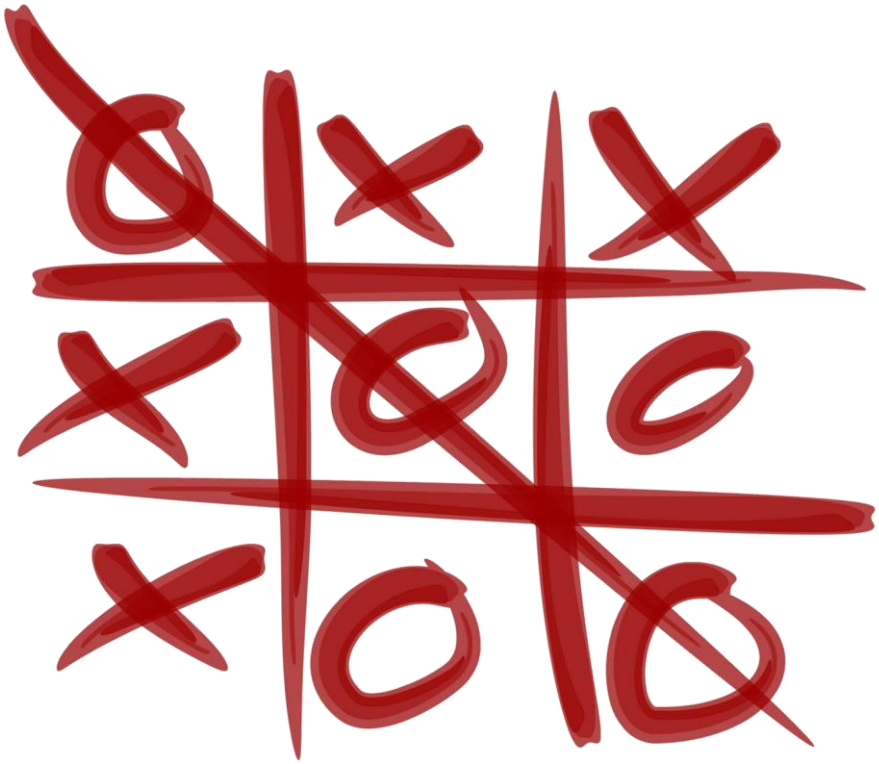
U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

INTELIGÊNCIA ARTIFICIAL
ABRIL 2016



ALGORITMOS MIN-MAX E ALFA-BETA NO "JOGO DO GALO"

Ricardo Dias Azevedo | up201403133
Rodrigo Flamínio Ribeiro | up201407863

Índice

Introdução

Um jogo pode ser definido, de forma simples, como um problema de procura constituído por alguns elementos, entre os quais se destacam: **estado inicial** (S_0), indicativo da configuração do jogo aquando do início, acompanhado da decisão de quem inicia o jogo, **função para gerar sucessores**, que indica as jogadas possíveis para um dado jogador num determinado estado, **teste de terminação**, que se constitui verdadeiro quando o jogo termina e falso na situação contrária, sendo os estados em que o jogo acaba designados por "estados terminais" e **função *utilidade***, capaz de atribuir um valor numérico ao jogo, ou seja, vitória, derrota ou empate [1].

De notar que, enquanto ambiente competitivo, um jogo com oponentes pressupõe a existência de um número de jogadores maior ou igual a dois e introduz uma variável significativa: a **incerteza.** Tal ocorre pelo facto de não possuirmos a capacidade de prever as jogadas do nosso adversário. Deste modo, apesar de se constituir como um problema de procura, não o faz de forma habitual, uma vez que estes não consideram a presença de um oponente. Por outro lado, este tipo de problema requer ainda um espaço de busca significativamente maior e o tempo para cada jogada reveste-se de particular importância [2]. Assim, podemos considerar que o impacto do outro é de extrema importância na nossa decisão de jogo.

No sentido da resolução destes jogos deparamo-nos, essencialmente, com dois grandes algoritmos: Algoritmo Min-Max e Algoritmo Alfa-Beta.

Algoritmo Min-Max

O algoritmo Min-Max corresponde a uma função recursiva, cujo principal objetivo é minimizar a derrota, maximizando a possibilidade de vitória [1].

Inicialmente aplicado em jogos *Zero-Sum* para dois jogadores (jogos em que a vitória de um traduz, necessariamente, a derrota de outro) encontra-se, atualmente, aplicado em situações bastante mais complexas.

Tendo por base uma heurística capaz de determinar a utilidade de cada jogada, este algoritmo implementa um valor minmax em cada possível estado [1], sendo Min a representação do segundo jogador e Max o jogador que inicia o jogo.

Iniciando o jogo no nível máximo, o presente algoritmo irá alternar entre o nível mínimo e o nível máximo, consoante estamos perante a jogado do adversário ou de quem chamou a função, respetivamente. Assim, vai realizando uma pesquisa em profundidade que, quando atinge um estado final, determina a sua utilidade. Este é comparado com o nó pai e, caso o nó pai seja o Min, este adotará o valor mais baixo entre o do nó pai e o da utilidade. Caso o pai se trate do Max, irá desejar o valor mais alto entre os dois (figura 1). De notar que este processo é realizado para todos os filhos desse nó pai, no sentido de permitir ao iniciador a melhor jogada possível para possibilitar ao máximo a vitória [1].

$$\text{MINIMAX-VALUE(n)} = \begin{cases} UTILITY(n) & \text{if } n \text{ is a terminal state} \\ max_{s \in \text{successors(n)}} \text{MINIMAX-VALUE(s)} & \text{if } n \text{ is a MAX node} \\ min_{s \in \text{successors(n)}} \text{MINIMAX-VALUE(s)} & \text{if } n \text{ is a MIN node} \end{cases}$$

Figura 1 – Função Min-Max [1]

Uma vez que este algoritmo realiza uma busca completa em profundidade, podemos admitir que, sendo **m** a profundidade máxima da nossa árvore e **b** todas as possíveis jogadas em cada ponto, a complexidade tempo temporal deste algoritmo pode definir-se como $\mathbf{O(b^m)}$. Por sua vez, a complexidade espacial pode definir-se por **O(bm)** [caso este algoritmo gere todas as ações em simultâneo], ou **O(m)** [quando as ações são geradas uma de cada vez] [1].

Algoritmo Alfa-Beta

De uma forma geral, este algoritmo pode definir-se como uma otimização do algoritmo Min-Max, na medida em que se implementa e atua de forma semelhante a este, mas tem a particularidade de solucionar o problema que lhe estava associado: quantidade exponencial de estados a examinar na profundidade da árvore de busca [1]. Assim, com este algoritmo tornou-se possível computar a solução correta, sem a necessidade de examinar todos os possíveis nós. Deste modo, quando aplicado a uma árvore *standard* de Min-Max, este algoritmo retorna o mesmo movimento que Min-Max, mas suprime ramos que não influenciam a decisão final [1], consumindo, por isso, menos tempo e memória.

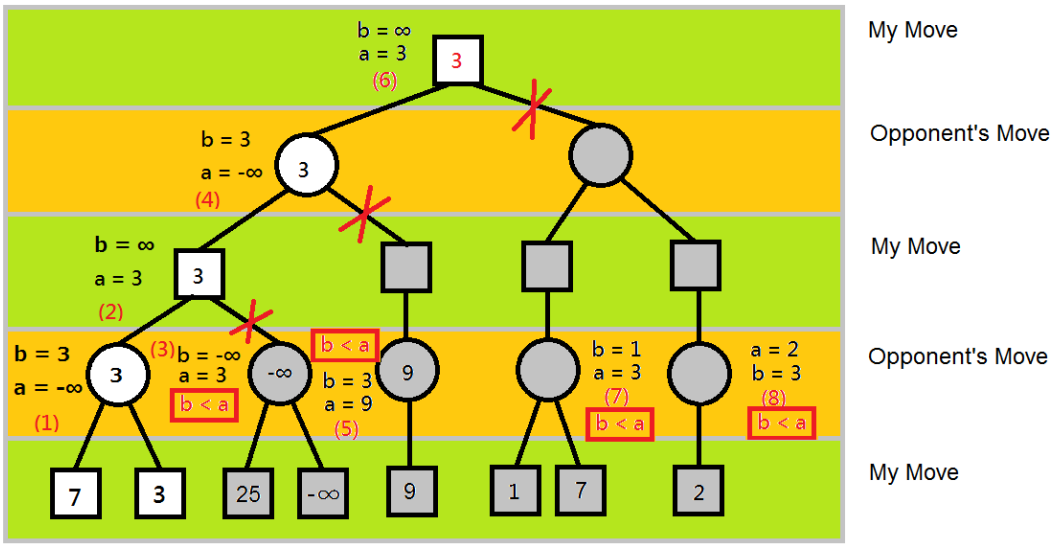


Figura 2 – Pesquisa Alfa-Beta [3]

Este algoritmo é definido por duas variáveis [1]:

- ✓ Alfa – O valor da melhor opção que encontramos até ao momento ao longo do caminho para Max;
- ✓ Beta – O valor da melhor opção que encontramos até ao momento ao longo do caminho para Min.

Assim, sempre que alfa for superior ou igual a beta o presente nó não será expandido, em prol de um mais favorável já encontrado [1].

Jogo do Galo

O Jogo do Galo consiste num jogo de oponentes para dois jogadores, sendo um representado por X e outro por O. Consiste numa matriz 3x3, sendo o objetivo ter três Xs ou Os em linha, sejam elas horizontais, verticais ou diagonais.

De notar que este jogo não é tão linear quanto a dualidade vitória/derrota, havendo margem para empate.

Min-Max e Alfa-Beta Aplicados ao Jogo do Galo

A nossa implementação dos algoritmos Min-Max e Alfa-Beta no Jogo do Galo teve por base a criação de uma função utilidade que, sempre que a matriz de jogo atingisse um estado final, fosse capaz de atribuir um valor a esse estado, que poderia variar entre -1 (perder), 0 (empatar) e +1 (ganhar) (figura 3). A estratégia de exploração da árvore teve por base a implementação, ao nível dos algoritmos, de um método de procura em profundidade recursivo. Com a utilização do algoritmo Alfa-Beta foram cortadas todas as porções da árvore de procura para as quais **$\alpha \geq \beta$**.

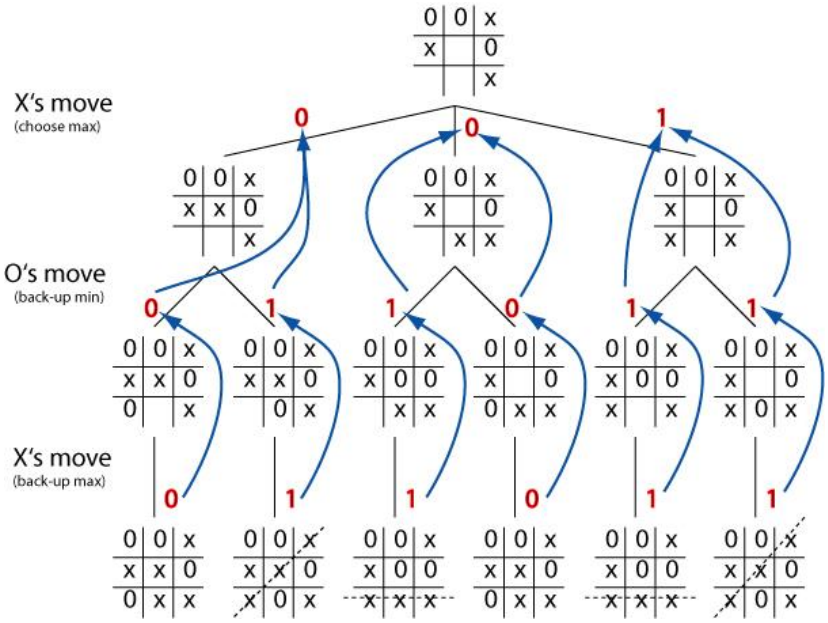


Figura 3 – Função utilidade aplicada ao Jogo do Galo [4]

Quanto às estruturas de dados utilizadas, recorremos a um *array* para memorizar as várias configurações do jogo, sendo este capaz de percorrer todos os nós da árvore visitados.

No que respeita os resultados, verificamos que o algoritmo Min-Max correu um número de nós significativamente maior quando comparado com Alfa-Beta. De notar que, devido à linguagem por nós utilizada (C++), apenas nos foi possível determinar o tempo total que o programa demora a correr, sem nos ser possível

individualizar o tempo necessário a cada jogada em função do algoritmo. Assim, o tempo total no jogo computador *vs* computador com os dois algoritmos foi: 0.058s para Min-Max e 0.004s para Alfa-Beta.

Em seguida encontram-se, esquematicamente representados, os resultados obtidos, considerando o número de nós expandidos pelo computador, a cada jogada, em função do algoritmo utilizado:

Jogada	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Min-Max	549946	59705	7332	935	198	47	14	5	2
Alfa-Beta	18311	2352	844	75	64	17	10	5	2

Tabela 1 – Quadro representativo do número de nós expandidos pelo computador de acordo com o algoritmo.

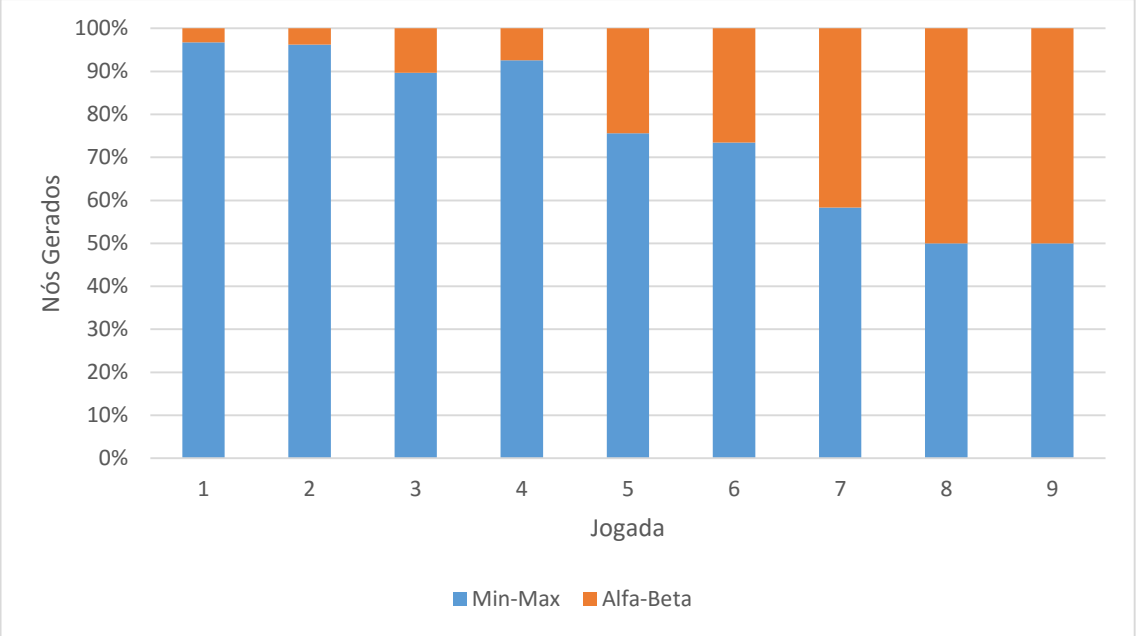


Gráfico 1 – Representação gráfica da tabela 1.

Conclusões

Após uma cuidada avaliação dos resultados obtidos, podemos afirmar que o algoritmo Alfa-Beta, enquanto otimização de Min-Max, foi claramente superior a este em termos de eficácia, uma vez que, por gerar menos nós, permite uma poupança de tempo e memória.

Assim, podemos seguramente afirmar que o algoritmo Alfa-Beta se revelou como o mais eficiente.

Bibliografia

[1] Russell, Stuarrt J.; Norvig, Peter, 2009. *Artificial Intelligence a Modern Aproach*. 3ª edição, Pearson Education, Inc., 13-978-0-13-604259-4

[2] Aula Teórica *Jogos com Oponentes*, Prof. Inês Dutra

[3] https://pjdelta.wordpress.com/2015/06/20/computer-chess-and-alpha-beta-pruning/

[4] http://snipd.net/minimax-algorithm-with-alpha-beta-pruning-in-c