Trabajo Final – Inteligencia Artificial I

2022

Contents

[1. Resumen 3](#_Toc175600191)

[2. Introducción 3](#_Toc175600192)

[3. Estado del arte 3](#_Toc175600193)

[3.1. Algoritmos de búsqueda 3](#_Toc175600194)

[3.1.1. Algoritmo de Búsqueda Primero en Anchura (BPA) 3](#_Toc175600195)

[3.1.2. Algoritmo A\* 5](#_Toc175600196)

[3.1.3. Algoritmo A\* modificado 6](#_Toc175600197)

[3.2. Generación automática de laberintos 8](#_Toc175600198)

[3.2.1. Randomized Prim’s Algorithm 8](#_Toc175600199)

[4. Especificación del agente 8](#_Toc175600200)

[5. Diseño del agente 8](#_Toc175600201)

[5.1. Algoritmos de búsqueda 8](#_Toc175600202)

[5.2. Generación automática de laberintos 8](#_Toc175600203)

[6. Desarrollo de la solución 9](#_Toc175600204)

[7. Ejemplo de aplicación 9](#_Toc175600205)

[8. Resultados 9](#_Toc175600206)

[9. Conclusión 9](#_Toc175600207)

[10. Referencias bibliográficas 9](#_Toc175600208)

# Resumen

# Introducción

La búsqueda de caminos óptimos o *pathfinding* es un problema frecuente en múltiples dominios. Por ejemplo, en el sector de los videojuegos, principalmente aquellos videojuegos donde el entorno tiene forma de tablero, se suelen utilizar algoritmos de búsqueda de caminos para determinar el camino más corto para mover objetos de una posición inicial a una cierta posición final. Uno de los algoritmos de búsqueda de caminos es el algoritmo A-estrella (**A\***).

El algoritmo A\* utiliza una función de evaluación que toma en cuenta el valor heurístico del nodo a evaluar desde el actual, *n*, y el coste real del camino recorrido para llegar a dicho nodo *n*, desde el nodo inicial. El algoritmo es una combinación entre búsquedas del tipo primero en anchura con primero en profundidad.

En este trabajo, se desarrollará en Python un laberinto 2D basado en un tablero dividido en casillas. El usuario podrá seleccionar la posición inicial (punto de partida), final (objetivo) y podrá colocar barreras que servirán de barreras al algoritmo de búsqueda. El algoritmo A\* será utilizado para encontrar el camino óptimo (si existe) que permita trasladarse desde la posición inicial hasta la posición final con el menor costo. A su vez, se planea desarrollar también un algoritmo de generación aleatoria de laberintos.

# Estado del arte

## Algoritmos de búsqueda

Un algoritmo de búsqueda es un conjunto sistemático de procedimientos diseñado para encontrar un elemento o un conjunto de elemento dentro de una estructura de datos como puede ser un grafo o un árbol, a la cual se la denomina **espacio de estados**. Existen diversos algoritmos de búsqueda, cada uno adecuado para diferentes situaciones (tamaño del conjunto de datos a explorar, requerimientos del problema, entre otros).

Los algoritmos de búsqueda pueden clasificarse en **informados** o **no informados**. Cuando el sistema agente posee algún tipo de información del medio se habla de técnicas de búsquedas informadas; sin embargo, si carece de conocimiento alguno, llevan el nombre de algoritmos de búsqueda no informadas. Los algoritmos de búsqueda no informados suelen ser más lentos y costosos en memoria. Entre los algoritmos de búsqueda más populares se encuentra el algoritmo A\* (A-estrella) y el algoritmo búsqueda primero en anchura.

Numerosos estudios tratan sobre la utilización de algoritmos de búsqueda para la búsqueda de caminos óptimos en el contexto de un laberinto 2D. El articulo realizado por Ade Candra et al. analiza la utilización directa del algoritmo A\* en este contexto, y muestra su lógica de funcionamiento en diagrama de bloques y su implementación en pseudocódigo, lo cual facilita su desarrollo en un entorno real de programación como Python.

### Algoritmo de Búsqueda Primero en Anchura (BPA)

El algoritmo de búsqueda primero en anchura (BPA), es un algoritmo de búsqueda no informado cuyo principio de funcionamiento consiste en ir explorando los nodos a medida que parten del origen. La búsqueda comienza por el nodo raíz y visita sucesivamente los nodos más cercanos a la raíz hasta que el nodo destino es alcanzado, o hasta que no queda ningún nodo por explorar, lo que ocurra primero. La función de evaluación de este algoritmo está dada por:

donde n representa el nodo a evaluar y g(n) representa el costo total desde el nodo de inicio hasta *n*, en este caso, g(n) denota el número de nodos que hace falta visitar hasta llegar a n a partir del nodo de inicio.

Durante el proceso de exploración del entorno, el algoritmo comienza analizando el nodo raíz y añade a una cola de prioridad a todos los nodos vecinos. En la cola, cada nodo se asocia con el valor dado por la función de evaluación (costo total desde la raíz en este caso), donde se les da mayor prioridad a los nodos de menor valor. De esta forma, puesto que la función de evaluación implica en definitiva la distancia del nodo *n* al nodo de inicio, BPA explora primero el nodo de la cola que se encuentre más cercano al nodo raíz.

Una vez comprobado que el nodo analizado no es el nodo destino, se añaden sus nodos vecinos a la cola de prioridady se reordena la cola en función de los valores de *f(n)*. El proceso se repite hasta que el nodo analizado corresponde al nodo destino, o hasta que no quedan nodos por analizar. El camino obtenido contiene el mínimo número de nodos, es decir, es el camino más corto medido en número de nodos.

La complejidad de tiempo del algoritmo es O(|N|+|A|) donde |N| es el número de nodos transitables del entorno y |A| es el número de aristas de cada nodo. El razonamiento es porque en el peor caso, cada vértice y cada arista será visitado por el algoritmo.

Una vez el algoritmo halla un camino, el mismo puede ser reconstruido a partir de la lista de nodos padres. En este sentido, se crea una lista *Camino* y se parte del nodo objetivo *o*, el nodo padre es obtenido a partir de padre[o], luego se obtiene el padre de este, y así sucesivamente hasta llegar al nodo de inicio s, luego, la lista es invertida para representar correctamente el orden de tránsito de los nodos.

#### Procedimiento algorítmico

El algoritmo de búsqueda puede ser descrito a través del siguiente pseudocódigo:

*BPA*(Grilla G, nodo\_fuente s, nodo\_objetivo o) {

**for** u ∈ V[G] **do**

{

distancia[u] = INFINITO;

padre[u] = NULL;

}

CrearCola(Q);

distancia[s] = 0;

Encolar(Q, s);

**while** !vacía(Q) **do**

{

u = extraer(Q);

**if** u == o **then**

{

break;

}

**for** v ∈ adyacencia[u] **and** v == TRANSITABLE **do**

{

**if** distancia[u]+1 < distancia[v] **then**

{

padre[v] = u

distancia[v] = distancia[u]+1

Encolar(Q, v);

}

}

}

}

### Algoritmo A\*

A diferencia del algoritmo BPA, el algoritmo A\* es un algoritmo de búsqueda informado que toma en cuenta una función heurística *h(n)* para el cálculo de la función de evaluación, además de la función de costo *g(n)*. La idea detrás de la función heurística, es que representa una estimación del costo del camino que existe entre el nodo *n* a evaluar y el nodo final. La función de evaluación del nodo *n* puede ser escrita en este caso como:

donde*,* en este caso, la distancia de Manhattan ha sido considerada como función heurística *h(n)*. Esta heurística suele ser bastante efectiva en grillas, ya que proporciona una estimación precisa del costo restante al objetivo sin sobreestimarlo.

La distancia de Manhattan representa la suma de la diferencia absoluta de las coordenadas de dos puntos A y B. En el desarrollo de este proyecto, el número de fila y de columna de la casilla a evaluar fueron considerados como coordenadas para el cálculo de la distancia de Manhattan. Así, tenemos que:

El proceso de exploración es exactamente igual que el descrito para el algoritmo BPA, con la única diferencia de que la función de evaluación *f(n)* ahora también toma en cuenta la función heurística *h(n)*.

Una vez el algoritmo halla un camino, el mismo puede ser reconstruido a partir de la lista de nodos padres de la misma forma que para el algoritmo BPA.

#### Procedimiento algorítmico

*A-estrella*(Grilla G, nodo\_fuente s, nodo\_objetivo o) {

**for** u ∈ V[G] **do**

{

distancia[u] = INFINITO;

padre[u] = NULL;

}

CrearCola(Q);

distancia[s] = 0;

heuristica[s] = DistanciaManhattan(s, o);

func\_eval[s] = distancia[s] + heuristica[s];

Encolar(Q, s);

**while** !vacía(Q) **do**

{

u = extraer(Q);

**if** u == o **then**

{

break;

}

**for** v ∈ adyacencia[u] **and** v == TRANSITABLE **do**

{

**if** distancia[u]+1 < distancia[v] **then**

{

padre[v] = u

distancia[v] = distancia[u]+1

heurística[v] = DistanciaManhattan(v, o);

func\_eval[v] = distancia[v] + heuristica[v];

Encolar(Q, v);

}

}

}

}

### Algoritmo A\* modificado

El artículo de Xiang Liu et al. propone modificaciones al algoritmo A\*, principalmente a su función heurística, y compara la calidad de las soluciones encontradas para laberintos “perfectos”, es decir, sin bucles ni regiones inaccesibles.

La función de evaluación propuesta por los autores toma la forma:

donde *n* es el nodo a evaluar y *j* es el nodo padre de *n*. La función *g(n)* sigue representando el costo del camino desde el nodo raíz hasta *n*, y *h(n)* es la distancia de Manhattan entre *n* y el nodo objetivo. En dicho artículo se muestra que la inserción de este término mejora la velocidad de convergencia ya que reduce considerablemente el número de nodos explorados.

La metodología de exploración y de reconstrucción de la solución es idéntica a la de los algoritmos anteriores.

#### Procedimiento algorítmico

La lógica detrás del algoritmo A\* modificado es muy similar a la de los anteriores. Esta se ilustra mediante el siguiente pseudocódigo:

*A-estrella-modificado*(Grilla G, nodo\_fuente s, nodo\_objetivo o) {

**for** u ∈ V[G] **do**

{

distancia[u] = INFINITO;

padre[u] = NULL;

}

CrearCola(Q);

distancia[s] = 0;

heuristica[s] = DistanciaManhattan(s, o);

func\_eval[s] = distancia[s] + heuristica[s];

Encolar(Q, s);

**while** !vacía(Q) **do**

{

u = extraer(Q);

**if** u == o **then**

{

break;

}

**for** v ∈ adyacencia[u] **and** v == TRANSITABLE **do**

{

**if** distancia[u]+1 < distancia[v] **then**

{

padre[v] = u

distancia[v] = distancia[u]+1

heurística[v] = DistanciaManhattan(v, o);

func\_eval[v] = distancia[v] + heuristica[v] + heurística[padre[v]];

Encolar(Q, v);

}

}

}

}

## Generación automática de laberintos

En cuanto a la generación automática de laberintos, el artículo de Peter Gabrovšek analiza comparativamente seis métodos de generación de laberintos. El autor propone inspeccionar el número de caminos sin salida, así como también, el número de casillas visitadas por diferentes agentes cómo método de caracterización de la dificultad del laberinto obtenido.

### Randomized Prim’s Algorithm

# Especificación del agente

# Diseño del agente

En esta sección se discute el principio de funcionamiento de los diferentes algoritmos implementados a lo largo del desarrollo del proyecto.

## Algoritmos de búsqueda

En el presente proyecto, tres algoritmos de búsqueda diferentes son analizados. Cabe destacar que para cada algoritmo el entorno es una grilla de 40x40 casillas. El algoritmo deberá partir de una casilla de inicio **(*nodo raíz)***hasta hallar la casilla final u objetivo ***(nodo destino)***. La metodología de exploración del entorno es lo que diferencia un algoritmo de otro.

## Generación automática de laberintos

Los algoritmos de generación de laberintos inspeccionados comienzan con una grilla rectangular que representa al propio laberinto. Generalmente, todas las casillas son consideradas como paredes y los algoritmos convierten paredes específicas en casillas transitables para formar el laberinto [1]. Otros algoritmos trabajan a la inversa, es decir, consideran inicialmente todas las casillas como transitables y luego van añadiendo paredes progresivamente. Algunos algoritmos incluso son capaces de trabajar de las dos formas. En general, los laberintos pueden tener bucles, pero en este caso nos concentraremos en aquellos laberintos simples.

En nuestro caso, consideraremos al laberinto como una *grilla* compuesta por *casillas* Las casillas están conectadas entre sí por medio de *vías*. El laberinto puede ser representado matemáticamente por medio de un *grafo*. Esta representación es intuitiva y permite escalar los algoritmos de generación a laberintos de mayor tamaño [2].

Se define **laberinto ideal** a aquel que respeta las siguientes restricciones:

1. El laberinto no posee bucles.
2. El laberinto no posee zonas inaccesibles.
3. El costo de moverse entre casillas adyacentes es siempre igual a 1.

Este algoritmo puede trabajar añadiendo paredes o casillas transitables a la estructura inicial.

# Desarrollo de la solución

# Ejemplo de aplicación

# Resultados

# Conclusión

# Referencias bibliográficas

[1] Gabrovšek, P. (2019). Analysis of maze generating algorithms. IPSI Transactions on Internet Research, 15(1), 23-30.

[2] Zhang Renping, Zhou Qingzhong (2009). Updated A\* Algorithm and its Application, Computer Systems & Applications, 98-100.

Stuart Russell; Peter Norvig (2010). Artificial Intelligence: A Modern Approach (3 edición). Prentice Hall.