TEC | Tecnológico de Costa Rica

Proyecto 1: PacMan - Minimax con Poda $\alpha - \beta$

Walter Morales Vásquez
2018212846
Brandon Ledezma Fernández
2018185574

05 de abril del 2021

Profesora:
María Auxiliadora Mora Cross

Escuela de Ingeniería en Computación
Unidad Desconcentrada de Alajuela
Carrera de Ingeniería en Computación
Principios de Sistemas Operativos
Grupo 20

Sede Interuniversitaria de Alajuela
Costa Rica

Tabla de contenidos

1. Introducción

La teoría de juegos forma una parte importante dentro del área de la inteligencia artificial, este trabajo busca aumentar el entendimiento sobre este tema al centrarse en el juego de adversarios y en los estados del juego. El presente sistema consiste en una implementación del clásico juego de **Pac-Man** que funcione de forma automatizada mediante el uso de algoritmos de inteligencia artificial, en específico se utilizará el algoritmo de **Minimax** con su variante de **Poda $\boldsymbol{\alpha}$ - $\boldsymbol{\beta}$** para definir el movimiento de Mr. Pac-Man.

El juego consiste en un sistema multi-agente en el que el personaje central será Mr. Pac-Man, el cuál se encargará de recorrer todo un tablero de juego en búsqueda de todos los puntos (*dots*) presentes para pasar al siguiente nivel. Sin embargo, en el laberinto de juego también se encontrarán los fantasmas (en este caso 2) que se encargarán de seguir a Mr. Pac-Man con el fin de alcanzarlo y quitarle sus vidas, no obstante, nuestro personaje principal podrá hacer uso de píldoras capaces de invertir los papeles para deshacerse de estos.

Para la implementación del sistema se deben de considerar las siguientes características:

- Es necesario elaborar una única heurística para Pac-Man y una para los fantasmas.
- Se deben crear 3 laberintos distintos, los cuales funcionarán como los niveles de juego.
- Se maneja un sistema de vidas (3 inicialmente) que se verá aumentado con el pasar de los niveles al sumarse 3 vidas nuevas.
- Únicamente existirán 2 fantasmas por laberinto y ambos se verán influenciados por la misma heurística.
- Es necesario llevar la puntuación de Mr. Pac-Man, esta se aumentará en 10 al comer un punto del laberinto, se sumarán 100 puntos al comer un fantasma y se agregarán 300 puntos cuando Mr. Pac-Man pase al siguiente nivel. Por otro lado, se perderán 100 puntos al perder una vida y cada vez que se avance a un cuadro vacío se restarán 10 puntos.
- Existen dos posibles estados finales del juego: el primero en caso de que Mr. Pac-Man gane al comer a todos los puntos en todos los laberintos o que termine perdiendo al verse reducidas todas sus vidas ante los fantasmas.

El juego deberá de contar también con características técnicas como:

- El juego deberá desarrollarse sobre el lenguaje de programación funcional **Racket**, tanto en el código logístico, como en la interfaz gráfica de usuario.
- Para el uso del algoritmo MiniMax con Poda α - β, es necesario tomar en cuenta que se trata de un sistema multi-agente, por lo tanto es vez de únicamente incluir una capa de minimización por rival será necesario incluir dos (una por cada fantasma).
- La profundidad del árbol de Minimax debe ser variable, de manera que se pueda jugar un mismo juego probando con diferentes niveles de profundidad para el árbol.
- Los fantasmas deberán seguir a Mr. Pac-Man de manera indefinida, moviéndose de manera que se reduzca la distancia en el estado normal del juego.
- Cuando los agentes fantasmas se vean afectados por una píldora de Mr. Pacman, buscarán alejarse de este y verán reducida su velocidad a la mitad.

2. Descripción de los Agentes y Componentes

A continuación se procederá a describir al agente correspondiente a Mr. Pac-Man y a los fantasmas.

2.1. Mr. Pac-Man:

2.1.1. Descripción y objetivo del agente:

El agente es un circulo con boca cuyo objetivo es comer todos los puntos presentes en un laberinto. Adicionalmente, debe estar pendiente de huir de agentes (fantasmas) que buscan cazarlo. Sin embargo, este agente puede capturar píldoras capaces de hacer que pueda comer a estos otros agentes.

2.1.2. Función del agente:

```
1  (define MAXINF 9999999999)
2  (define MININF -9999999999)
3  (define INIT-DEPTH 0)
4  (define MAX-DEPTH 6)
5
6  (define (minimax current-game-state)
7    (define (max-aux board pacman ghost1 ghost2 current-depth path-sum alpha beta)
8      (cond [(and (> MAX-DEPTH current-depth) (not (goal? pacman ghost1 ghost2)))
9              (let ([best-val MININF])
10               (for ([newpos (get-pacman-moves board pacman ghost1 ghost2)])
11                 (let ([val (min-aux (new-board-empty-at newpos board) newpos ghost1
       ghost2 (+ current-depth 1) (sum-cell path-sum newpos board ghost1 ghost2) #t alpha
       beta)])
12                   (set! best-val (if (< val best-val) best-val val)))
13                   #:break (>= best-val beta)
14                   (set! alpha (if (< alpha best-val) best-val alpha)))
15               best-val)]
16             [else (+ path-sum (eval board pacman-pos ghost1-pos is-scared1 ghost2-pos is-
       scared1 maze dots))]))
17
18   (define (min-aux board pacman ghost1 ghost2 current-depth path-sum conditional alpha
       beta)
19     (cond [(and (> MAX-DEPTH current-depth) (not (goal? pacman ghost1 ghost2)))
20         (cond [conditional
21                (let ([best-val MAXINF])
22                  (for ([newpos (get-ghost-moves board ghost1 pacman)])
23                    (let ([val (min-aux board pacman newpos ghost2 (+ current-depth 1) path
       -sum #f alpha beta)])
24                      (set! best-val (if (> val best-val) best-val val)))
```

```
25                        #:break (<= best-val alpha)
26                        (set! beta (if (> beta best-val) best-val beta)));)
27                    best-val)]
28                [else
29                 (let ([best-val MAXINF])
30                   (for ([newpos (get-ghost-moves board ghost2 pacman)])
31                     (let ([val (max-aux board pacman ghost1 newpos (+ current-depth 1) path
    -sum alpha beta)])
32                       (set! best-val (if (> val best-val) best-val val)))
33                       #:break (<= best-val alpha)
34                       (set! beta (if (> beta best-val) best-val beta)));)
35                    best-val)])]
36          [else (+ path-sum (eval board pacman ghost1 ghost2))]))
37
38  (define (minimax-aux board pacman ghost1 ghost2 current-depth path-sum)
39    (let ([best-move null]) (let ([best-val MININF])
40      (for ([newpos (get-pacman-moves board pacman ghost1 ghost2)])
41        (let ([val (min-aux (new-board-empty-at newpos board) newpos ghost1 ghost2 (+
    current-depth 1) (sum-cell path-sum newpos board ghost1 ghost2) #t best-val MAXINF)
    ])
42          (cond [(< best-val val) (set! best-move newpos) (set! best-val val)])))
43      best-move)))
44
45  (minimax-aux (game-state-board current-game-state)
46               (game-state-pacman current-game-state)
47               (game-state-ghost1 current-game-state)
48               (game-state-ghost2 current-game-state)
49               INIT-DEPTH
50               0))
```

2.1.3. El entorno de la tarea (PEAS):

- La medida de desempeño: La cantidad de vidas restantes, el tiempo empleado en terminar un laberinto, la puntuación final, la optimalidad del patrón de movimiento...
- El ambiente: El tablero de juego, los fantasmas, los puntos y las píldoras.
- Los actuadores: Capacidad de pasar de una celda contigua a otra.
- Los sensores: Su función de evaluación.

2.2. Fantasma:

2.2.1. Descripción y objetivo del agente:

Este agente es un fantasma de color rojo o naranja que busca capturar al primer agente (Mr. Pac-Man). Sin embargo, si el otro agente captura una píldora, los papeles se invertirán y este agente deberá huir de Pac-Man (la velocidad de movimiento del Fantasma en este estado es reducida a la mitad).

2.2.2. Función del agente:

```
(define (get-ghost-moves board ghost-pos pacman-pos is-scared)
  (dont-move ghost-pos (filter (    (npos) (and (is-valid? board npos) (no-pacman? npos
    pacman-pos (not is-scared))))
          ( list  (upward ghost-pos) (downward ghost-pos) (leftward ghost-pos) (rightward
    ghost-pos)))))

(define (ghost-move-filter oldpos newpos dir)
  (cond [(and (string=? dir "D") (= (sub1 (position-y oldpos)) (position-y newpos))) #f]
        [(and (string=? dir "U") (= (add1 (position-y oldpos)) (position-y newpos))) #f]
        [(and (string=? dir "R") (= (sub1 (position-x oldpos)) (position-x newpos))) #f]
        [(and (string=? dir "L") (= (add1 (position-x oldpos)) (position-x newpos))) #f]
        [else #t]))

(define (move-ghost current-game-state is-ghost1)
  (define (move-ghost-aux pacman ghost-moves best-move best-value scared)
    (cond [(null? ghost-moves) best-move]
          [(( if scared < >) (manhattan-distance (car ghost-moves) pacman) best-value)
           (move-ghost-aux pacman (cdr ghost-moves) (car ghost-moves) (manhattan-
    distance (car ghost-moves) pacman) scared)]
          [else (move-ghost-aux pacman (cdr ghost-moves) best-move best-value scared)]))

  (define current-ghost (if is-ghost1 (game-state-ghost1 current-game-state) (game-state-
    ghost2 current-game-state)))
  (define pacman (game-state-pacman current-game-state))
  (define ghost-moves (if (ghost-scared current-ghost)
                          ( filter  (lambda (newpos) (ghost-move-filter current-ghost newpos)
    )
                                    (get-ghost-moves (game-state-board current-game-state)
                                                     current-ghost
                                                     pacman (not (zero? (ghost-scared current-
    ghost)))))
                          (get-ghost-moves (game-state-board current-game-state)
                                           (ghost-position current-ghost)
                                           pacman-pos (not (zero? (ghost-scared current-
    ghost)))))))
  (move-ghost-aux pacman
```

```
30                  (cdr ghost-moves)
31                  (car ghost-moves)
32                  (manhattan-distance (car ghost-moves) pacman-pos)
33                  (zero? (ghost-scared current-ghost)))))
```

2.2.3. El entorno de la tarea (PEAS):

- La medida de desempeño: La cantidad de veces que logra capturar a Pac-Man, el tiempo que tarda en capturar a Pac-Man, la optimalidad del patrón de movimiento...
- El ambiente: El tablero de juego, las píldoras y Pac-Man.
- Los actuadores: Capacidad de moverse de una celda contigua a otra.
- Los sensores: La distancia de Manhattan que le indica si se acerca a su objetivo.

3. Descripción de la Función EVAL

Para el funcionamiento de esta función será necesario pasar por parámetro el tablero que se analizará, junto con las posiciones de Pac-Man y de los dos fantasmas, así como el contador de turnos que les queda a cada uno de los fantasmas para dejar de estar asustados por el efecto de la pastilla, el número de nivel en el que se encuentra el juego actualmente y la cantidad de puntos (*dots*) que se encuentran en el tablero actualmente.

Se evaluará cada una de las posiciones del tablero con respecto a Pac-Man, es decir, el valor del elemento que se encuentra en determinada celda es multiplicado por un número que será mayor en tanto se encuentre cerca de la posición de Pac-Man y menor en caso contrario (dando mayor valor a los tableros que tengan más celdas beneficiosas cercanas a Pac-Man, este valor es calculado utilizando la fórmula de la distancia de Manhattan). Cabe destacar que el valor que proporciona una celda con un punto será mayor en tanto queden menos puntos en el tablero.

Seguidamente, se procederá a restar la importancia que se le da a los fantasmas en el juego (lo que provoca que Pac-Man busque rutas lejanas a ellos). Esta importancia, de igual manera, también está multiplicada por un valor que incrementa conforme la posición del fantasma sea más cercana a la de Pac-Man. Sin embargo, si los fantasmas en ese momento dado se encuentran asustados por el efecto de la pastilla, el valor de la importancia de estos será positivo, de manera que Pac-Man intente acercarse a ellos.

4. Descripción de los Heurística de los Fantasmas

Por otro lado, la heurística de los fantasmas se ve influenciada principalmente por el algoritmo de distancia de Manhattan, el cuál hace uso de la fórmula:

$$[a - c] + [b - d]$$

Sin embargo, al verse los fantasmas constantemente atrapados en esquinas al intentar acercarse a Pac-Man, decidimos eliminarles la opción de retroceder en su camino, de esta manera, cuándo un fantasma decide irse por un camino para seguir a Pac-Man, continua por este mismo hasta que se encuentre con otra posible opción por donde moverse. Esto conllevó a un gran aumento en su capacidad para cazar a Pac-Man y en un comportamiento bastante similar al del juego original.

5. Pruebas de Rendimiento

A continuación se muestran las pruebas de rendimiento realizadas a la aplicación variando entre la profundidad del árbol de Minimax con Poda α - β (utilizando valores de profundidad de entre 3 y 10) y el laberinto utilizado (alternando entre los 3 tableros originales).

La primer columna de la tabla representa el número de laberinto por el que se está experimentando, la segunda columna indica la profundidad utilizada en el árbol de decisión de Pac-Man, la tercera columna muestra la puntuación alcanzada por la inteligencia artificial en la ejecución, la cuarta columna indica los "ticks"que fueron contados en la puesta en marcha de la aplicación con las características dadas y la quinta columna representa el tiempo real abarcado por la ejecución del programa sin interfaz. Por su parte, la última columna indica si el Pac-Man resultó ganador, perdedor o si el juego nunca concluyó.

Cabe destacar que se realizaron pruebas en dos computadoras diferentes y los tiempos reales de ejecución fueron bastante diferentes. A continuación se muestra las especificaciones de cada una de las computadoras, junto con su tabla correspondiente:

- **Modelo:** Laptop DELL Latitude 5480
- **Procesador:** Intel® Core™ i7-7820HQ CPU @ 2.90GHz × 8
- **Cantidad de RAM:** 8 Gb
- **Frecuencia RAM:** 2133 MHz
- **Velocidad de Almacenamiento:** 437,3 MB/s
- **GPU:** Intel® HD Graphics 630 (KBL GT2)

Número de Nivel	Profundidad	Marcador	Tiempo de Juego	Tiempo Real	Resultado
1	3	49	20	1,212s	Perdió
1	4	629	39	1,821s	Ganó
1	5	615	42	2,084s	Ganó
1	6	615	42	2,314s	Ganó
1	7	638	37	2,528s	Ganó
1	8	632	38	3,088s	Ganó
1	9	632	38	5,262s	Ganó
1	10	540	35	7,452s	Ganó
2	3	752	73	5,522s	Perdió
2	4	521	55	4,207s	Perdió
2	5	472	47	4,178s	Perdió
2	6	inf	inf	inf	Encicló
2	7	1316	99	11,006s	Ganó
2	8	1103	102	14,810s	Ganó
2	9	1394	104	24,317s	Ganó
2	10	1205	101	49,395s	Ganó
3	3	inf	inf	inf	Encicló
3	4	inf	inf	inf	Encicló
3	5	inf	inf	inf	Encicló
3	6	1666	172	19,031s	Ganó
3	7	1676	126	18,593s	Ganó
3	8	1681	125	27,284s	Ganó
3	9	2064	136	45,743s	Ganó
3	10	inf	inf	inf	Encicló

Cuadro 1: Tabla con los resultados de la primera computadora.

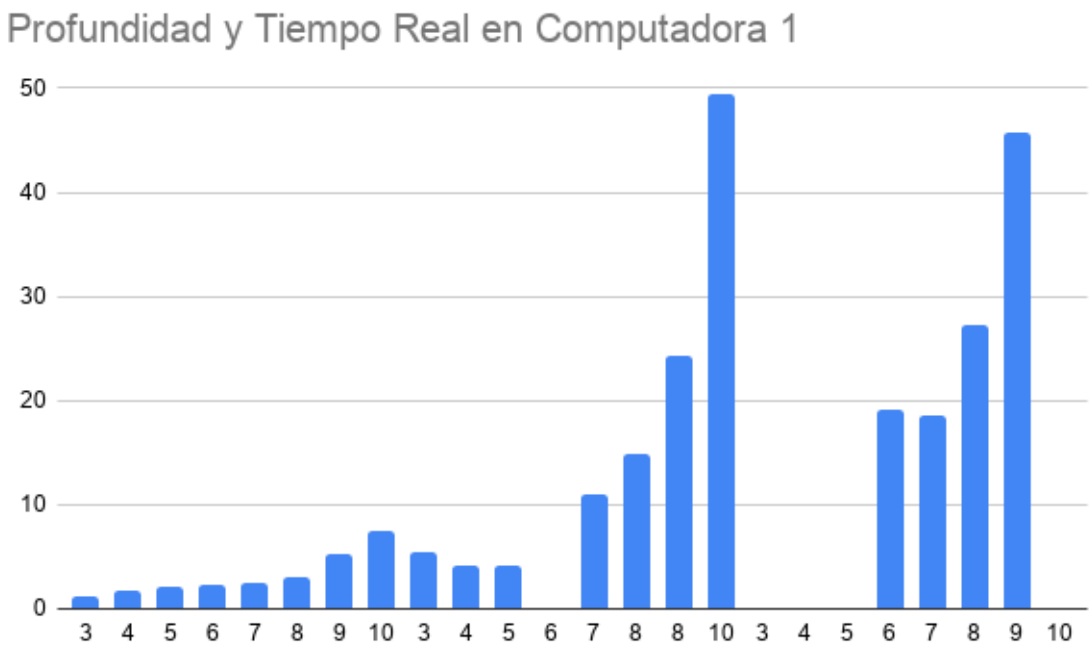

Figura 1: Gráfico con las comparaciones entre Profundidad y Tiempo Real en la primer computadora .

- **Modelo:** Asus ROG Strix g512li-bi7n10
- **Procesador:** Intel® Core™ i7-10750HQ CPU @ 2.60GHz × 12
- **Cantidad de RAM:** 8 GB
- **Frecuencia RAM:** 3200 MHz
- **GPU:** NVIDIA GeForce GTX 1650 Ti

Número de Nivel	Profundidad	Marcador	Tiempo de Juego	Tiempo Real	Resultado
1	3	49	20	2.672s	Perdió
1	4	629	39	4.368s	Ganó
1	5	615	42	5.511s	Ganó
1	6	615	42	7.092s	Ganó
1	7	638	37	9.398s	Ganó
1	8	632	38	16.467s	Ganó
1	9	632	38	33.591s	Ganó
1	10	540	35	66.092s	Ganó
2	3	752	73	10.492s	Perdió
2	4	521	55	9.572s	Perdió
2	5	472	47	11.188s	Perdió
2	6	inf	inf	inf	Encicló
2	7	1316	99	48.158s	Ganó
2	8	1103	102	93.282s	Ganó
2	9	1394	104	162.457s	Ganó
2	10	1205	101	312.914s	Ganó
3	3	inf	inf	inf	Encicló
3	4	inf	inf	inf	Encicló
3	5	inf	inf	inf	Encicló
3	6	1666	172	41.385s	Ganó
3	7	1676	126	58.890s	Ganó
3	8	1681	125	105.158s	Ganó
3	9	2064	136	218.765s	Ganó
3	10	inf	inf	inf	Encicló

Cuadro 2: Tabla con los resultados de la segunda computadora.

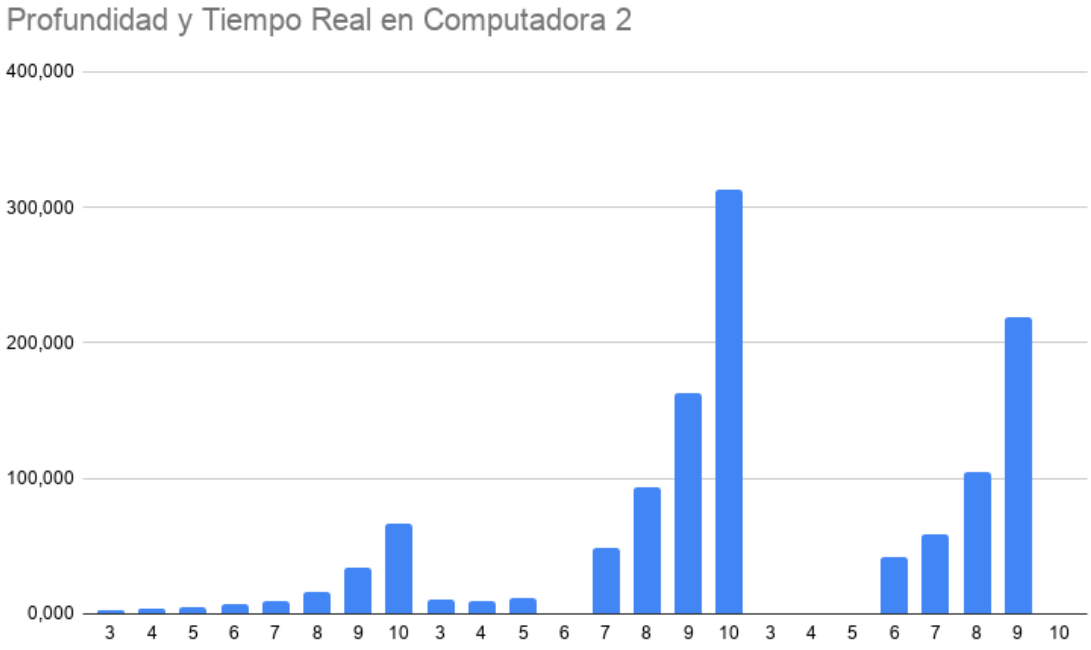

Figura 2: Gráfico con las comparaciones entre profundidad y tiempo real en la segunda computadora.

Los primeros diez valores en el eje x corresponden a la profundidad en el primer nivel, mientras que lo restantes grupos representan a los valores en los demás niveles.

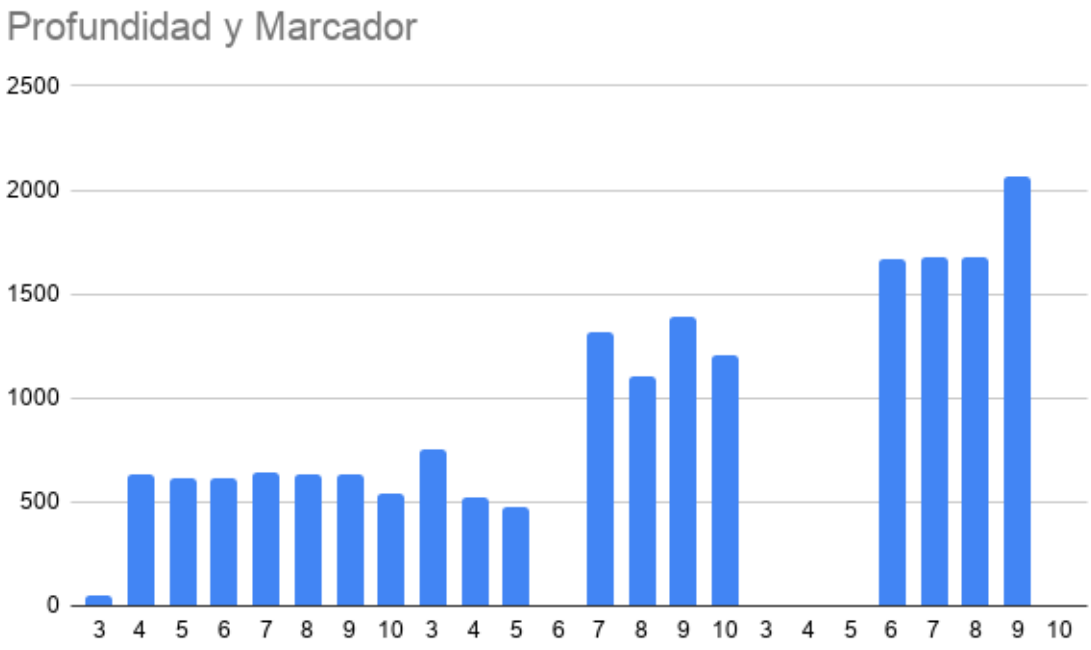

Figura 3: Gráfico con las comparaciones entre profundidad y puntuación.

Bibliografía

[1] ProfessionalAI. (2020, 23 abril). *Alpha Beta Pruning AI*. Recuperado de https://www.professional-ai.com/alpha-beta-pruning.html

[2] GeeksForGeeks. (2021, 21 Jan). *Sum of Manhattan distances between all pairs of points*. Recuperado de https://www.geeksforgeeks.org/sum-manhattan-distances-pairs-points/

[3] junzew. (2017, 15 abril). *rkt-pacman*. Recuperado de https://github.com/junzew/rkt-pacman

[4] Tonypoer. (2016, 28 octubre). *Implementing Minimax and Alpha-Beta Pruning Using Python*. Recuperado de https://tonypoer.io/2016/10/28/implementing-minimax-and-alpha-beta-pruning-using-python/

[5] Krivokuća, M. (s.f.). *Minimax with Alpha-Beta Pruning in Python*. Recuperado de https://stackabuse.com/minimax-and-alpha-beta-pruning-in-python/