***Connect Four AI Notebook***

1. **Introdução**

Este documento descreve a implementação em Python de um projeto de Inteligência Artificial do jogo Connect Four com três modos de jogo:

* Humano vs Humano
* Humano vs Computador (usando MCTS)
* Computador vs Computador (MCTS vs ID3)

O código utiliza Monte Carlo Tree Search (MCTS) para tomar decisões e gerar datasets para treinar uma árvore de decisão ID3 utilizando o cálculo de avaliação UCB1.

1. **Estrutura do código**

-Funções de UI (print\_board, show\_menu)

- TAD ConnectFourState e lógica de jogo

- TAD Node (uct\_search)

-TAD ID3Tree ()

- Dataset (generateDataset)

- Modos de Jogo (human\_vs\_human, human\_vs\_pc, pc\_vs\_pc)

- Main (execução principal)

1. **Detalhamento de Classes e Funções**
   1. **Classe ConnectFourState**

class ConnectFourState:

def \_\_init\_\_(self):

self.board = create\_board() # Tabuleiro 6x7

self.current\_player = 'X' # Jogador atual

self.last\_move = None # Última jogada

# Principais Métdos:

- clone(): Cria cópia do estado.

- get\_legal\_moves(): Retorna colunas não preenchidas.

- do\_move(col): Executa jogada na coluna especificada.

- is\_terminal(): Verifica fim do jogo.

- get\_winner(): Checa vitória horizontal/vertical/diagonal.

**3.2 Classe Node**

class Node:

def \_\_init\_\_(self, state, parent=None):

self.state = state # Estado do jogo

self.parent = parent # Nó pai

self.children = [] # Nós filhos

self.visits = 0 # Número de visitas

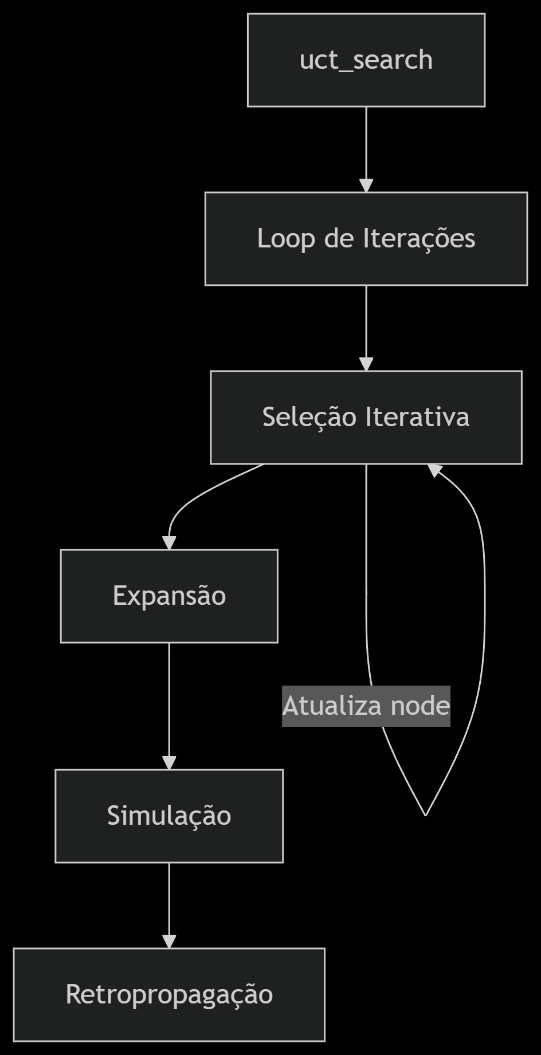
self.total\_value = 0 # Valor acumulado

# Métodos principais:

- select\_child(): Seleciona filho via UCB1.

- add\_child(): Adiciona novo nó filho.

* 1. **Função uct\_search**

****

Implementa o algoritmo MCTS em quatro fases:

1. Selection: Navega pela árvore usando UCB1.
2. Expansion: Adiciona novos nós à árvore.
3. Simulation: Joga aleatoriamente até o fim.
4. Backpropagation: Atualiza estatísticas dos nós.
5. **Implementação do MCTS**

**4.1 Estratégia de seleção**

def select\_child(self):

exploration\_constant = math.sqrt(2)

return max(self.children, key=lambda child: (child.total\_value / child.visits if child.visits > 0 else float('inf')) + exploration\_constant \* math.sqrt(math.log(self.visits) / (child.visits if child.visits > 0 else 1)))

* Fórmula UCB1: Balanceia exploração e exploração.
* Tratamento de nós não visitados: Atribui float('inf') para garantir prioridade.
* Utilizamos uma função anônima para calcular o UCB1 definindo a chave de comparação do max() e retornar o filho com maior avaliação.

**4.2 Gerador de Dataset**

def generateDataset(num\_games=100, iterations\_per\_move=1000, filename='MCTS\_dataset.csv'):

    dataset = []

    for \_ in range(num\_games):

        state = ConnectFourState()

        while not state.is\_terminal():

            board\_state = ''.join([''.join(row) for row in state.board])

            best\_move = uct\_search(state, iterations\_per\_move)

            dataset.append({'state': board\_state, 'move': best\_move})

            state = state.do\_move(best\_move)

    with open(filename, 'w', newline='') as csvfile:

        fieldnames = ['state', 'move']

        writer = csv.DictWriter(csvfile, fieldnames)

        writer.writeheader()

        writer.writerows(dataset)

    return dataset

* Salva os pares (state, best\_move) num arquivo .csv
* A representação do state em array vem a ser útil para não usar tanta memória.

1. **ID3Tree Procedure**
2. **Fluxo de Execução**

**Uma imagem com captura de ecrã, texto, diagrama, design

Os conteúdos gerados por IA poderão estar incorretos.**