Inteligência Artificial

Connect Four

Tiago Moita e Milton Lopes

March 24, 2017

Contents

1 Introdução

Os jogos com adversários onde por vezes o nosso adversário pode ser uma maquina e não simplesmente um ser humano já começam a tornar-se cada vez mais populares no século XXI, e isto devido a um grande avanço nos algoritmos para os resolverem.

Um jogo com adversário(maquina) pode ser intendido como um jogo onde após a jogada de um "Ser Humano", a maquina irá simular uma serie de jogadas para a frente(numero de jogadas á nossa escolha) e poderá assim fazer-nos frente de uma foram "Dinâmica" e Inteligente.

Hoje em dia existem alguns algoritmos para resolverem este tipo de jogos contra maquinas, alguns melhores que outros no que diz respeito a complexidade, tempo, memoria, etc... mas nesta disciplina vamos falar apenas de dois, nomeadamente o algoritmo Minimax, e o algoritmo Alfa-Beta.

2 Minimax Algoritmo

O algoritmo do Minimax, a maximização do ganho mínimo é um método para minimizar a possível perda máxima. Ele é utilizado na Filosofia, Teoria da Decisão, Teoria dos jogos e Estatística, todos com o mesmo objetivo de minimizar a perda máxima. O Minimax surgiu a partir do Zero-Sum Game, demonstrado por Von Neumann um brilhante matemático nascido em Budapeste em 1903 pelo qual recebeu o título de pai da teoria dos jogos em 1926.

Como funciona?

No algoritmo Minimax existe um nível de Max(onde queremos maximizar o valor da jogada) e um nível de Min(onde queremos minimizar o valor da jogada), e como este algoritmo será sempre para ser usado pela maquina, o algoritmo deve sempre devolver o valor de maximização.

Uma vez começado o algoritmo, o algoritmo ira gerar recursivamente sucessores da maquina(os que forem possíveis, depende da função SUCESSORS) e por sua vez os sucessores do utilizador se possíveis, e cada vez gera um sucessor tanto da maquina como do utilizador vai perguntando se já é um estado final (ganhou,empatou,perdeu) com a função TERMINALSTATE e caso não seja nenhum destes, podemos definir uma profundidade limite para que o jogo não simule muitas jogadas para a frente de forma a demorar muito tempo e tornando se assim pouco interativo com o utilizador, ou mesmo pode nem chegar a encontrar a solução se a profundidade for muito grande e o espaço de busca do jogo também o for.Quando o jogo encontra um estado terminal, ou chega á profundidade limite devolve-nos "Pontos" que é definido com a função UTILITY(dependendo das regras do jogo),e serão estes "Pontos" que nos farão optar pelo caminho de maior pontuação num nível de Max,e por outro lado num caminho de menor pontuação num nível de Min, pois queremos o melhor para o algoritmo e o pior para o utilizador na visão da maquina.

```
function MINIMAX_DECISION(state): returns an action
     inputs: state → estado corrente no jogo
     v ← MAX-VALUE(state)
     return the action in SUCCESSORS(state) with value v
function MAX-VALUE(state) returns a utility value
if TERMINAL_TEST(state) then
   return UTILITY(state)
end if
v ← -infinito
for s in SUCCESSORS(state) do
   v ← MAX(v, MIN-VALUE(s))
end for
return v
function MIN-VALUE(state) returns a utility value
if TERMINAL_TEST(state) then
   return UTILITY(state)
end if
v ← infinito
for s in SUCCESSORS(state) do
   v ← MIN(v, MAX-VALUE(s))
end for
return v
```

3 Alfa-Beta Algoritmo

O algoritmo Alfa-Beta é de certa forma uma variação do algoritmo Minimax, onde a sua única diferença ou sua única vantagem será o facto de ser mais inteligente ao ponto de não necessitar de visitar nós que não nos vão ser úteis e assim poupando-nos tempo e espaço de memoria.

Como funciona?

O algoritmo Alfa-Beta funciona de forma semelhante ao Minimax, ou seja existe um nível de Max(onde queremos maximizar o valor da jogada) e um nível de Min(onde queremos minimizar o valor da jogada), e como este algoritmo será sempre para ser usado pela maquina, o algoritmo deve sempre devolver o valor de maximização.

Uma vez começado o algoritmo, o algoritmo ira gerar recursivamente sucessores da maquina(os que forem possíveis, depende da função SUCESSORS) e por sua vez os sucessores do utilizador se possíveis, e cada vez gera um sucessor tanto da maquina como do utilizador vai perguntando se já é um estado final (ganhou,empatou,perdeu) com a função TERMINALSTATE e caso não seja nenhum destes, podemos definir uma profundidade limite para que o jogo não simule muitas jogadas para a frente de forma a demorar muito tempo e tornando se assim pouco interativo com o utilizador, ou mesmo pode nem chegar a encontrar a solução se a profundidade for muito grande e o espaço de busca do jogo também o for.Quando o jogo encontra um estado terminal, ou chega á profundidade limite devolve-nos "Pontos" que é definido com a função UTILITY(dependendo das regras do jogo),e serão estes "Pontos" que nos farão optar pelo caminho de maior pontuação num nível de Max,e por outro lado num caminho de menor pontuação num nível de Min, pois queremos o melhor para o algoritmo e o pior para o utilizador na visão da maquina. Uma vez devolvido o melhor valor,o algoritmo termina e passa a vez da jogada para o utilizador. Até agora este será apenas o processo de execução do Alfa-Beta que é semelhante ao Minimax, a variação será que o Alfa Beta em cada jogada carrega consigo variáveis de nome "alfa" e "beta" onde estas variáveis servem para a cada jogada o algoritmo perguntar se uma vez gerado o primeiro sucessor o algoritmo perguntar se "é possível" que para o próximos sucessores aquela jogada seja melhorada dependendo se estamos num nível de Max ou de Min, e caso não seja possível melhorar a jogada, o algoritmo simplesmente faz a poda desses nós, ou seja nem sequer os expande.

Por exemplo se estamos num nível de Max e temos no nó um valor de 10(profundidade 0), isso quer dizer que aquele nó só aceita novos valores se forem maiores que 10, logo se tivermos por exemplo dois sucessores(profundidade 1) deste nó no nível Min,onde o primeiro terá valor 10(profundidade 1) que foi o que originou o valor 10 do nível de max(profundidade 0), e no segundo nó do nível de Min(profundidade 1) ao gerar os seus sucessores(profundidade 2) e logo o primeiro gerado(profundidade 2) tiver por exemplo o valor 5,como no nível acima de Min eu só vou querer valores menores que 5, de que me

adianta verificar os sucessores seguintes(profundidade 2) se na profundidade 1 tenho um nó com valor maior que 5 e que logo á partida sabemos que o nó de Max(profundidade 0) o irá escolher.

E este será o diferencial do Alfa-Beta para com o Minimax que fará com que o algoritmo encontre na mesma a solução ótima , mas em menos tempo e gastando menos memoria.

```
function ALPHA-BETA-SEARCH(state): returns an action
     inputs: state → estado corrente no jogo
     v ← MAX-VALUE(state,-inf,+inf)
     return the action in SUCCESSORS(state) with value v
function MAX-VALUE(state,alfa,beta) returns a utility value
inputs: state, alfa → melhor alternativa para MAX, beta → melhor alternativa para MIN
if TERMINAL_TEST(state) then
   return UTILITY(state)
end if
v ← -infinito
for s in SUCCESSORS(state) do
   v ← MAX(v, MIN-VALUE(s,alfa,beta))
   if (v >= beta ) then
      return v // momento da poda
   end if
   alfa ← MAX(alfa,v)
end for
return v
function MIN-VALUE(state,alfa,beta) returns a utility value
if TERMINAL_TEST(state) then
   return UTILITY(state)
end if
v ← +infinito
for s in SUCCESSORS(state) do
   v ← MIN(v, MAX-VALUE(s,alfa,beta))
   if (v <= alfa ) then
      return v // momento da poda
   end if
   beta ← MIN(beta,v)
end for
return v
```

4 Connect Four

Connect Four é um jogo de estratégia de dois jogadores. Ele é reproduzido usando 42 pedras, geralmente 21 pedras vermelhas para um jogador e 21 pedras pretas para o outro jogador e uma grade vertical que é de 7 colunas de largura, cada coluna é capaz de armazenar um máximo de 6 pedras. Os dois jogadores jogam alternadamente. Um movimento consiste em um jogador soltar uma das suas pedras na coluna da sua escolha. Quando uma pedra é lançada em uma coluna, ele cai até atingir o fundo da coluna ou a pedra de topo nessa coluna. O objetivo do jogo é colocar 4 pedras em linha (quer horizontal, vertical ou diagonalmente). O primeiro jogador, que atingir este objetivo, ganha o jogo.

As funções de avaliação usadas são TERMINALSTATE e UTILITY, sendo o primeiro só para verificar se um dos jogadores já chegou a uma posição vencedora, e a outra verifica o quanto longe um jogador está de vencer após uma determinada jogada, sendo que que q UTILITY funciona da seguinte forma:

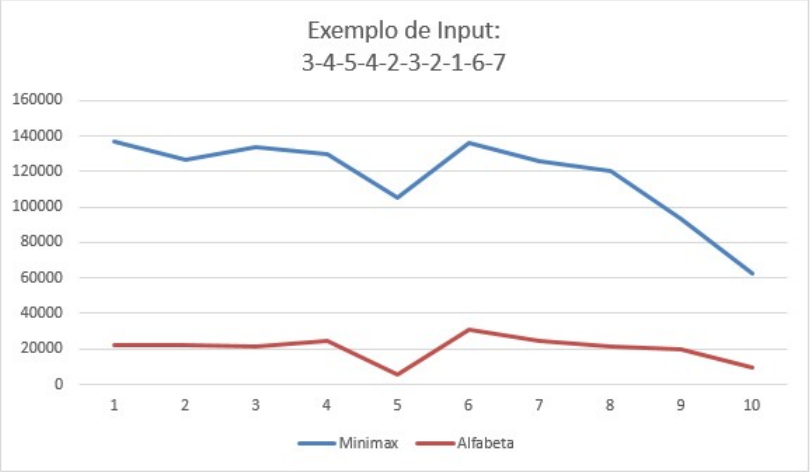
1. -50 para três Os, e nenhum Xs,
2. -10 para dois Os, e nenhum Xs,
3. -1 para um O, e nenhum Xs,
4. 0 para nenhuma pedra ou misturas entre Os e Xs
5. 1 para um X, e nenhum O
6. 10 para dois Os, e nenhum Xs,
7. 50 para três Os, e nenhum Xs.

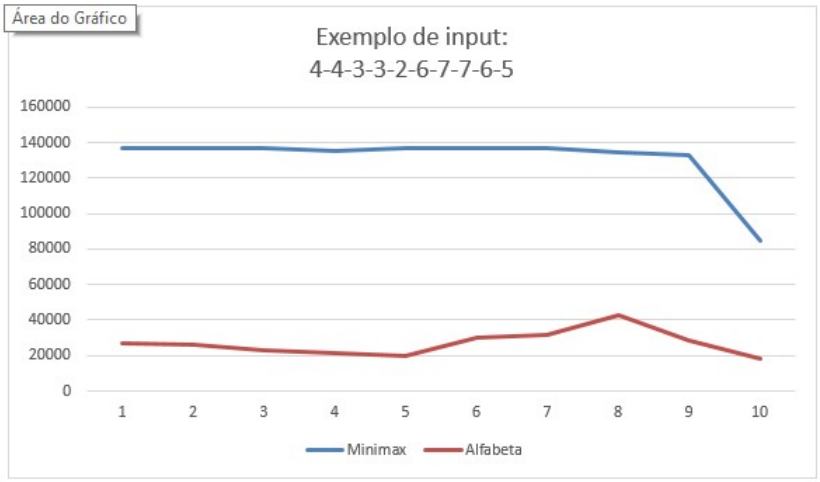
Embora esta função de avaliação esteja garantida para funcionar,não é necessariamente a melhor no sentido que não favorece ganhos rápidos.

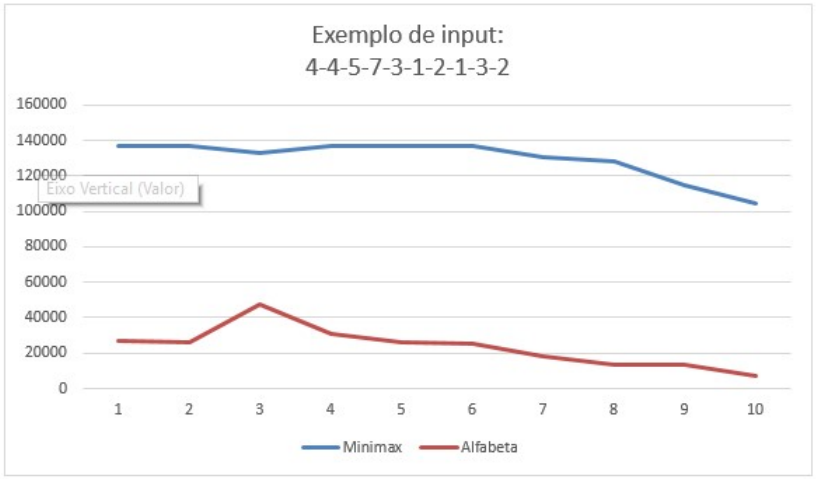
5 Minimax e o Alfa-beta aplicado ao Connect Four

O Minimax e o Alfabeta aplicados ao jogo do Connect Four funcionam de forma semelhante aparentemente, ou seja geram os mesmos sucessores, o diferencial será apenas no tempo e espaço gastos pelos dois. Em ambos algoritmos implementamos a função SUCESSORS-EU para o utilizador poder gerar o seu sucessor após a sua jogada,SUCESSORS-COMPUTADOR que serve em ambos os algoritmos para gerar os sucessores que o computador gera ao chamar as funções principais do Minimax e do Alfabeta,TERMINALSTATE que serve para verificar se o estado corrente será um estado final ou não,e por ultimo a função UTILITY que serve para gerar um pontuação segundo as regas definidas atrás pelo jogo do ConnectFour.

Nestes algoritmos utilizamos a classe Node que contem uma configuração apenas por questões de organização, pois era possível fazer sem esta classe. Ao executarmos ambos os algoritmos para as mesmas instâncias verificamos que o Alfabeta gera de facto menos nós, de tal dimensão que se pode considerar significativo. Ex: Para a instância: 1-2-3-4-5-6-7-1 dada pelo o utilizador ,o utilizador perde e com o algoritmo Minimax gera: 980063, enquanto que o Alfabeta gera apenas: 164396, ou seja a quantidade de nós podados pelo algoritmo Alfabeta são: 980063-164396=815667. Em ambos algoritmos definimos uma profundidade limite que no inicio do jogo irá dar muito jeito pelo facto de o espaço de busca deste jogo ser muito grande, logo este limite irá fazer com que o algoritmo não necessite de correr ate ao fim numa primeira fase , mas sim ate esse dado limite. De seguida apresentamos três exemplos da execução de ambos os algoritmos para três diferentes instâncias em 10 jogadas pré-definidas e obtemos os seguintes dados:




Área do Gráfico
Exemplo de input:
4-4-3-3-2-6-7-7-6-5
160000
140000
120000
100000
80000
60000
40000
20000
0
1
2
3
4
5
6
7
8
9
10
Minimax
Alfabeta


Exemplo de input:
4-4-5-7-3-1-2-1-3-2
160000
140000
120000
Eixo Vertical (Valor)
100000
80000
60000
40000
20000
0
1
2
3
4
5
6
7
8
9
10
Minimax
Alfabeta

6 Comentários Finais e Conclusões

Achamos os dois algoritmos muito interessantes, tanto pela sua funcionalidade como na particular recursividade que possuem. Em relação a sua performance damos nota excelente pelas sua caraterísticas próprias de pesquisar uma quantidade grande de possibilidades em pouco tempo, mostrando nos a melhor solução. Por fim, e com os gráficos apresentados tínhamos em mente que á medida que o jogo avançava os nós gerados pela maquina deviam ser menos, embora não foi o que nos apresentou os nossos gráficos para três instâncias diferentes, logo supomos que o numero de nós gerados varia de acordo com a boa ou a má jogada do utilizador.

7 Bibliografia

1. http://www.dcc.fc.up.pt/ ines/aulas/1617/IA/jogos.pdf
2. https://web.fe.up.pt/ eol/IA/IA0809/APONTAMENTOS/Alunos_MiniMax.pdf