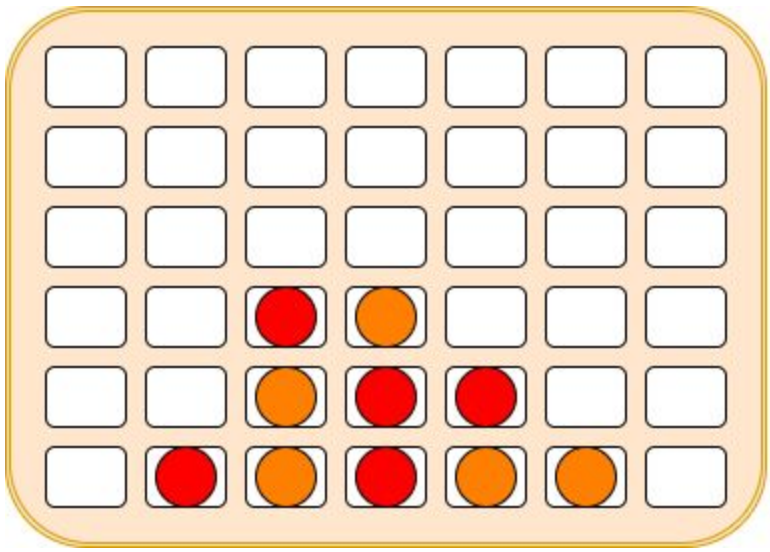
U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Connect Four



Ana Coutinho up200303059
Ana Germano up201105083

Inteligência Artificial
2016/2017

Índice

Introdução

O objectivo do presente trabalho é aprendermos a desenhar e implementar um programa que seja capaz de jogar o jogo *Connect Four* (em português quatro em linha). Para tal começamos por implementar uma versão mais simples do jogo, que permitia que duas pessoas jogassem uma contra a outra. A partir daqui tentamos desenvolver uma inteligência artificial que permita ao computador jogar contra uma pessoa, devendo o computador no mínimo conseguir um empate e nunca uma derrota.

Para a implementação da inteligência artificial necessária nós utilizamos os algoritmos de *Minimax* e *Alpha Beta Prunning* dados nas aulas. Estes são algoritmos de procura adversarial, muito utilizados em jogos como o jogo do galo ou o xadrez. O uso deste tipo de algoritmos prende-se com o facto de que nos permitem simular as jogadas possíveis e respectivas respostas e contra-respostas de forma a podermos escolher a jogada com mais chances de nos permitir ganhar.

Ao contrário dos algoritmos de pesquisa cega e de pesquisa guiada, estes algoritmos têm em conta a imprevisibilidade gerada pela existência de um oponente que o computador não sabe como vai jogar. É desta imprevisibilidade que resulta a necessidade de simular todas as possibilidades, não só das jogadas possíveis do computador mas também das do oponente.

Para conseguirmos usar este tipo de estratégia vamos usar uma árvore de pesquisa, em que cada nó vai representar um tabuleiro de jogo após uma jogada. Cada nó vai ter também associada uma utilidade e uma profundidade. [1]

Para podermos concluir se uma determinada jogada poderá dar origem a uma vitória, empate ou derrota, vamos ter que calcular a sua utilidade (que irá ser utilizada como uma heurística para avaliação da escolha). Este cálculo vai ser feito recorrendo a funções que se vão chamando recursivamente, até ser atingido um nó folha, altura em que é retornado um valor que vai sendo somado a outros valores até ao nó que efetuou a chamada inicial cuja utilidade vai ser o total das somas. Para cada uma das jogadas possíveis vamos seguir este algoritmo para podermos comparar as respectivas utilidades e finalmente podermos escolher a melhor.

A profundidade vai servir para limitarmos a pesquisa de forma a obtermos um resultado num espaço de tempo viável, uma vez que o espaço da árvore vai aumentar exponencialmente com o número das jogadas, aumentando assim o tempo que demoraria a fazer uma pesquisa completa até termos um jogo completo.[1]

Algoritmo Minimax

Este algoritmo define uma estratégia óptima para minimizar uma possível perda máxima. Utiliza uma heurística que é determinada pela função de utilidade, que por sua vez, calcula os valores nos estados finais que são gerados na árvore do jogo.[2]

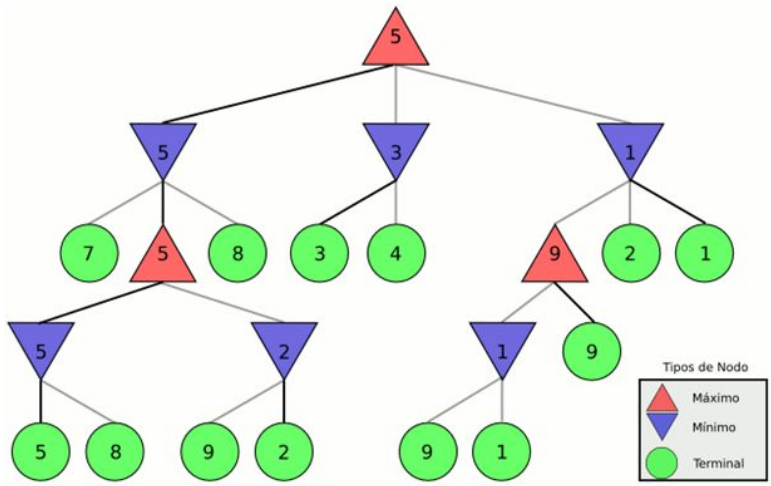


Figura 2 - Exemplo de execução do algoritmo Minimax[3]

Como podemos verificar pela imagem a ideia do algoritmo é muito simples, maximizar as jogadas que o computador faz e minimizar as jogadas do oponente. As chamadas de mínimo e máximo vão sendo feitas recursivamente, para quando chegarmos um nó terminal/folha os valores das utilidades sejam retornados. Quando estamos numa chamada de máximo vamos escolher a utilidade mais elevada, ou seja, maximizar a nossa jogada e quando estivermos numa chamada de mínimo vamos escolher a utilidade mais baixa, minimizando assim a jogada do oponente.

Algoritmo Alfa-Beta

O algoritmo Alfa-Beta funciona da mesma forma que o algoritmo minimax, a única diferença é que vai podando os nós à medida que desce na árvore. Estes nós são afastados pois não vão influenciar nas decisões finais do mínimo e do máximo. [2]

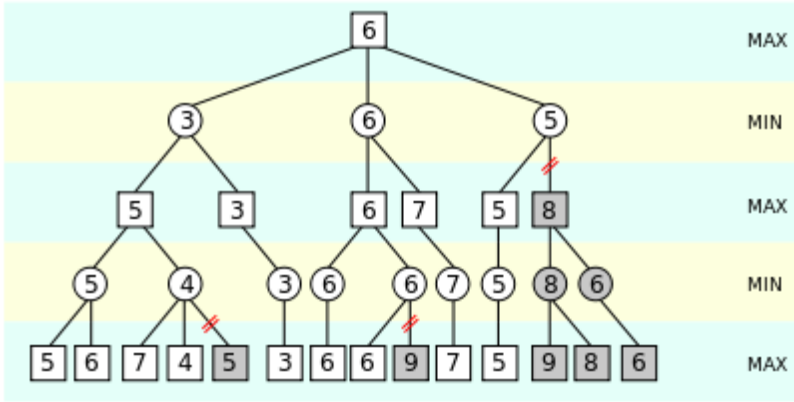


Figura 2 - Exemplo de execução do algoritmo Alfa-Beta[4]

Como se pode constatar esta estratégia vai conseguir evitar a exploração de ramos que podem ser considerados desnecessários. Olhando para a imagem vemos que o último ramo foi completamente podado. Isto acontece porque as chamadas de mínimo e máximo se vão alternando, uma chamada de mínimo é feita sobre vários valores de máximo e vice versa. Como consequência, quando estamos, por exemplo, numa chamada de mínimo e encontramos um valor baixo vindo de uma das chamadas de máximo seguintes, quando fazemos a comparação com os valores seguintes, se os mesmos forem superiores podemos podá-los.

Connect Four

É um jogo adversarial, neste caso com apenas dois adversários, bastante semelhante ao jogo do galo. O objetivo do jogo é colocar quatro peças em linha, quer horizontalmente, verticalmente ou diagonalmente. Os dois adversários vão então colocar as suas peças de forma a conseguirem ter 4 em linha, mas também de forma a que o oponente não consiga fazer o mesmo.

Para os efeitos do presente trabalho foram implementadas duas versões diferentes deste jogo. Uma em que um dos oponentes é uma inteligência artificial, ou seja, o computador vai decidir a sua próxima jogada seguindo um determinado algoritmo. Na outra versão é possível a duas pessoas jogarem uma contra a outra utilizando o mesmo computador como interface do jogo.

Existem várias outras implementações deste jogo disponíveis online e também versões do jogo em plástico para as crianças jogarem, pois trata-se de um jogo didático que ajuda as crianças a desenvolverem as suas capacidades de estratégia e planeamento. [5]

Os algoritmos Minimax e Alfa-Beta aplicados ao jogo Connect Four

O primeiro passo na implementação do jogo *Connect Four* foi perceber a dinâmica do jogo. Daqui passamos para a fase de implementação do código. Criamos uma classe tabuleiro para representar o tabuleiro de jogo, que não é mais uma *string* com ”-” quando está vazia, à qual vão sendo feitas modificações sempre que é feita uma jogada, ou seja, vamos substituindo os “-” por “X” ou “O”, consoante o jogador. Sempre que há uma alteração imprimimos o tabuleiro do jogo para que seja perceptível quem jogou e quais as jogadas ainda disponíveis.

Para os algoritmos poderem funcionar tornou-se necessário gerar os descendentes de cada jogada. De forma a conseguirmos isso criámos uma classe *Node*, cada *Node* vai ser constituído por uma matriz, uma utilidade, uma profundidade e uma coluna. Sempre que é a vez do computador jogar este pega na matriz do tabuleiro e transforma-a num *Node*, sendo que o *Node* inicial vai ter a utilidade, profundidade e coluna todos a zero. A partir deste *Node* vão ser gerados todos os seus descendentes, sendo que a profundidade vai incrementando em 1 por cada nível, ou seja, os filhos vão ter profundidade 1, os netos profundidade 2 e assim por diante. A utilidade vai ser calculada nas chamadas recursivas dos algoritmos e é este o factor chave que nos vai permitir tomar uma decisão. Finalmente temos o campo coluna, que vai guardar qual a coluna que deu origem à jogada em questão e que será o valor a ser devolvido por ambos os algoritmos para ser concretizada a jogada do computador.

Nesta implementação foi necessário termos especial cuidado na elaboração de um método que fosse calculando a utilidade das jogadas. De acordo com as instruções do enunciado do problema as pontuações de cada grupo de quatro casas poderiam ser: -50, -10, -1, 0, 1, 10, 50. Estes valores são obtidos contando o número de peças de cada jogador. Se um segmento de 4 casas tem 3 peças do oponente e nenhuma minha então vale -50, se tiver só duas vale -10, se for só uma -1. No caso em que só tem 3 peças minhas vai valer 50, 2 peças 10 e uma peça 1. Quando um segmento tem peças de ambos os jogadores o seu valor vai ser 0. para cada tabuleiro de um nó folha é feito este cálculo para todos os segmentos possíveis de quatro casas. Este valor é então retornado, através das chamadas recursivas, até ao nó que lhe deu origem, sofrendo diversas comparações ao longo das chamadas.

Existem 4 casos de paragem nestes algoritmos, quando num determinado Node temos uma configuração que nos deu a vitória, caso em que é retornada de imediato uma utilidade de 512, quando temos uma configuração de derrota, retornamos -512, quando temos uma situação de empate, ou seja o tabuleiro cheio

e ninguém fez 4 em linha, pelo que retornamos 0 e o caso em que chegamos a um nó folha da árvore de pesquisa e pretendemos retornar a utilidade.

No jogo *Connect Four* para termos uma pesquisa completa e termos a certeza se uma jogada dá origem a uma vitória, derrota ou empate deveríamos fazer a pesquisa até ao nível de profundidade 42, que equivaleria a ter todas as 42 casas do tabuleiro do jogo preenchidas. Contudo, como se trata de uma pesquisa em profundidade e o tamanho do espaço de pesquisa na árvore de decisão vai aumentando exponencialmente tal poderá não ser viável. Ao aumentarmos o nível de profundidade para 42 o computador vai ficar durante horas, possivelmente dias, a fazer a pesquisa para poder decidir a sua próxima jogada.

Nitidamente não é uma opção viável pois no mínimo precisaria de 4 jogadas para ganhar e para decidir cada jogada iria precisar de dias. Para ultrapassar este problema tentamos perceber qual seria o nível que nos daria um resultado fiável, mas num espaço de tempo muito menor. Começamos por experimentar um nível de profundidade de 1, mas os resultados não foram muito animadores pois o computador tinha um comportamento quase aleatório, uma vez que estava apenas a simular as suas possíveis respostas. Com um nível de 2 não melhorou muito. Com um nível de profundidade 3 percebemos que o computador precisava de simular pelo menos duas jogadas suas e uma do oponente para conseguir resultados satisfatórios. Após mais algumas tentativas concluímos que o nível de profundidade 8 daria já resultados ótimos, pois com 8 jogadas já dá para prever se é possível fazer quatro em linha ou não. Contudo devido a algumas limitações no processador da máquina utilizada fazer a pesquisa até este nível acabava por sobrecarregar demasiado o sistema e o tempo acabava por ser excessivo ou a memória da máquina não aguentava.

Assim sendo acabamos por optar fazer a pesquisa até ao nível 6, altura em que o computador já terá 3 jogadas suas e 3 do oponente, o que irá levar a uma boa aproximação de qual a melhor jogada.

Tendo implementado corretamente o algoritmo *Minimax* para o jogo *Connect Four*, bastou efetuar algumas alterações para fazer a implementação do algoritmo *Alfa-Beta*, nomeadamente passar o máximo e o mínimo encontrados, alfa e beta respectivamente, para em todas as chamadas podermos controlar se é necessário explorar o ramo em questão ou se não iremos conseguir obter um valor melhor e nesse caso podemos podar o ramo.

Resultados

Os resultados foram obtidos efetuando 10 jogos utilizando o algoritmo Minimax e 10 com o Alfa-Beta. Do total de jogos com cada algoritmo metade foi iniciada pelo computador e a outra metade pelo jogador. Os valores apresentados na tabela seguinte são uma média dos valores obtidos. Observou-se que apesar de o jogador partir com vantagem quando inicia o jogo acaba por perder na mesma ao fim de uma média de 8 a 9 jogadas. Todos os testes foram efetuados num computador com um processador Centrino vPro.

	Profundidade 6		Profundidade 5	
	Minimax	Alfa-Beta	Minimax	Alfa-Beta
1ª jogada	1 411.8 ms	259.5 ms	262.5 ms	72 ms
jogadas intermédias	587.2 ms	76.3 ms	119.7 ms	36.9 ms
última jogada	231.3 ms	9.5 ms	68.5 ms	6.5 ms

Tabela 1 - Média dos tempos obtidos nos jogos.

Como se pode verificar pela tabela apresentada existe uma diferença muito grande entre os tempos requeridos pelo Minimax e pelo Alfa-Beta, diferença essa que é acentuada na primeira jogada. Utilizando uma profundidade de pesquisa de 5, conseguimos tempos de resposta melhores, mantendo a fiabilidade da resposta, ou seja o computador consegue na mesma ganhar o jogo. Com uma profundidade de pesquisa de 6 o comportamento do computador vai sofrer ligeiras alterações, mas o tempo vai aumentar substancialmente. É de esperar que com tempos superiores os tempos de resposta aumentem.

De seguida apresentamos os gráficos comparativos entre os tempos em dois jogos com o algoritmo Minimax, um iniciado pelo computador e outro pelo jogador e entre os tempos em dois jogos com o Alfa-Beta, com as mesmas condições.

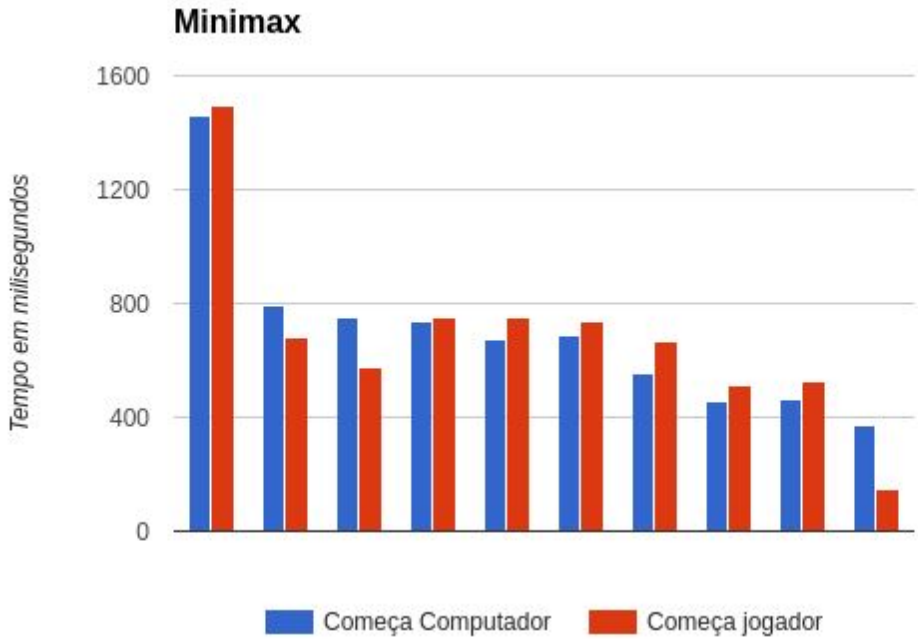


Gráfico 1 - Comparação dos tempos em dois jogos com o algoritmo Minimax, com pesquisa a profundidade 6.

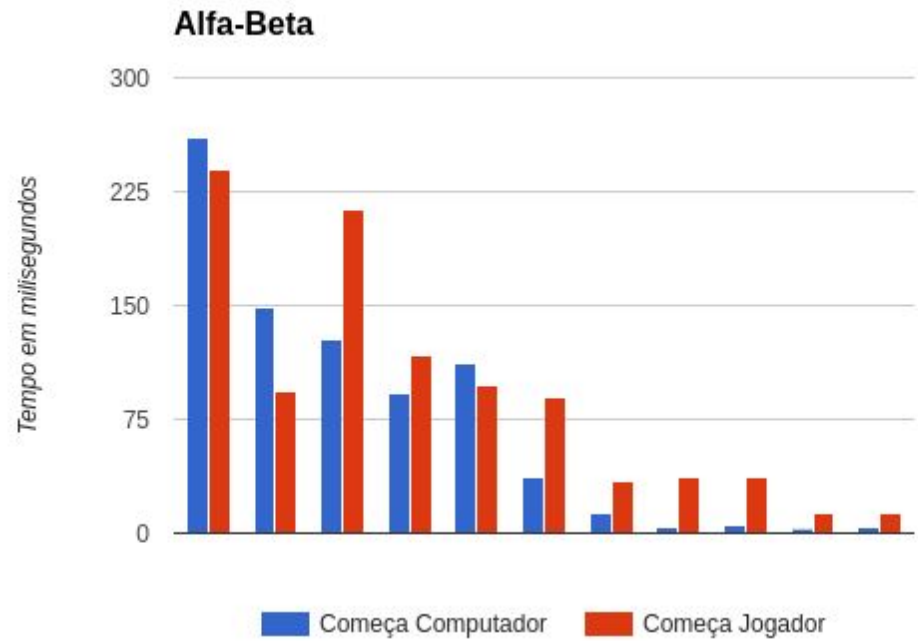


Gráfico 2 - Comparação dos tempos em dois jogos com o algoritmo Alfa-Beta, com pesquisa a profundidade 6.

Comentários e Conclusões

Após conseguirmos implementar o algoritmo de *Minimax* com sucesso passamos à fase de testes para verificar possíveis erros no nosso programa. Foram efetuados vários despistes até conseguirmos corrigir todos os problemas e o algoritmo passar a responder adequadamente. Nesta altura passamos à implementação do algoritmo de *Alfa-Beta*, que foi muito mais simples por já termos o *Minimax* corretamente implementado.

Jogamos diversas vezes contra o computador, usando quer um algoritmo quer o outro e de todas as vezes o computador ganhou. Foram efetuados testes com mais do que uma pessoa e os resultados mantêm-se. Tal como esperado ambos os algoritmos conseguem simular o passos suficientes do jogo para poderem ter uma previsão acurada da jogada que levará o computador à vitória.

Apesar de ambos os algoritmos estarem construídos para conseguir uma vitória não vai tentar conseguir uma vitória o mais rapidamente possível. Entre dois caminhos com a mesma pontuação irão escolher o primeiro, independentemente se é esse que dá a vitória mais cedo ou não.

Os resultados obtidos foram os esperados, tendo-se verificado que o computador coloca as suas peças quase sempre de forma a ficar com 3 em linha de duas formas diferentes e assim se vir a sua jogada a ser cortada numa das formas, consegue na mesma efetuar 4 em linha usando o outro grupo.

Bibliografia

[1] - http://www.cs.cornell.edu/courses/cs2110/2014sp/assignments/a4/A4ConnectFour.pdf (em 25/03/2017)
[2] - Stuart J. Russell and Peter Norvig, 1995, Artificial Intelligence: A Modern Approach
[3] - https://inteligenciartificial2kc.wordpress.com/2015/07/09/busqueda-entre-adversarios/, imagem (em 20/03/2017)
[4] - https://pt.wikipedia.org/wiki/Poda_(computa%C3%A7%C3%A3o), imagem (em 20/03/2017)
[5] - http://www.knowledgeadventure.com/games/connect-four/ (em 26/03/2017)