algoritmos de búsquedas de camino más cortos es muy útil para brindar una solución y mejorar los problemas que ocurren en las evacuaciones reales de la realidad peruana. Esto debido a que con esta solución los usuarios tendrán el beneficio de obtener la ruta de escape más corta en un tiempo menor comparado al tiempo en que se demora una persona promedio sin usar un sistema de evacuación inteligente. Además, como consecuencia, será posible reducir el número de personas afectadas en un siniestro y brindar la posibilidad de que más personas estén a salvo cuando se necesite evacuar una superficie.

Por otro lado, los resultados de la comparación de algoritmos nos demuestran que el uso del algoritmo A estrella es recomendable para escenarios que no cuentan con la presencia de obstáculos dinámicos en el trayecto, ya que este tiene el menor tiempo de ejecución. Sin embargo, la comparación de estos algoritmos también nos demuestra que el algoritmo D estrella es el más adecuado para escenarios donde se presenten obstáculos dinámicos. Es decir, para una implementación real de este sistema en evacuaciones ante siniestros, este algoritmo es el más adecuado debido a que es capaz de encontrar la ruta de evacuación más eficiente y segura tomando en cuenta obstáculos que se presentan sobre el camino en cualquier momento.

TRABAJOS FUTUROS

Como una siguiente etapa de la investigación, se tiene como objetivo realizar una implementación real de este sistema de evacuación inteligente. Para esto, una propuesta ideal sería implementar el algoritmo planteado junto con lógica de negocio personalizada para cada plano o establecimiento. Este sistema desarrollado se podría desplegar como un servicio en la nube y exponer las funcionalidades y sus resultados mediante un API. Por otro lado, así como se sugiere en Ying et al. (2017), se puede tomar en cuenta añadir una funcionalidad adicional para dar soporte a entornos que incluyen ascensores, escaleras y ambientes con múltiples niveles.

Además, para confirmar su aplicabilidad en entornos reales, también se propone la creación de una aplicación móvil que haga uso de la geolocalización. Así, la aplicación podría ser capaz de renderizar una interfaz gráfica del plano y ubicar al agente en el mapa y enviar esta ubicación al API desarrollado para obtener una ruta de evacuación eficiente en tiempo real. Del mismo modo, es importante considerar el comportamiento de las personas en las evacuaciones ya que muchas podrían no usar el sistema debido a la desesperación, angustia, miedo, heridas o lesiones. Es por ello que también se sugiere implementar un "modo de ayuda" en el que el sistema detecte que no hay interacción por parte del usuario y automáticamente le proponga unirse a un grupo de personas para que estos puedan ayudar al usuario y así evacuar juntos hacia la salida más cercana usando el camino más corto.

Adicionalmente, se encontró que algunas literaturas revisadas consideran implementaciones en espacios de menores dimensiones, es por esto que también se podría considerar el uso de dispositivos IoT como sensores, beacons, etc. para lograr obtener una ubicación más exacta en ambientes cerrados. Por otro lado, es importante tomar en cuenta la latencia del internet dentro del establecimiento, es por ello que también se debería considerar construir un modelo que no necesite conexión a internet y tenga acceso a una red local que ejecute el algoritmo y provea la información necesaria.

Por último, también es importante contemplar escenarios donde se puedan encontrar resultados inesperados o contradictorios, como por ejemplo que el algoritmo A estrella y D estrella brinden el mismo tiempo de ejecución en un caso en específico. Es por esto que se sugiere que en un trabajo futuro se tomen en cuenta variables como el número de agentes que el algoritmo es capaz de manejar en paralelo y el tiempo total considerando el tiempo de

ejecución del algoritmo y el tiempo de desplazamiento del agente. Además, se debe considerar la revisión e implementación del algoritmo D* Lite el cuál es una versión más optimizada de el algoritmo D* ya que usa una estructura de datos diferente y así comparar los diferentes tiempos de ejecución bajo múltiples escenarios.

REFERENCIAS

Chancay, A. J. (2018). Estudio estadístico de la peligrosidad sísmica

Del ecuador mediante la teoría de valores extremos. Recuperado el 2022, de http://www.dspace.espol.edu.ec/xmlui/bitstream/handle/123456789/48894/D-CD110104.pdf?sequence=1&isAllowed=y

Chen, L. W., & Liu, J. X. (2018). Demo: EasyGO - A rapid indoor navigation and evacuation system using smartphones through internet of things technologies. *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, 835–837. https://doi.org/10.1145/3241539.3267721

Chen, X., Li, W., Nie, Y., Chen, R., He, K., Liu, Y., & Zhang, Y. (2019). A stage-wise path planning approach for crowd evacuation in buildings. *ACM International Conference Proceeding Series*, 15–20. https://doi.org/10.1145/3328756.3328757

INDECI (2022). Reporte de Emergencias]. Recuperado el 2022, de https://app.powerbi.com/view?r=eyJrIjoiNTFkOWRhYWQtYmMwMS00OWNmLTg4ZTctNjZjYTc1OTIyN2M0IiwidCI6IjNlZWNkMjZlLTlhNTUtNDg4MC04ODEyLWEzMGZjZGU3OGEyZCJ9&pageName=ReportSectioncd99edcca07a5ff10551&pageName=ReportSection16ae97874d87a478978d

Dineshpathak. (02 de 03 de 2022). *Algorithmic Complexity*. Obtenido de Devopedia: https://devopedia.org/algorithmic-complexity

Espindola, V. H., & Perez, X. (2018). *¿Qué son los SISMOS, dónde ocurren y cómo se miden?* Recuperado el 2022, de https://www.amc.edu.mx/revistaciencia/images/revista/69_3/PDF/QueSonSismos.pdf

Gu, T., Wang, C., & He, G. (2018). A VR-based, hybrid modeling approach to fire evacuation simulation. *Proceedings - VRCAI 2018: 16th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*. https://doi.org/10.1145/3284398.3284409

Gutenschwager, K., Volker, S., Radtke, A., & Zeller, G. (2012). The shortest path: Comparison of different approaches and implementations for the automatic routing of vehicles. *Proceedings - Winter Simulation Conference*, 3312–3323. https://doi.org/10.1109/WSC.2012.6465023

Firecity. (s.f.). Recuperado el June de 2022, de Firecity Peru: https://firecityperu.com

Fournier, J.-C. (2009). Graph Theory and Applications. Iste.

Haghani, M., & Sarvi, M. (2016). Human exit choice in crowded built environments: Investigating underlying behavioural differences between normal egress and emergency evacuations. *Fire Safety Journal*, *85*, 1–9. https://doi.org/10.1016/J.FIRESAF.2016.07.003

Hridi, A. P., Das, D., Anjum, M. M., & Das, T. (2016). Faster Evacuation after Disaster. *Proceedings of the 7th Annual Symposium on Computing for Development - ACM DEV '16*, 1–4. https://doi.org/10.1145/3001913.3006632

Iizuka, Y., & Iizuka, K. (2015). Disaster Evacuation Assistance System Based on Multi-agent Cooperation. *2015 48th Hawaii International Conference on System Sciences*, 173–181. https://doi.org/10.1109/HICSS.2015.30

IORDAN, A. E. (2012). Development of an Interactive Environment Used for Simulation of Shortest Paths Algorithms. *Annals of the Faculty of Engineering Hunedoara - International Journal of Engineering*, *10*(3), 97–102. Retrieved from http://flagship.luc.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=102165580&site=ehost-live

Kocay, W., & Kreher, D. L. (n.d.). *Graphs, algorithms, and optimization*. Retrieved from http://library1.org/_ads/3A31F89895F49C8FDD85E577A7BFAB3B

Koenig, S., & Likhachev, M. (2002). D* Lite [Review of D* Lite]. AAAI-02, 476–483.

Koo, J., Kim, Y. S., Kim, B.-I., & Christensen, K. M. (2013). A comparative study of evacuation strategies for people with disabilities in high-rise building evacuation. *Expert Systems with Applications*, *40*(2), 408–417. https://doi.org/10.1016/j.eswa.2012.07.017

Masudur Rahman Al-Arif, S. M., M Iftekharul Ferdous, A. H., & Hassan Nijami, S. (2012). Comparative Study of Different Path Planning Algorithms: A Water based Rescue System. *International Journal of Computer Applications*, *39*(5), 975–8887. Retrieved from https://pdfs.semanticscholar.org/e59d/c7af0604fced7d124d4f755917725c7892e9.pdf

Michigan Technological University. (s.f.). *Michigan Technological University*. Recuperado el 2022, de Michigan Tech: https://www.mtu.edu/geo/community/seismology/learn/earthquake-measure/magnitude/

Microsoft. (n.d.). System.Diagnostics Namespace. Retrieved May 19, 2019, from https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics?view=netframework-4.8

Muntean, P. (2016). *Mobile Robot Navigation on Partially Known Maps using a Fast A Star Algorithm Version*. Recuperado de http://arxiv.org/abs/1604.08708

Pariona, W. (2019). Evaluación de un sistema de búsqueda de rutas de evacuación eficientes de un establecimiento usando el algoritmo D estrella. *Segundo Congreso Internacional de Ingeniería de Sistemas.* Lima: Universidad de Lima.

Pelechano, N., & Badler, N. (2006). Modeling Crowd and Trained Leader Behavior during Building Evacuation. *IEEE Computer Graphics and Applications*, *26*(6), 80–86. https://doi.org/10.1109/MCG.2006.133

Recibir información u orientación sobre preparación ante emergencias y desastres (s.f). *Servicio - Instituto Nacional de Defensa Civil - Gobierno del Perú.* Recuperado el 14 noviembre de 2022, de https://www.gob.pe/10148-recibir-informacion-u-orientacion-sobre-preparacion-ante-emergencias-y-desastres

Sathyaraj, B. M., Jain, L. C., Finn, A., & Drake, S. (2008). Multiple UAVs path planning algorithms: A comparative study. *Fuzzy Optimization and Decision Making*, *7*(3), 257–267. https://doi.org/10.1007/s10700-008-9035-0

Securitech. (s.f.). *Securitech*. Recuperado el Junio de 2022, de

https://securitechperu.com/web/sistema-saci/audio-evacuacion-y-perifoneo/

Septiana, R., Soesanti, I., & Setiawan, N. A. (2016). Evaluation function effectiveness in Wireless Sensor Network routing using A-star algorithm. *2016 4th International Conference on Cyber and IT Service Management*, 1–5. https://doi.org/10.1109/CITSM.2016.7577519

Sistema de Alerta Sísmica del Perú (SASPe) (s.f). *Orientación - Instituto Geofísico del Perú -Gobierno del Perú.* Recuperado el 10 noviembre de 2022, de https://www.gob.pe/9677-sistema-de-alerta-sismica-del-peru-saspe

Skiena, S. (2008). *The Algorithm Design Manual*. *Stony Brook, NY: Telos Pr*. https://doi.org/10.1007/978-1-84800-070-4

Stentz, A. (1994). *Optimal and Efficient Path Planning for Partially-Known Environments*. Retrieved from http://www.frc.ri.cmu.edu/~axs/doc/icra94.pdf

Thulasiraman, K., & Swamy, M. N. S. (2011). Graphs: Theory and Algorithms. In *Graphs: Theory and Algorithms* (pp. 1–30). https://doi.org/10.1002/9781118033104.ch1

Vokřínek, J., Komenda, A., & Pěchouček, M. (2010). Cooperative agent navigation in partially unknown urban environments. *ACM International Conference Proceeding Series*, 41-48.

Yahja, A., Stentz, A., Singh, S., & Brumitt, B. L. (n.d.). Framed-quadtree path planning for mobile robots operating in sparse environments. Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146). https://doi.org/10.1109/robot.1998.677046

Ying, Z., Zi-min, Z., & Jian, C. (2017). EvacAgent. *Proceedings of the 2017 International Conference on Artificial Intelligence, Automation and Control Technologies - AIACT '17*, 1–7. https://doi.org/10.1145/3080845.3080872

articulo

INFORME DE ORIGINALIDAD

9%	3%	0%	8%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	Submitted to Universidad de Lima Trabajo del estudiante	7%
2	repositorio.ulima.edu.pe Fuente de Internet	1%
3	www.revistareduca.es Fuente de Internet	<1%
4	200.13.202.26 Fuente de Internet	<1%
5	sedici.unlp.edu.ar Fuente de Internet	<1%
6	docplayer.es Fuente de Internet	<1%
7	lajornadaweb.com.ar Fuente de Internet	<1%
8	repositorioacademico.upc.edu.pe Fuente de Internet	<1%
9	sandbox.ijcaonline.org Fuente de Internet	<1%