**Guião: vídeo do projeto IA (Connected Four)**

1. **Abertura e Motivação (0:00 – 0:20) [Gonçalo]**

[introdução]

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) [Gonçalo]**

[descrição do problema]

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.

[connected\_four.py]

As regras de jogo foram implementadas em 3 classes:

A classe GameMeta: [classe]

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

A classe MCTSMeta: [classe]

- Define o coeficiente de exploração.

A classe ConnectState: [classe]

- 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 [ver função], que executa um movimento numa coluna escolhida e atualiza o estado de jogo; a função get\_legal\_moves [ver função], que retorna uma lista das colunas onde ainda é possível jogar; e a função check\_win [ver função], que verifica se o jogador atual venceu.

1. **MCTS (1:20 – 2:20) [Gil]**

[capítulo 3]

O MCTS é um algoritmo de busca que utiliza o critério Upper Bound Confidence para avaliar cada ramo de uma árvore [mostrar fórmula]

O nosso código implementa as 4 fases do MCTS: seleção [select\_node()], expansão [expand()], simulação [rollout()] e retro propagação [back\_propagate()], usadas assim para cada ciclo da função search() [função] que repete as etapas até ao tempo limite do mesmo.

- Para equilibrar a exploration e exploitation, usámos a função get\_exploration() [função], 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 [classe].

**4. ID3/ Árvore de Decisão (2:20-3:20) [Gil]**

[capítulo 3.2. geração do dataset, função generate\_dataset()]

Para a segunda abordagem foi treinada uma árvore de decisão com um dataset de jogadas do quatro em linha usando o mcts.

Este recolheu pares (estado, movimento) para ensinar a árvore a prever as jogadas.

[decision\_tree\_builder.py]

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

- A entropia [fórmula da entropia e função]

- O ganho de informação para cada feature f, onde é selecionada a feature que maximiza esse ganho [fórmula de ganho e função]

- Recursão [função id3 -> linha child = id3(...)]

[função id3 no geral]

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

[iris\_test.py, dando scroll]

Validámos a nossa implementação do ID3, usando o dataset Iris, onde discretizámos variáveis contínuas e treinámos a árvore de pesquisa.

**5. Resultados (3:20-4:00) [Rita]**

[ game.py, mostrar as funções de cada modo]

Após isto, já é possível jogar o jogo completo, no qual foram criadas 4 funções/modos: jogador contra jogador; jogador contra mcts; jogador contra árvore e mcts contra árvore.

[arvore contra mcts-> improviso]

O MCTS processa 77 rollouts por segundo, tem um tempo médio de decisão de aproximadamente 3 segundos e um win rate de 100% contra uma árvore de decisão em 3 jogos. [mostrar cada função que gera os resultados]

**6. Discussão de Resultados (4:00 – 4:45) [Rita]**

[gráfico de barras do número de jogadas]

Para além dos aspetos falados anteriormente, também é possível analisar que o MCTS tende a escolher colunas centrais, especialmente a coluna 3. Isto reflete a estratégia de controlo do centro, pois maximiza o número de linhas de quatro possíveis e permite flexibilidade. No entanto, causa um viés no modelo ID3, que vai aprender melhor a coluna 3 do que o resto das amostras, baixando assim a acurácia.

[tabela de comparação]

No geral, cada abordagem tem os seus pontos fortes e fracos.

O MCTS é adaptável a diferentes jogos, é dinâmico, mas tem um alto custo computacional. Já a árvore de decisão é rápida em termos de previsão, mas tem uma baixa robustez em cenários de alta dimensão.

Em cenários simples o ID3 consegue competir razoavelmente bem, mas em cenários complexos o MCTS supera-a.

**7. Conclusão (4:45 – ~5:00) [Rita]**

[ conclusão]

Conclui-se assim que o ID3 é útil para demonstrar como a inteligência artificial pode tomar decisões com base em exemplos passados, mas não reage bem a situações fora do padrão. O MCTS pelo contrário, é mais adaptável e inteligente em tempo real.