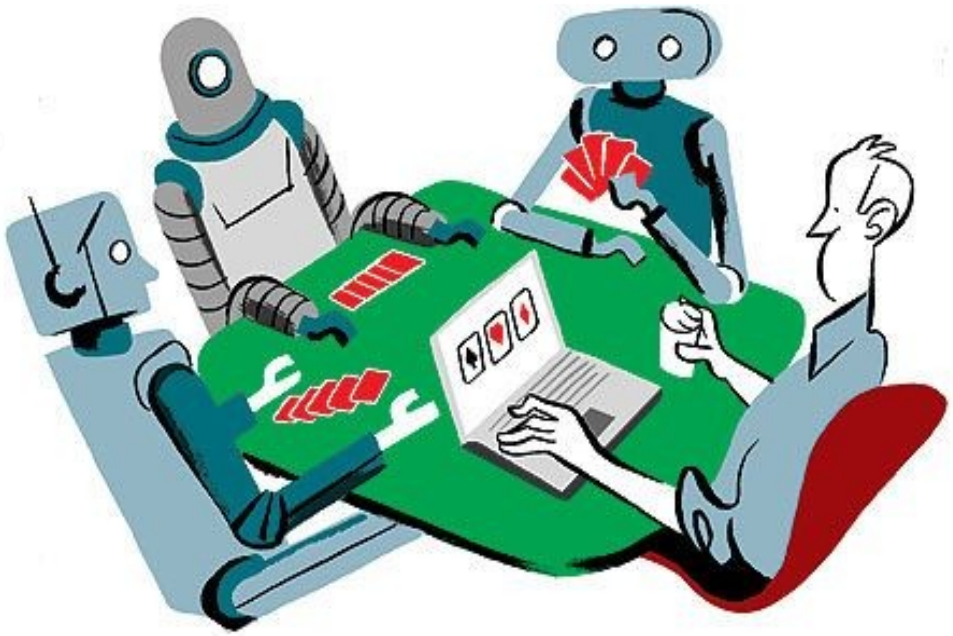
PRACTICA 1

TECNICAS DE BUSQUEDA

SISTEMAS INTELIGENTES



JOSE ÁNGEL PARDO CERDÁN

Índice

Índice de tablas

Índice de figuras

1. Formulación del problema

En este problema se asigna como agente el balón de fútbol, que ha de llegar a través de una cuadrícula, y esquivando los obstáculos, a la posición de portería (estado final) desde una posición o estado de inicio.

Nos enfrentamos a un problema en el que un agente (en nuestro caso una pelota de fútbol) tiene que encontrar mediante diversas estrategias de búsqueda (tanto informada como no informada) el camino hacia el gol.

Se nos ha proporcionado una serie de paquetes con sus respectivas clases en Java para realizar esta implementación. Nuestra tarea ha sido realizar 8 agentes que se encuentran dentro del paquete "Search", para lo que previamente hemos tenido que definir nuestro problema:

Estado: representa el estado en el que nos encontramos dentro del campo en un momento concreto.

Estado inicial: es el estado en el que comienza el agente. Puede ser cualquier recuadro o posición dentro de la cuadrícula, siempre y cuando no esté ocupado por un obstáculo. El estado inicial no debe estar rodeado de obstáculos, es decir, el agente se debe poder mover a otra posición.

Estado final: es uno/s estado/s en la que se encuentra nuestra portería.

Acción sucesor: Los movimientos disponibles del agente son a cualquier posición lindante con la posición actual, que se hallarán a partir de los ángulos con los que estemos jugando. Estas posiciones las podemos llamar sucesores y se darán en forma de pares (estamos en una cuadrícula).

Test objetivo: determina si un estado es objetivo o no. Es nuestro problema, un estado será objetivo si podemos llegar a él desde el estado donde se encuentra el agente.

Coste de un camino: en esta acción se le asigna un coste numérico a cada camino. Es una medida de rendimiento basada habitualmente en una función de coste individual.

Camino: lista de enteros que le dirán al agente qué acción de entre las posibles tomará. Esta lista tomará valores entre 0 y numAngles-1 en todas sus posiciones. Este camino será la solución que encuentre nuestro agente para llegar desde el estado inicial al estado final.

Solución: la solución es el camino más óptimo, según el algoritmo de búsqueda empleado, que va desde el estado inicial al final.

1.1 Ejercicio Propuesto

Encontrar el camino para que el balón llegue a la casilla blanca

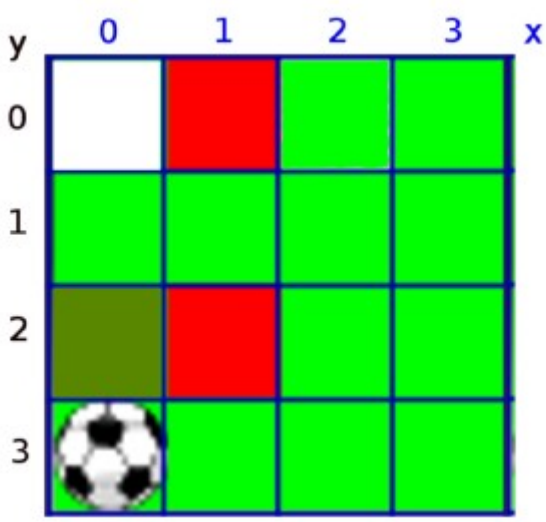


Ilustración 1: Ejemplo para hacer el árbol de búsqueda a mano

1.1.1 Grafo de búsqueda por coste uniforme (hoyos = 1)

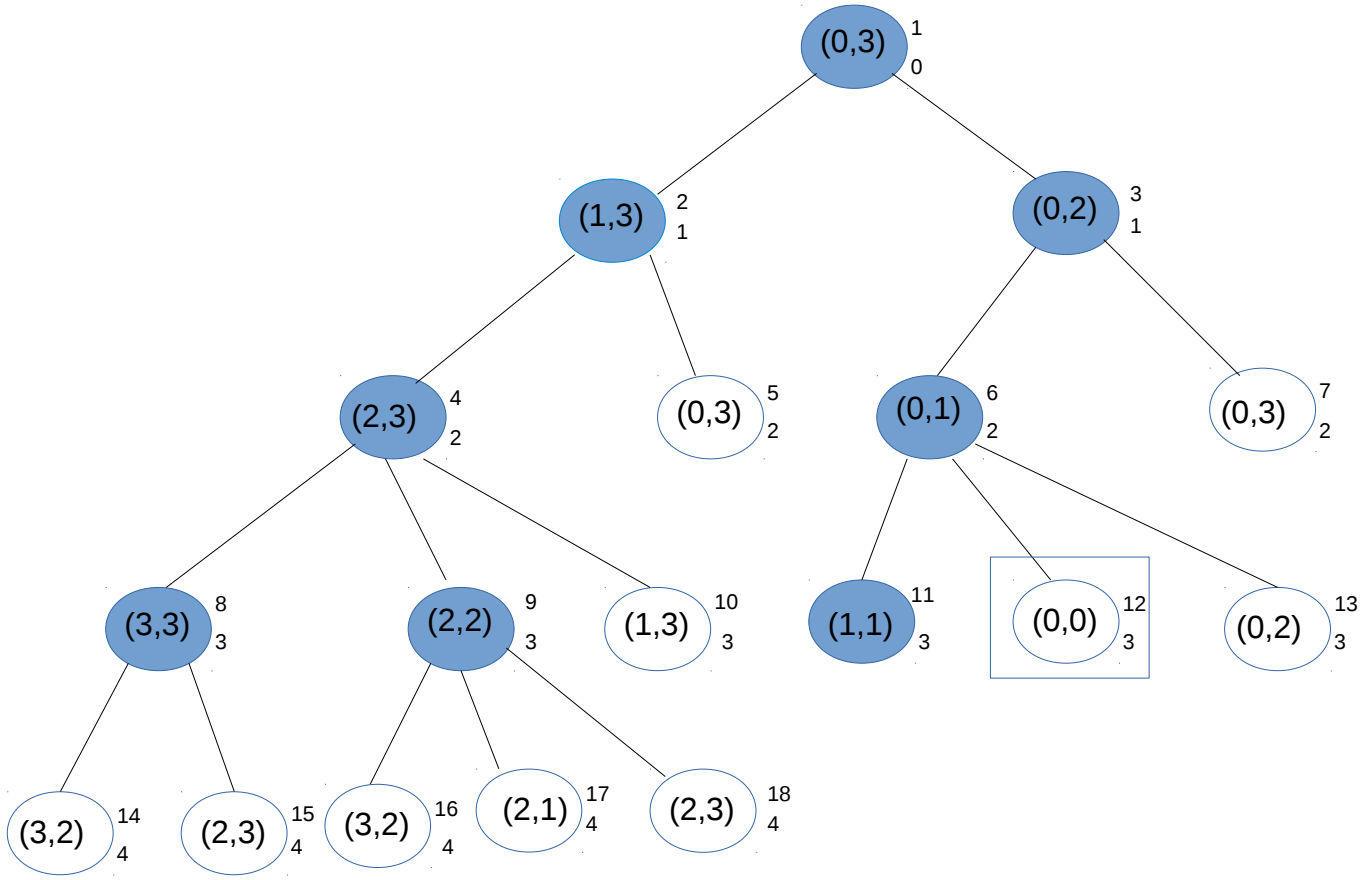


Ilustración 2: Grafo de búsqueda por coste uniforme del ejemplo propuesto (hoyos = 1)

Camino solución: (0,3), (0,2), (0,1), (0,0)

Coste: 3

Profundidad: 3

Nodos generados: 20

Nodos expandidos: 8

Nodos explorados: 9

1.1.2 Grafo de búsqueda por coste uniforme (hoyos = 10)

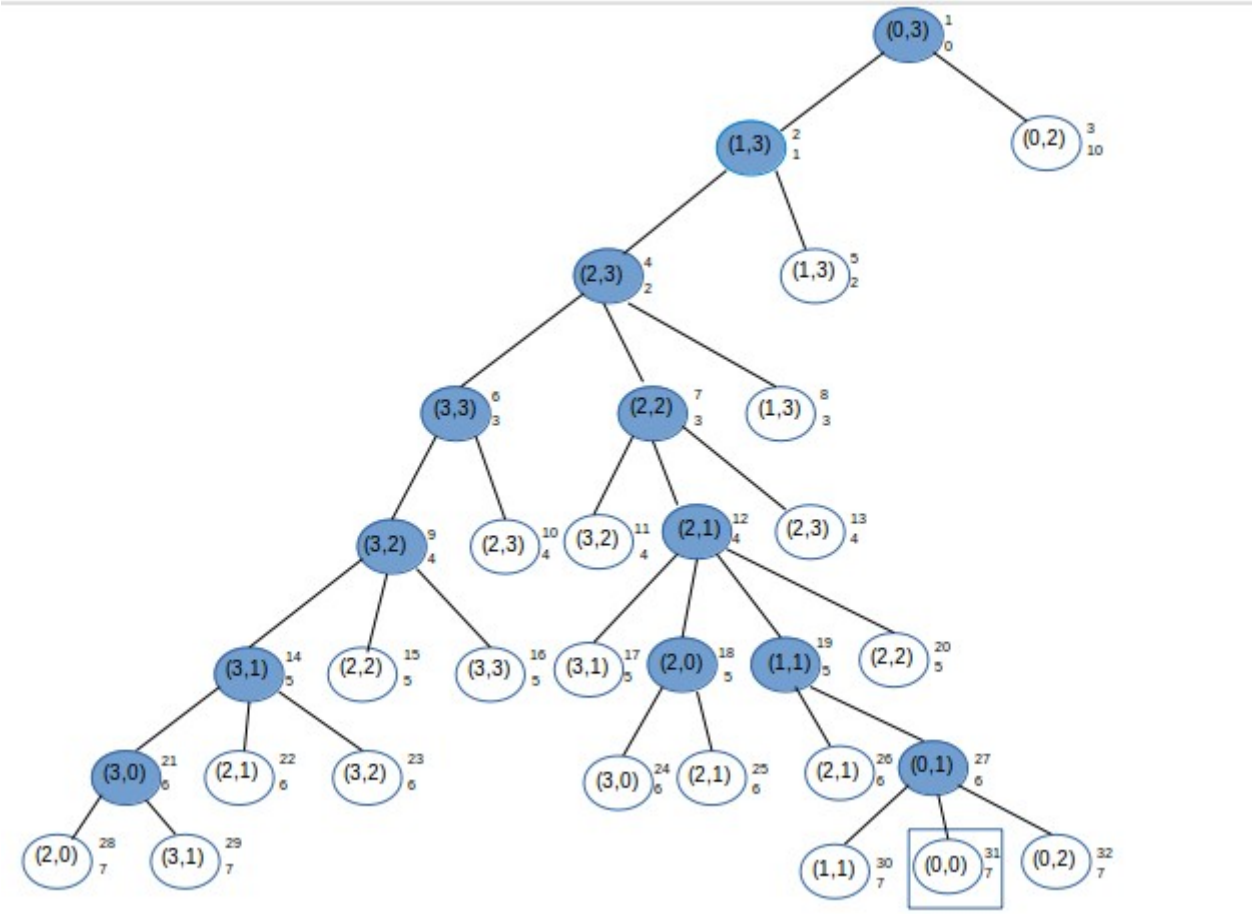


Ilustración 3: Grafo de búsqueda por coste uniforme del ejemplo propuesto (hoyos = 10)

Camino solución: (0,3), (1,3), (2,3), (2,2), (2,1), (1,1), (0,1), (0,0)

Coste: 7

Profundidad: 7

Nodos generados: 32

Nodos expandidos: 12

Nodos explorados: 13

2. Agentes implementados usando estrategias de búsqueda no informada

2.1 Búsqueda en anchura

En este algoritmo lo que hacemos es ir cogiendo el primer nodo que esté en la frontera y así asegurarnos que exploraremos todos los nodos del mismo nivel antes de pasa al siguiente. Tal como nos muestra el pseudocódigo de la ilustración 3.

```
BFS(grafo G, nodo_fuente s)
{
   // recorremos todos los vértices del grafo inicializándolos a NO_VISITADO,
   // distancia INFINITA y padre de cada nodo NULL
   for u ∈ V[G] do
   {
      estado[u] = NO_VISITADO;
      distancia[u] = INFINITO; /* distancia infinita si el nodo no es alcanzable */
      padre[u] = NULL;
   }
   estado[s] = VISITADO;
   distancia[s] = 0;
   Encolar(Q, s);
   while !vacia(Q) do
   {
      // extraemos el nodo u de la cola Q y exploramos todos sus nodos adyacentes
      u = extraer(Q);
      for  v ∈ adyacencia[u]  do
      {
         if estado[v] == NO_VISITADO then
         {
              estado[v] = VISITADO;
              distancia[v] = distancia[u] + 1;
              padre[v] = u;
              Encolar(Q, v);
         }
      }
   }
}
```

Ilustración 4: Pseudocódigo búsqueda en anchura

2.2 Búsqueda en profundidad

En este algoritmo seleccionamos el primero de los hijos que generó el último nodo explorado (los introducimos en el array de frontera en el orden inverso del que se generaron para poder hacerlo). Tal como nos muestra el pseudocódigo de la ilustración 4.

```
DFS(grafo G)
   PARA CADA vertice u ∈ V[G] HACER
          estado[u] ← NO_VISITADO
          padre[u] ← NULO
   tiempo ← 0
   PARA CADA vertice u ∈ V[G] HACER
          SI estado[u] = NO_VISITADO ENTONCES
                 DFS_Visitar(u,tiempo)
```

```
DFS_Visitar(nodo u, int tiempo)
   estado[u] ← VISITADO
   tiempo ← tiempo + 1
   d[u] ← tiempo
   PARA CADA v ∈ Vecinos[u] HACER
          SI estado[v] = NO_VISITADO ENTONCES
                 padre[v] ← u
                 DFS_Visitar(v,tiempo)
   estado[u] ← TERMINADO
   tiempo ← tiempo + 1
   f[u] ← tiempo
```

Ilustración 5: Pseudocódigo búsqueda en profundidad

2.3 Búsqueda con coste uniforme

En el algoritmo se almacena cada nodo con su valor g y se insertan los nuevos nodos expandidos en una estructura dinámica (una cola con prioridad) en orden ascendente según su valor de g; la búsqueda de coste uniforme es completa al ser los costes números enteros positivos, siendo no acotada la sucesión de valores g, por lo que si el nodo meta existe en el espacio de estados, será expandido alguna vez; y óptima, puesto que al expandir todos los nodos del espacio de estados por valores crecientes de g, cuando se expande el primer nodo meta, éste será el nodo meta de menor valor de g.

```
{búsqueda de coste uniforme}
abierta ←s0
Repetir Si
vacío?(abierta) entonces
devolver(negativo)
nodo←primero(abierta)
Si meta?(nodo)entonces
devolver(nodo)
sucesores←expandir(nodo)
Para cada n∈sucesores hacer
n.padre←nodoordInsertar(n,abierta,g)
Fin{repetir}
```

Ilustración 6: Pseudocódigo Búsqueda por coste uniforme

3. Agentes implementados usando estrategias de búsqueda informada

3.1 Búsqueda primero el mejor

La búsqueda primero el mejor es un caso en el cual se seleccionar un nodo para la expansión basada en una función de evaluación f(n).

Se expande primero, aquel nodo que tiene mejor evaluación y se escoge el que parece ser el mejor.

```
void construirGrafo(){
	Lista abiertos,visitados;
	Nodo actual=Nodo con Pos. Inicial;
	abiertos.introduce(actual); //mete el nodo inicial a la lista

	mientras(abiertos no vacío y no acabar){
		actual=cogeMejorHeuristica(abiertos);
		visiatos.introduce(actual);
		Si (Actual está en GOL) {
			hacerRecorrido(actual);
			acabar=true;
		} si no {
			expandirYOrdenar(actual, abiertos);
			mientras(abiertos.cabeza contenida en visitados){
				abiertos.borrarCabeza;
			}
		}
	}
}
```

```
void expandirYOrdenar(nodo, lista){
	Cola hijos;
	para cada posible ángulo{
		posición = simular movimiento
		si (posición es válida) {
			metemos la posición en hijos, indicando su padre,
			movimiento, profundidad y heurística
		}
	}

	Mientras(hijos no vacío){
		Nodo = cabeza de hijos;
		Desde 0 a lista.longitud hacer{
			Si(nodo.heuristica<lista.coger(indice).heuristica){
				Lista.introduce(nodo) ;
				Índice = lista.longitud;
			}
		}
	}
}
```

Ilustración 7: Pseudocódigo búsqueda primero el mejor

3.2 A* (A-estrella)

El algoritmo A* utiliza una función de evaluación f(n)=g(n)+h′(n), donde h′(n) representa el valor heurístico del nodo a evaluar desde el actual, n, hasta el final, y g(n), el coste real del camino recorrido para llegar a dicho nodo, n, desde el nodo inicial. A* mantiene dos estructuras de datos auxiliares, que podemos denominar abiertos, implementado como una cola de prioridad (ordenada por el valor f(n) de cada nodo), y cerrados, donde se guarda la información de los nodos que ya han sido visitados. En cada paso del algoritmo, se expande el nodo que esté primero en abiertos, y en caso de que no sea un nodo objetivo, calcula la f(n) de todos sus hijos, los inserta en abiertos, y pasa el nodo evaluado a cerrados.

```
construirGrafo(){
        Lista lista, visitados;
        Nodo actual(PosInicial);
        lista.añade(actual);
        mientras (la lista no esté vacía y no hayamos acabado) hacer
                actual=cogerMejorHeuristicaDeLaLista(lista);
                Si (actual está en GOL) hacer
                hacer recorrido;
                finmientras;
        finSI;
        Sino hacer
                visitados.añade(actual);
                ExpandirYOrdenar (actual,lista);
        finSiNO;
        eliminarPrimeroSiEstaEnVisitados(lista);
finmientras;
}
```

Ilustración 8: Pseudocódigo A parte1*

```
expandirYOrdenar(Nodo,lista){
        Position iPosition;
        Qeue hijos;
        j=0;
        desde i=0 hasta i<possibleAngles hacer
                iPosition=simularMovimientos(posInicial,posibleAngles[i]);
                Si (iPosition valido) hacer
                        Nodo aux(iPosition);
                        aux.asignarPadre;
                        aux.asignarMovimiento;
                        aux.asignarProfundidad;
                        aux.asignarHeuristica;
                        aux.asignarDistanciaRealAcumulada;
                        aux.asignarCosteAcumulado;

                        hijos.añadir(aux);
                finHacer;
                mientras (hijos no sea vacia)
                        aux=hijos.sacaPrimerHijo;
                        recorrer lista desde i=0 hasta su tamaño y hacer:
                                Si aux.distanciaRealAcumulada<lista.distanciaRealAcumulada
                                        Si aux.profunidad<=lista.profundidad
                                                lista.añade(i,aux)
                                                i=tamañoLista;
                                        SiNo
                                                insertarEnLaListaTeniendoEncuentaSuProfundidadYHeuristica
                                                i=tamañoLista;
                                        finSiNO;
                                        SiNO si (llegamos al final de la lista)
                                                lista.añade(i+1,aux)
                                                i=lista.tamaño;
                        finHacer;
                finmientras;
}
```

Ilustración 9: Pseudocódigo A parte2*

4. Estudio del comportamiento de los algoritmos

4.1 Cuatro ángulos

4.1.1 Tamaño 4x4

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	4	0,022	8	16	23	24
Búsqueda Profundidad	3	0,02	9	20	26	27
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	4	0,029	5	8	11	12
Búsqueda A-Estrella	3	0,011	9	18	27	28

Tabla 1: Cuatro ángulos tamaño 4x4

4.1.2 Tamaño 8x8

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	11	0,066	42	105	114	115
Búsqueda Profundidad	7	0,01	43	129	152	153
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	9	0,032	48	144	158	159
Búsqueda A-Estrella	11	0,011	49	137	150	151

Tabla 2: Cuatro ángulos tamaño 8x8

4.1.3 Tamaño 16x8

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	7	0,008	56	159	209	210
Búsqueda Profundidad	8	0,023	57	132	177	178
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	14	0,058	63	176	189	190
Búsqueda A-Estrella	14	0,088	86	274	302	303

Tabla 3: Cuatro ángulos tamaño 16x8

4.1.4 Tamaño 32x16

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	28	0,17	315	1051	1112	1113
Búsqueda Profundidad	28	0,084	139	405	406	407
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	29	0,124	195	507	532	533
Búsqueda A-Estrella	28	0,18	320	1085	1142	1143

Tabla 4: Cuatro ángulos tamaño 32x16

4.1.5 Tamaño 32x32

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	35	0,126	677	2103	2165	2166
Búsqueda Profundidad	46	0,178	715	2236	2291	2292
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	44	0,143	814	2959	3024	3025
Búsqueda A-Estrella	54	0,084	563	1531	1543	1544

Tabla 5: Cuatro ángulos tamaño 32x32

4.1.6 Tamaño 40x40

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	55	0,081	899	2650	2722	2723
Búsqueda Profundidad	55	0,273	1216	4295	4383	4384
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	58	0,192	1094	3381	3451	3452
Búsqueda A-Estrella	55	0,227	1221	4417	4504	4505

Tabla 6: Cuatro ángulos tamaño 40x40

4.2 Seis ángulos

4.2.1 Tamaño 4x4

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	4	0,021	10	30	33	34
Búsqueda Profundidad	3	0,031	12	34	49	50
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	3	0,003	6	11	17	18
Búsqueda A-Estrella	3	0,01	6	11	16	17

Tabla 7: Seis ángulos tamaño 4x4

4.2.2 Tamaño 4x8

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	8	0,013	45	184	214	215
Búsqueda Profundidad	10	0,009	33	95	111	112
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	10	0,021	36	124	155	156
Búsqueda A-Estrella	8	0,042	42	145	188	189

Tabla 8: Seis ángulos tamaño 4x8

4.2.3 Tamaño 16x8

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	19	0,156	243	1264	1359	1360
Búsqueda Profundidad	10	0,125	48	178	211	212
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	8	0,082	41	132	171	172
Búsqueda A-Estrella	51	0,117	963	4966	5277	5278

Tabla 9: Seis ángulos tamaño 16x8

4.2.4 Tamaño 32x16

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	21	0,2	309	1651	1732	1733
Búsqueda Profundidad	19	0,034	127	466	513	514
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	39	0,221	329	1573	1586	1587
Búsqueda A-Estrella	50	0,177	508	2335	2380	2381

Tabla 10: Seis ángulos tamaño 32x16

4.2.5 Tamaño 32x32

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	165	0,638	4277	22963	22964	22965
Búsqueda Profundidad	151	4,747	10652	59167	60045	60046
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	194	3,189	7451	40138	40374	40375
Búsqueda A-Estrella	71	0,892	3608	20596	21109	21110

Tabla 11: Seis ángulos tamaño 32x32

4.2.6 Tamaño 40x40

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	89	1,285	5755	31708	32065	32066
Búsqueda Profundidad	61	0,391	1059	4604	4785	4786
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	75	0,57	573	2193	2239	2240
Búsqueda A-Estrella	230	4,76	14246	78226	78696	78697

Tabla 12: Seis ángulos tamaño 40x40

4.3 Ocho ángulos

4.3.1 Tamaño 4x4

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	3	0,006	14	21	64	65
Búsqueda Profundidad	3	0,017	45	99	247	248
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	4	0,017	8	14	18	19
Búsqueda A-Estrella	4	0,008	25	47	95	96

Tabla 13: Ocho ángulos tamaño 4x4

4.3.2 Tamaño 8x8

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	8	0,391	829	3442	5742	5743
Búsqueda Profundidad	8	0,191	173	401	735	736
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	10	0,816	831	3336	4576	4577
Búsqueda A-Estrella	9	0,048	114	250	329	330

Tabla 14: Ocho ángulos tamaño 8x8

4.3.3 Tamaño 16x8

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	8	0,397	921	3454	5804	5805
Búsqueda Profundidad	13	0,809	1441	5399	7165	7166
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	12	0,305	451	1176	15552	15553
Búsqueda A-Estrella	10	0,026	114	347	390	391

Tabla 15: Ocho ángulos tamaño 16x8

4.3.4 Tamaño 32x16

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	21	18,352	25766	142308	171665	171666
Búsqueda Profundidad	15	6,968	13256	64812	84594	84595
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	28	3,805	11277	38870	43085	43086
Búsqueda A-Estrella	21	2,989	22550	127048	154076	154077

Tabla 16: Ocho ángulos tamaño 32x16

4.3.5 Tamaño 32x32

Algoritmos	Interacciones	Tiempo	Expandidos	Explorados	Generados	Simulados
Búsqueda Anchura	No goal	-	-	-	-	-
Búsqueda Profundidad	40	227,72	73510	298509	327451	327452
Búsqueda Coste Uniforme						
Búsqueda Primero Mejor	37	877,353	120430	600737	667485	667486
Búsqueda A-Estrella	No goal	-	-	-	-	-

Tabla 17: Ocho ángulos tamaño 32x32

5. Conclusiones

Podemos observar que, respecto a los datos obtenidos, cuando calculamos los distintos caminos con ángulo = 4 se obtienen mejores resultados que en los distintos ángulos, pues expande menos nodos por lo que mejora el tiempo de búsqueda.

También observamos que el mejor de los algoritmos implementados para tamaños muy grandes es el de A-Estrella y el peor en profundidad, aunque para tamaños relativamente pequeños el resultado entre los distintos algoritmos no varia mucho.

En cuanto al tiempo de ejecución podemos deducir que a mayor numero de nodos simulados el tiempo de búsqueda sera mas elevado llegando incluso a no encontrar el camino correcto hacia la solución (No Goal).

6. Bibliografía

http://misiniciosenjava.

http://es.slideshare.net/

http://www.dma.fi.upm.es/java/

http://es.wikipedia.org