# MEMORIA PRÁCTICA 4

Alejandro Santorum - alejandro.santorum@estudiante.uam.es
Sergio Galán Martín - sergio.galanm@estudiante.uam.es
Inteligencia Artificial
Práctica 3 Pareja 9

6 de mayo de 2019

## Contents

# 1 Introducción

Llegamos a la práctica más competitiva de este curso: El Torneo. Nuestro objetivo será diseñar y programar la mejor heurística para que juegue eficientemente al conocido juego `Conecta 4`. En la primera sección expondremos los jugadores finales subidos al torneo, y en la segunda sección explicaremos otra parte del trabajo realizado, que se centra en atacar el problema utilizando nuestros conocimientos sobre aprendizaje automático.

# 2 Heurísticas

A continuanción enumeraremos algunas de las heurísticas diseñadas y las ideas que nos llevaron a ellas.

## 2.1 Primer intento

Como primer acercamiento para familiarizarnos con las heurísticas en lisp para el Connect4 lo que hicimos fue "completar" al jugador bueno, es decir, hacer que comprobara no solo las lineas fichas de abajo, izquierda, derecha y abajo-izquierda, sino las fichas en todas las posibles direcciones. Esto no dio mucho fruto, sino que de hecho el jugador era peor.

Primera heurística

```
(defun heuristica (estado)
  ; current player standpoint
  (let* ((tablero (estado-tablero estado))
         (ficha-actual (estado-turno estado))
         (ficha-oponente (siguiente-jugador ficha-actual)))
    (if (juego-terminado-p estado)
        (let ((ganador (ganador estado)))
          (cond ((not ganador) 0)
                ((eql ganador ficha-actual) +val-max+)
                (t +val-min+)))
      (let ((puntuacion-actual 0)
            (puntuacion-oponente 0))
        (loop for columna from 0 below (tablero-ancho tablero) do
              (let* ((altura (altura-columna tablero columna))
                     (fila (1- altura))
                     (abajo (contar-abajo tablero ficha-actual columna
                        fila))
                     (der (contar-derecha tablero ficha-actual columna
                        fila))
                     (izq (contar-izquierda tablero ficha-actual columna
                        fila))
                     (abajo-der (contar-abajo-derecha tablero ficha-actual
                         columna fila))
                     (arriba-izq (contar-arriba-izquierda tablero
                        ficha-actual columna fila))
                     (abajo-izq (contar-abajo-izquierda tablero
                        ficha-actual columna fila))
                     (arriba-der (contar-arriba-derecha tablero
                        ficha-actual columna fila)))
                (setf puntuacion-actual
                      (+ puntuacion-actual
              (cond ((= abajo 0) 0)
                    ((= abajo 1) 10)
                                 ((= abajo 2) 100)
                                 ((= abajo 3) 1000))
                               (cond ((= der 0) 0)
                                     ((= der 1) 10)
                                     ((= der 2) 100)
                                     ((= der 3) 1000))
                               (cond ((= izq 0) 0)
                                     ((= izq 1) 10)
                                     ((= izq 2) 100)
                                     ((= izq 3) 1000))
                               (cond ((= abajo-izq 0) 0)
                                     ((= abajo-izq 1) 10)
```

```
                                ((= abajo-izq 2) 100)
                                ((= abajo-izq 3) 1000))
                          (cond ((= abajo-der 0) 0)
                                ((= abajo-der 1) 10)
                                ((= abajo-der 2) 100)
                                ((= abajo-der 3) 1000))
                          (cond ((= arriba-izq 0) 0)
                                ((= arriba-izq 1) 10)
                                ((= arriba-izq 2) 100)
                                ((= arriba-izq 3) 1000))
                          (cond ((= arriba-der 0) 0)
                                ((= arriba-der 1) 10)
                                ((= arriba-der 2) 100)
                                ((= arriba-der 3) 1000)))))
                (let* ((altura (altura-columna tablero columna))
                       (fila (1- altura))
                       (abajo (contar-abajo tablero ficha-oponente columna
                          fila))
                       (der (contar-derecha tablero ficha-oponente columna
                          fila))
                       (izq (contar-izquierda tablero ficha-oponente columna
                           fila))
                       (abajo-der (contar-abajo-derecha tablero
                          ficha-oponente columna fila))
                       (arriba-izq (contar-arriba-izquierda tablero
                          ficha-oponente columna fila))
                       (abajo-izq (contar-abajo-izquierda tablero
                          ficha-oponente columna fila))
                       (arriba-der (contar-arriba-derecha tablero
                          ficha-oponente columna fila)))
                  (setf puntuacion-oponente
                        (+ puntuacion-oponente
                           (cond ((= abajo 0) 0)
                                 ((= abajo 1) 10)
                                 ((= abajo 2) 100)
                                 ((= abajo 3) 1000))
                           (cond ((= der 0) 0)
                                 ((= der 1) 10)
                                 ((= der 2) 100)
                                 ((= der 3) 1000))
                           (cond ((= izq 0) 0)
                                 ((= izq 1) 10)
                                 ((= izq 2) 100)
                                 ((= izq 3) 1000))
                           (cond ((= abajo-izq 0) 0)
                                 ((= abajo-izq 1) 10)
                                 ((= abajo-izq 2) 100)
                                 ((= abajo-izq 3) 1000))
                           (cond ((= abajo-der 0) 0)
                                 ((= abajo-der 1) 10)
                                 ((= abajo-der 2) 100)
                                 ((= abajo-der 3) 1000))
                           (cond ((= arriba-izq 0) 0)
                                 ((= arriba-izq 1) 10)
                                 ((= arriba-izq 2) 100)
                                 ((= arriba-izq 3) 1000))
                           (cond ((= arriba-der 0) 0)
                                 ((= arriba-der 1) 10)
                                 ((= arriba-der 2) 100)
                                 ((= arriba-der 3) 1000))))))
(- puntuacion-actual puntuacion-oponente)))))
```

## 2.2 Segundo intento

Tras ver que la anterior heurística no daba muy buenos resultados, decidimos hacer otra empezando de cero. Para ello, nos pusimos a pensar en cómo puntuar adecuadamente un estado en función de lo bueno que es un estado, es decir, maximizar el número de posibles líneas aliadas y minimizar el número de posibles líneas enemigas. Con esto en mente, decidimos crear unas funciones auxiliares que se encargaran de comprobar todas las posibles líneas verticales y horizontales del tablero, y asignarles una puntuación de 0 en caso de que

haya fichas de ambos jugadores, 10 elevado al número de fichas aliadas en esa línea, o -(10 elevado al número de fichas enemigas). Al subir el jugador al servidor vimos que tenía un porcentaje de victorias bastante alto, con lo que confirmamos que ibamos por buen camino. También añadir que variamos el 10 a otros valores y vimos que, por ejemplo, con 2 mejoraba algo el porcentaje de victoria.

Segunda heurística

```
; Auxiliary functions
(defun check-line (ficha prim sec ter cuar)
  (let ((ficha-o (cond ((= ficha 1) 0)
                       ((= ficha 0) 1)))
        (vals (list prim sec ter cuar)))
    (if (= (count ficha-o vals) 0)
        (expt 10 (count ficha vals))
      0)))

(defun check-horiz (tablero ficha-actual)
  (let ((puntuacion 0))
    (loop for fila from 0 below (tablero-alto tablero) do
          (loop for col from 0 to 3 do
                (let* ((prim (obtener-ficha tablero (+ col 3) fila))
                       (sec (obtener-ficha tablero (+ col 2) fila))
                       (ter (obtener-ficha tablero (+ col 1) fila))
                       (cuar (obtener-ficha tablero col fila)))
                  (setf puntuacion
                    (+ puntuacion
                       (check-line ficha-actual prim sec ter cuar))))))
    puntuacion))

(defun check-vert (tablero ficha-actual)
  (let ((puntuacion 0))
    (loop for col from 0 below (tablero-ancho tablero) do
          (loop for fila from 0 to 2 do
                (let* ((prim (obtener-ficha tablero col (+ fila 3)))
                       (sec (obtener-ficha tablero col (+ fila 2)))
                       (ter (obtener-ficha tablero col (+ fila 1)))
                       (cuar (obtener-ficha tablero col fila)))
                  (setf puntuacion
                    (+ puntuacion
                       (check-line ficha-actual prim sec ter cuar))))))
    puntuacion))

(defun check-diag-ppal (tablero ficha-actual)
  (let ((puntuacion 0))
    (loop for col from 0 to 3 do
          (loop for fila from 3 to 5 do
                (let* ((prim (obtener-ficha tablero col fila))
                       (sec (obtener-ficha tablero (+ col 1) (- fila 1)))
                       (ter (obtener-ficha tablero (+ col 2) (- fila 2)))
                       (cuar (obtener-ficha tablero (+ col 3) (- fila 3)))
                          )
                  (setf puntuacion
                    (+ puntuacion
                       (check-line ficha-actual prim sec ter cuar))))))
    puntuacion))

(defun check-diag-neg (tablero ficha-actual)
  (let ((puntuacion 0))
    (loop for col from 0 to 3 do
          (loop for fila from 0 to 2 do
                (let* ((prim (obtener-ficha tablero col fila))
                       (sec (obtener-ficha tablero (+ col 1) (+ fila 1)))
                       (ter (obtener-ficha tablero (+ col 2) (+ fila 2)))
                       (cuar (obtener-ficha tablero (+ col 3) (+ fila 3)))
                          )
                  (setf puntuacion
                    (+ puntuacion
                       (check-line ficha-actual prim sec ter cuar))))))
    puntuacion))

;Heuristica
```

```
(defun f-eval-horiz-vert-lines (estado)
  ; current player standpoint
  (let* ((tablero (estado-tablero estado))
         (ficha-actual (estado-turno estado))
         (ficha-oponente (siguiente-jugador ficha-actual)))
    (if (juego-terminado-p estado)
        (let ((ganador (ganador estado)))
          (cond ((not ganador) 0)
                ((eql ganador ficha-actual) +val-max+)
          (t +val-min+)))
      (let ((puntuacion-actual 0)
            (puntuacion-oponente 0))
        (setf puntuacion-actual
          (+ puntuacion-actual
             (check-horiz tablero ficha-actual)
             (check-vert tablero ficha-actual)))
        (setf puntuacion-oponente
          (+ puntuacion-oponente
             (check-horiz tablero ficha-oponente)
             (check-vert tablero ficha-oponente)))
(- puntuacion-actual puntuacion-oponente)))))
```

## 2.3 Tercer intento

Una vez visto que con la heurística anterior ya teníamos un considerable porcentaje de victorias, pensamos que lo más adecuado sería hacer pequeñas modificaciones sobre la que ya teníamos para pulirla y mejorar poco a poco su eficiencia. Tras varios intentos que no mejoraron la eficiencia, decidimos buscar en internet artículos sobre el Connect4 y encontramos la tesis de Victor Allis. De las múltiples estrategias que allí se exponen, decidimos que la mejor en relación mejora/tiempo disponible de implementación era la de dar más puntuación a los tres en raya (3 fichas aliadas y ninguna enemiga en la misma línea) que estuvieran en una altura par si somos los que hemos jugado primero y o dar mas puntuación a los que estuvierian en altura impar si somos los que hemos jugado segundos. Con esto mejoramos algo el porcentaje de victorias de nuestro jugador.

Tercera heurística

```
; Auxiliary functions
(defun parity (tablero)
  (let* ((num 0))
    (loop for col from 0 below (tablero-ancho tablero) do
          (setf num (+ num (altura-columna tablero col))))
    (mod num 2)))

(defun get-height (prim sec ter cuar hp hs ht hc)
  (cond ((null prim) (mod hp 2))
        ((null sec) (mod hs 2))
        ((null ter) (mod ht 2))
        ((null cuar) (mod hc 2))))

(defun check-line-height-ln (tablero ficha prim sec ter cuar hp hs ht hc)
  (let ((ficha-o (cond ((= ficha 1) 0)
                       ((= ficha 0) 1)))
        (vals (list prim sec ter cuar)))
    (if (= (count ficha-o vals) 0)
        (cond ((= (count ficha vals) 1) 2)
              ((= (count ficha vals) 2) (calculate-value-filter tablero
                 prim sec ter cuar hp hs ht hc))
              ((= (count ficha vals) 3) (if (=(parity tablero) (get-height
                  prim sec ter cuar hp hs ht hc))
                                            8
                                           12))
              (T 1))
      0)))


(defun calculate-value (tablero fila)
  (if (= (parity tablero) (mod fila 2))
      3
    2))
```

```
32
33 (defun calculate-value-filter (tablero prim sec ter cuar hp hs ht hc)
34   (let ((puntuacion 0))
35     (if (null prim) (setf puntuacion (+ puntuacion (calculate-value
          tablero hp))))
36     (if (null sec) (setf puntuacion (+ puntuacion (calculate-value tablero
           hs))))
37     (if (null ter) (setf puntuacion (+ (calculate-value tablero ht))))
38     (if (null cuar) (setf puntuacion (+ (calculate-value tablero hc))))
39     puntuacion))
40 (defun check-horiz-ln (tablero ficha-actual)
41   (let ((puntuacion 0))
42     (loop for fila from 0 below (tablero-alto tablero) do
43           (loop for col from 0 to 3 do
44                 (let* ((prim (obtener-ficha tablero (+ col 3) fila))
45                        (sec (obtener-ficha tablero (+ col 2) fila))
46                        (ter (obtener-ficha tablero (+ col 1) fila))
47                        (cuar (obtener-ficha tablero col fila)))
48                   (setf puntuacion
49                     (+ puntuacion
50                        (check-line-height-ln tablero ficha-actual prim sec
                               ter cuar fila fila fila fila))))))
51     puntuacion))
52
53 (defun check-vert-ln (tablero ficha-actual)
54   (let ((puntuacion 0))
55     (loop for col from 0 below (tablero-ancho tablero) do
56           (loop for fila from 0 to 2 do
57                 (let* ((prim (obtener-ficha tablero col (+ fila 3)))
58                        (sec (obtener-ficha tablero col (+ fila 2)))
59                        (ter (obtener-ficha tablero col (+ fila 1)))
60                        (cuar (obtener-ficha tablero col fila)))
61                   (setf puntuacion
62                     (+ puntuacion
63                        (check-line-height-ln tablero ficha-actual prim sec
                               ter cuar (+ fila 3) (+ fila 2) (+ fila 1) fila)
                             )))))
64     puntuacion))
65 (defun check-diag-ppal-ln (tablero ficha-actual)
66   (let ((puntuacion 0))
67     (loop for col from 0 to 3 do
68           (loop for fila from 3 to 5 do
69                 (let* ((prim (obtener-ficha tablero col fila))
70                        (sec (obtener-ficha tablero (+ col 1) (- fila 1)))
71                        (ter (obtener-ficha tablero (+ col 2) (- fila 2)))
72                        (cuar (obtener-ficha tablero (+ col 3) (- fila 3)))
                             )
73                   (setf puntuacion
74                     (+ puntuacion
75                        (check-line-height-ln tablero ficha-actual prim sec
                               ter cuar fila (- fila 1) (- fila 2) (- fila 3))
                             )))))
76     puntuacion))
77
78 (defun check-diag-neg-ln (tablero ficha-actual)
79   (let ((puntuacion 0))
80     (loop for col from 0 to 3 do
81           (loop for fila from 0 to 2 do
82                 (let* ((prim (obtener-ficha tablero col fila))
83                        (sec (obtener-ficha tablero (+ col 1) (+ fila 1)))
84                        (ter (obtener-ficha tablero (+ col 2) (+ fila 2)))
85                        (cuar (obtener-ficha tablero (+ col 3) (+ fila 3)))
                             )
86                   (setf puntuacion
87                     (+ puntuacion
88                        (check-line-height-ln tablero ficha-actual prim sec
                               ter cuar fila (+ fila 1) (+ fila 2) (+ fila 3))
                             )))))
89 puntuacion))
90
91 ;heuristic
92 (defun heuristica (estado)
```

```
  ; current player standpoint
  (let* ((tablero (estado-tablero estado))
     (ficha-actual (estado-turno estado))
     (ficha-oponente (siguiente-jugador ficha-actual)))
    (if (juego-terminado-p estado)
    (let ((ganador (ganador estado)))
      (cond ((not ganador) 0)
        ((eql ganador ficha-actual) +val-max+)
         (t +val-min+)))
      (let ((puntuacion-actual 0)
            (puntuacion-oponente 0))
        (setf puntuacion-actual
          (+ puntuacion-actual
             (check-horiz-ln tablero ficha-actual)
             (check-vert-ln tablero ficha-actual)
             (check-diag-ppal-ln tablero ficha-actual)
             (check-diag-neg-ln tablero ficha-actual)))
        (setf puntuacion-oponente
          (+ puntuacion-oponente
             (check-horiz-ln tablero ficha-oponente)
             (check-vert-ln tablero ficha-oponente)
             (check-diag-ppal-ln tablero ficha-oponente)
             (check-diag-neg-ln tablero ficha-oponente)))
(- puntuacion-actual puntuacion-oponente)))))
```

## 2.4 Jugadores finales

Los jugadores finales fueron segundo intento cambiándole la ponderación a check-line a 2 en vez de a 10, tercer intento, y tercer intento cambiandole la ponderación de check-line-height a 10 en vez de a 12. Los dos primeros ya habían participado en algun torneo y vimos que tenían porcentajes de victoria superiores al 85%, pero el tercero lo subimos para el último torneo y, al no estar aún los resultados, nos es imposible saber si mejoró o no.

## 2.5 Evaluación

Para evaluar la eficiencia de los primeros jugadores también hicimos un pequeño código para jugar 100 partidas contra aleatorio, 50 con aleatorio jugando primero y 50 con aleatorio jugando segundo. De ahí sacábamos el winratio contra el jugador aleatorio. Sin embargo, a partir del denominado segundo intento ya teníamos 100% de winratio, por lo que no fue muy útil de cara a evaluar la eficiencia de los últimos jugadores.

Winrate vs Aleatorio

```
(setf winratio 0)
(loop for match from 1 to 50 do
      (let ((value (partida *jugador-aleatorio* *jugador-test* 4)))
        (setf winratio (+ winratio (cond ((null value) 0)
                                         (t value))))))
(loop for match from 51 to 100 do
      (let ((value (partida *jugador-test* *jugador-aleatorio* 4)))
        (setf winratio (+ winratio (cond ((null value) 0)
                                         ((= value 0) 1)
                                         ((= value 1) 0))))))
(print winratio)
```

# 3 Aproximación al aprendizaje automático

En esta sección se va a exponer el desarrollo de la idea de entrenar una **Inteligencia Artificial** y que sea esta la que evalúe por nosotros un tablero, es decir, nosotros no le vamos a dar las reglas para que calcule como de bueno (o malo) es un estado del tablero, sino que serán los algoritmos de aprendizaje automático los que infieran las reglas que evalúan dichos estados.

En un primer momento se pensó en utilizar un algoritmo de **aprendizaje reforzado**, pero esto se vió pronto comprometido con el detalle de tener que transportar todo lo diseñado e implementado a Lisp. Además, no podíamos almacenar grandes datos ya precalculados, ya que el fichero que se subía al torneo tenía que tener un peso limitado.

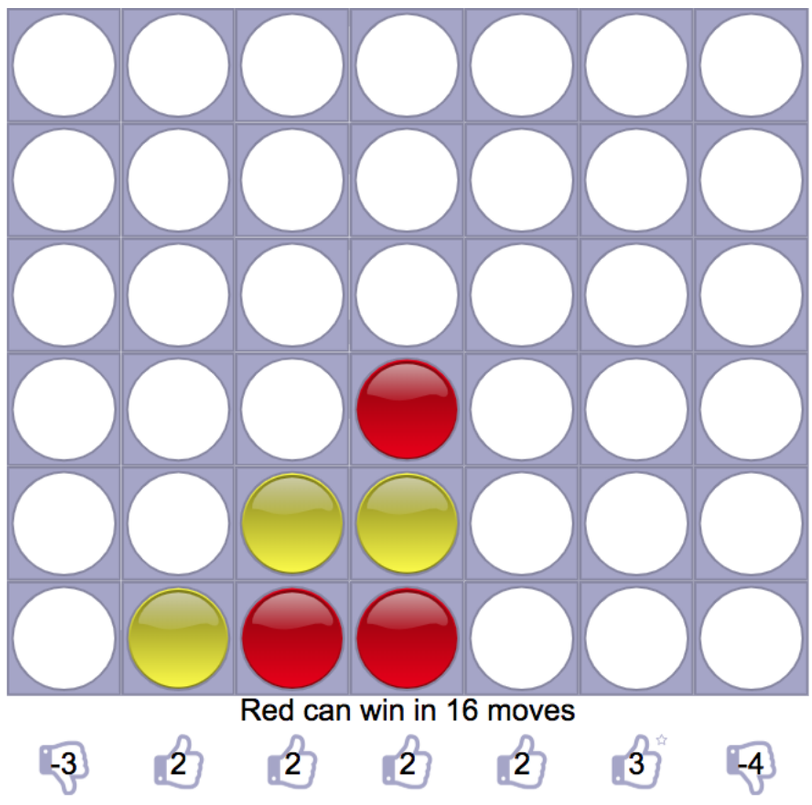
Por el mismo motivo también fueron rechazados otros algoritmos que necesitasen para su evaluación una gran coste computacional, ya que el tiempo de ejecución era limitado en el servidor del torneo.

Aclarado esto, procedemos a buscar algoritmos que no necesiten apoyo de datos precalculados, como sería el caso de algoritmos de aprendizaje automático que necesitarían **tablas de transposición** y **árboles de búsqueda** entre otras estructuras para su implementación y ejecución. Además, buscamos algoritmos que una vez entrenados podamos obtener sus pesos o parámetros y pasarlos fácilmente a Lisp.

Con estas limitaciones sabíamos que posiblemente no sacaríamos nada relativamente útil de cara al torneo, pero eso no nos amedrentó y decidimos intentarlo, aunque fuese solo por aprender y encontrarnos con problemas que nos haciesen crecer.

## 3.1 Recolección de datos

La primera fase que definimos en nuestra estrategia es la recolección de datos. La pregunta es, ¿qué datos podemos buscar para entrenar una IA para que prediga cómo de bueno es un estado de un tablero?. Pues bien, esto fue rápido de resolver ya que nos encontramos una página web (https://connect4.gamesolver.org/) en la cual el sistema juega de forma perfecta, aportándote además un valor numérico de cómo de bueno es un movimiento.



Tal y como se muestra en la foto, el movimiento de la primera columna tiene un valor $-3$, lo que quiere decir que es un mal movimiento y acabaríamos perdiendo la partida si el oponente juega perfecto. Por otro lado, el movimiento en las columnas 2, 3, 4, 5 y 6 tienen una puntuación positiva, lo que quiere decir que ganaríamos la partida con esos movimientos si suponemos que jugamos de forma perfecta. Una puntuación de 0 quiere decir que la partida acabará en empate si ambos jugadores juegan de forma perfecta.

Comentar que el sistema de esta página juega de forma perfecta debido a que desarrolla todo el árbol de juego en `C++`, algo que ya hemos comentado era imposible para nosotros. Se puede encontrar más información de cómo está implementado el algoritmo de juego de esta página en el siguiente enlace: http://blog.gamesolver.org/

Viendo que esta página juega de forma perfecta y nos aporta un valor numérico para cada movimiento, podemos utilizarlo para intentar evaluar un estado. De esta forma, consideramos que el valor de un estado es el valor que se le daba al último movimiento, es decir, el valor del movimiento que hizo acabar en ese estado. Por lo tanto, nuestro objetivo en este momento es buscar aleatoriamente un estado **no final**, coger todas las puntuaciones de los movimientos a partir de ese estado y asignarle esos valores a los estados alcanzados realizando el movimiento correspondiente al valor.

Por lo tanto, nuestro principal problema en estos momentos es cómo buscar un estado aleatorio. Por suerte, la *url* de la página es modificada para cada movimiento. La *url* del tablero de la imagen es https://connect4.gamesolver.org/?pos=324443, donde cada dígito de 324443 es la columna donde se ha movido en cada turno (indexado en 1). En el ejemplo, rojo ha jugado primero en la columna 3, seguido de un movimiento en la 2, en tres turnos consecutivos se ha movido en la columna 4 y finalmente en la 3.

Sabiendo esto, se programa una función en python que obtiene una secuencia aleatoria de entre 5 y 35 dígitos, los cuales estarán entre 1 y 7 (columnas) con una frecuencia menor o igual que 6 (número de filas).

Función generadora de estados

```
PIECES = ["1", "1", "1", "1", "1", "1",
          "2", "2", "2", "2", "2", "2",
          "3", "3", "3", "3", "3", "3",
          "4", "4", "4", "4", "4", "4",
          "5", "5", "5", "5", "5", "5",
          "6", "6", "6", "6", "6", "6",
          "7", "7", "7", "7", "7", "7"]

MAX_PIECES = 42 # Maximum number of pieces on a Connect4 board
MAX_PLAYS = 35  # Maximum number of plays
MIN_PLAYS = 5   # Minimum number of plays

################################################################
#       It generates a random board pattern. The pattern length is
#       chosen randomly between MIN_PLAYS and MAX_PLAYS, and the
#       played column in each turn is also chosen randomly
################################################################
def generate_board_pattern():
        board = ""
        # Auxiliary pieces array
        pieces_aux = PIECES.copy()
        # Getting randomly the number of plays
        n_plays = randint(MIN_PLAYS, MAX_PLAYS)
        for i in range(n_plays):
                # Getting random piece position
                pos = randint(0,MAX_PIECES-i-1)
                # Extracting piece and adding it to the board pattern
                    expression
                board += pieces_aux.pop(pos)
        return board
```

Ahora que tenemos el generador de estados disponible, utilizamos la librería de python `Selenium`, con la que podremos entrar en la página introduciendo la *url* generada y coger sus datos. A continuación se muestra la función que itera tantas veces como le hayamos

dicho, generando un patrón de tablero aleatorio y iniciando un hilo paralelo que calculará las características más significativas de un tablero y la puntuación que le da al mismo la página web.

Scraping

```
################################################################
#       It creates a webdriver (in this case ChromeDriver) and loops
#       'n_examples' times, getting a random board pattern, search it
#       and starting a parallel thread that will calculate the desired
#       board features (storing them in a file)
################################################################
def scrape_main(n_examples):
        start_time = time.time()

        # Getting init board pattern
        pattern = generate_board_pattern()
        first_url = MAIN_URL + pattern

        # Initializing webdriver (Chrome in this case)
        driver = webdriver.Chrome()
        # Opening it for the first time
        driver.get(first_url)

        try:
                # Main loop
                for i in range(n_examples):
                        print("N example: ", i)
                        # Opening a new tab to use it later
                        next_tab = "tab"+str(i+1)
                        new_tab_command = "window.open('about:blank', \'"+
                            next_tab+"\');"
                        driver.execute_script(new_tab_command)

                        # Get button you are going to click by its id (
                            also you could
                        # find_element_by_css_selector to get element by
                            css selector)
                        button_element = driver.find_element_by_id('
                            hide_solution_div')

                        # Getting points
                        time.sleep(SLEEP_TIME)
                        points_array = []
                        for j in range(0,7):
                                elem = driver.find_element_by_xpath('//*[
                                    @id="sol' + str(j) + '"]')
                                points_array.append(elem.get_attribute('
                                    innerHTML'))

                        # Creating a new thread to calculate and store
                            features
                        features_thread = thr.Thread(
                                target = features_main,
                                args = (pattern, points_array,)
                        )
                        features_thread.start()

                        # Closing current tab
                        driver.close()

                        # Switching tabs
                        driver.switch_to.window(next_tab)

                        # Random URL generator
                        pattern = generate_board_pattern()

                        next_url = MAIN_URL+pattern
                        driver.get(next_url)
                return n_examples
        except Exception as err:
                print("Something went wrong at iteration: "+str(i)+" ==> "
```

```
                    + str(err))
60                  return i
61          finally:
62                  driver.quit()
63                  end_time = time.time()
64                  print("Time used: "+str(end_time - start_time)+" seconds")
```

La función que ejecutan los hilos paralelamente, **features_main**, será descrita en secciones posteriores.
Como este programa puede fallar en cualquier momento, ya sea porque se ha perdido la conexión *wifi* o porque la página nos ha prohibido el acceso por constante abuso (control que al final no parecía tener), se ha diseñado un sistema por el cual la función volverá a ser ejecutada si se da el caso de que salta una excepción y el número de ejemplos *"scrapeados"* no corresponde con el deseado:

Scrape main

```
1 if __name__ == "__main__":
2         # Number of desired examples
3         M = 20000
4         # Writing header file (features names)
5         init_features_file()
6         # Preparing counters in error case
7         exec_times = 0
8         aux_counter = M
9         while aux_counter > 0:
10                ret = scrape_main(aux_counter)
11                exec_times += 1
12                print("Return from scrape_main (time "+str(exec_times)+"):
                      "+str(ret))
13                aux_counter -= ret
```

Esto resulta muy útil sobre todo si tus recursos son limitados y tienes que dejar el ordenador ejecutando el programa durante la noche o en cualquier otro momento en el que no haya supervisión del ingeniero.

## 3.2 Atributos del estado del tablero

Hemos estado viendo como recoger la información de la página web https://connect4.gamesolver.org, sobre todo la puntuación numérica para cada movimiento.

Ahora vamos a programar en python el propio juego Conecta 4, con el que podremos manejar cada estado:

Estructura tablero e inserción

```
1 class Board():
2
3     def __init__(self, nrows , ncols):
4         # Number of board rows
5         self.nrows = nrows
6         # Number of board columns
7         self.ncols = ncols
8         # Board is represented as a matrix
9         self.board = np.zeros((nrows, ncols))
10        # Array of heigths (number of pieces placed in each col)
11        self.height = np.zeros(ncols)
12
13    ################################################
14    #   It inserts a piece in a given column
15    #   User has to check the board status
16    #   before inserting the piece, because
17    #   this function does not check it
18    ################################################
19    def insert(self , piece , col):
20        counter = 0
21        for i in range(self.nrows):
22            counter += np.abs(self.board[i][col])
23        pos = self.nrows - (counter+1)
24        self.board[int(pos)][col] = piece
25        self.height[col] += 1
26        return int(pos)
```

Nos interesa ahora que dado un patrón de un estado (secuencia de dígitos que redirecciona un estado en la página conocida), podamos obtener la instancia de dicho tablero en ese estado en concreto. Para ello tenemos que tener en cuenta (cosa que también lo tiene la página web) que los patrones que generamos pueden ser finales, incluso hace ya muchas jugadas atrás, por lo que a la hora de construir el tablero deberemos truncar el patrón si se diese el caso que ya fuese final, sin necesidad de seguir leyendo el resto de dígitos.

Función constructora a partir de patrones

```
    ################################################
    #   It builds a board given a status pattern,
    #   truncating it if the pattern represents
    #   a winner status
    ################################################
    def build_pattern(self, pattern):
        length = len(pattern)
        current_board = ""
        current_piece = 1
        for i in range(length):
            col_move = int(pattern[i])-1 # Pattern indexes in 1
            inserted_row = self.insert(current_piece, col_move)
            if self.winner(inserted_row, col_move, current_piece):
                self.go_back(col_move)
                return current_board, current_piece
            current_board += pattern[i]
            current_piece = 0-current_piece # Swaping player move
        return current_board, current_piece
```

Comentar que las funciones **winner** y **go_back** evalúan si un estado ya es un estado final, o elimina el último movimiento en una columna dada respectivamente.

Vista la forma en que implementamos un tablero en Python, vamos a centrarnos en uno de los aspectos más importantes en este proyecto: ¿qué atributos o característica tiene el estado de un tablero? ¿Cuáles son las peculiaridades que hacen un estado de un tablero tener una puntuación positiva o negativa?
Estas preguntas no son fáciles de contestar, posiblemente quien lo haya hecho mejor haya ganado el torneo, ya que habrá hecho mejores heurísticas.

Nosotros en el momento en que estábamos desarrollando este modelo de aprendizaje automático aún no habíamos ideado nuestras mejores heurísticas, por lo que los atributos que escogimos no se corresponden en su totalidad a los que escogeríamos en este momento.

En ese momento decidimos probar cómo afecta a la evaluación de un estado los siguientes atributos: · Número total de fichas en el tablero · Distancia media entre fichas aliadas · Distancia media entre fichas enemigas · Amenazas de 2-en-raya aliadas · Grupos de 2-en-raya aliados · Amenazas de 2-en raya enemigas · Gruposde 2-en-raya enemigos · Amenazas de 3-en-raya aliadas · Grupos de 3-en-raya aliados · Amenazas de 3-en raya enemigas · Gruposde 3-en-raya enemigos Estos datos son recogidos y almacenados en un array mediante la siguiente función:

Obtención de atributos

```
    def get_features(self, piece):
        features = []
        # Total number of pieces
        features.append(sum(self.height))
        # Ally mean distance
        features.append(self._mean_distance(piece))
        # Opponent mean distance
        features.append(self._mean_distance(-piece))
        # Ally #2-in-a-row
        block, eff = self._calculate_N_in_a_row(piece, 2)
        features.append(block)
        features.append(eff)
        # Opponent #2-in-a-row
        block, eff = self._calculate_N_in_a_row(-piece, 2)
        features.append(block)
        features.append(eff)
        # Ally #3-in-a-row
```

```
        block, eff = self._calculate_N_in_a_row(piece, 3)
        features.append(block)
        features.append(eff)
        # Opponent #3-in-a-row
        block, eff = self._calculate_N_in_a_row(-piece, 3)
        features.append(block)
        features.append(eff)
        return features
```

Comentar que a día de hoy, hubiésemos eliminado la distancia media entre fichas (muy poco relevante), hubiésemos añadido el tema de la paridad, es decir, el jugador que juega primero tiene preferencia por las casillas de altura impar y el que juega segundo tiene preferencia por las casillas de altura par, por lo que las amenazas de dichas alturas sería recompensadas con un mayor valor; y por otro lado, hubiésemos añadido que las amenazas situadas encima de otras amenazas son inútiles, por lo que no las hubiésemos contado con esta función, mejorando la calidad de los atributos obtenidos.
Otro atributo posiblemente interesante sería un "mapa de calor" de las fichas, ya que se premia jugar por el medio del tablero al tener más presencia y mayor número de posibles amenazas. No obstante esto sería difícil de codificar y aún es un tema de estudio.

Llegado este momento, es la hora de recordar cual era la función que ejecutaban paralelamente los hilos ejecutados al hacer *scraping*: **features_main**.
Este función comprueba si el tablero obtenido es final. Esto lo hace comprobando si el array de puntuaciones *scrapeadas* está vacío, ya que no hay más movimiento posibles y por lo tanto sin puntuaciones.
En el caso de que no sea un estado final, se construye el tablero utilizando **build_pattern** y para cada posible movimiento se calculan los atributos del estado y se guarda en un fichero al array de atributos junto con la puntuación objetina. Si es un estado final se construye el tablero hasta el penúltimo movimiento (eliminando un dígito del patrón) y se hace el mismo proceso.

Función que ejecutan los hilos

```
############################################################
#   It builds a board given a pattern and calculates its
#   features, storing them into a file
############################################################
def features_main(pattern, points_array):
    if empty_points(points_array)==True:
        board = Board(NROWS, NCOLS)
        current_pattern, current_piece = board.build_pattern(pattern)
        points = get_points(current_pattern)
        for i in range(NCOLS):
            if points[i] != '-':
                # Getting child board
                board.insert(current_piece, i)
                # Getting features of this board
                features_array = board.get_features(current_piece)
                # Getting the father state
                board.go_back(i)
                # Adding points
                features_array.append(int(points[i]))

    else:
        board = Board(NROWS, NCOLS)
        current_pattern, current_piece = board.build_pattern(pattern)
        for i in range(NCOLS):
            if points_array[i] != '':
                # Getting child board
                board.insert(current_piece, i)
                # Getting features of this board
                features_array = board.get_features(current_piece)
                # Getting the father state
                board.go_back(i)
                # Adding points
                features_array.append(int(points_array[i]))
    # Writing features in a file
    store_features(FEATURES_FILE, features_array)

```

```
37
38 ############################################################
39 #   It stores in a file the board features and its points
40 ############################################################
41 def store_features(filename, features_array):
42     # Red light
43     lock.acquire()
44     # Writing file
45     f = open(filename, "a")
46     f.write(str(features_array)[1:len(str(features_array))-1])
47     f.write("\n")
48     f.close()
49     # Green light
50     lock.release()
51
52
53 ############################################################
54 #   It checks if an array of points is empty
55 ############################################################
56 def empty_points(points_array):
57     for i in range(NCOLS):
58         if points_array[i] != '':
59             return False
60     return True
```

Ejecutando en conjunto las funciones expuestas en estas dos secciones durante un par de noches hemos conseguido reunir cerca de **200.000 ejemplos**, con los que entrenaremos a nuestro modelo.

## 3.3 Entrenamiento

Ha llegado el momento que estábamos esperando: entrenar nuestro modelo de inteligencia artificial con los datos obtenidos.

Como hemos comentado en la introducción de esta sección, nuestro abanico de modelos está limitado por razones obvias. Lo que se ha intentado en un primer momento es realizar una **regresión lineal multidimensional**. Aunque se denomine regresión lineal, también se han añadido nuevos atributos que son combinaciones de los atributos simples obtenidos del tablero, es decir, si denominamos $x_1, x_2, ..., x_{11}$ a los 11 atributos analizados de cada estado, hemos calculado todas las combinaciones cuadráticas entre ellos: $x_1^2, x_2^2, ..., x_{11}^2, x_1x_2, ..., x_1x_3, ...$

De esta forma nuestro objetivo será minimizar el error entre $y_i$ (puntuación obtenida de la página web), y $h_\theta(x^i) = \theta_0 x_0^i + \theta_1 x_1^i + ... + \theta_n x_n^i$ *con* $i = 1, ..., M$ donde $n = número\ de\ atributos\ totales$, $\theta_j = parámetros\ de\ entrenamiento$ y $M = número\ de\ ejemplos\ de\ entrenamiento$.

Código de entrenamiento

```
1 def linear_regression(filename, train_test_h, feat_degree):
2     # Reading file
3     data = np.loadtxt(filename, delimiter=SEP, usecols=range(Y_COL))
4     ydata = np.loadtxt(filename, usecols=Y_COL)
5
6     # Feature preparation
7     poly_feat = PolynomialFeatures(feat_degree)
8     xdata = poly_feat.fit_transform(data)
9
10    # Train and test set
11    xdata_train = xdata[:train_test_h]
12    xdata_test = xdata[train_test_h:]
13    ydata_train = ydata[:train_test_h]
14    ydata_test = ydata[train_test_h:]
15
16    # Linear regression (actually polynomial regression)
17    lin_reg = LinearRegression()
18    # Fitting
19    lin_reg.fit(xdata_train, ydata_train)
20    # Predicting
21    y_pred = lin_reg.predict(xdata_test)
```

```
22
23      print("YPRED:")
24      print(y_pred)
25
26      # The coefficients
27      print('Coefficients: \n', lin_reg.coef_)
28      # The mean squared error
29      print("Mean squared error: %.2f" % mean_squared_error(ydata_test,
            y_pred))
30      # Explained variance score: 1 is perfect prediction
31      print('Variance score: %.2f' % r2_score(ydata_test, y_pred))
32
33      print("My test: ", my_test(y_pred, ydata_test, 1))
```

Mostramos a continuación un ejemplo de ejecución:

```
[SantorumPC:Connect4 santorum$ python3 fitting.py
YPRED:
[  1.9127369   -3.24232768  -4.08593064 ...  -8.72644781 -11.996267
   8.4054666 ]
Coefficients:
 [-1.69694714e-14  1.07608560e+00  1.92516084e+00 -2.46266262e+00
 -2.25692373e+00 -5.53842630e-01  1.05599275e+00 -8.20068856e-01
  6.16997961e+00  6.94528781e+00 -7.88989446e-01 -1.71318143e+01
 -2.20385973e-03 -3.34708642e-01  3.64791211e-02  6.32158289e-03
 -2.33511556e-03 -1.89350111e-02  8.84463702e-03  1.24329768e-02
 -1.40283157e-01  3.58220765e-03  1.96067448e-01 -3.14858203e-01
  1.93240038e-03  3.78261093e-01  7.55413761e-02  7.16701497e-02
 -2.77422224e-01 -9.38760527e-01 -2.72088863e-01  4.93847032e-01
  2.51453957e+00  7.61150096e-01  3.53395940e-02  1.81411459e-01
 -5.92604116e-02  2.62710417e-01 -8.06892236e-01 -2.62879674e-01
 -1.14456039e-01 -5.74907057e-01  1.86099128e-02  3.80146184e-02
  3.33727067e-03 -2.25667333e-02 -3.58061602e-02  3.92094965e-02
  8.93740922e-03  1.53554749e-01  1.26758586e-03 -2.45177050e-03
  1.18130014e-02 -1.31959574e-01 -1.96844545e-01 -1.52534530e-02
 -1.11691271e-02 -1.05373801e-02 -8.55506792e-03  6.79279586e-03
 -1.98309079e-02 -2.44793770e-02 -2.11996461e-02  2.97317451e-02
 -8.51419157e-03  8.41797612e-02  1.15750492e-02  3.00720186e-01
 -5.94806844e-03 -1.36394268e-01 -6.23057753e-02  2.54573858e-01
 -1.21065146e-02  9.73735040e-02 -4.75325760e-01  1.36394276e-02
 -1.77247316e-01  6.45348263e-01]
Mean squared error: 73.41
Variance score: 0.23
My test:  59.36882969110806
SantorumPC:Connect4 santorum$
```

## 3.4 Conclusiones

Las conclusiones son varias:

· En primer lugar, con hipersuperficies de grado 2 ya tenemos 78 atributos más el término independiente o *intercepter*. Pasar esto a mano es posible, con mucho trabajo, pero posible. No obstante queda claro que no podemos aumentar el grado a uno superior, ya que el número de atributos, y por lo tanto de parámetros de entrenamiento, aumentan exponencialmente, haciendo imposible pasarlo a mano a código Lisp.

· Las predicicciones no son una maravilla. El error medio cuadrático es entorno a 70, cuando los valores que intentamos predecir están entre $-17$ y 17. Esto indica que las predicciones no son nefastas, pero inútiles de cara al torneo. Una heurística sencillita jugaría posiblemente mejor.

· El valor denominado "My test" simboliza el porcentaje de jugadas que se han etiquetado correctamente dentro de dos categorías: buena jugada o mala jugada. Esto quiere decir que con una simple una regresión lineal de grado 2 tenemos una precisión del 60%, lo que hace pensar que un modelo de **clasificación** como **Random Forest** o **Regresión Logística**, hubiese tenido unos resultados aceptables.

· Si aumentamos el grado, las predicciones mejoran. Esto puede indicar que si hubiésemos

podido utilizar un **algoritmo de aprendizaje reforzado** o **redes neuronales profundas** hubiésemos sacado unos resultados bastante interesantes.

Finalmente no hemos subido ningún jugador basado en este trabajo, pero es algo con mucho potencial de mejora y que, de haberse tenido menos limitaciones técnicas, estamos seguro que hubiésemos dado la sorpresa.

Todo el **código** de esta sección se encuentra en el repositorio público del siguiente enlace: **https://github.com/AlejandroSantorum/Connect4_HeuristicPrediction**