
# Universidad de Granada

Doble Grado en Ingeniería Informática y Matemáticas

# Inteligencia Artificial

*Memoria de la Práctica 3*

Autor:
Jesús Muñoz Velasco

Curso 2024-2025

# Índice

# 1. Podas

A continuación se desarrollan las distintas podas que se han implementado con el objetivo de mejorar la eficacia y la eficiencia del algoritmo `minimax`.

## 1.1. Alpha-Beta

En primer lugar se ha implementado la poda Alpha-Beta básica. Esta consiste en añadir dos parámetros a la función `minimax` con el objetivo de evitar explorar ramas que no van a mejorar el resultado actual. Dichos parámetros representan unas cotas inferior y superior (respectivamente) de cada nodo. Conforme se llama al método recursivamente se producen 2 actualizaciones:

- •) **Actualización hacia abajo:** Cuando un nodo genera sus hijos les asigna a cada uno su valor alpha y su valor beta.
- •) **Actualización hacia arriba:** Cuando un nodo es evaluado (ya sea por ser nodo hoja o por haber sido evaluados todos sus descendientes) se actualizan los valores `alpha` y `beta` del nodo padre. Esto depende de qué tipo de nodo sea:
  - Si es un nodo `MAX` actualiza el `beta` del padre por su valor `alpha` si y solo si $alpha_{hijo} < beta_{padre}$
  - Si es un nodo `MIN` actualiza el `alpha` del padre por su valor `beta` si y solo si $beta_{hijo} > alpha_{padre}$

  De esta forma si se va a evaluar un nodo y cumple que $alpha >= beta$ entonces se deja de seguir evaluando (evaluando a sus descendientes) y se produce la poda

## 1.2. Poda Probabilística

Después de la implementación anterior se ha añadido una versión que suaviza la condición de poda de forma que define un umbral (que se ha puesto a 10) y en este caso la condición de poda sería que $alpha + UMBRAL >= beta$. De esta forma se podan ramas que tienen mucha probabilidad de ser podadas (dado que `alpha` solo puede crecer como se ha visto en el punto anterior). Aún así esto no deja de ser probabilístico y podría ignorar la rama donde se encuentra la solución que daría el minimax. Es por esto que hay mejores opciones.

## 1.3. Ordenación de movimientos

Este mecanismo consiste en elegir el orden en el que se evalúan los nodos descendientes de un nodo padre. Para ello se ordenan de mayor a menor según su valor heurístico. Esto provoca que se evalúen primero las ramas más prometedoras aumentando la probabilidad de cota. Este método sí promete dar la solución exacta y en la práctica se ha visto que es el más eficiente en relación tiempo-resultados.

# 2. Heurísticas

A continuación se describe el proceso que se ha seguido para la obtención de los resultados

## 2.1. Heurística Básica

En primer lugar se desarrolló una heurística básica, basada en la dada en el tutorial. En este punto se vio que al tener en cuenta los factores como la distancia a la meta, si está en la casa o si está en la recta final y modificando ligeramente los pesos ya se conseguía ganar al ninja 1. Esta heurística es la denominada `valoracion1` en el código proporcionado.

## 2.2. Heurística Avanzada

Basada en la heurística anterior se buscó ganar ahora al ninja 2. Para ello se valoró positivamente si en la jugada actual podía comer alguna ficha rival valorando además cómo de buena era la ficha que había comido (basándose en la distancia a la meta). Con esto se consiguió vencer al ninja 2 (aunque por algún motivo dejé de ganar al ninja 1).

## 2.3. Heurística Definitiva

Terminando la heurística anterior (valorando barreras y rebotes) se vio que no mejoraba mucho la heurística anterior y que con pequeños cambios en los pesos variaba mucho las victorias o derrotas con respecto a los ninjas. Es por ello que se decidió hacer un nuevo enfoque. Para ello se desarrolló esta heurística pero definiendo todos los pesos al principio de la misma y de forma que leyera dichos valores de un archivo externo (en este caso `'pesos.csv'`). Después se copió esta heurística pero de forma que leyera los pesos de otro archivo (`'pesos2.csv`).

Después se desarrolló un programa externo (`'generar_pesos.py'`) cuyo objetivo era mediante un algoritmo genético ir mejorando los pesos actuales. Se le dio como semilla los valores iniciales del archivo `'pesos.csv'` que tenían los valores de la heurística anterior pero adaptados a esta (poniendo a 0 todos los casos que no se valoraban como barreras y rebotes).

De esta forma el programa hacía una mutación sobre los valores de `'pesos.csv'` y lo guardaba en `'pesos2.csv'`. Después enfrentaba ambas heurísticas en los dos casos posibles (según quién empezara). En caso de ganar las 2 partidas la segunda heurística, esta sustituía a la primera y se repetía el proceso. Además para la mutación se tuvo en cuenta la cantidad de partidas ganadas consecutivas que llevaba la heurística "ganadora"variando cada vez menos los pesos hasta un umbral inferior.

Lamentablemente al ser tan lenta la ejecución no logró mejorarse mucho y no consiguió ganar al ninja 3 en ningún caso. Si se quiere ver el desarrollo del programa auxiliar se puede consultar en el Anexo.

## 3. Análisis de resultados

El análisis de las Heurísticas ha resultado en lo siguiente (donde * indica que el ninja comienza la partida como jugador 1):

| Heurística | Ninja 1 | Ninja 1* | Ninja 2 | Ninja 2* | Ninja 3 | Ninja 3* |
|---|---|---|---|---|---|---|
| valoracion0 (Id=2) | ✓ | ✓ | × | × | × | × |
| valoracion1 (Id=3) | × | × | ✓ | ✓ | × | × |
| valoracion2 (Id=7) | × | × | × | × | × | × |

En todas se ha elegido el algoritmo de la poda ordenada por ser el que mejor resultado ha proporcionado.

## 4. Conclusiones

A pesar de los esfuerzos no se ha conseguido vencer a más de dos ninjas simultáneamente. Es posible que heurísticas más sencillas ofrezcan mejores resultados (igual que se vio que la primera heurística conseguía vencer al ninja 1 mientras no se ha vuelto a conseguir dicho resultado).

## 5. Anexo (Algoritmo genético)

A pesar de pedir explícitamente la no inclusión de código en el presente documento, ya que no va a estar incluido en la entrega añado el código del programa con el algoritmo genético. Si esto va a penalizar se puede omitir su lectura.

```
#!/usr/bin/env python3
import os
import csv
import random
import subprocess
from datetime import datetime

# Configuración
NUM_GENERACIONES = 10000
ID_11 = 4
ID_12 = 5
FICHERO_PESOS_11 = "pesos.csv"
FICHERO_PESOS_12 = "pesos2.csv"
LOG_PATH = "log_evolucion.csv"

# Rangos de cada parámetro
rangos = [
    (0, 100),   # Máximo beneficio distancia
    (0, 100),   # Mínimo beneficio distancia
    (0, 100),   # Máximo Comer Ficha
    (0, 100),   # Mínimo Comer Ficha
    (-100,100), # Comer propia
```

```
23      (-100, 0),  # Ficha en Casa
24      (0, 100),   # Ficha en Casilla Segura
25      (0, 100),   # Ficha en Recta Final
26      (0, 100),   # Ficha en Casilla Final
27      (-100, 0),  # Rebote
28      (0, 100),   # Barrera
29  ]
30
31  # === Funciones auxiliares ===
32
33  def crear_individuo():
34      return [round(random.uniform(r[0], r[1]), 2) for r in rangos]
35
36  # def mutar(individuo, tasa_mutacion=0.3, desviacion=2.5):
37  #     nuevo = individuo[:]
38  #     # for i in range(len(nuevo)):
39  #     #     if random.random() < tasa_mutacion:
40  #     #         min_val, max_val = rangos[i]
41  #     #         nuevo[i] = round(random.uniform(min_val, max_val), 2)
42  #     if random.random() < tasa_mutacion:
43  #             min_val, max_val = rangos[i]
44  #             cambio = random.uniform(-desviacion, desviacion)
45  #             nuevo_valor = nuevo[i] + cambio
46  #             # Asegurar que el nuevo valor esté dentro de los rangos
47  #             nuevo[i] = round(max(min_val, min(max_val, nuevo_valor)), 2)
48  #     return nuevo
49
50  def mutar(individuo, contador_mejoras, tasa_mutacion=0.3, desviacion_max=50.0,
51              desviacion_min=2.5):
52      """
53      Mutación adaptativa: cuanto más mejora, más suave es la mutación.
54
55      :param individuo: lista de floats (pesos)
56      :param contador_mejoras: cuántas veces ha ganado consecutivamente
57      :param tasa_mutacion: probabilidad de mutar cada peso
58      :param desviacion_max: desviación inicial (cuando no ha ganado nada)
59      :param desviacion_min: desviación mínima (si ha ganado muchas veces)
60      """
61      nuevo = individuo[:]
62
63      # A partir de 10 victorias seguidas la desviación será constante
64      contador_mejoras=min(contador_mejoras,10)
65      desviacion = ((desviacion_min - desviacion_max)/10)*contador_mejoras
66                      + desviacion_max
67
68      for i in range(len(nuevo)):
69          if random.random() < tasa_mutacion:
70              min_val, max_val = rangos[i]
71              cambio = random.uniform(-desviacion, desviacion)
72              nuevo_valor = nuevo[i] + cambio
```

```
73              nuevo[i] = round(max(min_val, min(max_val, nuevo_valor)), 2)
74      return nuevo
75
76  def guardar_pesos(pesos, fichero):
77      with open(fichero, "w") as f:
78          f.write(",".join(map(str, pesos)) + "\n")
79
80  def cargar_pesos(fichero):
81      with open(fichero, "r") as f:
82          return [float(x) for x in f.readline().strip().split(",")]
83
84  # guardar_log(generacion, hijo, "Victoria" if resultado_2 == 1 else "Derrota", ID_12)
85  def guardar_log(generacion, pesos, resultado, empieza):
86      encabezado = [
87          "generacion",
88          "empieza",
89          "min_distancia",
90          "max_distancia",
91          "comer_propia",
92          "max_comer",
93          "min_comer",
94          "ficha_casa",
95          "casilla_segura",
96          "recta_final",
97          "casilla_final",
98          "rebote",
99          "barrera",
100         "resultado",
101         "timestamp"
102     ]
103     existe = os.path.exists(LOG_PATH)
104     with open(LOG_PATH, mode="a", newline="") as f:
105         writer = csv.writer(f)
106         if not existe:
107             writer.writerow(encabezado)
108         writer.writerow([generacion, empieza] + pesos
109                         + [resultado, datetime.now().isoformat()])
110
111 def ejecutar_partida(id1, id2):
112     resultado = subprocess.run(
113         ["./ParchisGame", "--p1", "AI", str(id1), "jugador1", "--p2",
114          "AI", str(id2), "jugador2", "--no-gui"],
115         capture_output=True, text=True
116     )
117     salida = resultado.stdout
118
119     if "Ha ganado el jugador 2" in salida:
120         return 2
121     elif "Ha ganado el jugador 1" in salida:
122         return 1
```

```
123     else:
124         return 0  # Empate o error
125 
126 # === Bucle principal ===
127 
128 if __name__ == "__main__":
129 
130     # Cargo la información del
131     try:
132         campeon = cargar_pesos(FICHERO_PESOS_11)
133         print("Campeón inicial cargado desde pesos.csv")
134     except FileNotFoundError:
135         campeon = crear_individuo()
136         guardar_pesos(campeon, FICHERO_PESOS_11)
137         print("No se encontró pesos.csv. Generado nuevo campeón aleatorio.")
138 
139     contador_victorias=0
140 
141     for generacion in range(1, NUM_GENERACIONES + 1):
142 
143         hijo = mutar(campeon, contador_mejoras=contador_victorias)
144         guardar_pesos(hijo, FICHERO_PESOS_12)
145 
146         resultado_1 = ejecutar_partida(ID_11, ID_12)
147         guardar_log(generacion, hijo,
148                     "Victoria" if resultado_1 == 2 else "Derrota", ID_12)
149 
150         puntuacion_id12 = 0
151 
152         if resultado_1 == 2:
153             print(f"Gen {generacion}, Empieza id {ID_11}: ¡Nuevo campeón!")
154             puntuacion_id12 += 0.5
155 
156             # Solo juega la segunda si gana la primera
157             resultado_2 = ejecutar_partida(ID_12, ID_11)
158             guardar_log(generacion, hijo,
159                     "Victoria" if resultado_2 == 1 else "Derrota", ID_12)
160 
161             if resultado_2 == 1:
162                 print(f"Gen {generacion}, Empieza id {ID_12}: ¡Nuevo campeón!")
163                 puntuacion_id12 += 0.5
164             else:
165                 print(f"Gen {generacion}, Empieza id {ID_12}: Derrota o empate")
166         else:
167             print(f"Gen {generacion}, Empieza id {ID_11}: Derrota o empate")
168 
169         guardar_log(generacion, hijo,
170                     "Victoria" if resultado_1 == 2 else "Derrota", ID_11)
171 
172 
```

```
173         # Si gana 1: 0.5, si gana las 2: 1 y si pierde las 2: 0
174
175
176         if puntuacion_id12 == 1: # Que gane las 2
177             contador_victorias=0
178             campeon = hijo
179             guardar_pesos(campeon, FICHERO_PESOS_11)
180         elif puntuacion_id12 == -1: # Se quita esta opción
181             contador_victorias=0
182             for i in range(len(hijo)):
183                 campeon[i] = (campeon[i]+hijo[i])/2   # La media
184             guardar_pesos(campeon, FICHERO_PESOS_11)
185         else:
186             contador_victorias+=1
187
188
189     print("Optimización finalizada.")
190
```