# Algoritmo A\*

El algoritmo A\*...

costo total = heuristica + costo de camino

Se utilizó el módulo “headpq” el cual proporciona funciones que permiten trabajar con colas de prioridad, los cuales falicitarán el orden de los elementos en una lista según el costo total que tengan

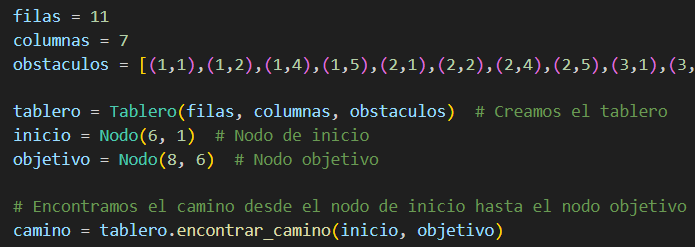
## Codigo

El algoritmo A\* se realizó con programacion orientada a objetos ya que la capacidad de crear clases y objetos facilita su implementación

Para el algoritmo se crearon dos clases con los siguientes atributos:

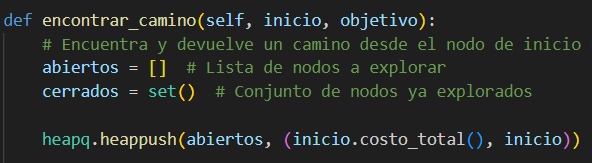
* **Nodo**
  + **x** → Posición X del nodo en el tablero
  + **y** → Posición Y del nodo en el tablero
  + **costo\_g** → Costo del camino desde el nodo inicial hasta este nodo
  + **costo\_h** → Heurística: Costo desde este nodo hasta el objetivo
  + **padre** → Nodo padre de este nodo en el camino
* **Tablero**
  + **filas →** Número de filas del tablero
  + **columnas** → Número de columnas del tablero
  + **obstaculos** → Lista de posiciones de obstáculos en el tablero

Al crear el objeto “*tablero*” se le asignan los valores de los atributos *filas*, *columnas* y *obstáculos* y luego se crean los objetos “*inicio*” y “*objetivo*”, los cuales indican la posicion inicial y final del recorrido a realizar .Posteriormente se llama al método *encontrar\_camino()* en el objeto “*tablero”* para encontrar un camino entre el nodo “inicio” y “objetivo” en el tablero y almacenar el resultado en la variable *camino* (*figura…*).



*figura…*

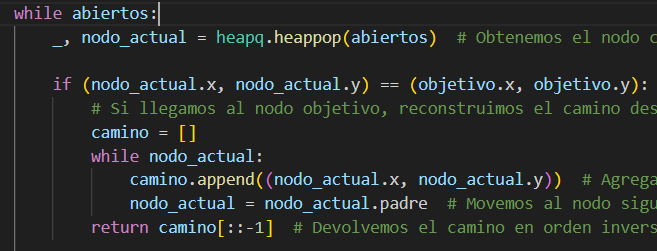
Dentro del método *encontrar\_camino()* se crea una lista llamada *abiertos* la cual almacenará los nodos a explorar y se crea un conjunto llamado *cerrados* que almacenará los nodos ya explorados. Cabe aclarar que la lista es una colección de elementos de forma ordenada los cuales se pueden acceder utilizando un índice y el conjunto es una colección de elementos únicos no ordenados, esto quiere decir, que no se pueden acceder mediante índices y no se repiten. Luego de crear la lista y el conjunto, se procede a almacenar el nodo “inicio” a la lista *abiertos* utilizando la función *heappush* del módulo que se mencionó anteriormente “*headpq*” el cual tiene como parámetro una tupla con el costo total calculado en la clase *Nodo* y el objeto nodo (*figura…*).



*figura…*

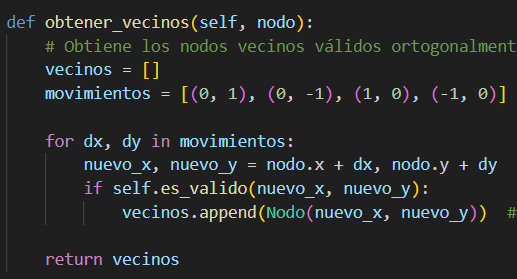
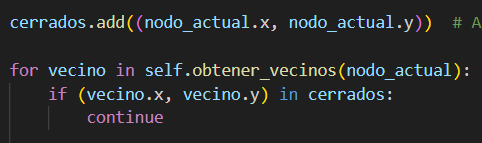
Mientras haya elementos en la lista *abiertos* se ejecutará el bucle *while*, en el cual se realizarán todos los cálculos necesarios para obtener el camino óptimo.

De la lista *abiertos* se obtiene el nodo el nodo con menor costo total y luego se analiza si es el nodo objetivo. Si ese es el caso, se define una lista la cual almacenará las posiciones X e Y de los nodos que generaron el camino optimo para asi poder devolverlos a la funcion principal de donde se llamo a la funcion *encontrar\_camino*. Para completar dicha lista, se analizan los nodos padres (*figura…*).



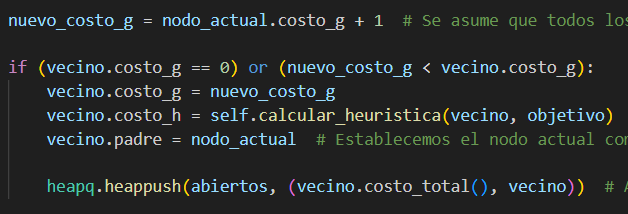
*figura…*

Si el nodo actual no es el objetivo, se procede a almacenarlo en el conjunto *cerrados*. Luego de este almacenamiento, se analizan los vecinos del conjunto (*figura*…) y se verifica antes de realizar cálculos si se puede avanzar o no dependiendo de las dimensiones del tablero y de los obstáculos (*figura…*).



*figura… figura…*

A una nueva variable llamada *nuevo\_costo\_g*, se le suma un valor de 1 ya que ese es el costo al avanzar una posición. Luego, si no se ha calculado el nodo vecino o el nodo vecino tiene un costo mayor a la variable *nuevo\_costo\_g*, entonces se procede a asignarle los atributos correspondientes a al nodo vecino (costo de ganancia y heuristica) y asignando el nodo analizado como el nodo padre de este vecino y finalmente se agrega este objeto a la lista *abiertos* (figura…).



*figura…*

Se procede a seguir analizando los vecinos y los nodos con menor costo hasta alcanzar el camino óptimo.