Memoria de prácticas de Sistemas Inteligentes 2023/2024

Sergio González Velasco y Minerva Jiménez Álvarez

Grupo B

sergio.gonzalez.35@alu.uclm.es

minerva.jimenez@alu.uclm.es

En este trabajo se ha presentado un rescate urbano realizado por un sistema de inteligencia artificial para ayudar a un equipo de rescate, el agente debe de encontrar la ruta mas rápida y segura en un entorno con diferentes obstáculos ya bien sean peligros, sitios bloqueados por donde el agente no puede pasar y posibles peligros fatales, para ello se van a implementar posibles estrategias, en el caso de la primera parte se va a realizar búsqueda no informada y búsqueda heurística, para la segunda parte aprendizaje por refuerzo donde añadimos el factor aleatorio del entorno y penalización al movimiento lo que permite una mayor cercanía a un entorno real.

Tabla de contenido

1.- Práctica 1: Búsqueda en Espacio de Estados

1.1- Introducción

-A lo largo de esta práctica se desea desarrollar una situación que representa un rescate urbano. Se trata de un agente robótico, que debe de recorrer el entorno y encontrar la ruta más rápida y segura para rescatar a las supuestas personas que hay atrapadas. El entorno estará representado con una cuadrícula de un tamaño determinado. Cada casilla de la cuadrícula será de un tipo entre cinco diferentes, que pueden ser el inicio, bloqueado, libre, con personas atrapadas o peligros potenciales. El agente debe de tener estas variables en cuenta a la hora de determinar una ruta de rescate de la mejor manera posible para que sea óptima.

-El objetivo es implementar diferentes algoritmos a un mismo problema tanto de búsqueda no informada (primero en anchura y primero en profundidad) como de búsqueda informada (A* y búsqueda voraz) usando heurísticas para resolverlo de diferentes formas. Además, como trabajo extra se han implementado dos nuevos algoritmos de búsqueda no informada (profundidad limitada y profundidad iterativa).

1.2- Representación del problema

-El problema en sí viene representado en forma de diccionario y mediante la estructura de datos json, en el cual vienen los tipos, en formato (x, y) (coordenadas), anteriormente mencionados. Éstos son:

- Inicio: posición inicial en la que comienza el programa. Determina la posición de salida del agente robótico en coordenadas dentro del tablero que representa el problema, para empezar la ruta de rescate desde ese punto.
- Bloqueados: tipo que representa un obstáculo para el agente. No podrá avanzar o atravesar las casillas del tablero que se encuentren bloqueadas.
- Destinos: casilla que representa un posible destino final, es decir, posición del tablero en la que se encuentra un grupo de personas atrapadas. Éste será uno de los objetivos a los cuales el agente debe de llegar de la forma más rápida y segura posible.
- Peligros: áreas que pueden representar un atraso en el avance del agente si éste decide atravesarlas, que tendrán la penalización de incrementar en gran medida el coste de avance del agente (en un entorno real puede ser, por ejemplo, un derrumbe o un incendio).
- Problema: representa el tamaño del espacio en el que se desarrolla la acción de rescate de personas atrapadas. Al estar el problema integrado en forma de tablero, su tamaño viene dado por:
    - Filas: número de filas del tablero.
    - Columnas: número de columnas del tablero.

-Pasaremos ahora a hacer una descripción de cómo se representan y manejan los datos en nuestro código implementado para esta práctica.

Clase problema:

-En nuestra implementación de la práctica, la clase problema representa toda la información contenida en dicho diccionario json y se ha inicializado con un constructor que recoge de forma paramétrica dicho diccionario para que el cliente introduzca los datos de forma cómoda sin alterar el código. Se inicializa dicho problema en la variable ***ciudad***. A continuación, se inicializan en variables el estado inicial de nuestro agente, destino, peligros y bloqueados. Se ha optado por hacer esto en formato de tupla, ya que el coste computacional de acceder a dicho contenido es 1, en lugar de iterar, además de poder hacer comparaciones entre un posible estado del agente y el estado final o bloqueado de forma eficiente.

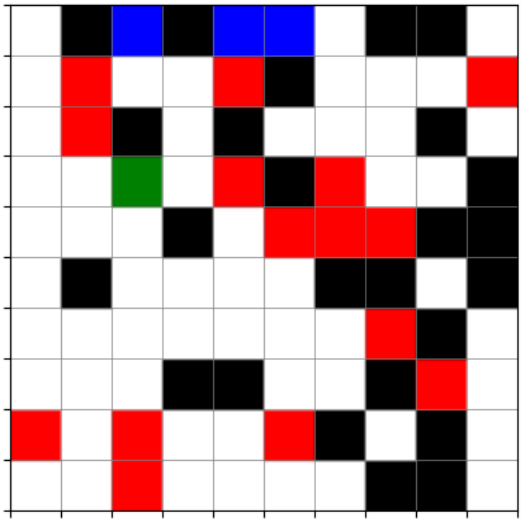
-Posteriormente, en la clase Problema se inicializan diferentes variables más que se usarán para ir almacenando y acumulando las estadísticas de cada rescate efectuado por una misma estrategia de búsqueda para posteriormente hacer una media de todas en un método perteneciente a esta misma clase. Además, se añade otro método auxiliar para resetear las estadísticas (ponerlas todas a cero) para almacenar las de otra ejecución diferente.

-Dentro de esta clase tenemos también varios métodos que nos servirán para realizar ciertas acciones que deberemos usar posteriormente, tales como determinar si una casilla es peligrosa, si está bloqueada, si es válida (es decir, que no se sale del rango del tamaño del tablero y además no está bloqueada) y un método que calcula la heurística de Manhattan, posteriormente usada en la búsqueda informada y que veremos y analizaremos más a fondo posteriormente en esta memoria.

-Por último, tenemos en esta clase los métodos encargados de llamar a los diferentes componentes necesarios para resolver un problema dado por aquel método de búsqueda deseado, mostrando posteriormente las estadísticas globales de la resolución total del problema: búsqueda en anchura, búsqueda en profundidad, búsqueda en profundidad limitada, búsqueda en profundidad iterativa, búsqueda en primero el mejor y búsqueda en A*. Este apartado se explicará más detalladamente posteriormente en el desarrollo de esta memoria.

Clase Estado:

-En la teoría, un estado es la representación de todo el sistema, entorno y la posición del agente en un momento determinado del tiempo. Viene a representar el equivalente a una foto de nuestro problema en diferentes etapas a nivel temporal. Vemos un esquema visual que puede representar un estado inicial de nuestro agente, en el que, para facilitar la comprensión, se han asignado colores diferentes a cada cuadrícula del problema para representar cada uno de los tipos de casillas existentes: negro es una casilla bloqueada, azul es una casilla de persona atrapada, rojo es una casilla de peligro y verde la posición inicial de la que parte nuestro agente nuestro agente. De igual forma, el verde también representa la posición actual en el tiempo del robot, por lo cual en cada movimiento esta casilla verde se va moviendo.



-En nuestra implementación hemos definido dicha clase mediante dos variables, la fila y la columna que representan la posición actual del agente y que son inicializadas en el constructor. Se implementan también los métodos *__eq__* y *__hash__* para poder hacer comparaciones entre diferentes estados y que sean hasheables, es decir, que la transformación en un valor hash de dicho estado coincida con otro, indicando así que un estado es igual a otro, además del método *__repr__* (para la correcta salida de las estadísticas por pantalla), que serán todos ellos necesarios para el correcto funcionamiento de diferentes partes del código que explicaremos más adelante.

Clase Acción:

-En la teoría, una acción representa uno de los posibles movimientos que se pueden hacer para cambiar de un estado a otro nuevo.

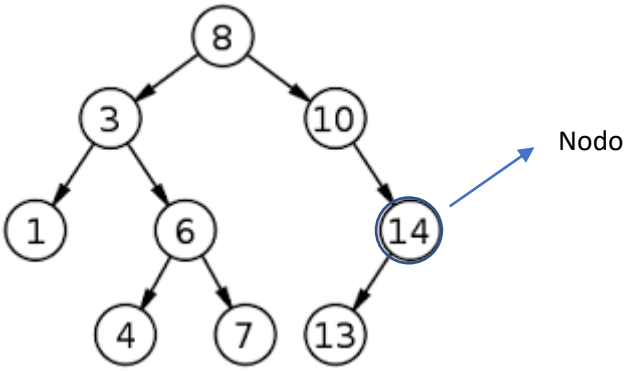
-En nuestra implementación, contiene una variable con el coste de la acción a realizar y otra variable de tipo string que contiene uno de los posibles movimientos a realizar (UP, RIGHT, DOWN o LEFT) aplicado a un estado concreto. El constructor recoge estos dos parámetros y los inicializa, calculando el coste de la acción en un método denominado *calcular_coste*, que permite calcular el coste de una acción. Para ello, se le parametriza el estado concreto, así como la variable *ciudad* que contiene toda la información del problema (las casillas) y, dependiendo de si dicho estado es o no un estado de peligro se le asigna el coste de 5 si lo es, o, en caso contrario, un coste de 1 si no lo es, tal y como se especifica en el guion de la práctica. Esto se calcula mediante una comparación if/else de si dicho estado se encuentra contenido en peligros, usando para ello uno de los métodos ya explicadas de la clase problema; *es_peligro*.

Clase Nodo:

-En la teoría, un nodo, en concreto de un árbol de búsqueda, representa cada uno de los puntos de su ramificación. En nuestra implementación del problema un nodo contiene, además de un estado de la clase Estado, el estado Padre, si es que lo posee, el coste acumulado para llegar a este nodo, la profundidad del nodo y la acción de la clase Acción con la cual se ha llegado a él, si es que no es un nodo inicial. También se le asigna una id, la cual va a ser utilizada para los algoritmos de búsqueda informadas como posteriormente se va a detallar. Concretamente, el coste se calcula sumando el coste calculado de realizar la acción para llegar a este nodo con el

coste del padre del nodo, si lo tiene, si no, se trata del nodo raíz y se considera coste nulo, es decir, cero.

-Al igual que ocurre con la clase Estado, se han definido los métodos *__eq__* y *__hash__*, además del método *__lt__*, pudiendo con este último comparar si un id es superior a otro, lo que permite realizar una comparación para determinar si un nodo es posterior a otro. En este caso, el método *__eq__* utiliza el hash para comparar la igualdad entre nodos.

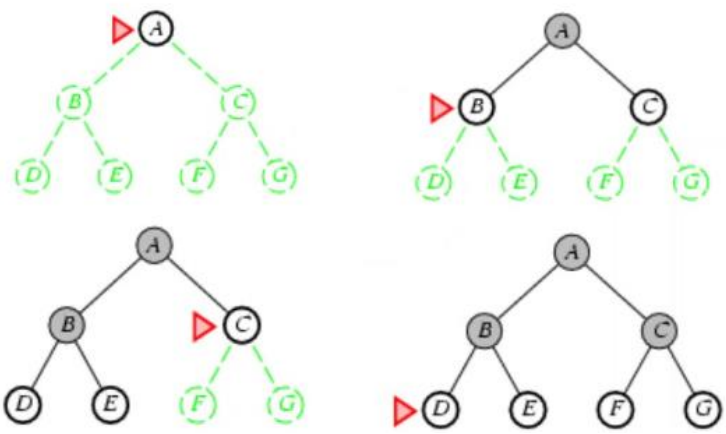


1.3- Estrategias de búsqueda

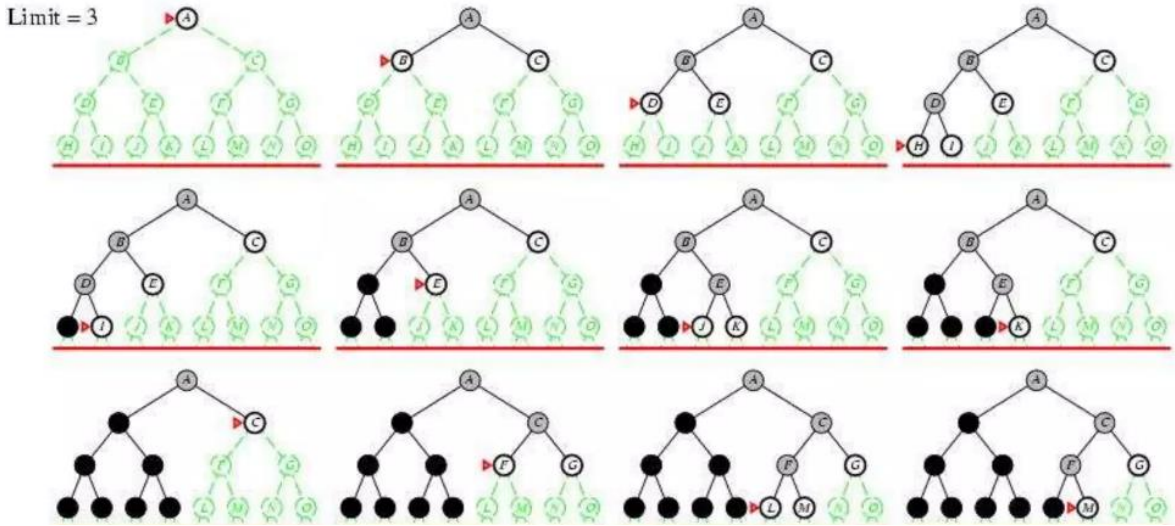
-Una estrategia de búsqueda es la forma en la que el algoritmo recorre un entorno, ya sea con información previa de dicho entorno (búsqueda informada) o sin conocimiento previo (búsqueda no informada) para asegurar la llegada a un estado final determinado. Al ser nuestro problema un árbol de búsquedas, en nuestro caso partimos de un nodo inicial hasta un nodo final.

-Para esta búsqueda se han dividido las estrategias de búsqueda a implementar en dos:

- **Búsqueda no informada**: en esta estrategia el agente desconoce la información procedente del entorno. Para ello realiza cuatro búsquedas, en anchura, en profundidad, en profundidad limitada y en profundidad iterativa.
    - Búsqueda en anchura: el agente realiza una búsqueda y trata de encontrar el camino más corto a un nodo final expandiendo y examinando todos los nodos de la misma profundidad antes de pasar a la siguiente. Esto representa un inconveniente si tenemos un árbol procedente de un problema grande.



- Búsqueda en profundidad: el agente recorre el árbol expandiendo desde la raíz hasta los nodos extremos, si ese nodo extremo no es final, recursivamente vuelve atrás y recorre el siguiente. La diferencia además de esta estrategia de búsqueda es que tenemos un conjunto de nodos visitados, con esto nos aseguramos de que si un nodo ha sido ya visitado no sea necesario expandirlo de nuevo.



- Búsqueda en profundidad limitada: en este caso, esta estrategia es como la búsqueda en profundidad básica, pero con una pequeña modificación; determinamos una profundidad, el algoritmo iniciará la búsqueda de forma normal, sin embargo, en lugar de llegar hasta los nodos extremos del árbol, si llega a la profundidad determinada sin encontrar el nodo final en esa rama vuelve hacia arriba del todo y continúa con el siguiente, repitiendo de nuevo el proceso. Es decir, se expande el árbol en profundidad hasta una profundidad especificada a partir de la cual no se puede seguir expandiendo, por lo que se vuelve atrás para seguir la búsqueda en la siguiente rama.

- Búsqueda en profundidad iterativa: finalmente, la búsqueda en profundidad iterativa es, a grandes rasgos, como una mezcla entre la búsqueda en anchura y en profundidad. En ésta, vamos explorando todos los nodos hasta un cierto límite de profundidad mediante la estrategia de búsqueda en profundidad hasta llegar al nodo final, o bien recorrer todo hasta el límite de profundidad sin éxito, caso en el que se expande en uno el límite de profundidad (de ahí la iteración) y se vuelve a comenzar desde la raíz, repitiendo el proceso de nuevo.

- **Búsqueda informada**: en esta versión de estrategia de búsqueda el agente posee información del entorno. En este caso la heurística será el coste estimado más barato desde el nodo N hasta un nodo objetivo (heurística de Manhattan). Para ello se usan dos implementaciones:
  - Primero el mejor: En búsqueda en primero el mejor sólo se tiene en cuenta la heurística para calcular la siguiente acción a realizar y se expande hasta el nodo objetivo más cercano únicamente con dicho valor.

- A*: Por otro lado, en el caso de A* se tiene en cuenta el coste de la acción acumulada además del de la heurística para expandir el árbol desde el nodo raíz al objetivo más cercano.

Heurística de Manhattan:

-En esta práctica, para las estrategias de búsqueda informada hemos usado la heurística de Manhattan, basada en un concepto llamado distancia de Manhattan. Suele ser una medida de distancia muy usada en optimización y en algoritmos de búsqueda, como es nuestro caso, además, concretamente suele usarse en implementaciones sobre inteligencia artificial.

La distancia de Manhattan consiste en calcular la distancia entre dos puntos de una cuadrícula, sumando los valores absolutos de las diferencias de sus coordenadas. De esta forma, se consigue calcular la menor distancia entre dos casillas, pero mediante movimientos verticales y horizontales evitando así las diagonales por las que no se puede avanzar en una cuadrícula.

La fórmula matemática para calcular esta distancia, que está implementada en el método de la heurística Manhattan en la clase Problema, es la siguiente:

Siendo C1(x1, y1) la primera casilla y C2(x2, y2) la segunda casilla, siendo x las filas e y las columnas que dan dichas coordenadas, entonces la distancia de Manhattan entre las dos casillas se calcula como:

$$D_m = |x1\text{-}x2|+|y1\text{-}y2|$$

En nuestra práctica, esta heurística es aceptable e ideal por el hecho de que podemos guiarnos por las distancias sin confundirnos con distancias diagonales, que no están permitidas. En este caso, no proporciona optimalidad en cuanto al cálculo de la solución a un problema concreto, pero el equilibrio que nos ofrece en cuanto a intensificación y diversificación la propia combinación del algoritmo de búsqueda junto con la heurística nos proporciona una buena solución en un tiempo razonable.

Esto es realmente el hecho importante, ya que en este tipo de problemas no existe un algoritmo eficiente para su resolución, por lo que con este modo de implementarlo estaríamos consiguiendo una buena solución en poco tiempo, superando la utilidad que proporcionaría el obtener la solución óptima, pero en un intervalo de tiempo inabordable o no factible.

-Ahora, en cuanto a la implementación de toda esta estructura de búsquedas en nuestro código, se ha optado por implementar una clase abstracta denominada Search que posteriormente heredan otras clases correspondientes a los algoritmos de las estrategias de búsqueda previamente explicadas.

Clase Search:

-La clase Search posee, primeramente, varios métodos abstractos que heredarán las clases de las estrategias de búsqueda. Éstos son tres, los métodos para insertar y extraer un nodo y para comprobar si una lista de nodos se encuentra vacía. Esto se debe a que la forma de insertar y extraer los nodos de una lista de nodos es la diferencia que hay en el comportamiento a la hora de la implementación en código de los algoritmos de cada estrategia de búsqueda.

-Por otro lado, en esta clase tenemos cierta información almacenada en variables: la ciudad, que se le pasará a la clase, el tiempo de ejecución, dos listas de nodos, los abiertos (frontera) y los cerrados (ya explorados para no volver a expandirlos), siendo la primera de ellas una estructura diferente dependiendo del tipo de búsqueda (informada o no informada), el número de nodos expandidos y el de nodos generados, el coste total, y, finalmente, el estado (posición) del nodo objetivo al que se está tratando de llegar en cada ejecución.

-A continuación, tenemos dos métodos ya implementados dentro de la clase que se usarán posteriormente en otros métodos también implementados de la clase. Primero, tenemos un método para generar los sucesores de un nodo pasado como parámetro. Éste funciona de la siguiente manera: teniendo definidas en una variable (lista de tuplas) las posibles acciones a realizar (arriba, derecha, abajo, izquierda), y una lista vacía de sucesores, para cada posible acción, la aplicamos al nodo dado, sumando los valores correspondientes a su fila y columna, almacenando el resultado en nuevas variables con las que finalmente se compone un nuevo estado.

Hecho esto, usamos una de las funciones ya explicadas de la clase Problema para comprobar si el nuevo estado es válido. En caso de serlo, almacenamos la acción realizada para llegar al nuevo estado de forma decodificada, es decir, en formato String, creamos el nuevo nodo con la información nueva (el estado, la acción, el padre, y su id, sumándole 1 al id de su padre), y finalmente lo añadimos a la lista de sucesores que devolverá la función al explorar todas las posibles acciones para encontrar todos los sucesores, no sin antes aumentar el número de nodos expandidos.

El siguiente método es simplemente el usado para decodificar la acción, que recibe una tupla correspondiente a una de las de la lista de tuplas que define cada posible movimiento y mediante una estructura de if/elif devuelve el String correspondiente a la acción recibida.

-Ahora, tenemos otros dos métodos implementados que corresponden al propio algoritmo de búsqueda. El primero se usará para las búsquedas por anchura, profundidad, primero el mejor y A*, y el segundo para las búsquedas de profundidad limitada y profundidad iterativa. El segundo es muy parecido al primero con una ligera modificación.

Comenzando por el primero, este algoritmo recibe el estado del nodo objetivo y lo guarda en una variable, ya que se debe de usar esta información para calcular la heurística de Manhattan. Posteriormente, en una variable almacenamos el tiempo inicial del algoritmo y después insertamos el nodo de inicio del rescate en la lista de nodos abiertos. Indicamos ahora el rescate a efectuar por pantalla y comenzamos el bucle, que no terminará hasta que la lista de nodos abiertos se encuentre vacía, usando para comprobar esto el método abstracto implementado en aquella clase correspondiente al modo de búsqueda que se esté ejecutando.

Dentro del bucle, primero extraemos un nodo de la lista de nodos abiertos, de nuevo según la implementación de este método abstracto en la clase que corresponda a la ejecución,

comprobamos si este nodo corresponde al objetivo a alcanzar comparando sus estados (es decir, su posición o sus coordenadas), para lo cual se usa el método *__hash__*, en cuyo caso actualizaremos el coste de la clase al coste del nodo (que es el suyo propio más todos los anteriores acumulados desde el inicio), después almacenamos el tiempo final en otra variable, que posteriormente usamos para restarle el inicial calculando así el tiempo de ejecución, que guardamos en otra variable. Ahora llamaríamos al método para recuperar el camino, que explicaremos posteriormente, almacenando su salida en la variable *camino*. Finalmente devolvemos el resultado de la llamada a la función para generar estadísticas, que también explicaremos posteriormente, pasándole el camino como parámetro.

Por otro lado, si el nodo no es el objetivo, primero comprobamos si dicho nodo no se encuentra en la lista de nodos cerrados (para ello usa también la función *__hash__*), en cuyo caso generamos sus sucesores guardándolos en una variable de tipo lista que posteriormente recorremos insertando cada sucesor en la lista de nodos abiertos mediante el método abstracto implementado en la clase correspondiente. Finalmente se añade el nodo ya expandido (recordemos que se suma el nodo expandido dentro de la función para generar sucesores) en la lista de nodos cerrados, para no volver a expandirlo si llegamos a él de nuevo. Si el nodo ya estaba en la lista de cerrados no se vuelve a expandir. Después, se vuelve a empezar el bucle sacando un nuevo nodo de la lista de nodos abiertos según corresponda.

Por otro lado, el segundo método es muy parecido, su comportamiento es el mismo solo que se diferencia en que recibe una variable más como parámetro que corresponde a la profundidad máxima que queremos explorar, y en que dentro del bucle y al no ser el nodo actual el objetivo, además de comprobar si el nodo no se encuentra en la lista de cerrados, se comprueba también que su profundidad sea menor estricto que la profundidad máxima pasada, para no expandir los nodos más allá de dicha profundidad. Además, después comprobamos si la lista de nodos abiertos está vacía, en cuyo caso se mostraría por pantalla que no se ha podido llegar al objetivo en la profundidad establecida, y el método devolvería el valor -1.

-Por último, en esta clase, tenemos los dos métodos mencionados y usados en los algoritmos anteriores. Primero tenemos el método para recuperar el camino al encontrar un nodo objetivo, consistente en un bucle que añade a una lista la acción del nodo objetivo pasado como parámetro en formato String y posteriormente a la variable donde se almacena el nodo se le asigna el padre del actual, volviendo a repetir el proceso hasta que el nodo cuya acción se ha añadido no tenga padre, es decir, que es el nodo inicial donde comenzó la búsqueda. Este método devuelve la lista con el camino seguido para encontrar el nodo objetivo, aunque dicha lista se encuentra al revés.

Finalmente, tenemos el método para generar las estadísticas individuales de un solo rescate, que muestra por pantalla el número de nodos generados, el de nodos expandidos, el tiempo de ejecución, la longitud de la solución (del camino), el coste de este camino, y la solución (la lista del camino mostrada del revés para que vaya desde el nodo inicial hasta el nodo objetivo). Posteriormente, suma todas estas estadísticas individuales a las estadísticas globales de *ciudad* para poder finalmente mostrar desde la clase Problema las estadísticas globales de todos los rescates efectuados.

Clase Anchura:

-Esta clase hereda la clase Search e implementa los tres métodos abstractos. Primero, dentro de su constructor llama al de la clase superior para adquirir sus variables, y después declara la variable donde se almacenarán los nodos abiertos como una de tipo lista.

Ahora pasamos a implementar los métodos abstractos. El primero de ellos es el de insertar un nodo en una lista. Para ello le pasamos el nodo y la lista en la cual lo queremos insertar. En este caso, al ser búsqueda en anchura, el orden en el que debemos explorar los nodos es FIFO, por lo que agregamos los nodos al final de la lista mediante un append, y finalmente aumentamos en uno los nodos generados.

El segundo método es para extraer el siguiente nodo a manejar de una lista que pasamos como parámetro. De nuevo, como necesitamos orden FIFO, lo que hacemos es usar un pop del elemento en la posición 0 (la primera) de la lista. Esta función nos permite devolver el nodo en dicha posición además de eliminarlo de la lista.

Por último, el tercero nos permite determinar si una lista está vacía. En esta práctica, solo necesitamos comprobar esto para la lista de nodos abiertos, que, como en esta estrategia la hemos inicializado en formato lista simple, lo hacemos simplemente comprobando si su función *__len__* devuelve el valor cero, devolviendo así true o false según corresponda.

Clase Profundidad:

-Esta clase se usa tanto para la profundidad como profundidad limitada, puesto que la diferencia la marcamos en el método que inicia la búsqueda y controla la profundidad, esta clase de nuevo hereda de la clase Search, implementando los tres métodos abstractos. En cuanto al constructor, hace exactamente lo mismo que la clase Anchura, llama al de la clase superior y declara la variable para guardar los nodos abiertos como una de tipo lista.

En cuanto a los métodos abstractos, el primero, para insertar un nodo en una lista, lo implementamos añadiendo el nodo en la posición 0 de la lista mediante un insert y posteriormente sumamos uno a los nodos generados. Esto es así para garantizar el orden LIFO que requiere la búsqueda en profundidad.

El segundo, para extraer el siguiente nodo, es exactamente igual que en la clase anchura, sacamos el primer elemento de la lista, que devolvemos y eliminamos de la lista. La diferencia es que ahora este elemento será el último que se añadió a la lista, no el primero.

Finalmente, el tercero, de nuevo se encuentra implementado de la misma forma que la clase Anchura, puesto que de nuevo nuestra variable en la que guardamos los nodos abiertos es de tipo lista, sacamos su longitud de la misma manera.

Clase ProfundidadIterativa:

-Esta clase, heredando también de la clase Search, se usa para la estrategia de búsqueda por profundidad iterativa. En el constructor, de nuevo tenemos lo mismo que en las anteriores, solo que esta vez, se añade algo más. Se inicializa la variable de profundidad máxima a uno siendo el punto de partida. La variable que guarda los nodos abiertos sigue de nuevo siendo de tipo lista.

El resto de las implementaciones de los métodos abstractos de la clase son exactamente iguales que en la clase anterior, la clase de Profundidad. Pues en este caso, la única diferencia con ella viene por parte de controlar la profundidad de exploración, que, como ya hemos visto, viene controlada principalmente por el método correspondiente ya implementado en la clase Search y por los métodos que llaman y usan esta estrategia de búsqueda en la clase inicial, la clase Problema, que serán explicados posteriormente.

Clase PrimeroElMejor:

-Esta clase hereda nuevamente de la clase Search y, al igual que las anteriores, llama al constructor de la clase superior. En este caso, como se trata de búsqueda informada, implementamos la variable para guardar los nodos abiertos como una Priority Queue, ya que de esta forma se guardarán los nodos junto a un valor dado por la heurística impuesta para poder sacar siempre el mejor nodo a escoger.

El primer método abstracto, para insertar un nodo a, en este caso, la Priority Queue, primero declara una variable para guardar el valor correspondiente que devuelve el método para calcular el valor correspondiente a la heurística, pasándole el nodo actual y el nodo objetivo actual. Después, se añade la tupla (valor, nodo) a la Priority Queue, actuando este valor como la prioridad de esta estructura de datos. Por último, no nos olvidemos de sumar uno al número de nodos generados.

En cuanto al segundo, para extraer un nodo de la Priority Queue, lo que hacemos es guardar en una variable el resultado de hacer un get sobre la Priority Queue, que nos devuelve la tupla con la mayor prioridad que, en este caso, sería aquella cuyo valor de la heurística fuera el más bajo, y además elimina dicha tupla de la Priority Queue. Entonces devolvemos el elemento uno de la tupla, que correspondería al nodo extraído.

En tercer y último lugar, tenemos el método para comprobar si la Priority Queue se encuentra vacía, que esta vez, al no tratarse esta estructura de una lista, cambia. Usamos ahora el método qsize, devolviendo true o false según el resultado de comparar el valor devuelto por qsize con cero.

Clase AEstrella:

-Esta es la última clase que hereda de la clase Search. En cuanto al constructor, actúa exactamente de la misma forma que la clase anterior, la clase PrimeroElMejor.

El cambio en esta clase se encuentra aquí, en la implementación del primer método abstracto, el que se utiliza para insertar un nodo a la Priority Queue, aunque la diferencia es también mínima en cuanto a código. Lo único que cambia es que, a la hora de insertar la tupla, en lugar de insertar (valor, nodo) insertamos (coste + valor, nodo), siendo aquí el coste el total, es decir, el acumulado de toda la rama recorrida o movimientos efectuados hasta llegar al nodo actual que estamos insertando a la Priority Queue.

El resto de los métodos abstractos, tanto el de extraer nodo como el de comprobar si la Priority Queue está vacía son exactamente iguales y se comportan de la misma forma que en la clase anterior.

1.4- Evaluación experimental

-Por último, en nuestro código tenemos el main del programa, que se encarga de recibir el archivo json que contiene el problema que queremos resolver y pasárselo a una variable de tipo Problema, que lo construye. A continuación, se añaden a una lista todas las coordenadas objetivo en forma de tupla del problema mediante un bucle, que se interpretan como un objeto de tipo Estado y como un objeto de tipo Nodo también. Finalmente tenemos una serie de comentarios con las llamadas a las diferentes formas de búsqueda que podemos ejecutar, que envían la lista de objetivos previamente comentada. Para usar cada una, seleccionar el json deseado y descomentar la línea correspondiente a la estrategia de búsqueda deseada, teniendo en cuenta que para la búsqueda en profundidad limitada hay que seleccionar también el límite deseado.

Ahora, como comentamos anteriormente, pasaremos a explicar cómo funcionan los métodos de la clase Problema que se encargan de ejecutar el algoritmo especificado para el problema, que son los métodos a los que llama el main.

En general, casi todos (excepto el de búsqueda en profundidad iterativa) funcionan de la misma manera y con la misma estructura, por lo que nos centraremos principalmente en estas dos implementaciones diferentes.

Método genérico (para anchura, profundidad, profundidad limitada, primero el mejor y A*):

-Estos métodos tienen una implementación sencilla. Primero se indica por pantalla el algoritmo que se está ejecutando, para posteriormente iterar en un bucle la ejecución del algoritmo correspondiente para cada uno de los objetivos de la lista recibida. Esto se hace llamando al método de iniciar búsqueda correspondiente desde la clase del tipo de búsqueda que se está ejecutando, por lo que usa dicho método que ya se encuentra implementado en la clase Search con los métodos abstractos implementados de la propia clase que lo ejecuta. Cuando finaliza el algoritmo para todos los objetivos muestra las estadísticas globales de la ejecución.

Método para profundidad iterativa:

-En el método para ejecutar la búsqueda en profundidad iterativa. Dentro del bucle lo primero que hacemos es establecer en una variable la profundidad máxima en uno y, de nuevo, como anteriormente, iteramos la ejecución de esta búsqueda por cada objetivo, guardando en una variable el resultado de su ejecución. Sin embargo, recordemos que en el algoritmo para la búsqueda limitada había una ocasión en la que se devolvía el valor -1; que no se halle el objetivo en el límite de profundidad establecido.

Sabiendo esto, al terminar la primera ejecución comprobamos si el valor devuelto es -1, en cuyo caso y mientras la solución siga siendo este valor, aumentamos en uno la profundidad máxima y volvemos a ejecutarlo con este nuevo límite para el mismo objetivo. Al cambiar de objetivo la profundidad máxima vuelve a fijarse en uno para empezar la búsqueda. Finalmente, y de igual forma que en el anterior, se muestran las estadísticas globales de ejecución al terminar con todos los objetivos del problema.

Análisis de rendimiento de los diferentes algoritmos:

-Ahora nos centraremos en ejecutar tres instancias diferentes de un problema de diferentes tamaños cada una y comparar el rendimiento en cuanto a tiempo de ejecución, nodos generados, nodos expandidos y coste total de cada una de las estrategias de búsqueda para cada instancia. Para ello organizaremos toda esta información en tres tablas diferentes, una por cada instancia. Las instancias que usaremos para ello son:

- instance-20-20-33-8-33-2023.json
- instance-100-100-1557-8-1557-2023.json
- instance-1000-1000-110592-15-110592-2023.json

-Comenzamos por la primera de ellas, con un entorno de tamaño 20 x 20 casillas y ocho objetivos a rescatar, veremos las estadísticas por cada objetivo y las estadísticas globales, es decir, la media de todos los objetivos por cada estrategia de búsqueda. La profundidad limitada está configurada a profundidad máxima de 145, mínimo para llegar a todos los objetivos.

ESTRATEGIA	OBJETIVO	T. EJECUCIÓN	GENERADOS	EXPANDIDOS	COSTE
ANCHURA	(14, 4)	0.0002128	52	14	3
PROFUNDIDAD	(14, 4)	0.0003401	151	45	45
PROF LIMITADA	(14, 4)	0.0004389	151	45	45
PROF ITERATIVA	(14, 4)	0.0000871	19	5	3
PRIMERO MEJOR	(14, 4)	0.0001586	13	3	3
A*	(14, 4)	0.0002539	21	5	3
ANCHURA	(10, 8)	0.0011713	340	98	9
PROFUNDIDAD	(10, 8)	0.0027235	1182	338	221
PROF LIMITADA	(10, 8)	0.0014963	636	183	181
PROF ITERATIVA	(10, 8)	0.0010708	270	80	19
PRIMERO MEJOR	(10, 8)	0.0001602	31	9	9
A*	(10, 8)	0.0005617	80	22	9
ANCHURA	(1, 4)	0.0023367	844	238	24
PROFUNDIDAD	(1, 4)	0.0020019	971	275	314
PROF LIMITADA	(1, 4)	0.0025346	989	282	176
PROF ITERATIVA	(1, 4)	0.0019725	840	240	86
PRIMERO MEJOR	(1, 4)	0.0003311	61	16	24
A*	(1, 4)	0.0012054	170	47	18
ANCHURA	(14, 16)	0.0031945	646	182	17
PROFUNDIDAD	(14, 16)	0.0010356	538	154	173
PROF LIMITADA	(14, 16)	0.0018902	538	154	173
PROF ITERATIVA	(14, 16)	0.0007167	297	86	33
PRIMERO MEJOR	(14, 16)	0.0004398	52	13	17
A*	(14, 16)	0.0005731	127	32	13

ANCHURA	(13, 19)	0.0033390	938	265	25
PROFUNDIDAD	(13, 19)	0.0011483	599	171	193
PROF LIMITADA	(13, 19)	0.0011876	567	162	181
PROF ITERATIVA	(13, 19)	0.0007739	339	97	37
PRIMERO MEJOR	(13, 19)	0.0002657	68	18	25
A*	(13, 19)	0.0010637	199	52	17
ANCHURA	(14, 19)	0.0031701	874	247	20
PROFUNDIDAD	(14, 19)	0.0010619	556	159	178
PROF LIMITADA	(14, 19)	0.0013955	556	159	178
PROF ITERATIVA	(14, 19)	0.0007601	334	95	36
PRIMERO MEJOR	(14, 19)	0.0002165	64	16	20
A*	(14, 19)	0.0007304	156	40	16
ANCHURA	(8, 6)	0.0016289	465	133	11
PROFUNDIDAD	(8, 6)	0.0014956	737	211	233
PROF LIMITADA	(8, 6)	0.0017534	739	211	133
PROF ITERATIVA	(8, 6)	0.0008257	358	105	31
PRIMERO MEJOR	(8, 6)	0.0002112	41	12	11
A*	(8, 6)	0.0005440	96	26	11
ANCHURA	(9, 4)	0.0007607	298	86	12
PROFUNDIDAD	(9, 4)	0.0008071	380	111	118
PROF LIMITADA	(9, 4)	0.0008278	380	111	118
PROF ITERATIVA	(9, 4)	0.0004389	188	56	18
PRIMERO MEJOR	(9, 4)	0.0001129	30	8	12
A*	(9, 4)	0.0005956	102	29	10
ANCHURA	GLOBALES	0.0019767	557.125	157.875	15.125
PROFUNDIDAD	GLOBALES	0.0013267	639.25	183	184.375
PROF LIMITADA	GLOBALES	0.0014405	569.5	163.375	148.125
PROF ITERATIVA	GLOBALES	0.0008317	330.625	95.5	32.875
PRIMERO MEJOR	GLOBALES	0.0002370	45	11.875	15.125
A*	GLOBALES	0.0006910	118.625	31.625	12.125

-Seguiremos con la información recogida sobre la ejecución de la segunda instancia mediante todas las estrategias de búsqueda. Tiene un entorno de 100 x 100 casillas y, de nuevo hay 8 objetivos a rescatar en él, pero esta vez centrémonos en los 4 primeros objetivos y en las estadísticas globales de cada variable a comparar. La profundidad limitada está configurada a profundidad máxima de 4300, mínimo para llegar a todos los objetivos.

ESTRATEGIA	OBJETIVO	T. EJECUCIÓN	GENERADOS	EXPANDIDOS	COSTE
ANCHURA	(42, 13)	0.0701979	23810	7078	174
PROFUNDIDAD	(42, 13)	0.0538670	12317	3640	5422
PROF LIMITADA	(42, 13)	0.0429590	12317	3640	5422
PROF ITERATIVA	(42, 13)	0.0304854	10765	3216	324
PRIMERO MEJOR	(42, 13)	0.0015385	367	107	198
A*	(42, 13)	0.0570871	5450	1586	106
ANCHURA	(14, 78)	0.0385539	15228	4498	115
PROFUNDIDAD	(14, 78)	0.1079642	23164	6874	10289
PROF LIMITADA	(14, 78)	0.0760407	21079	6234	7447
PROF ITERATIVA	(14, 78)	0.0566560	19723	5859	1345
PRIMERO MEJOR	(14, 78)	0.0009593	235	67	115
A*	(14, 78)	0.0186965	2433	706	85
ANCHURA	(92, 22)	0.0411631	15939	4712	137
PROFUNDIDAD	(92, 22)	0.0029344	1384	427	647
PROF LIMITADA	(92, 22)	0.0029339	1384	427	647
PROF ITERATIVA	(92, 22)	0.0023205	1024	311	147
PRIMERO MEJOR	(92, 22)	0.0011562	296	84	153
A*	(92, 22)	0.0208562	2677	783	81
ANCHURA	(87, 89)	0.0092061	3906	1141	55
PROFUNDIDAD	(87, 89)	0.0119627	4852	1462	2223
PROF LIMITADA	(87, 89)	0.0116321	4852	1462	2223
PROF ITERATIVA	(87, 89)	0.0055253	2039	599	57
PRIMERO MEJOR	(87, 89)	0.0004151	98	29	65
A*	(87, 89)	0.0040690	639	179	33
ANCHURA	GLOBALES	0.0404246	14652.625	4334.875	113.125
PROFUNDIDAD	GLOBALES	0.0494964	11230.875	3338.375	4452.125
PROF LIMITADA	GLOBALES	0.0390214	10504.875	3117.875	3889.875
PROF ITERATIVA	GLOBALES	0.0245461	8828.0	2626.25	504.125
PRIMERO MEJOR	GLOBALES	0.0011115	250	72.75	122.125
A*	GLOBALES	0.0226032	2679.75	784.25	75.875

-Pasamos a la última tabla, que recoge la información de, al igual que la anterior, la ejecución los primeros 4 objetivos y las estadísticas globales, esta vez de la tercera instancia, con un entorno de tamaño 350 x 700 casillas. La profundidad limitada está configurada a profundidad máxima de 1300, mínimo para llegar a todos los objetivos.

ESTRATEGIA	OBJETIVO	T. EJECUCIÓN	GENERADOS	EXPANDIDOS	COSTE
ANCHURA	(58, 82)	1.8851422	581166	157671	558
PROFUNDIDAD	(58, 82)	0.0449933	13967	3785	4850
PROF LIMITADA	(58, 82)	0.7302412	198083	53776	1794
PROF ITERATIVA	(58, 82)	0.0216871	7895	2134	474
PRIMERO MEJOR	(58, 82)	0.0087543	1772	486	578
A*	(58, 82)	0.4311293	36773	9967	442
ANCHURA	(305, 309)	2.4996090	599413	162651	566
PROFUNDIDAD	(305, 309)	3.2431642	179487	48726	60926
PROF LIMITADA	(305, 309)	1.1711312	288868	78493	1692
PROF ITERATIVA	(305, 309)	0.9623598	293445	79580	894
PRIMERO MEJOR	(305, 309)	0.0081577	1704	465	576
A*	(305, 309)	0.8662951	107398	28897	438
ANCHURA	(173, 570)	0.6524724	214732	58264	289
PROFUNDIDAD	(173, 570)	41.9594541	596620	161347	203747
PROF LIMITADA	(173, 570)	1.9550584	507850	137911	1299
PROF ITERATIVA	(173, 570)	0.3324650	115103	31227	489
PRIMERO MEJOR	(173, 570)	0.0032664	829	222	309
A*	(173, 570)	0.1871994	27743	7465	217
ANCHURA	(141, 85)	2.6144986	682062	185064	646
PROFUNDIDAD	(141, 85)	14.8214923	390835	105866	133354
PROF LIMITADA	(141, 85)	0.4480424	111027	30219	1774
PROF ITERATIVA	(141, 85)	0.0556733	13710	3734	730
PRIMERO MEJOR	(141, 85)	0.0082502	2090	574	726
A*	(141, 85)	0.5462919	64934	17490	498
ANCHURA	GLOBALES	1.5913315	438543.53	118995.27	453.67
PROFUNDIDAD	GLOBALES	22.3345531	395997.73	107167.87	135104.33
PROF LIMITADA	GLOBALES	1.2583198	307971.27	83656.87	1659.13
PROF ITERATIVA	GLOBALES	0.2786646	82227.33	22296.07	609.4
PRIMERO MEJOR	GLOBALES	0.0057630	1407.6	388.4	490.87
A*	GLOBALES	0.5194619	56630	15274.13	350.2

-Ahora, teniendo todos los datos organizados y expuestos en las tablas, pasaremos a hacer un análisis comparativo de cada estrategia de búsqueda basándonos en la información que podemos sacar de las tablas. Primero, sacaremos de forma individual las propiedades que podemos sacar de cada estrategia dados los datos expuestos:

Estrategia de búsqueda por Anchura:

- En general, ofrece un rendimiento moderado en cuanto a tiempo de ejecución, tanto en problemas de tamaño pequeño como en los de tamaño mediano y grande.
- En cuanto a nodos generados, revisando los objetivos individualmente, el rendimiento (menos nodos generados) suele depender del caso específico, es decir, si el objetivo se encuentra en profundidades mayores será peor que si se encuentra en profundidades menores. Según las instancias ejecutadas, parece ser que el rendimiento en esta variable empeora conforme aumenta el tamaño del problema, resultando en un rendimiento moderado en problemas pequeños y algo mayor en problemas de mayor tamaño.
- Por otro lado, fijándonos en los nodos expandidos, vemos que ocurre algo parecido, disminuyendo el rendimiento conforme aumenta el tamaño de problema. En general, un rendimiento moderado o incluso mediocre.
- Por último, en cuanto a coste de la solución hallada, en general esta estrategia de búsqueda ofrece costes muy bajos para problemas de cualquier tamaño. Este aspecto es su mayor punto a favor.

Estrategia de búsqueda por Profundidad:

- En relación al tiempo de ejecución, esta estrategia ofrece un rendimiento moderado en problemas de tamaño pequeño, dependiendo también de la fila en la que se encuentre el objetivo, pero a mayor tamaño de problema el tiempo de ejecución se dispara, haciéndolo considerablemente ineficiente.
- Respecto a los nodos generados, ofrece un rendimiento moderado, incluso mediocre, dependiendo también de la fila del objetivo, lo cual hace decaer a esta estrategia.
- Fijándonos en los nodos expandidos, de nuevo es muy dependiente de la fila en la que se encuentre el objetivo, aunque, en general, presenta un rendimiento moderado de esta variable.
- Finalmente, los costes que presenta esta estrategia son generalmente pésimos independientemente del tamaño del problema, lo cual no la hace atractiva para problemas que no sean muy específicos para ella.

Estrategia de búsqueda por Profundidad Limitada:

- Primeramente, en cuanto al tiempo de ejecución presenta una eficiencia moderada, casi eficiente, consistentemente sin importar el tamaño del problema.
- Siguiendo con los nodos generados, sin duda esta variante presenta mejoras sobre su estrategia base, aunque cabe destacar que depende de la profundidad límite establecida, que dependerá de aquel objetivo cuya profundidad sea la mayor.

- Con respecto a los nodos expandidos pasa lo mismo que en el punto anterior, sin duda se mejoran las prestaciones con respecto a la estrategia base, consiguiendo una eficiencia moderada.
- Por último, en términos de coste, como es obvio visto los anteriores puntos, éste también se mejora, aunque aun así los costes con este tipo de estrategia siguen siendo poco eficientes, poco interesantes.

Estrategia de búsqueda por Profundidad Iterativa:

- Esta estrategia, en general, ofrece una mejora bastante consistente con respecto a su otra variante y su caso base, ofreciendo unos tiempos de ejecución bastante eficientes sin importar el tamaño del problema. También hay que tener en cuenta que este tiempo es por iteración, de las cuales puede haber muchas, dependiendo de la profundidad del objetivo, por lo tanto, también del tamaño del problema. Con lo cual cabe analizar si su uso nos es beneficioso con respecto a las otras dos variantes.
- Por otro lado, los nodos generados son también mucho menores, ofreciendo un mayor rendimiento en este aspecto, de forma moderada.
- En cuanto a los nodos expandidos, pasa lo mismo también, se mejora bastante, ofreciendo un rendimiento moderado en general.
- En términos de coste, la mejora es abismal, ofreciendo costes mucho menores que las otras dos variantes haciendo de ésta la más interesante de las tres en casi todos los aspectos. Aunque, en general, ofrece un coste moderado con relación al problema a resolver.

Estrategia de búsqueda por Primero el Mejor:

- Comenzando con el tiempo de ejecución, esta estrategia presenta una eficiencia significativamente mayor que todos los anteriores, siendo una eficiencia alta y consistente con respecto al tamaño del problema, haciéndola muy interesante.
- Seguimos con los nodos generados, que sin duda presentan una eficiencia sumamente alta, siendo acorde con respecto al tiempo de ejecución.
- Con respecto a los nodos expandidos, de nuevo en la misma línea que antes, la eficiencia es altísima, que junto con el punto anterior hacen a esta estrategia muy atractiva también en cuanto a rendimiento espacial.
- Finalmente, el coste de las soluciones que proporciona esta estrategia es también muy eficiente, siendo costes muy bajos con respecto al resto de algoritmos. En general, es un algoritmo muy interesante en todos los aspectos, aplicable a problemas de cualquier tamaño.

Estrategia de búsqueda por A*:

- Llegando a la última de las estrategias, ésta, en cuanto a tiempo de ejecución presenta una eficiencia alta y consistente sin importar el tamaño del problema.
- En los nodos generados, podemos observar una eficiencia bastante alta para problemas de tamaño pequeño, aunque menor con respecto a la estrategia anterior, sin embargo,

el número se dispara para problemas de tamaño mayor, dando un duro golpe a su eficiencia, que baja en picado, aunque sigue siendo una eficiencia bastante moderada.

- Siguiendo con los nodos expandidos, acorde a los generados, su rendimiento es también bastante alto en problemas pequeños, pero de nuevo se disparan enormemente a mayor tamaño de problema, bajando su eficiencia que es entonces también bastante moderada.

1.5- Conclusiones

-Como conclusiones de todo lo expuesto anteriormente podemos decir:

- Para problemas pequeños, la búsqueda en anchura es relativamente adecuada para problemas pequeños debido a su eficiencia moderada con respecto a coste, pero su rendimiento cae en picado conforme aumenta el tamaño del problema, además de que la eficiencia espacial no es muy alta.
- La estrategia de búsqueda en profundidad, salvo en problemas específicos quizá, resulta ser a menudo ineficiente debido al alto coste y tiempo de ejecución que proporciona, siendo mayores aun conforme aumenta el tamaño del problema.
- La profundidad limitada, aunque presenta una mejora con respecto a la normal, depende mucho de un factor sobre el cual en problemas reales no siempre tenemos control como es la profundidad máxima del objetivo, con lo cual sigue siendo ineficiente en general.
- La profundidad iterativa, de forma general está bastante balanceada con respecto a las otras dos, haciéndola interesante para problemas de diferentes tamaños al combinar lo mejor de la estrategia de búsqueda por anchura con lo mejor de la búsqueda en profundidad.
- La estrategia de búsqueda por primero el mejor presenta un excelente rendimiento en todos los aspectos de forma bastante consistente, no perdiendo tanta eficiencia como el resto conforme aumenta el tamaño del problema.
- La estrategia de búsqueda por A* presenta también un buen balance en todos los aspectos, incluso mejor que la profundidad iterativa, aunque su eficiencia con respecto a la anterior es significativamente peor ya que, al tener en cuenta el coste sus decisiones pueden resultar mucho más costosas para evitar una casilla de peligro que el propio coste de atravesarla sin más. Aun así, puede resultar interesante también para diferentes problemas.
- Cabe destacar también que, sobre todo en cuanto a coste, las estrategias basadas en la búsqueda en profundidad los las que peores costes ofrecen, lo cual, además de su baja eficiencia espacial, los hace la peor estrategia de búsqueda no informada, ya que la de anchura, para compensar la ineficiencia espacial al menos ofrece costes mucho mejores.
- Finalmente, podemos concluir con la significativa mejora en la resolución de problemas con estrategias de búsqueda informada sobre las no informadas, siendo en general la de primero el mejor la más ideal para este tipo de problemas al ofrecer las mejores prestaciones en todos los aspectos.

2.- Práctica 2: Aprendizaje por Refuerzo

2.1- Introducción

-De forma similar a la primera práctica, nos encontramos en el mismo dominio del problema, que no es otra cosa que un rescate urbano. Tenemos a un agente que desea encontrar la ruta más rápida y segura para rescatar a supuestas personas atrapadas. De forma idéntica, el entorno está representado por una cuadrícula, con la diferencia de que pueden existir riesgos fatales para el agente con recompensas negativas bastante mayores y, de forma equivalente, tenemos a las personas a rescatar con recompensas positivas. Además, nos encontramos con un entorno no determinista, por lo que es probable que las acciones que desea realizar el agente no se produzcan dónde quiere o debe de hacerlas.

El objetivo, al igual que en la primera práctica, es implementar las estrategias de aprendizaje por refuerzo, en este caso mediante el algoritmo Q-Learning y la parte no continua, que implementa el algoritmo de iteración de valores con la idea de coger la política que contempla el algoritmo Q-Learning tras su ejecución, valorarla y mejorarla.

La estrategia que se ha deseado realizar es definir en varias clases, las dos principales, el Entorno y el Agente, en las cuales se va a ir desarrollando el cómo interactúan el uno con el otro y sus roles.

2.2- Representación del problema

El problema viene representado en forma de diccionario, de igual forma que en la anterior práctica, en estructura json, pero esta vez con información adicional que nos permite el desarrollo de un entorno de aprendizaje por refuerzo.

- Inicio: posición inicial en la que comienza el programa. Determina la posición de salida del agente robótico en coordenadas dentro del tablero que representa el problema, para empezar la ruta de rescate desde ese punto.
- Bloqueados: tipo que representa un obstáculo para el agente. No podrá avanzar o atravesar las casillas del tablero que se encuentren bloqueadas.
- Atrapados: casilla que representa un posible destino final, es decir, posición del tablero en la que se encuentra un grupo de personas atrapadas. Éste será uno de los objetivos a los cuales el agente debe de llegar de la forma más rápida y segura posible además de ser un estado de recompensa positiva para el agente.
- Peligros: áreas que pueden representar un atraso en el avance del agente si éste decide atravesarlas, que tendrán la penalización de incrementar en gran medida el coste de avance del agente (en un entorno real puede ser, por ejemplo, un derrumbe o un incendio).
- Peligros fatales: otra casilla de un posible destino final, con la diferencia de que este destino supone una recompensa negativa además de que hace terminar la iteración del algoritmo en un determinado episodio. Supone, a grandes rasgos, que el agente ya no puede continuar.

- Ciudad: representa el tamaño del espacio en el que se desarrolla la acción de rescate de personas atrapadas. Al estar el problema integrado en forma de tablero, su tamaño viene dado por:
    - Filas: número de filas del tablero.
    - Columnas: número de columnas del tablero.
- Estocasticidad: representa el no determinismo del entorno. Con ello reflejamos la aleatoriedad del entorno y que el agente, aunque desee ir a una determinada dirección, el entorno aleatorio le hace ir por otra dirección.
- Penalización del entorno: ya que la función de un agente es salvar a las posibles personas atrapadas lo más eficazmente posible, se penaliza el movimiento en forma de una pequeña recompensa negativa. A través de iteraciones esto nos es útil para determinar la convergencia por lo que, para el agente, cada movimiento cuenta.

Clase Entorno

-La información viene contenida en un formato json por lo que tras cargar la información del diccionario en la variable *ciudad* y mediante el constructor se almacena en forma de parámetros junto con parámetros adicionales útiles para otros métodos que se irán desarrollando. Al igual que en la anterior práctica, todo de forma paramétrica y, de igual forma, se ha optado por un formato de tuplas, ya que nos asegura el coste computacional de 1 lo que implica una mayor eficiencia a la hora del acceso de datos en entornos grandes.

Dentro de la clase entorno disponemos de varios métodos auxiliares que nos sirven para determinar si ciertas acciones posibles son válidas o no en diferentes aspectos. Para ello se usan los métodos *es_bloqueado*, que determina si un estado concreto está bloqueado y por ende el agente no puede continuar, el método *es_valido*, que nos permite conocer si un futuro estado está fuera de los límites del entorno, otro método para conocer si un estado es un posible destino con el método *es_destino* y un método que nos permite conocer si una posible acción lleva a un estado válido con *es_accion_valida*.

-Los métodos que interaccionan con los agentes se han definido mediante la estructura siguiente, ya que esta interacción no le corresponde al agente, él sólo desea saber en qué estado está y moverse hasta llegar a un posible destino, los cuales cada uno es un método:

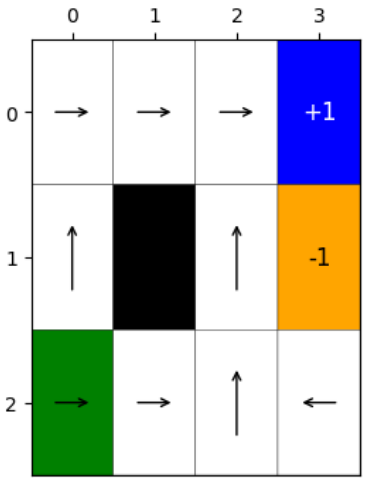
- Mover al agente → Aplicar una acción → Aplicar estocasticidad del entorno → Obtener recompensa
    - Método *mover_agente*: se parametriza la intención del agente o no de explorar en formato booleano, en cuyo caso se elige una acción al azar y se procede a aplicarla. Si no es así, se aplica la acción deseada por el agente en formato de parámetro junto con el estado actual del que se parte. Cabe destacar que los posibles movimientos vienen dados en formato [0,1,2,3] siendo equivalentes a UP, RIGHT, DOWN, LEFT respectivamente.
    - Método *aplicar_accion*: en función de una acción y un estado se transita hacia un nuevo estado, pero, al ser un entorno aleatorio, generamos un número entre cero y uno y, si es superior al valor de la estocasticidad del entorno, llamamos al método encargado de aplicarlo. En caso contrario, aplicamos la acción

generando el nuevo estado y llamando al método de obtener dicha recompensa para retornarlos como variables.

- Método *aplicar_estocasticidad*: tras ser llamado por el método anterior, al igual que en los apuntes, se ha elegido que la aleatoriedad del entorno mueva al agente en las perpendiculares. Si, por ejemplo, tenemos una estocasticidad de un 80%, el agente puede irse por las perpendiculares, un 10% para cada una respectivamente. Lo controlamos restando uno a la estocasticidad para obtener el restante y dividirlo entre dos, tras generar un número entre cero y el restante, se valora si se va a una de las perpendiculares o a la otra en función de si está en la primera mitad del restante o en la segunda respecto al número generado.

- Método *obtener_recompensa*: tras recibir el parámetro estado, en función del estado al que se va a transitar, obtiene la recompensa correspondiente en función de donde se encuentra. Si se trata de una recompensa final positiva o negativa, un peligro, o simplemente una casilla blanca que aplica la penalización del entorno. En estos métodos se usa el valor hash de un estado como hemos comentado anteriormente para la realización de *if estado in x.*

-Podemos ver una representación de un problema tras la ejecución de los algoritmos

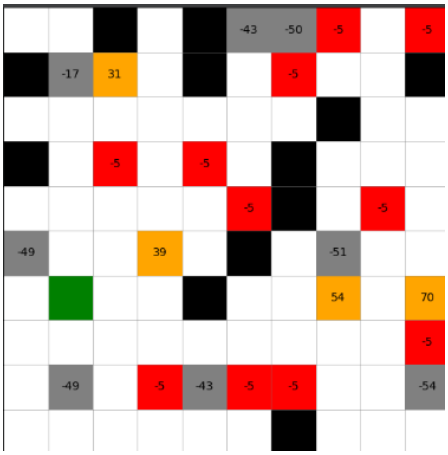


Clase Estado

-Representa, de forma idéntica a la primera práctica, a todo el sistema; al entorno y la posición del agente en un momento determinado, equivalentemente a una foto.

De igual forma, hemos representado el estado del problema en su clase correspondiente mediante las variables fila y columna, que representan la posición del agente, y que son inicializadas en el constructor con los métodos *__eq__* y *__hash__* para poder comparar entre estados y que sean hasheables, útiles para nuestras implementaciones como se desarrollará más adelante.

Hemos usado de igual forma el método *__repr__* para la visualización de estadísticas y depuración a lo largo de la práctica.



Clase Agente

-Esta clase representa al agente, que desea moverse a una nueva casilla aplicando una acción en función de su estado actual y de saber si ha llegado o no, así como obtener la recompensa asociada. Como se ha comentado, es el entorno el que reacciona a sus acciones tal y como se ha representado arriba, aquí se implementan las distintas estrategias que vienen dadas por el algoritmo Q-Learning y el algoritmo de Iteración de Políticas. Para ello se ha seguido un sencillo esquema:

- Algoritmo Q-Learning:
    - Ejecutar el algoritmo con un número n de episodios → realizar un episodio → Actualizar la Q-Tabla en cada episodio → Al finalizar obtenemos la política e imprimimos las estadísticas.
    - Se ha optado por usar épsilon para encontrar un equilibrio entre exploración y explotación y evitar que haya una convergencia hacia un estado determinado pudiendo haber recompensas mayores en otros lugares, además de aplicar el decaimiento de épsilon que nos permite que el agente cada vez explore menos en cada episodio, de forma similar con el decaimiento de Alpha, que nos ayuda a que no se dispare el tiempo de ejecución, los valores se estabilicen y haya convergencia a medida que el algoritmo itera. Se han aplicado las fórmulas de los apuntes.
- Algoritmo de Iteración de Políticas, partiendo de la política obtenida del algoritmo anterior se desea evaluarla y mejorarla:
    - Iteración de políticas → Evaluación de política → Mejora de política.
    - Para la iteración de políticas se tienen en cuenta un número de iteraciones y en cada iteración se realizan los siguientes pasos:
        - Evaluación de la política actual.
        - Actualización de la tabla de utilidades.
        - Mejorar la política actual que tendrá en cuenta la siguiente iteración.

-Para la realización de todo lo anterior, hemos declarado unas constantes que nos facilitan los cálculos en los métodos de la clase agente y nos permiten agilizarlos y codificarlos en formato deseado para las estadísticas.

-El constructor se inicializa con las variables encargadas de la ejecución del algoritmo:

- El propio entorno el cual va a interactuar con el agente y viceversa.
- Inicializamos la Q-Tabla a cero para cada uno de los posibles movimientos de cada estado, [0,1,2,3] siendo equivalente a UP, RIGHT, DOWN, LEFT respectivamente.
- Los valores Alpha, gamma y épsilon, que corresponden a la tasa de aprendizaje, la tasa de descuento de futuras recompensas y el factor de exploración.
- Valores auxiliares que nos permiten parametrizar las estadísticas, como la política obtenida para cada algoritmo, un método auxiliar de Alpha para aplicar su decaimiento también parametrizado, al igual que el decaimiento de épsilon y el número de episodios para el algoritmo

2.3- Algoritmos implementados

Parte Q-Learning del Algoritmo

Algoritmo Q-learning

Inicializar $\hat{Q}$ aleatoriamente o arbitrariamente (p.e. a 0.0)
for episodio $= 1, 2, \ldots$ **do**
  Inicializar s
  repeat
    Seleccionar la acción a de acuerdo a $\pi^*(\hat{Q})$
    Ejecutar la acción a en s
    Obtener del entorno el estado s' y la recompensa r
    Actualizar $\hat{Q}(s, a)$:
      Si s' es final: $\hat{Q}(s,a) = (1-\alpha)\cdot\hat{Q}(s,a) + \alpha\cdot[r]$
      Si s' **no** es final: $\hat{Q}(s,a) = (1-\alpha)\cdot\hat{Q}(s,a) + \alpha\cdot\left[r + \gamma\cdot max_{a'}\hat{Q}(s',a')\right]$
    $s \leftarrow s'$
  until s es final
return Generar la política óptima π^* a partir de $\hat{Q}$

-Como se ha comentado, el agente desea realizar un movimiento de un estado a otro, para ello lo vamos a hacer de forma sencilla con el método mover agente:

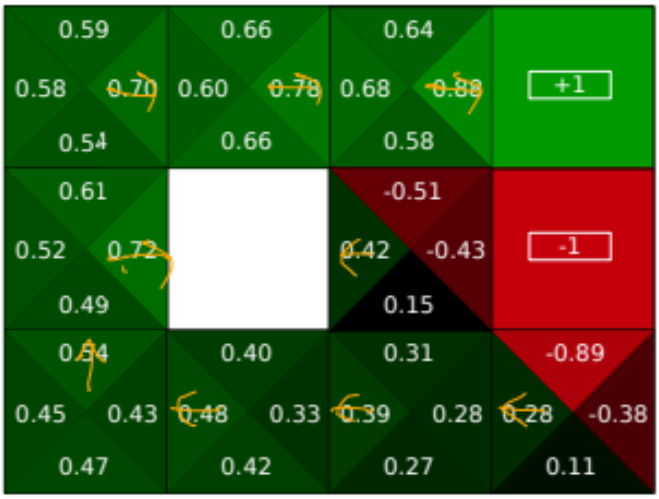
- Método *mover_agente*: se genera un número entre cero y uno, si es inferior o superior al valor de épsilon se inicializa como True o False, respectivamente, que indica si el agente desea explorar y no realizar la acción deseada. Tras esto se llama al método de la clase Entorno, *mover_agente*, explicado anteriormente, que es el encargado de reaccionar a dicho intento de movimiento. Tras esto obtenemos el nuevo estado del agente y la recompensa asociada.

-El método que se encarga de ejecutar el algoritmo mediante el esquema implementado anteriormente, así como todos los pasos realizados hasta obtener una política y llegar a una convergencia es el siguiente:

- Método *ejecutar_algoritmo*: en función del número de episodios que ha sido recibido como parámetro, creamos una variable de tiempo inicial y en un simple bucle for

realizamos los n episodios. Tras finalizar, obtenemos el tiempo final de ejecución y llamamos al método que obtiene la política.

- Método *episodio*: primeramente, el agente desea moverse por el entorno hasta encontrar un estado de destino, se inicializa el estado inicial y el booleano que controla que hemos llegado en False. Mientras no lleguemos al destino el agente va a recoger de la Q-Tabla el máximo valor para las posibles acciones en ese estado llamando a *max_q*. Cuando tiene la acción seleccionada llama a mover_agente para obtener el nuevo estado y la recompensa, y después valora si es un estado de destino, y, si lo es, se inicializa el valor de destino a True, que finaliza el episodio. Se aplican los decaimientos de épsilon y de Alpha y actualizamos la Q-Tabla al llegar al destino. En caso de no llegar al destino continua en el bucle y actualiza su estado actual.
- Método *max_q*: para ello de entre las posibles acciones válidas de un vector obtenemos los índices que mayor valor tienen y en caso de tener varios índices con empate se opta por escoger uno al azar. Cabe destacar que hemos implementado una forma de evitar que el agente elija acciones no válidas; asignamos un valor muy bajo a la recompensa de esa acción para descartarla y que el agente no transite por ahí, evitando así que el agente se disperse mucho. Sin esta implementación el agente opta por intentar atravesar paredes, aunque sea una acción no válida en el entorno.
- Método *actualizar_qtabla*: es la aplicación directa de la fórmula de los apuntes, si es un estado de destino o no recibido como parámetro, aplicamos una formula u otra para actualizar la Q-Tabla y retornamos el valor.
- Método *obtener_política*: vamos recorriendo los diferentes estados posibles, sin contar los no transitables ni los destinos finales el valor máximo de la Q-Tabla para ese estado llamando al método *max_q*.
- Método *estadísticas*: en función del parámetro que indica desde donde ha sido llamado, ya bien sea desde la parte de Q-Learning o la parte de Iteración de Políticas se imprimen por pantalla unas estadísticas u otras, así como el mapa de políticas proporcionadas en el Campus Virtual.



Parte Iteración de Políticas

Algoritmo de evaluación de políticas

```
1: function POLICYEVALUATION(MDP< S, A, T, R, γ >, π)
2:     U ← [0, ..., 0]
3:     repeat
4:         δ ← 0
5:         for each s ∈ S do
6:             U'(s) ← R(s) + γ Σ_s' T(s, π(s), s') U(s')        ▷ Sólo considera π(s)
7:             if |U'(s) − U(s)| > δ then
8:                 δ ← |U'(s) − U(s)|
9:         U ← U'
10:    until δ < ε
11:    return U
```

Algoritmo de mejora de la política

```
1: function POLICYIMPROVEMENT(MDP< S, A, T, R, γ >, U^π)
2:     for each s ∈ S do
3:         π'(s) ← argmax_a R(s) + γ Σ_s' T(s, a, s') U^π(s')
4:     return π'
```

-Como se ha comentado, parte de un sencillo esquema el cual itera sobre una política ya dada, al contrario que la iteración de valores, y vamos a usar la política proporcionada por el algoritmo Q-Learning para evaluarla y mejorarla en cada iteración hasta un total de n iteraciones dado por el valor "self.num_episodios". Aquí hay que pensar distinto a cómo se plantea el Q-Learning ya que aquí no hay ningún agente que se mueva. Para cada estado hay que actualizar su utilidad esperada a través de su recompensa en la tabla de utilidades, se aplica a todos los estados a los que se pueda llegar y se parte de la misma información del agente, así como el acceso a la información del entorno y de las variables distintas usadas anteriormente. Para ello se han usado los diferentes métodos:

- Método *iteracion_de_politicas*: dado un número n de episodios sobre los estados posibles, inicializamos la Tabla de Utilidades. Tras las indicaciones del profesor se han integrado también los estados finales, a los cuales les corresponde el valor de su recompensa, el resto salvo los no válidos se han inicializado a cero. Posteriormente se recoge el tiempo inicial en variable, se itera sobre ese número de episodios llamando al método *evaluacion_de_politica* al cual pasamos como parámetro los estados posibles. Tras evaluar, llamamos al método *mejorar_politica* y recogemos el tiempo de ejecución tras finalizar las iteraciones restándole el tiempo inicial. Llamamos al método de estadísticas y finalizamos retornando la política mejorada (recordemos que el método estadísticas contiene una variable booleana que indica qué estadísticas imprimir, en este caso la parte de Iteración de Políticas).
- Método *evaluacion_de_politica*: teniendo en cuenta todos los estados posibles, iteramos sobre ellos, para el estado actual si no es un estado de destino y para el siguiente estado si nos encontramos en un entorno estocástico debemos de tener en cuenta la probabilidad de transicionar al estado siguiente y a sus dos perpendiculares y realizar dicho sumatorio. Para ello seleccionamos el estado siguiente a un estado almacenado en la política llamando a *siguiente_estado* y llamamos al método de *transicion_y_utilidad,* que realiza el sumatorio previamente mencionado cuyo proceso se detalla más adelante, posteriormente actualizamos la tabla de utilidades siguiendo la fórmula de los apuntes y finalizamos con dicha iteración.

- Método *mejorar_politica*: utilizamos como valor auxiliar un diccionario vacío para obtener la política mejorada, un método que nos permite evaluar el valor máximo para una acción posible en un estado inicializado a cero y puesto que vamos a encontrar un máximo dicho valor se inicializa a menos infinito, se itera sobre los estados posibles y si es un estado válido, para cada uno de los sucesores (válidos) de dicho estado, se llama al método encargado de obtener la probabilidad de transicionar multiplicado a la utilidad de ese estado siguiente llamando al método *transición_y_utilidad*, si el valor encontrado es mejor, se obtiene dicho nuevo estado y hayamos la acción que nos ha llevado a ese nuevo estado, cabe destacar que aquí si debemos de tener en cuenta la estocasticidad para una acción en concreto, tal y como se señala en el ejemplo de los apuntes, actualizamos la política auxiliar mejorada y reseteamos los valores para volver a iterar sobre ese estado, tras esto retornamos la política mejorada que va a utilizar la siguiente iteración de la evaluación de políticas.
- Método *transición_y_utilidad*: aplicamos la fórmula de los apuntes que determina la probabilidad de transicionar de un estado a otro dada una acción. En este caso, al encontrarnos un entorno estocástico, debemos de tenerlo en cuenta para las probabilidades de sus perpendiculares. Para ello obtenemos inicialmente la acción que lleva al estado siguiente desde el actual, ambos recibidos como parámetros, luego en una estructura if/elif, para cada acción aplicamos el sumatorio de la probabilidad de realizar la acción deseada y la probabilidad de cada una de las perpendiculares de ser válidas, posteriormente se retorna como valor, por ejemplo para para estocasticidad del entorno, por ejemplo para una estocasticidad de un 80% -> T80%(s,a1,s1)*U(s1) + T10%(s,a2,s2)*U(s2) + T10%(s,a3,s3)*U(s3) que corresponde a la probabilidad de esa acción y a sus perpendiculares.
- Métodos auxiliares que facilitan la legibilidad y la ejecución de los métodos anteriores:
    - *decodificar_accion*: dada una acción en formato (x, y) entre los posibles valores de [(-1, 0), (0, 1), (1, 0), (0, -1)] obtiene el String correspondiente a esa acción a UP, RIGHT, DOWN, LEFT.
    - *sucesores*: obtiene los estados adyacentes válidos de un estado en concreto, similar al usado en la primera práctica.
    - *siguiente_estado*: dado un estado obtiene el siguiente almacenado en la política de Q-Learning, recordemos que se parte de esta política inicialmente para empezar el algoritmo de Iteración de Políticas.

Métodos main con diferentes implementaciones

-Dadas las diferentes implementaciones posibles en función de cada una de las variables, tal y como se ha sugerido en el esquema de la memoria, se han de parametrizar todos los valores. En un primer main se han inicializado en formato de vector posibles valores para cada uno de los parámetros, se han parametrizado dos json para realización de pruebas y mediante el uso de itertools se seleccionan valores aleatorios posibles y se almacenan como semilla para, en un bucle for, ir inicializando los Algoritmos para cada semilla hasta un total de cinco. Por otro lado, si se desea una implementación singular e ir manualmente probando valores muy concretos, se tiene otro main adicional para realizar un algoritmo con valores personalizados e implementados de forma concreta sin el uso de una semilla aleatoria.

2.4- Evaluación experimental

-Para la evaluación experimental hemos usado inicialmente el escenario de los apuntes correspondiente al tema cinco, un mundo de tamaño 4 x 3 casillas almacenado en formato json con el nombre lesson5-rl. Para la evaluación de los experimentos primero hemos optado por reproducir diferentes semillas y analizar los resultados obtenidos para ambos algoritmos. Las semillas posibles, que se seleccionan de forma aleatoria de entre las posibles hasta un total de cinco, son las siguientes:

- penalizacion_entorno = [-0.04, -0.5, -5]
- penalizacion_peligro = [-5, -1, -10]
- estocasticidad_entorno = [0.7, 0.8, 0.9]
- numero_episodios = [100, 500, 1000]
- alpha = [0.2, 0.5, 0.9]
- gamma = [0.9, 0.5, 0.2]
- epsilon = [0.2, 0.3, 0.5]
- decaimiento_epsilon = [0.05, 0.01, 0.02, 0.005]
- decaimiento_alpha = [0.999, 0.995, 0.99]

-Para el Q-Learning se han tenido en cuenta diferentes parámetros de Alpha, Gamma y Épsilon y poseer diferentes escenarios, así como diferentes decaimientos de Alpha y Épsilon para ver sus influencias en dicho escenario.

Semilla 1 (lesson5-rl)

Valores iniciales:

- Penalización del entorno: -0.5
- Penalización de peligro: -5
- Estocasticidad del entorno: 0.9
- Número de episodios: 100
- Alpha: 0.9
- Gamma: 0.9
- Épsilon: 0.2
- Decaimiento de épsilon: 0.01
- Decaimiento de Alpha: 0.999

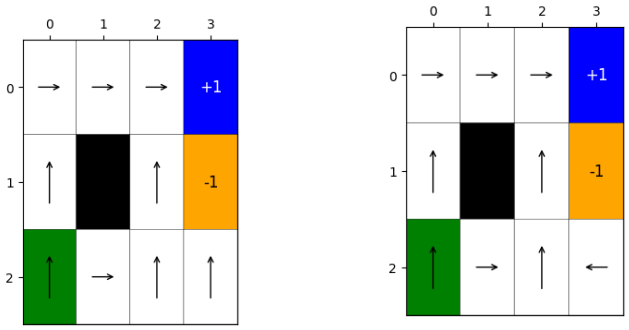
Estadísticas del algoritmo de Q-Learning:

- Número de episodios: 100
- Estado inicial: (2,0)
- Personas a rescatar: [(0, 3)]
- Tamaño de la ciudad: 3 x 4
- Tiempo de ejecución del Algoritmo: 0.05 segundos

Q-Tabla:

```
[[[ 0.         -0.14170231 -1.50386797  0.        ]
  [ 0.          0.39990293  0.         -1.23595422]
  [ 0.          0.8265309  -0.76096252 -0.77125117]
  [ 0.          0.          0.          0.        ]]

 [[-0.76419033  0.         -1.73782032  0.        ]
  [ 0.          0.          0.          0.        ]
  [-0.28614995 -0.96552031 -1.04061004  0.        ]
  [ 0.          0.          0.          0.        ]]

 [[-1.11596639 -2.06234731  0.          0.        ]
  [ 0.         -1.07178565  0.         -2.06593046]
  [-0.24141767 -1.2911882   0.         -1.66037283]
  [-0.96552031  0.          0.         -1.48577047]]]
```

Política obtenida y gráfico de visualización de políticas (Q-Learning vs Iteración de Políticas) :



Estadísticas del algoritmo de Iteración de políticas:

- Número de iteraciones realizadas: 100
- Tiempo de ejecución: 0.02400040626525879

Tabla de utilidades final:

Estado	Valor
(0, 0)	-0.760847
(0, 1)	-0.260018
(0, 2)	0.296274
(0, 3)	1.000000
(1, 0)	-1.116286
(1, 2)	-0.305018
(1, 3)	-1.000000
(2, 0)	-1.457913
(2, 1)	-1.193790
(2, 2)	-0.856531
(2, 3)	-1.238790

Análisis:

-Tenemos un entorno casi determinista en su totalidad con un valor de Alpha elevado, que permite en este caso que el algoritmo omita el conocimiento previo y sólo considere información reciente, al haber estocasticidad debemos de decaer dicho valor en cada iteración, el valor de gamma es elevado también que hace que el algoritmo considera importante las recompensas futuras en lugar de las actuales junto con un valor de épsilon que favorece la exploración de un 20%.

-El entorno penaliza considerablemente en función de las recompensas finales, por lo que el algoritmo considera más importante en el caso del estado (2,3) ir a una recompensa final, aunque sea un peligro fatal, que seguir moviéndose hasta la persona a rescatar. El resto tiene un comportamiento esperable, tratando de llegar a la persona a rescatar, vemos como el método de iteración de políticas mejora respecto a la situación del estado (2,3) impidiendo que caiga en un peligro fatal.

Semilla 2 (lesson5-rl)

Valores iniciales:

- Penalización del entorno: -5
- Penalización de peligro: -1
- Estocasticidad del entorno: 0.7
- Número de episodios: 500
- Alpha: 0.5
- Gamma: 0.9
- Épsilon: 0.2
- Decaimiento de épsilon: 0.01
- Decaimiento de Alpha: 0.999

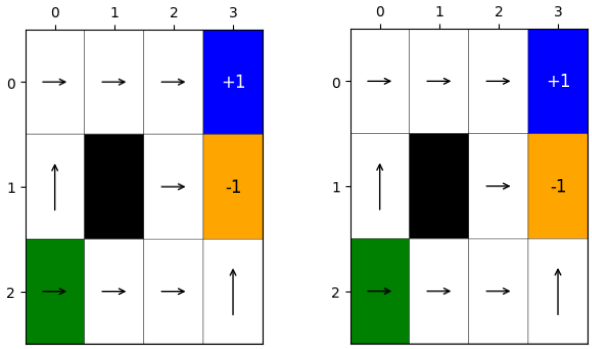
Estadísticas del algoritmo de Q-Learning:

- Número de episodios: 500
- Estado inicial: (2,0)
- Personas a rescatar: [(0, 3)]
- Tamaño de la ciudad: 3 x 4
- Tiempo de ejecución del Algoritmo: 0.21 segundos

Q-Tabla:

```
[[[  0.         -12.50827802 -20.56067044   0.        ]
  [  0.          -8.17187434   0.         -13.20370072]
  [  0.          -1.14089408  -6.85644667  -6.41615587]
  [  0.           0.           0.           0.        ]]

 [[-16.86076485   0.         -22.60348814   0.        ]
  [  0.           0.           0.           0.        ]
  [ -8.40615645  -2.58071654  -7.92263073   0.        ]
  [  0.           0.           0.           0.        ]]

 [[-22.33444172 -18.23120368   0.           0.        ]
  [  0.         -14.18097306   0.         -19.19082104]
  [-13.00181114  -8.94447673   0.         -13.19599811]
  [ -2.68216806   0.           0.          -9.49396811]]]
```

Política obtenida y gráfico de visualización de políticas (Q-Learning vs Iteración de Políticas):



Estadísticas del algoritmo de Iteración de políticas:

- Número de iteraciones realizadas: 500
- Tiempo de ejecución: 0.1170339584350586

Tabla de utilidades final:

Estado	**Valor**
(0, 0)	-11.996496
(0, 1)	-8.414594
(0, 2)	-5.419991
(0, 3)	1.000000
(1, 0)	-12.557793
(1, 2)	-7.777711
(1, 3)	-1.000000
(2, 0)	-14.008378
(2, 1)	-11.608057
(2, 2)	-10.488979
(2, 3)	-7.046012

Análisis:

Este entorno es algo más ruidoso con una estocasticidad de un 70% y se ha asignado un valor Alpha intermedio que equilibra la tasa de aprendizaje que equilibre el aprendizaje previo y el nuevo así como un valor gamma también elevado que le da importancia a las recompensas futuras y un factor de exploración de un 20% junto con una penalización muy agresiva, como consecuencia vemos que esto ha involucrado un tiempo de ejecución ligeramente superior por lo que la convergencia se ha retrasado ligeramente, pero vemos como el agente para garantizar la máxima recompensa trata de llegar a cualquiera de las dos recompensas que más cerca encuentre al contrario que pasaba con la anterior semilla, ambos algoritmos devuelven la misma política.

Semilla 3 (lesson5-rl)

Valores iniciales:

- Penalización del entorno: -0.04
- Penalización de peligro: -1
- Estocasticidad del entorno: 0.7
- Número de episodios: 500
- Alpha: 0.2
- Gamma: 0.5
- Épsilon: 0.2
- Decaimiento de épsilon: 0.01
- Decaimiento de Alpha: 0.999

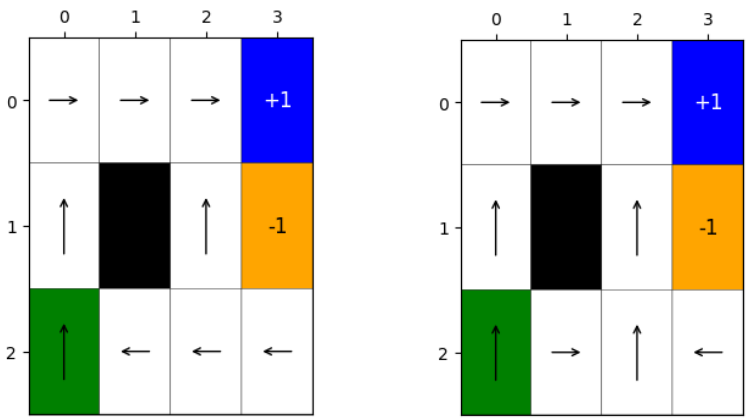
Estadísticas del algoritmo de Q-Learning:

- Número de episodios: 500
- Estado inicial: (2,0)
- Personas a rescatar: [(0, 3)]
- Tamaño de la ciudad: 3 x 4
- Tiempo de ejecución del Algoritmo: 0.30 segundos

Q-Tabla:

```
[[[ 0.          0.04859385 -0.0386      0.        ]
  [ 0.          0.24169366  0.         -0.0178611 ]
  [ 0.          0.70925932 -0.00514519 -0.00485103]
  [ 0.          0.          0.          0.        ]]

 [[-0.01697909  0.         -0.05960456  0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.07331358 -0.3247791  -0.35512488  0.        ]
  [ 0.          0.          0.          0.        ]]

 [[-0.04875651 -0.06913147  0.          0.        ]
  [ 0.         -0.07459602  0.         -0.0680317 ]
  [-0.08903426 -0.09180149  0.         -0.08004174]
  [-0.32148762  0.          0.         -0.26326616]]]
```

Política obtenida y gráfico de visualización de políticas (Q-Learning vs Iteración de Políticas):



Estadísticas del algoritmo de Iteración de políticas:

- Número de iteraciones realizadas: 500
- Tiempo de ejecución: 0.11552071571350098

Tabla de utilidades final:

Estado	Valor
(0, 0)	-0.019601
(0, 1)	0.068325
(0, 2)	0.309499
(0, 3)	1.000000
(1, 0)	-0.046860
(1, 2)	-0.006675
(1, 3)	-1.000000
(2, 0)	-0.060896
(2, 1)	-0.059933
(2, 2)	-0.056951
(2, 3)	-0.134933

Análisis:

Tenemos otro entorno relativamente ruidoso en torno a una estocasticidad de un 70%, un valor bajo de Alpha que no favorece el aprendizaje y tenga más en cuenta el conocimiento previo, una tasa de descuento equilibrada entre favorecer recompensas actuales y futuras y un factor de exploración bajo de un 20%, vemos que en Q-Learning al tener una penalización del entorno bajo de -0.04 el agente en el estado (2,2) se mueve en una dirección que no parece ser la más idónea pero se debe a la influencia del entorno ruidoso y a un valor bajo de Alpha que ha hecho retrasar la convergencia, tal y como se ve en el tiempo de ejecución que es algo más elevado, esto se mejora con la parte de iteración de políticas que determina la mejor acción para ese estado.

Semilla 4 (instance-10-10-12-4-11-1111--rl)

-Para este entorno se ha inicializado un decaimiento de épsilon más bajo debido al mayor tamaño del problema y favorecer todavía más a la exploración entre iteraciones

Valores iniciales:

- Penalización del entorno: -5
- Penalización de peligro: -10
- Estocasticidad del entorno: 0.8
- Número de episodios: 1000
- Alpha: 0.9
- Gamma: 0.5
- Épsilon: 0.2

- Decaimiento de épsilon: 0.005
- Decaimiento de Alpha: 0.999

Estadísticas del algoritmo de Q-Learning:

- Número de episodios: 1000
- Estado inicial: (6,1)
- Personas a rescatar: [(5, 3), (6, 9), (1, 2), (6, 7)]
- Tamaño de la ciudad: 10 x 10
- Tiempo de ejecución del Algoritmo: 0.25 segundos

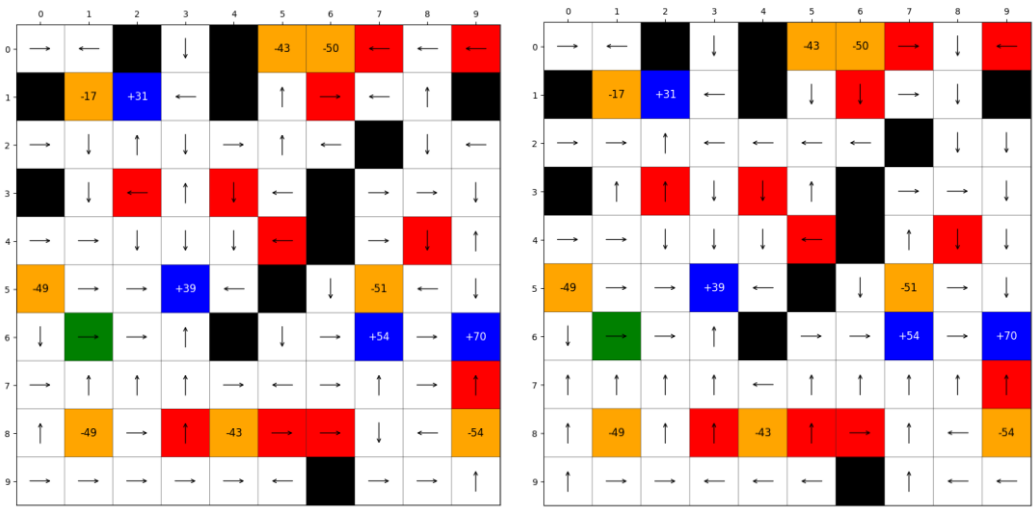
Q-Tabla:

```
[[[  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]]

 [[  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.         -1.65462941 10.25870235]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]]

 [[  0.          0.          0.          0.        ]
  [ -5.62574    -5.62574    -1.97370968  -3.88293264]
  [  0.         -0.88670534  -3.76621768  -1.65462941]
  [ -2.76169913  -2.76169913  -1.65462941  -2.50408433]
  [  0.          0.         -3.30925882  -1.65462941]
  [  0.          0.          0.         -1.65462941]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]]

 [[  0.          0.          0.          0.        ]
  [ -6.90102633  -7.37620216  -6.54467613   0.        ]
  [ -3.57535041  -3.71040111  -3.25025192  -3.14937426]
  [ -1.65462941  -3.30925882  -2.42967987  -1.92840926]
  [ -1.65462941  -1.65462941   0.         -1.68188473]
  [ -1.65462941   0.         -3.30925882   0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]
  [  0.          0.          0.          0.        ]]
```

```
[[  0.          -6.65119319 -16.21536823   0.        ]
 [ -5.96706579  -4.52837624  -6.17307666  -6.39258955]
 [ -5.7971781   -6.10776112   6.09381693  -4.5       ]
 [  0.          -1.3877916    6.87689876  -1.5048635 ]
 [ -3.30925882  -3.30925882   1.49066457  -1.65462941]
 [ -1.92840926   0.           0.          -1.65462941]
 [  0.           0.           0.           0.        ]
 [  0.           0.           0.           0.        ]
 [  0.           0.           0.           0.        ]
 [  0.           0.           0.           0.        ]]

[[  0.           0.           0.           0.        ]
 [ -5.33740784   9.71351732 -18.65862599 -16.21536823]
 [ -4.5         34.14471995  -2.30898331  -4.5       ]
 [  0.           0.           0.           0.        ]
 [ -1.65462941   0.           0.          27.31880437]
 [  0.           0.           0.           0.        ]
 [  0.         -16.87722     -1.92840926   0.        ]
 [  0.           0.           0.           0.        ]
 [  0.           0.           0.           0.        ]
 [  0.           0.           0.           0.        ]]

[[-16.21536823 -19.41779135  -9.07853137   0.        ]
 [ -7.42153406  -0.30553622  -7.29522927  -7.36200545]
 [ -4.5          9.42402998  -4.53057046  -5.24653825]
 [ 32.38179456   0.           0.           0.        ]
 [  0.           0.           0.           0.        ]
 [  0.          -3.50241082  -3.03547897   0.        ]
 [ -1.65462941  -1.65462941  -1.65462941  -3.03547897]
 [  0.           0.           0.           0.        ]
 [  0.           0.           0.           0.        ]
 [  0.           0.           0.           0.        ]]

[[ -8.94726378  -8.7504278   -8.97482779   0.        ]
 [ -6.16755129 -18.74191967 -16.21536823 -17.32243795]
 [ -4.41561587  -6.84285095  -6.63142286  -6.51208064]
 [  5.31049479  -5.11620583  -3.30925882  -4.9860166 ]
 [  0.         -16.2951968  -23.75061248 -23.75061248]
 [ -4.14254869  -3.50241082  -3.30925882  -3.03547897]
 [ -1.65462941   2.88728407  -3.30925882  -2.15688869]
 [ 11.04645165   0.           0.           0.        ]
 [  0.           0.           0.          -0.78336581]
 [  0.           0.           0.           0.        ]]
```

```
[[ -9.14557726 -17.32243795 -16.21536823   0.        ]
 [  0.           0.           0.           0.        ]
 [-16.21536823 -11.0741548  -20.09450861 -18.68741086]
 [ -1.83782641 -14.22981294  -4.91195665  -4.40233229]
 [  0.           0.           0.           0.        ]
 [ -3.30925882  -1.65462941  -1.92840926 -15.33688265]
 [ -3.30925882  -1.65462941   0.          -3.58303867]
 [  0.          -1.65462941   0.          -3.30925882]
 [ -1.65462941  -1.65462941  -1.65462941   0.        ]
 [  0.           0.           0.           0.        ]]

[[ -5.08191359  -3.62769907   0.           0.        ]
 [-16.21536823  -5.01888116   0.          -5.38213061]
 [ -6.06881317  -5.7602005    0.          -5.79948278]
 [ -5.7971781   -5.52505584   0.          -6.18532273]
 [-14.22981294  -6.17925217   0.         -17.21943293]
 [ -6.39242324   0.           0.          -5.42419911]
 [  0.           0.           0.           0.        ]
 [ -1.65462941   0.           0.           0.        ]
 [ -1.65462941   0.           0.          -1.65462941]
 [  0.           0.           0.           0.        ]]]
```

Política obtenida y gráfico de visualización de políticas (Q-Learning vs Iteración de políticas) :



Estadísticas del algoritmo de Iteración de políticas:

- Número de iteraciones realizadas: 1000
- Tiempo de ejecución: 2.448000431060791

Tabla de utilidades final:

Estado	Valor	Estado	Valor	Estado	Valor	Estado	Valor
(0, 0)	-8.738.095	(3, 1)	-6.710.032	(5, 8)	5.068.722	(8, 3)	-13.245.687
(0, 1)	-9.345.238	(3, 2)	-7.564.826	(5, 9)	23.253.436	(8, 4)	-43.000.000
(0, 3)	-2.119.782	(3, 3)	-1.706.971	(6, 0)	-8.861.874	(8, 5)	-14.501.695
(0, 5)	-43.000.000	(3, 4)	-10.845.610	(6, 1)	-5.678.383	(8, 6)	-9.442.985
(0, 6)	-50.000.000	(3, 5)	-8.908.738	(6, 2)	-0.527361	(8, 7)	1.107.001
(0, 7)	-14.433.950	(3, 7)	-8.377.958	(6, 3)	10.573.632	(8, 8)	-4.607.465
(0, 8)	-9.930.847	(3, 8)	-7.349.646	(6, 5)	1.433.104	(8, 9)	-54.000.000
(0, 9)	-13.972.339	(3, 9)	-3.811.254	(6, 6)	16.670.117	(9, 0)	-9.969.020
(1, 1)	-17.000.000	(4, 0)	-9.696.787	(6, 7)	54.000.000	(9, 1)	-11.156.757
(1, 2)	31.000.000	(4, 1)	-5.616.967	(6, 8)	23.563.030	(9, 2)	-9.266.894
(1, 3)	7.200.546	(4, 2)	-0.535410	(6, 9)	70.000.000	(9, 3)	-9.369.042
(1, 5)	-9.094.157	(4, 3)	10.533.876	(7, 0)	-8.944.887	(9, 4)	-10.897.617
(1, 6)	-14.553.982	(4, 4)	-0.787060	(7, 1)	-8.002.754	(9, 5)	-10.084.131
(1, 7)	-9.232.230	(4, 5)	-10.760.261	(7, 2)	-5.683.133	(9, 7)	-4.917.059
(1, 8)	-8.776.332	(4, 7)	-8.761.987	(7, 3)	-1.441.024	(9, 8)	-7.197.197
(2, 0)	-6.331.791	(4, 8)	-8.216.082	(7, 4)	-7.726.410	(9, 9)	-10.578.879
(2, 1)	-3.329.477	(4, 9)	3.890.570	(7, 5)	-4.698.864		
(2, 2)	7.140.061	(5, 0)	-49.000.000	(7, 6)	2.284.294		
(2, 3)	-1.869.297	(5, 1)	-1.346.023	(7, 7)	17.023.809		
(2, 4)	-6.289.999	(5, 2)	10.546.861	(7, 8)	6.191.882		
(2, 5)	-8.416.144	(5, 3)	39.000.000	(7, 9)	18.309.594		
(2, 6)	-9.094.157	(5, 4)	10.560.647	(8, 0)	-11.027.955		
(2, 8)	-8.286.800	(5, 6)	-0.881953	(8, 1)	-49.000.000		
(2, 9)	-6.938.842	(5, 7)	-51.000.000	(8, 2)	-10.385.538		

Análisis:

-Con una penalización moderadamente alta y una estocasticidad de un 80% el algoritmo tiene una tasa elevada de aprendizaje, por lo que sobrescribe continuamente la información vieja favoreciendo el aprendizaje, una tasa de descuento equilibrada entre la importancia de las recompensas futuras e inmediatas y un factor de exploración de un 20%, vemos cómo el algoritmo converge adecuadamente hacia las casillas de recompensa finales donde se encuentran las personas rescatadas.

Semilla 5 (instance-10-10-12-4-11-1111--rl)

Valores iniciales:

- Penalización del entorno: -0.04
- Penalización de peligro: -10
- Estocasticidad del entorno: 0.8
- Número de episodios: 500
- Alpha: 0.5
- Gamma: 0.9

- Épsilon: 0.5
- Decaimiento de épsilon: 0.05
- Decaimiento de Alpha: 0.99

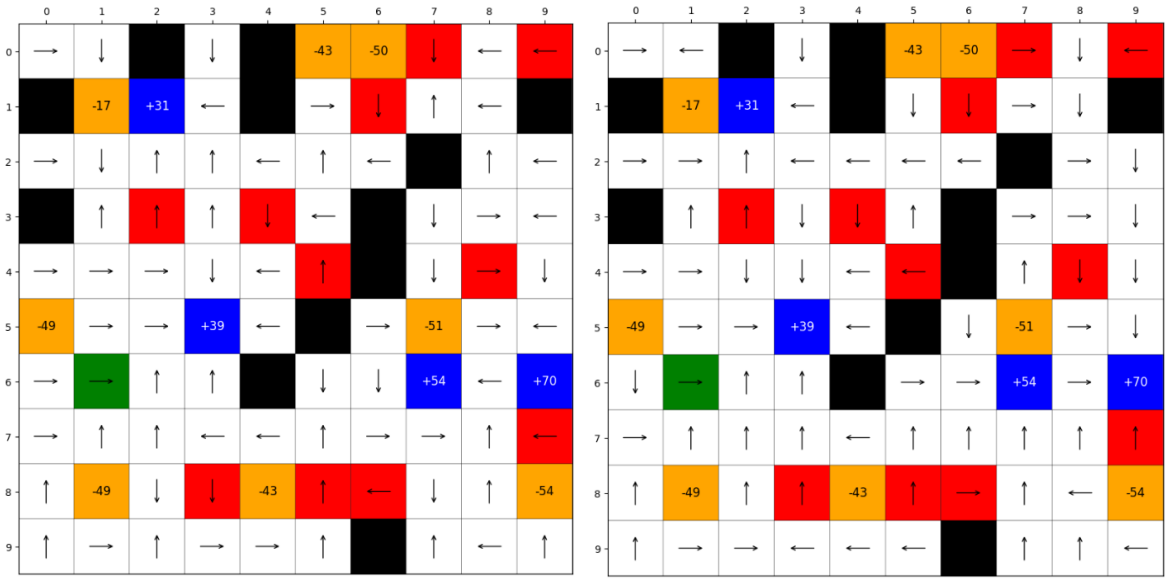
Estadísticas del algoritmo de Q-Learning:

- Número de episodios: 500
- Estado inicial: (6,1)
- Personas a rescatar: [(5, 3), (6, 9), (1, 2), (6, 7)]
- Tamaño de la ciudad: 10 x 10
- Tiempo de ejecución del Algoritmo: 0.13 segundos

Q-Tabla:

```
[[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00 -2.24166807e-02  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [-1.39660560e-02  0.00000000e+00 -1.39660560e-02  9.39042302e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [-1.51594736e+01  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  3.09733560e+00  0.00000000e+00]
  [ 2.59074511e+01 -1.21275789e-02 -3.03189472e+00  0.00000000e+00]
  [-2.05782035e-02 -2.24166807e-02 -2.48691176e-02 -2.05782035e-02]
  [ 0.00000000e+00 -2.05782035e-02 -3.03373320e+00 -1.21275789e-02]
  [ 0.00000000e+00 -1.21275789e-02 -1.21275789e-02 -1.21275789e-02]
  [-3.03189472e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [-2.12606481e+00 -3.99911279e+00 -3.05543607e+00  0.00000000e+00]
  [ 5.21047930e+00 -2.36977509e-02 -1.21275789e-02 -2.24166807e-02]
  [-3.74411797e-02 -3.03189472e+00 -3.03189472e+00 -3.03189472e+00]
  [-2.05782035e-02 -1.21275789e-02 -1.21275789e-02 -2.24166807e-02]
  [-3.04218383e+00  0.00000000e+00 -3.04218383e+00 -3.03189472e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[ 0.00000000e+00 -2.24166807e-02 -1.48562841e+01  0.00000000e+00]
  [-3.31813075e-01  3.43325579e+00 -2.39764543e-02 -2.24166807e-02]
  [-3.03189472e+00  1.58151157e+01 -2.24166807e-02 -2.24166807e-02]
  [-1.21275789e-02 -1.21275789e-02  3.86448321e+01  0.00000000e+00]
  [-3.03189472e+00 -3.03189472e+00 -1.93432899e-01  5.60135042e+00]
  [-1.39660560e-02  0.00000000e+00  0.00000000e+00 -2.24166807e-02]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [-2.64666878e-02  1.42037736e+01 -1.48562841e+01 -1.48562841e+01]
  [-2.24166807e-02  3.51493970e+01 -1.42447595e-02 -1.39660560e-02]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [-1.21275789e-02  0.00000000e+00  0.00000000e+00  2.49287855e+01]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[-1.48562841e+01  7.25170426e-01 -2.64666878e-02  0.00000000e+00]
  [-3.78295867e-02  6.68421703e+00 -3.50718210e-02 -3.87222499e-02]
  [ 1.63247333e+01 -2.05782035e-02 -3.06554578e-02 -2.18592738e-02]
  [ 3.47461895e+01  0.00000000e+00 -1.21275789e-02 -1.21275789e-02]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[-2.24166807e-02  3.22420283e-01 -2.05782035e-02  0.00000000e+00]
  [ 2.39272916e+00 -3.07796527e-02 -1.48562841e+01 -1.44354632e+01]
  [ 5.64724443e+00 -2.85838683e-02 -2.64666878e-02 -2.99281113e-02]
  [-2.05782035e-02 -2.24166807e-02 -3.03189472e+00  5.48609554e+00]
  [ 0.00000000e+00 -1.30371473e+01 -1.30371473e+01 -2.27376341e-02]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[-1.21275789e-02 -1.48647348e+01 -1.39660560e-02  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [-3.04162642e+00 -3.03189472e+00 -3.36893929e-02 -1.48562841e+01]
  [-1.39660560e-02 -1.30371473e+01 -1.21275789e-02 -1.52471262e-02]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[-2.05782035e-02 -2.05782035e-02  0.00000000e+00  0.00000000e+00]
  [-1.48562841e+01 -2.39764543e-02  0.00000000e+00 -1.48660158e+01]
  [-2.58714843e-02 -2.83051649e-02  0.00000000e+00 -3.00591421e-02]
  [-3.03189472e+00 -2.24166807e-02  0.00000000e+00 -2.52997745e-02]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.21275789e-02]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]]
```

Política obtenida y gráfico de visualización de políticas (Q-Learning vs Iteración de Políticas):



Estadísticas del algoritmo de Iteración de políticas:

- Número de iteraciones realizadas: 500
- Tiempo de ejecución: 1.4139976501464844

Tabla de utilidades final:

Estado	Valor	Estado	Valor	Estado	Valor	Estado	Valor
(0, 0)	-0.471429	(3, 1)	1.470121	(5, 8)	12.641512	(8, 3)	-9.623451
(0, 1)	-1.078571	(3, 2)	-4.479702	(5, 9)	28.592076	(8, 4)	-43.000000
(0, 3)	5.127607	(3, 3)	5.900636	(6, 0)	0.310395	(8, 5)	-10.787225
(0, 5)	-43.000000	(3, 4)	-6.941208	(6, 1)	3.354081	(8, 6)	-5.892511
(0, 6)	-50.000000	(3, 5)	-0.084173	(6, 2)	7.446535	(8, 7)	8.986003
(0, 7)	-10.415954	(3, 7)	0.592250	(6, 3)	15.932327	(8, 8)	4.330241
(0, 8)	-0.982081	(3, 8)	1.583422	(6, 5)	9.123741	(8, 9)	-54.000000
(0, 9)	-10.392832	(3, 9)	4.510190	(6, 6)	22.391416	(9, 0)	-1.095287
(1, 1)	-17.000000	(4, 0)	-1.145971	(6, 7)	54.000000	(9, 1)	-2.647259
(1, 2)	31.000000	(4, 1)	3.360073	(6, 8)	29.279280	(9, 2)	-0.393147
(1, 3)	12.919017	(4, 2)	7.464178	(6, 9)	70.000000	(9, 3)	-0.678431
(1, 5)	-0.243753	(4, 3)	16.279205	(7, 0)	0.456726	(9, 4)	-2.461372
(1, 6)	-10.132811	(4, 4)	6.919921	(7, 1)	1.491430	(9, 5)	-1.563910
(1, 7)	-0.462439	(4, 5)	-7.236240	(7, 2)	3.339226	(9, 7)	3.643036
(1, 8)	0.245897	(4, 7)	-0.022382	(7, 3)	6.520808	(9, 8)	1.772702
(2, 0)	1.694107	(4, 8)	-4.385637	(7, 4)	0.418323	(9, 9)	-2.030919
(2, 1)	4.335266	(4, 9)	11.177548	(7, 5)	4.143501		
(2, 2)	12.879400	(5, 0)	-49.000000	(7, 6)	10.261756		
(2, 3)	6.052743	(5, 1)	6.817922	(7, 7)	22.760292		
(2, 4)	2.034037	(5, 2)	16.305536	(7, 8)	13.744087		
(2, 5)	0.757218	(5, 3)	39.000000	(7, 9)	18.687204		
(2, 6)	-0.243753	(5, 4)	15.905996	(8, 0)	-2.307309		
(2, 8)	0.772547	(5, 6)	6.366566	(8, 1)	-49.000000		
(2, 9)	1.802704	(5, 7)	-51.000000	(8, 2)	-1.635482		

Análisis:

-Ahora el entorno penaliza muy poco el movimiento, a pesar de penalizar moderadamente los peligros, tenemos un Alpha de 0.5, Gamma de 0.9 y un Épsilon que favorece la exploración en un 50% así como una estocasticidad de un 80%, debido a la poca penalización el Agente parece explorar mucho más respecto a la anterior ejecución junto con el valor de Épsilon, debido a que Alpha está en 0.5 hay un balance entre aprovechar la información adquirida y utilizar la información anterior lo que hace que el agente aprenda mas lentamente y en la mitad de episodios por lo que la convergencia se retrasa más respecto al anterior, además al tener el gamma alto, se favorece mas las acciones que conducen a recompensas a largo plazo por lo que al tener poca penalización el agente trata de buscar recompensas superiores junto con un factor de exploración elevado que decae significativamente poco en cada iteración.

2.5- Conclusiones

-En el aprendizaje por refuerzo nos encontramos con un entorno más cercano a lo real debido a la presencia de la estocasticidad, que aleatoriza las posibles acciones del agente, cuyo comportamiento está sujeto a diferentes variables que hacen que sus decisiones sean radicalmente distintas aun estando en el mismo entorno:

- Valores altos de Alpha pueden favorecer la adaptación del agente en entornos estocásticos dado a que este valor implica una mayor tasa de aprendizaje, pero valores bajos retrasan la convergencia. Además, es necesario aplicarle una tasa de decaimiento para que en futuras iteraciones los valores se estabilicen y no sobre aprenda.
- Valores altos de Gamma favorecen posibles recompensas futuras y el agente puede considerar converger en posibles recompensas mayores más lejos de su punto de partida, al contrario que con valores bajos, que se centra más en la recompensa inmediata que en la planificación a largo plazo.
- Valores altos de Épsilon favorecen la exploración y evitan la convergencia en estados finales cercanos al punto de inicio, pero valores bajos evitan posibles recompensas mayores alejadas y provoca estancamientos en óptimos locales. Para encontrar un equilibrio a medida que se itera es ideal implementar el decaimiento de épsilon en entornos grandes, no se tiene por qué explorar continuamente.
- La iteración de políticas aplicada a la política que extraemos del Q-Learning permite subsanar la absoluta incertidumbre, fruto de la estocasticidad del entorno y la variación de las variables anteriormente mencionadas y obtener políticas mejoradas.

-Es muy interesante saber el cómo el aprendizaje por refuerzo está sujeto a tantas variaciones y es influenciado por tantas variables, y ver cómo interaccionan entre sí a la hora de tenerlas en cuenta en un entorno real. También podemos observar que en entornos más pequeños la incertidumbre parece ser menor por razones obvias, además de apreciarse también la importancia de un código eficiente en entornos grandes.

3.- Bibliografía

- Base teórica: Apuntes de Sistemas Inteligentes – Campus Virtual – Universidad de Castilla La-Mancha.
- Microsoft Copilot (GPT-4): consultas teóricas sobre conceptos, limpieza de código, chequeo de eficiencia en algún que otro método poco eficiente.
- Consultas externas a través de correo-e con el profesorado de la asignatura.