

Razón Artificial

La ciencia y el arte de crear videojuegos

- [Blog](#)
- [Artículos](#)
 - [Matemáticas y Física](#)
 - [Programación y Algoritmos](#)
 - [Desarrollo de videojuegos](#)
 - [Inteligencia Artificial](#)
 - [Recursos](#)
 - [Todos los artículos](#)
- [Proyectos](#)
 - [Generic Game Engine](#)
- [Sobre mí](#)
- [Contacto](#)

El salto del caballo, backtracking

Escrito por adrigm el 8 de enero de 2010 en [Inteligencia Artificial](#), [Programación](#) | [10 Comentarios](#).




El salto del caballo es un viejo problema que existe mucho antes que los ordenadores. Consiste en, dada una posición inicial, en un tablero de ajedrez recorrer todas las casillas del tablero únicamente con los movimientos del caballo sin repetir ninguna casilla.

Este problema antiguamente se resolvía “a mano” probando posibles soluciones y anotando las que eran erróneas, es decir método prueba-error. Con la llegada de los ordenadores se solucionó mediante técnicas de inteligencia artificial. Usando algoritmos de búsqueda de caminos.

1	16	11	6	3
10	5	2	17	12
15	22	19	4	7
20	9	24	13	18
23	14	21	8	25

He creado una versión en Python de este problema típico de IA. Yo no se usar de momento los árboles en Python que sería lo ideal para resolver este problema así que lo he hecho de la manera que se me ha ido ocurriendo.

Partiendo de la idea de que el caballo puede mover como máximo a 8 posiciones diferentes el algoritmo consiste en pasar a una función los posibles movimientos del caballo descartar los que estén fuera del tablero y las casillas ya visitadas y probar con el resto. Si una posición se queda sin posibles siguientes movimientos, volver a la anterior posición y probar la siguiente posible. Lo he implementado con una función recursiva, se llama así misma.

	3		2	
4				1
				
5				8
	6		7	

Heurística

En cuanto a la heurística, hay un método que dice que se debe visitar primero las casillas con menos movimientos posibles. Así que he hecho una función que analiza los posibles movimientos de los vecinos y ordenada la lista de movimientos de menor a mayor.

El juego es de un tablero de 8×8 , es decir 64 posiciones diferentes, que son las de un tablero de ajedrez, aunque se puede ajustar el juego para que halle la solución de un tablero de $n \times n$

Bueno dejo el código después del salto.

```
001  # -*- coding: utf-8 -*-
002
003  #####
004  # Salto del Caballo
005  # Versión: 0.2
006  # Autor: Adrigm
```

?

```
007 # Email: adrigm.admin@gmail.com
008 # Web: www.adrigm.es
009 # Licencia: GPL
010 #####
011
012 # Bibliotecas.
013 import random
014 import sys
015
016 # Funciones.
017 # -----
018
019 # Dibuja el tablero.
020 def dibujar(tablero):
021     tab = """
022     for f in range(8):
023         for c in range(8):
024             if tablero[f][c] < 10:
025                 celda = "" + str(tablero[f][c])
026             else:
027                 celda = str(tablero[f][c])
028             tab = tab + celda + " "
029         tab = tab + "\n"
030     print tab
031
032 # Devuelve una lista con las coordenadas del caballo.
033 def pos_c(tablero):
034     for f in range(8):
035         for c in range(8):
036             if tablero[f][c] == "CC":
037                 pos = [f, c]
038                 return pos
039
040 # Devuelve 1 si el tablero está lleno.
041 def lleno(tablero):
042     for f in range(8):
043         for c in range(8):
044             if tablero[f][c] == "__":
045                 return 0
046     return 1
047
048 # Mueve desde CC desde inicial hasta final.
049 def mover(inicial, final, tablero, contador):
050     tablero[inicial[0]][inicial[1]] = contador
051     tablero[final[0]][final[1]] = "CC"
052
053 # Retrocede una posición.
054 def retroceder(inicial, final, tablero):
055     tablero[inicial[0]][inicial[1]] = "__"
056     tablero[final[0]][final[1]] = "CC"
057
058 # devuelve una lista con los movimientos posibles.
059 def posibles(mov, pos, tablero):
```

```

060     posibles = []
061     for n in range(8):
062         if (pos[0] + mov[n][0]) in range(8) and (pos[1] + mov[n][1])
063             if tablero[pos[0] + mov[n][0]][pos[1] + mov[n][1]] == &
064                 v = [pos[0]+mov[n][0], pos[1]+mov[n][1]]
065                 posibles.append(v)
066     return posibles
067
068 # Ordena dos listas en función de la primera.
069 def ordenar(lista, listb):
070     if listb == []:
071         return listb
072     else:
073         for i in range(len(lista)):
074             for j in range(len(lista) - 1):
075                 if lista[j] > lista[j+1]:
076                     listb[j], listb[j+1] = listb[j+1], listb[j]
077                     lista[j], lista[j+1] = lista[j+1], lista[j]
078     return listb
079
080 # Ordena los valores de manera que primero se mueva a las menos prot
081 def heuristica(pos_mov, tablero, mov):
082     n_pos = []
083     for n in range(len(pos_mov)):
084         pos = len(posibles(mov, pos_mov[n], tablero))
085         n_pos.append(pos)
086     return ordenar(n_pos, pos_mov)
087
088 # Función recursiva principal.
089 def caballo(contador):
090     contador += 1
091     pos = pos_c(tablero)
092     mov = [[2, 1], [1, 2], [-2, 1], [2, -1], [-1, 2], [1, -2], [-2,
093
094     pos_mov = posibles(mov, pos, tablero)
095     pos_mov = heuristica(pos_mov, tablero, mov)
096
097     for i in range(len(pos_mov)):
098         pos_ant = pos
099         mover(pos, pos_mov[i], tablero, contador)
100         dibujar(tablero)
101         if lleno(tablero):
102             sys.exit()
103         pos = pos_c(tablero)
104         caballo(contador)
105         retroceder(pos, pos_ant, tablero)
106         pos = pos_c(tablero)
107         dibujar(tablero)
108
109 # -----
110
111 # Crea un tablero de 8x8.
112 tablero = range(8)

```

```
113 for n in tablero:
114     tablero[n] = range(8)
115
116 # Pone el tablero a 0.
117 for f in range(8):
118     for c in range(8):
119         tablero[f][c] = ""
120
121 # Posición inicial caballo.
122 tablero[random.randint(0,7)][random.randint(0,7)] = "CC"
123
124 # Comienza el programa.
125 dibujar(tablero)
126 contador = 0
127 caballo(contador)
```

Comparte esto:



10 Comentarios en "El salto del caballo, backtracking"



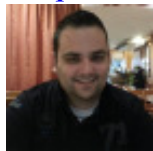
1. *jacob* dice:
[13/06/2011 a las 1:27 am](#)

oie, que representa este pedazo de código?

tablero[f][c]
es de la línea 24.

lo pregunto porque lo intenté correr pero da error de sintaxis en esa parte.

[Responder](#)



2. *adrigm* dice:
[13/06/2011 a las 1:30 am](#)

Es para Python 2.x asegúrate que no estás usando una versión 3.x

[Responder](#)



3. *Richard* dice:
[16/04/2012 a las 22:00 pm](#)

Disculpa, en Python 2.7 Portable me bota error de sintaxis en la línea 21. Agradecería tu ayuda.

[Responder](#)



o [adrigm](#) dice:

[16/04/2012 a las 22:24 pm](#)

son comillas que el resultado de sintaxis ha cambiado seria:

```
tab = ""
```

en el resto del código igual.

[Responder](#)



4. [karina](#) dice:

[22/04/2012 a las 18:20 pm](#)

Disculpa que representa esto:

```
"CC"
```

cuando corro el programa me muestra error en esas partes
gracias

[Responder](#)



5. [karina](#) dice:

[22/04/2012 a las 18:21 pm](#)

22/04/2012 a las 18:20 pm

Disculpa que representa esto:

” "CC"” cuando corro el programa me muestra error en esas partes
gracias

[Responder](#)



o [adrigm](#) dice:

[22/04/2012 a las 18:22 pm](#)

Qué error te da?

[Responder](#)6. *Llanos* dice:[16/11/2012 a las 19:32 pm](#)

alguien sabe cómo se puede hacer en Java?

[Responder](#)7. *cemete* dice:[21/12/2012 a las 10:50 am](#)

porqué pones “tablero[f]1” y cómo puedo cambiarlo para que no me dé error? Uso python 2.7

[Responder](#)8. *cemete* dice:[21/12/2012 a las 11:27 am](#)

He quitado el 1 y lo he dejado como tablero[f] y parece que funciona, pero ahora me da este error en la línea 123: ‘str’ object does not support item assignment

[Responder](#)

Deja un comentario

Introduce tu comentario aquí...

[Razón Artificial](#) por Adrián Guerra Marrero | © 2010-2013

El contenido de este sitio esta bajo una licencia [Creative Commons Atribución-CompartirIgual 3.0 España](#)