Guião: vídeo do projeto IA (Connected Four)

1. Abertura e Motivação (0:00 – 0:20)

- Introdução do projeto

Um jogador de Quatro-em-Linha precisa de analisar centenas de milhões de posições para se tornar invencível. Nós testámos duas abordagens — ainda longe dessa perfeição —: Pesquisa em Árvore de Monte Carlo e uma Árvore de Decisão. O objetivo deste projeto é compreender o funcionamento, as vantagens e as limitações de cada abordagem na resolução de jogos com imensas decisões por tomar.

2. Definição do Problema (0:20 – 1:20)

- Explicação do jogo

O Quatro-em-Linha é um jogo de estratégia para dois jogadores, jogado num tabuleiro vertical de 6 linhas por 7 colunas (42 casas). A cada jogada, um jogador escolhe uma coluna; a sua peça “cai” até ocupar a linha mais baixa disponível. Ganha quem conseguir alinhar quatro peças consecutivas — horizontal, vertical ou diagonal.

- Explicação do código das regras de jogo

Para implementar as regras do jogo em Python, foram criadas 3 classes:

A classe GameMeta:

- Contém constantes do jogo, como os jogadores, resultados e tamanho do tabuleiro.

A classe MCTSMeta:

- Define o coeficiente de exploração.

A classe ConnectedState:

- Inicializa o estado de um jogo, com o preenchimento inicial do tabuleiro, a escolha do primeiro jogador, contador de jogadas, vencedor, etc.

- A parte disso, contém funções que sustentam o jogo. Por exemplo, a função move, que executa um movimento numa coluna escolhida e atualiza o estado de jogo; a função get\_legal\_moves, que retorna uma lista das colunas onde ainda é possível jogar; e a função check\_win, que verifica se o jogador atual venceu.

3. MCTS (1:20 – 2:20)

- Explicação do código

- Inclusão de teoria (Fases do MCTS, UCT)

O nosso código implementa as 4 fases do MCTS:

- Seleção (função select\_node()): percorre a árvore da raiz até uma folha, usando a fórmula de Upper Confidence Bound para escolher sempre o filho que a maximize;

- Expansão (função expand()): cria nós-filho na árvore sempre que encontra um estado não totalmente explorado;

- Simulação (função rollout()): joga aleatoriamente até ao fim, retornando o resultado;

- E retro propagação (função backpropagate()): atualiza recursivamente as estatísticas de visitas e vitórias, de folha à raiz.

- Para equilibrar a exploration e exploitation, usámos a função get\_exploration, para calcular dinamicamente o coeficiente de exploração. Assim, dependendo do número de visitas do nó pai, o valor pode diminuir, levando a uma exploração maior no início e mais leve no final.

- Cada nó é representado por uma instância da classe Node.

- Por fim, no método search() repetimos as quatro etapas do MCTS até esgotar o tempo limite fornecido.

4. ID3/ Árvore de Decisão (2:20-3:20)

- Menção rápida do dataset Iris

Para validar a nossa implementação do ID3, utilizamos inicialmente o dataset Iris, onde discretizámos variáveis contínuas e treinámos a árvore de pesquisa.

- Gerar dataset

De seguida, gerámos um dataset de jogadas do Quatro em linha usando o MCTS. Este recolhia pares (estado, movimento) para ensinar a árvore de decisão a prever as jogadas.

- Explicação do código do ID3

- Inclusão de teoria (Cálculos da árvore de decisão)

O nosso Algoritmo de Dichotomização Iterativa 3 (ID3) está organizado em 3 passos:

- A entropia, para um no com rótulos y

- O ganho de informação para cada feature f, onde é selecionada a feature que maximiza esse ganho

- Recursão: dividimos o conjunto pelo valor escolhido e repetimos até aos rótulos serem puros ou atingirmos profundidade máxima

A função id3() implementa essa recursão e a função predict() percorre a árvore para rotular novos estados.

5. Resultados (3:20-4:00)

- Mostrar células e explicar jogadas

- Desempenho do MCTS

- Não jogar um jogo inteiro, mas fazer algumas jogadas para demonstração ao vivo

6. Discussão de Resultados (4:00 – 4:45)

- Pontos fortes e fracos de cada algoritmo e sua comparação

- Análise de gráficos do notebook

7. Conclusão (4:45 – ~5:00)

- Conclusão breve