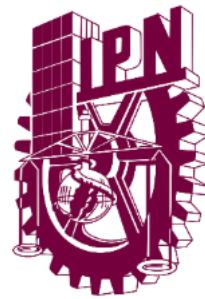




INSTITUTO POLITECNICO  
NACIONAL  
ESCUELA SUPERIOR DE COMPUTO



---

## ALGORITMO MINIMAX Y PODA ALFA BETA

---

ALUMNA:

MIRELES SILVESTRE JOSELYN GUADALUPE



## INTELIGENCIA ARTIFICIAL

GRUPO: 6CV2

NOMBRE DEL PROFESOR: Andrés García Floriano

# Juego de Gato 4x4 con Algoritmo Minimax y Poda Alfa-Beta

## 1. Introducción

El presente proyecto consiste en el desarrollo de un juego de Gato (Tic-Tac-Toe) utilizando un tablero de 4x4. A diferencia del juego tradicional de 3x3, el aumento del tamaño del tablero incrementa significativamente la complejidad del problema, lo que hace necesario el uso de algoritmos de búsqueda optimizados.

Para la toma de decisiones de la computadora se implementó el algoritmo **Minimax con poda alfa-beta**, técnica ampliamente utilizada en el área de Inteligencia Artificial para juegos adversariales.

## 2. Objetivo

Desarrollar un juego de Gato en un tablero de 4x4 que permita enfrentar a un jugador humano contra la computadora, utilizando el algoritmo Minimax con poda alfa-beta, implementado en al menos dos lenguajes de programación.

## 3. Reglas del juego

- El tablero está compuesto por una matriz de 4x4.
- Participan dos jugadores:
  - Jugador humano: X
  - Computadora: O
- Gana el jugador que logre colocar cuatro símbolos iguales de manera consecutiva en:
  - Una fila
  - Una columna
  - Una diagonal
- Si todas las casillas son ocupadas y no existe un ganador, el juego termina en empate.

## 4. Algoritmo Minimax

El algoritmo Minimax es un método de búsqueda que analiza todas las posibles jugadas futuras del juego. Se basa en dos tipos de jugadores:

- **MAX**: intenta maximizar el valor de la jugada (computadora).
- **MIN**: intenta minimizar el valor de la jugada (jugador humano).

Cada estado del juego es evaluado con un valor numérico que representa qué tan favorable es para la computadora.

## 5. Poda Alfa-Beta

La poda alfa-beta es una optimización del algoritmo Minimax que permite reducir el número de nodos evaluados en el árbol de búsqueda. Utiliza dos valores:

- **Alfa ( $\alpha$ )**: el mejor valor encontrado hasta el momento para el jugador MAX.
- **Beta ( $\beta$ )**: el mejor valor encontrado hasta el momento para el jugador MIN.

Cuando se detecta que una rama no puede mejorar el resultado actual, dicha rama es descartada, mejorando considerablemente el rendimiento.

## 6. Optimización para tablero 4x4

Debido al crecimiento exponencial del árbol de búsqueda en un tablero de 4x4, se implementó un **límite de profundidad** junto con una **función heurística** para evaluar estados intermedios del juego.

Esta estrategia permite conservar el uso del algoritmo Minimax con poda alfa-beta sin afectar el desempeño del programa.

## 7. Lenguajes de programación utilizados

- **Python:** implementación del juego en modo consola utilizando Minimax con poda alfa-beta y profundidad limitada.
- **C++:** implementación alternativa del mismo algoritmo, demostrando la portabilidad de la solución.

## 8. Resultados

La computadora es capaz de tomar decisiones competitivas, respondiendo de forma óptima a los movimientos del jugador humano.

Gracias a la poda alfa-beta, el tiempo de respuesta se mantiene adecuado incluso con el aumento de complejidad del tablero.

**Python:**

```

1 import math
2
3 PLAYER = 'X'
4 AI = 'O'
5 EMPTY = ' '
6
7 MAX_DEPTH = 4
8
9 def create_board():
10    return [[EMPTY for _ in range(4)] for _ in range(4)]
11
12 def print_board(board):
13    print("\n")
14    for row in board:
15        print(" | ".join(row))
16    print("-" * 13)
17
18 def check_winner(board):
19    lines = []
20
21    for i in range(4):
22        lines.append(board[i])
23        lines.append([board[j][i] for j in range(4)])
24
25    lines.append([board[i][i] for i in range(4)])
26    lines.append([board[i][3 - i] for i in range(4)])
27
28    for line in lines:
29        if all(cell == AI for cell in line):
30            return AI
31        if all(cell == PLAYER for cell in line):
32            return PLAYER
33
34    if all(board[i][j] != EMPTY for i in range(4) for j in range(4)):
35        return "EMPATE"
36
37    return None
38
39 def heuristic(board):
40    score = 0
41    lines = []
42
43    for i in range(4):
44        lines.append(board[i])
45        lines.append([board[j][i] for j in range(4)])
46
47    lines.append([board[i][i] for i in range(4)])
48    lines.append([board[i][3 - i] for i in range(4)])
49
50    for line in lines:
51        if PLAYER not in line:
52            score += line.count(AI)
53        if AI not in line:
54            score -= line.count(PLAYER)
55
56    return score
57
58 def minimax(board, depth, alpha, beta, is_max):
59    result = check_winner(board)
60    if result == AI:
61        return 100
62    if result == PLAYER:
63        return -100
64    if result == "EMPATE":
65        return 0
66    if depth == MAX_DEPTH:
67        return heuristic(board)
68
```

```

58 def minimax(board, depth, alpha, beta, is_max):
59     if is_max:
60         best = -math.inf
61         for i in range(4):
62             for j in range(4):
63                 if board[i][j] == EMPTY:
64                     board[i][j] = AI
65                     value = minimax(board, depth + 1, alpha, beta, False)
66                     board[i][j] = EMPTY
67                     best = max(best, value)
68                     alpha = max(alpha, best)
69                     if beta <= alpha:
70                         break
71             return best
72     else:
73         best = math.inf
74         for i in range(4):
75             for j in range(4):
76                 if board[i][j] == EMPTY:
77                     board[i][j] = PLAYER
78                     value = minimax(board, depth + 1, alpha, beta, True)
79                     board[i][j] = EMPTY
80                     best = min(best, value)
81                     beta = min(beta, best)
82                     if beta <= alpha:
83                         break
84             return best
85
86 def best_move(board):
87     best_score = -math.inf
88     move = None
89
90     for i in range(4):
91         for j in range(4):
92             if board[i][j] == EMPTY:
93                 board[i][j] = AI
94                 score = minimax(board, 0, -math.inf, math.inf, False)
95                 board[i][j] = EMPTY
96                 if score > best_score:
97                     best_score = score
98                     move = (i, j)
99
100    return move
101
102 def main():
103     board = create_board()
104
105     while True:
106         print_board(board)
107
108         r, c = map(int, input("Ingresa fila y columna (0-3): ").split())
109         if board[r][c] != EMPTY:
110             print("Casilla ocupada")
111             continue
112
113         board[r][c] = PLAYER
114
115         if check_winner(board):
116             break
117
118         ai = best_move(board)
119         board[ai[0]][ai[1]] = AI
120
121         if check_winner(board):
122             break
123
124         print_board(board)

```

```
111
112 def main():
113     board = create_board()
114
115     while True:
116         print_board(board)
117
118         r, c = map(int, input("Ingresa fila y columna (0-3): ").split())
119         if board[r][c] != EMPTY:
120             print("Casilla ocupada")
121             continue
122
123         board[r][c] = PLAYER
124
125         if check_winner(board):
126             break
127
128         ai = best_move(board)
129         board[ai[0]][ai[1]] = AI
130
131         if check_winner(board):
132             break
133
134     print_board(board)
135     print("Resultado:", check_winner(board))
136
137 if __name__ == "__main__":
138     main()
```


Ingresa fila y columna (0-3): 0 0

X		0		

Ingresa fila y columna (0-3): 1 0

X		0		
X			0	

Ingresa fila y columna (0-3): 3 0

X		0		
X			0	
0				
X				

Ingresa fila y columna (0-3): 3 3

X		0		0
X			0	
0				
X				X

```
X |   |   | X
```

```
-----  
Ingresa fila y columna (0-3): 0 3  
Casilla ocupada
```

```
X | 0 |   | 0
```

```
-----  
X | 0 | 0 |
```

```
-----  
0 |   | X |
```

```
-----  
X |   |   | X
```

```
-----  
Ingresa fila y columna (0-3): 0 2
```

```
X | 0 | X | 0
```

```
-----  
X | 0 | 0 | 0
```

```
-----  
0 |   | X |
```

```
-----  
X |   |   | X
```

```
-----  
Ingresa fila y columna (0-3): 1 3  
Casilla ocupada
```

```
X | 0 | X | 0
```

```
-----  
X | 0 | 0 | 0
```

```
-----  
0 |   | X |
```

```
-----  
X |   |   | X
```

```
-----  
Ingresa fila y columna (0-3): 2 3
```

```
X | 0 | X | 0
```

```
-----  
X | 0 | 0 | 0
```

```
-----  
0 | 0 | X | X
```

```
-----  
X |   |   | X
```

```
-----  
Ingresa fila y columna (0-3): 3 2
```

```
X | 0 | X | 0
```

```
-----  
X | 0 | 0 | 0
```

```
-----  
0 | 0 | X | X
```

```
-----  
X | 0 | X | X
```

```
-----  
Resultado: 0
```

```
C:\Users\asus\Desktop\Matriz\Matriz1.c
```

## C++:

```
lime > joss > Escrevendo > IA > gato4x4.cpp > ...
1  #include <iostream>
2  #include <vector>
3  #include <limits>
4
5  using namespace std;
6
7  const char PLAYER = 'X';
8  const char AI = 'O';
9  const char EMPTY = ' ';
10 const int SIZE = 4;
11 const int MAX_DEPTH = 4;
12
13 vector<vector<char>> board(SIZE, vector<char>(SIZE, EMPTY));
14
15 void printBoard() {
16     cout << "\n";
17     for (int i = 0; i < SIZE; i++) {
18         for (int j = 0; j < SIZE; j++) {
19             cout << board[i][j];
20             if (j < SIZE - 1) cout << " | ";
21         }
22         cout << "\n";
23         cout << "-----\n";
24     }
25 }
26
27 char checkWinner() {
28     for (int i = 0; i < SIZE; i++) {
29         if (board[i][0] != EMPTY &&
30             board[i][0] == board[i][1] &&
31             board[i][1] == board[i][2] &&
32             board[i][2] == board[i][3])
33             return board[i][0];
34
35         if (board[0][i] != EMPTY &&
36             board[0][i] == board[1][i] &&
37             board[1][i] == board[2][i] &&
38             board[2][i] == board[3][i])
39             return board[0][i];
40     }
41
42     if (board[0][0] != EMPTY &&
43         board[0][0] == board[1][1] &&
44         board[1][1] == board[2][2] &&
45         board[2][2] == board[3][3])
46         return board[0][0];
47
48     if (board[0][3] != EMPTY &&
49         board[0][3] == board[1][2] &&
50         board[1][2] == board[2][1] &&
51         board[2][1] == board[3][0])
52         return board[0][3];
53
54     return ' ';
55 }
```

```

56
57     int heuristic() {
58         int score = 0;
59
60         for (int i = 0; i < SIZE; i++) {
61             int aiCount = 0, playerCount = 0;
62             for (int j = 0; j < SIZE; j++) {
63                 if (board[i][j] == AI) aiCount++;
64                 if (board[i][j] == PLAYER) playerCount++;
65             }
66             if (playerCount == 0) score += aiCount;
67             if (aiCount == 0) score -= playerCount;
68         }
69         return score;
70     }
71
72     int minimax(int depth, bool isMax, int alpha, int beta) {
73         char winner = checkWinner();
74         if (winner == AI) return 100 - depth;
75         if (winner == PLAYER) return depth - 100;
76         if (depth == MAX_DEPTH) return heuristic();
77
78         bool movesLeft = false;
79         for (auto &r : board)
80             for (char c : r)
81                 if (c == EMPTY) movesLeft = true;
82
83         if (!movesLeft) return 0;
84
85         if (isMax) {
86             int best = numeric_limits<int>::min();
87             for (int i = 0; i < SIZE; i++) {
88                 for (int j = 0; j < SIZE; j++) {
89                     if (board[i][j] == EMPTY) {
90                         board[i][j] = AI;
91                         int value = minimax(depth + 1, false, alpha, beta);
92                         board[i][j] = EMPTY;
93                         best = max(best, value);
94                         alpha = max(alpha, best);
95                         if (beta <= alpha) return best;
96                     }
97                 }
98             }
99             return best;
100        } else {
101            int best = numeric_limits<int>::max();
102            for (int i = 0; i < SIZE; i++) {
103                for (int j = 0; j < SIZE; j++) {
104                    if (board[i][j] == EMPTY) {
105                        board[i][j] = PLAYER;
106                        int value = minimax(depth + 1, true, alpha, beta);
107                        board[i][j] = EMPTY;
108                        best = min(best, value);
109                        beta = min(beta, best);
110                        if (beta <= alpha) return best;
111                    }
112                }
113            }
114            return best;
115        }
116    }
117

```

```

115     }
116 }
117
118 pair<int, int> bestMove() {
119     int bestScore = numeric_limits<int>::min();
120     pair<int, int> move = {-1, -1};
121
122     for (int i = 0; i < SIZE; i++) {
123         for (int j = 0; j < SIZE; j++) {
124             if (board[i][j] == EMPTY) {
125                 board[i][j] = AI;
126                 int score = minimax(0, false,
127                                     numeric_limits<int>::min(),
128                                     numeric_limits<int>::max());
129                 board[i][j] = EMPTY;
130
131                 if (score > bestScore) {
132                     bestScore = score;
133                     move = {i, j};
134                 }
135             }
136         }
137     }
138     return move;
139 }
140
141 int main() {
142     while (true) {
143         printBoard();
144
145         int r, c;
146         cout << "Ingresa fila y columna (0-3): ";
147         cin >> r >> c;
148
149         if (r < 0 || r >= SIZE || c < 0 || c >= SIZE || board[r][c] != EMPTY) {
150             cout << "Movimiento invalido\n";
151             continue;
152         }
153
154         board[r][c] = PLAYER;
155
156         if (checkWinner() != ' ') break;
157
158         auto ai = bestMove();
159         board[ai.first][ai.second] = AI;
160
161         if (checkWinner() != ' ') break;
162     }
163
164     printBoard();
165     char winner = checkWinner();
166     if (winner == PLAYER) cout << "Ganaste\n";
167     else if (winner == AI) cout << "Gana la computadora\n";
168     else cout << "Empate\n";
169
170     return 0;
171 }
172

```