Poda α-β aplicada em *gomoku*

Emmanuel Podestá Jr., Gustavo Zambonin*
Inteligência Artificial (UFSC – INE5430)

- **Funções de heurística e utilidade**

  A função heurística $H(T)$, onde T é um nodo do grafo que caracteriza um tabuleiro do jogo *gomoku*, foi definida como a soma da quantidade de n-uplas, $n \in \{2, 3, 4\}$. Tomando v_n como cada uma destas somas, é possível calcular v_{n+1}, assumindo alguns fatos na construção do raciocínio:

  – $v_{n+1} > \sum v_n$, para que o algoritmo sempre busque construir as maiores n-uplas;
  – um valor inicial para v_2, a menor n-upla que a função considera;
  – existem $15 - n + 1$ possíveis arranjos de elementos consecutivos com o mesmo valor em uma palavra binária de tamanho 15;
  – $\lceil \frac{15^2}{2} \rceil = 113$ é o maior número de peças de um jogador num tabuleiro, o que implica $\frac{113}{15} \approx 7.5$ linhas de peças iguais;
  – e existem 3 orientações onde um arranjo pode ser considerado como n-upla.

  Então, $v_{n+1} = \sum v_n \times 3 \times 7.5 \times 113 \times (15 - n + 1)$, e a função heurística é definida como $H(T) = \sum_{n=2}^{4} v_n$.

  A função de utilidade foi definida de forma semelhante, levando em conta a dificuldade de representar todos os nodos possíveis do grafo em virtude da quantidade de tabuleiros possíveis. Dessa forma, foi necessário especificar uma profundidade limite para o algoritmo minimax. Este fato dificulta a introdução de um fator heurístico relacionado à profundidade do grafo, pois seria custoso armazenar e processar tal informação ao longo da execução do algoritmo. Então, tomando $U(T)$ como a função utilidade, estendeu-se a análise de n-uplas até $n = 5$, o que significa $U(T) = \sum_{n=2}^{5} v_n$.

  Modificações importantes na heurística ainda devem ser inseridas, como avaliar bloqueio de n-uplas por peças do adversário, e verificar casos onde existem grupos separados de n-uplas que podem levar a uma quíntupla com apenas uma jogada.

- **Otimizações e estratégias**

  O algoritmo minimax com poda α-β foi utilizado para buscar melhores jogadas. A estrutura de dados que representa o tabuleiro é uma lista de listas simples, onde cada elemento pode ser um inteiro em $0, 1, -1$: respectivamente espaço vazio e dois jogadores. Dado uma configuração de entrada, o algoritmo verificará possíveis movimentos ao redor das peças já inseridas no tabuleiro, armazenará os possíveis movimentos em uma lista, e vários cenários de jogadas com cada uma dessas coordenadas serão testados, recursivamente. A avaliação de melhor jogada será feita sobre os valores de cada um dos tabuleiros. Após o processo de verificação e poda, a melhor jogada escolhida, juntamente com sua pontuação, é retornada para que o algoritmo possa proceder.

- **Detecção de fim de jogo e sequências**

  A detecção de fim de jogo é feita a partir de um método `victory`, que percorre toda a representação matricial do tabuleiro procurando por quíntuplas da mesma cor. Após cada jogada, essa verificação é aplicada e $U(T)$ é chamada caso o resultado seja verdadeiro. De modo similar, sequências são verificadas com uma variante do método acima que observa apenas n-uplas de um jogador e tamanho específicos. O funcionamento específico tem base em `itertools.groupby` da linguagem Python, que agrupa símbolos semelhantes e sua quantidade em um objeto iterável, como uma lista.

*`{emmanuel.podesta,gustavo.zambonin}@grad.ufsc.br` — todos os algoritmos utilizados podem ser encontrados também neste repositório.