UNIVERSIDAD DE GRANADA

Práctica 3: mancala

Juan Ocaña Valenzuela

3 de junio de 2018

Índice

1. Diario de trabajo

Jueves, 31 de mayo de 2018

Me hubiese gustado empezar esta práctica antes, pero quien me conozca sabe que Java y yo no nos llevamos bien, y estuvo prohibiéndome usar la aplicación del mancala, pese a tener Java 8 instalado.

Además, Ubuntu 18.04 también se unió a la fiesta, actuando de mago y haciendo desaparecer ventanas con la ayuda de su ayudante GTK. Por si fuera poco, la distribución que elegí para reemplazar Ubuntu, Antergos, decidió romper sus repositorios media hora antes de que yo empezase a instalarla, quedándome sin Linux hasta el lunes 28 por la mañana.

Una vez toda esta serie de catastróficas desdichas se solucionó, instalé todo lo que tenía que instalar de nuevo, y me puse a... estudiar AC (tenemos exámenes, es lo que hay). Cuando me cansé de la optimización de código con gcc, decidí que era un buen momento para retar al no tan temible GreedyBot, y ya de paso, ponerme a pensar en cómo desarrollar una heurística curiosa.

Del lunes hasta el miércoles, el mancala se ha adueñado de mi tiempo, de mis ideas, y hasta de mis sueños (esto último no es una exageración), y ha dado como resultado un pseudo-código que describe más o menos lo que quiero que haga mi bot. Bot que, hasta la fecha, no tiene nombre.

Aquí adjunto el resultado de mis cavilaciones:

```
#-----------------------------------------------------#
#   Ideas heuristica: Mancala                         #
#-----------------------------------------------------#

#Idea 1
#-------------------------------------------------------

func h(pos_actual):
    var h = metidas_granero() - dadas_rival()
    var ultima_pos = pos_actual + pos_actual.semillas

    return h

func metidas_granero():
    if pos_actual.semillas >= pos_actual:
        return 1
    else:
        return 0Las optimizaciones hacen lo que tienen que hacer.

func dadas_rival():
    if pos_actual.semillas > pos_actual:
```

```
22          return pos_actual.semillas - pos_actual
23      else:
24          return 0
25
26  #-------------------------------------------------------
27
28  #Idea 1.1: unificar metidas_granero() y dadas_rival() en valor_jugada()
29  #-------------------------------------------------------
30
31  func h(pos_actual):
32      var h = valor_jugada()
33      var ultima_pos = pos_actual + pos_actual.semillas
34
35      return h
36
37  func valor_jugada():
38      if pos_actual.semillas >= pos_actual:
39          return pos_actual.semillas - pos_actual
40      else:
41          return 0
42
43  #-------------------------------------------------------
44
45  #Idea 2: anadida comprobacion de poder robar
46  #-------------------------------------------------------
47
48  func h(pos_actual):
49      var h = valor_jugada()
50      var ultima_pos = pos_actual + pos_actual.semillas
51
52      if ultima_pos.es_mia() and not ultimaPos.es_granero():
53          if ultima_pos.semillas == 0 and ultima_pos.enfrente().semillas >
            ↪  0:
54              h = h + ultima_pos.enfrente().semillas + 1
55
56      return h
57
58  func valor_jugada():
59      if pos_actual.semillas >= pos_actual:
60          return pos_actual.semillas - pos_actual
61      else:
62          return 0
63
```

```
64  func enfrente():
65      return pos(oponente, 7 - pos_actual)
66
67  #-------------------------------------------------------
68
69  #Idea 3: anadida comprobacion de si una casilla tiene semillas
70  #-------------------------------------------------------
71
72  func h(pos_actual):
73
74      var h
75      var ultima_pos
76
77      if pos_actual.semillas > 0:
78          h = valor_jugada()
79          ultima_pos = pos_actual + pos_actual.semillas
80
81          if ultima_pos.es_mia() and not ultimaPos.es_granero():
82              if ultima_pos.semillas == 0 and
83              ultima_pos.enfrente().semillas > 0:
84                  h = h + ultima_pos.enfrente().semillas + 1
85
86
87      else:
88          #Valor muy bajo para que nunca lo tome.
89          h = -70
90
91      return h
92
93  func valor_jugada(pos_actual):
94      if pos_actual.semillas >= pos_actual:
95          return pos_actual.semillas - pos_actual
96      else:
97          return 0
98
99  func enfrente(pos_actual):
100     return pos(oponente, 7 - pos_actual)
101 #-------------------------------------------------------
102
103 #Idea 4: tiene en cuenta que cuando das mas de seis semillas al rival,
104 #vuelven a ti (mas vueltas).
105 #Corregido calculo de valor_jugada(). Se debe restar
106 #pos_actual - pos_actual.semillas, no al contrario.
```

```
107 #------------------------------------------------------
108
109 const CASILLAS_VUELTA_COMPLETA = 13
110
111 func h(pos_actual):
112
113     var h
114     var ultima_pos
115
116     #Para que una ficha "salte" del lado contrario al nuestro dando
117     #la vuelta, necesitamos tener un numero de semillas mayor al que
118     #tiene que recorrer contando el granero,
119     #es decir, 6 + pos. Como pueden dar mas de una vuelta, calculamos
120     #modulo(semillas, pos+6), y asi vemos cuantas se introducen en el
121     #granero, ademas de ver cuantas semillas damos al rival en caso de
122     #dar vueltas.
123     var vueltas = pos_actual.semillas%(pos_actual+6)
124
125
126     if pos_actual.semillas > 0:
127         h = valor_jugada(pos_actual, vueltas)
128
129         #Calculamos ultima posicion:
130         if pos_actual > semillas:
131             ultima_pos = posicion_actual - posicion_actual.semillas
132
133         elif pos_actual == pos_relativa(pos_actual, vueltas):
134             ultima_pos = tu_granero
135
136         elif (pos_relativa(pos_actual, vueltas) > pos_actual and
137         pos_relativa(pos_actual, vueltas) < pos_actual+6:
138         #se queda en el otro lado
139             ultima_pos = enfrente(semillas -
140             vueltas*CASILLAS_VUELTA_COMPLETA)
141
142         else: #se queda en nuestro lado
143             ultima_pos = pos_actual - pos_relativa(pos_actual, vueltas)
144
145         #Robo:
146         if ultima_pos.es_mia() and not ultimaPos.es_granero():
147             if ultima_pos.semillas == 0 and
148             ultima_pos.enfrente().semillas > 0:
149                 h = h + ultima_pos.enfrente().semillas + 1
```

```
150
151
152     else:
153         #Valor muy bajo para que nunca lo tome.
154         h = -70
155
156     return h
157
158 func valor_jugada(pos_actual, vueltas):
159     if pos_actual.semillas >= pos_actual:
160         if vueltas > 1:
161             return pos_actual - 6*vueltas + vueltas
162         else:
163             return pos_actual - pos_actual.semillas + 1
164     else:
165         return 0
166
167 func enfrente(pos_actual):
168     return pos(oponente, 7 - pos_actual)
169
170 #Esta funcion devuelve la posicion relativa a la actual.
171 #Valores positivos: a la derecha.
172 #Valores negativos: a la izquierda.
173 func pos_relativa(pos_actual, vueltas):
174     return semillas - vueltas*CASILLAS_VUELTA_COMPLETA)
175
176 #-------------------------------------------------------
```

Con este puñado de ideas, me dispongo a encajarlas tanto en el sistema proporcionado, como en un algoritmo minimax con poda alfa-beta. Espero salir bien parado.

Sábado, 2 de junio de 2018

Entre ayer y hoy he conseguido adaptar la heurística al simulador proporcionado, pero el proceso me ha generado alguna que otra duda. La profesora de prácticas ya me comentó en una tutoría que no había contemplado la opción de poder simular movimientos, y al principio lo rechacé, pensando que sería echar por tierra todo el trabajo que había hecho hasta ahora, pero pensándolo en frío, quizás no sea tan mala idea.

De momento, voy a seguir implementando el minimax con poda alfa-beta, manteniendo la heurística intacta. Si funciona, perfecto; si no, consideraré una opción más simple, como la de simular un movimiento y quedarme con el que más semillas meta en mi granero. He oído que ha sido la elección de bastante gente (quién sabe, igual es mejor que mi código, y es mucho menos laborioso...), pero no quiero dejar de intentar mi opción hasta que no vea que sólo estoy perdiendo el tiempo. Ya tuve que dejar de lado mi primera opción en la anterior práctica, y no quiero que vuelva a suceder.
Intentaré dejarlo todo funcionando para esta noche, y dedicar el día de mañana a corregir cosas puntuales, como hacer ganar al bot. Espero de verdad que todo marche bien.

Actualización nocturna: he implementado tanto la heurística como el algoritmo minimax con poda alfa-beta, corregido varios errores, mayormente de *casting* y sintaxis, y únicamente falta implementar el método nextMove para poner el bot en marcha.
Además, ya tiene nombre: **Patrick Dotimas**.

Domingo, 3 de junio de 2018

Esta mañana he terminado de implementarlo todo. Había olvidado cambiar el nombre del bot en *main.cpp*, pero más allá de eso, la compilación ha funcionado sin problemas, ya que anoche solucioné todos los problemas de casting.

Había llegado la hora de Patrick, pero, como todo el mundo esperaba, aún le queda mucho camino por recorrer. Seleccionaba el movimiento 7, cosa que no debería poder hacer, y aparecía en el log un jugador 3 que no sé muy bien qué pintaba ahí. Revisando la implementación del algoritmo todo parece estar bien, así que voy a probar con heurísticas diferentes, algo más simples, para ver si el error se soluciona.

Probando con el enfoque simplista de $h = nuestroGranero - graneroDelOponente$, Patrick vuelve a las andadas. Sin embargo, se ha empeñado en escoger siempre la casilla 6, haya lo que haya, y no entiendo muy bien por qué.

Tras cerca de dos horas, sigue igual de cabezón, seleccionando siempre la casilla seis, aunque esté vacía. Y si le añado la condición de que no considere la casilla si está vacía, realiza el movimiento 7 y explota. No lo entiendo.

Actualización: toda esta situación se debía a una serie de errores tontos que desembocaba en desastre. Finalmente, Patrick Dotimas es un bot funcional y ha conseguido derrotar al GreedyBot, tanto de J1 como de J2. Ha sido un duro e intenso viaje, pero ahora Patrick debe enfrentarse a lo peor: la desafiante liga oficial de Mancala.

2. Descripción del algoritmo

Utilizaremos el algoritmo minimax con poda alfa-beta, junto con nuestra heurística, para valorar cada movimiento.
Este algoritmo realizará una búsqueda recursiva en forma de árbol, el cual estará compuesto por nodos MAX (nosotros) y nodos MIN (el adversario), simulando estados hasta llegar a las hojas del árbol. El valor de un nodo vendrá determinado por los valores de sus hijos de la siguiente forma:

- Nodo MAX: tendrá el valor del máximo de sus hijos.
- Nodo MIN: tendrá el valor del mínimo de sus hijos.

Esto es así ya que debemos suponer que el oponente (MIN) siempre va a intentar minimizar nuestro beneficio, y nosotros lo maximizaremos.
Sin embargo, esto generará un árbol inmenso (en nuestro caso, para cada estado generará seis hijos de forma recursiva), por lo que recurriremos a la técnica de poda alfa-beta para evitar recorrer gran parte del árbol.

- Alfa representa el máximo valor de lo recorrido para MAX. Se reemplazará con el valor más alto que encontremos.
- Beta es el valor de la mejor opción encontrada para MIN. Se reemplazará con el valor más bajo.

En cada casilla, evaluaremos tantos turnos como profundidad indiquemos en la llamada al método.

3. Descripción de la heurística utilizada

Se ha utilizado la siguiente heurística para determinar el movimiento a realizar:

$$h = nuestroGranero - graneroDelOponente$$

Que, traducido a nuestro código, sería:

```
1  int PatrickDotimas::h(GameState state) {
2
3    //Contamos las semillas de nuestro granero menos las del del
     ↪  contrario:
4    int h = state.getScore(patrick) - state.getScore(oponente);
5
6    return h;
7  }
```

4. Bibliografía

Páginas y manuales consultados para realizar la práctica:
https://www.ittc.ku.edu/publications/documents/Gifford_ITTC-FY2009-TR-03050-03.pdf
https://www.experts-exchange.com/questions/21628387/Mancala-heuristics-if-you're-familiar.html
http://www.staff.science.uu.nl/~bodla101/d.gam/mancala.html