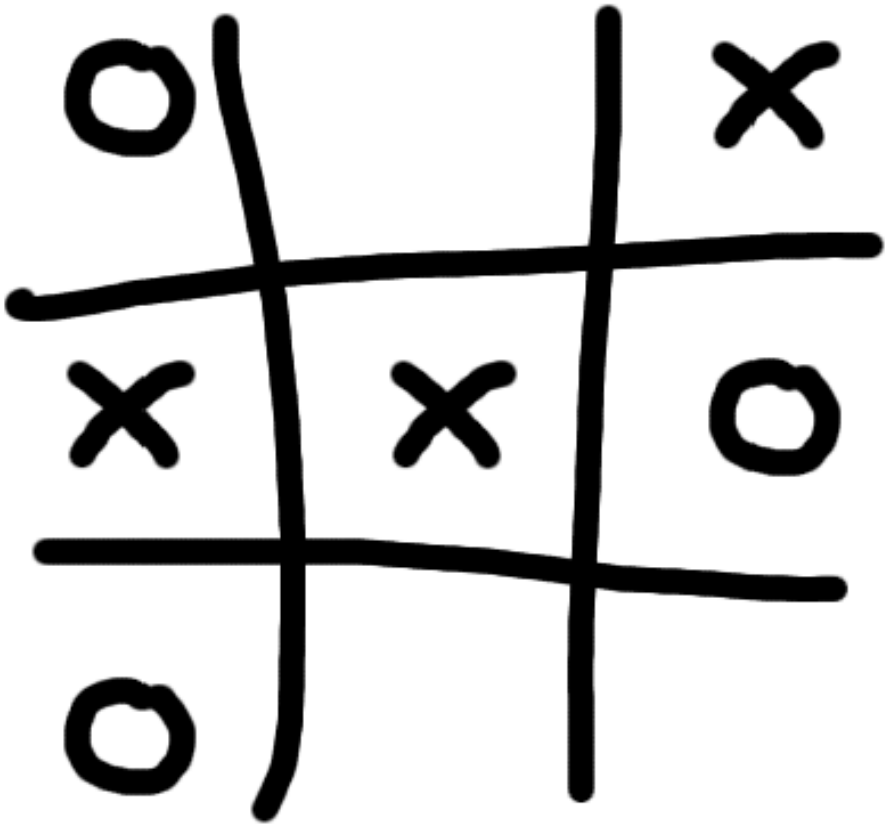
U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Inteligência Artificial

Jogo do Galo (TicTacToe)



Carlos Bernardes
201303743
Célio Rodrigues
201303171
Cláudia Correia
201304727

FCUP,2016

1. Introdução

O presente trabalho tem como objetivo o estudo e implementação de algoritmos em jogos com oponentes. Estes jogos, pertencentes a um grupo muito específico de problemas de Inteligência Artificial, caracterizam-se pelo tipo de ambiente multiagente competitivo devido ao facto de terem participação de mais do que um agente que, naturalmente, têm objetivos que entram em conflito (pois querem os dois ganhar, e um tem que perder). Num jogo, a presença de um oponente implica incerteza, o que torna mais difícil a tomada de uma decisão. As dificuldades ao nível de implementação nestes algoritmos de jogos passam pela ausência de conhecimento das jogadas exatas do oponente e não pela falta de informação, isto é, assim que é feita uma jogada, não existe tempo para calcular as consequências desta e, por esse motivo, o jogador terá de fazer a melhor escolha baseado na experiencia passada. Podemos então, definir um jogo como um problema de busca em que existe um estado inicial, uma função que gera sucessores (estados), um teste para verificar se um estado é terminal, uma função de utilidade e um limite de profundidade (dependente do espaço de procura).

2. Algoritmo MiniMax

O algoritmo MiniMax é um método para minimizar a perda máxima possível, ou seja, minimizar a perda e maximizar o ganho.
Neste jogo, temos dois jogadores, A e B, sendo que em cada passo o A quer maximizar a probabilidade de ganhar e o B quer minimizar as chances de isso acontecer. Para maximizar a probabilidade de ganhar, o jogador A tem de ter conhecimento do melhor movimento que pode fazer, no entanto, para isso necessita de conhecer também os melhores movimentos possíveis que o B tem à sua disponibilidade. A forma de tornar isto possível passa por minimizar a probabilidade de B ganhar e depois maximizar a probabilidade de A ganhar. Assim, podemos encontrar a melhor jogada a partir do fim do jogo, caminhando por todas as opções válidas.
É necessário fazer a expansão da árvore de procura desde o nó inicial até todos os nós terminais. Depois usamos uma função de utilidade para classificar cada nó terminal (se perdeu, ganhou ou empatou) e recursivamente retornamos essa utilidade para os níveis superiores onde vamos tomar a decisão da jogada a tomar.

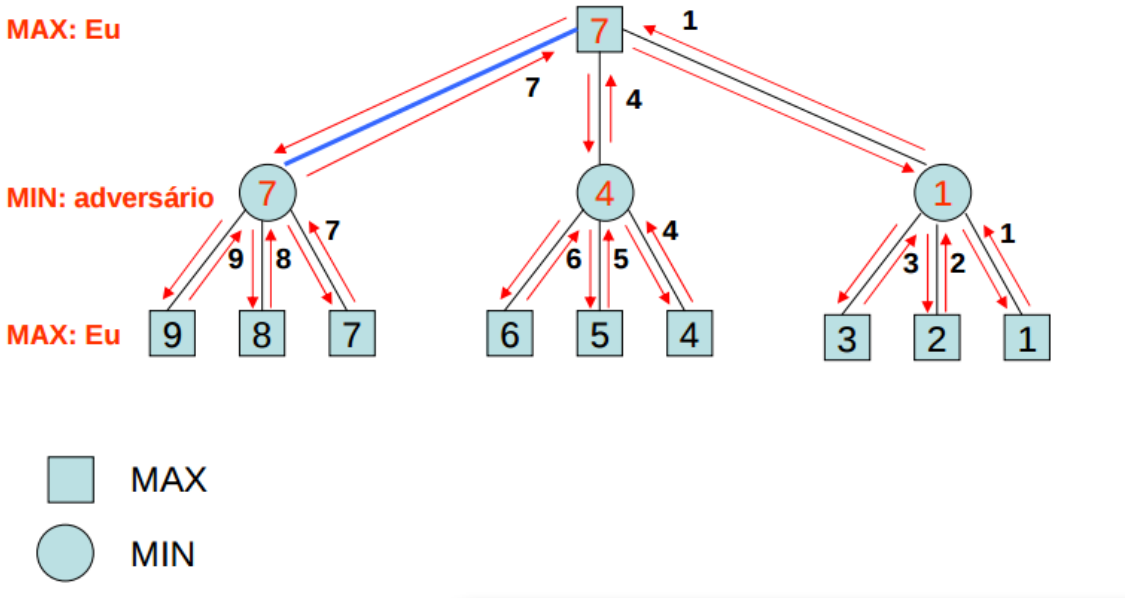
```
function MINIMAX_DECISION(state) returns an action
   inputs: state -> estado corrente no jogo
   v <- MAX-VALUE(state)
   return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- -infinito
   for s in SUCCESSORS(state) do
     v <- MAX(v, MIN-VALUE(s))
   return v

function MIN-VALUE(state) returns a utility value
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- infinito
   for s in SUCCESSORS(state) do
     v <- MIN(v, MAX-VALUE(s))
   return v
```

A seguinte imagem pretende explicar melhor o conceito do algoritmo tomando um exemplo.

No primeiro nível da árvore pretendemos calcular a utilidade máxima dos 3 nós (estados no TicTacToe) do nível 2. Que por sua vez, estas são as menores utilidades de todos os nós do nivel 3. Ou seja, no nível 1, o primeiro nó, escolhe o Max entre (7,4,1), por sua vez, o nó com utilidade 7 (nível 2) irá escolher o minímo nó entre os nós do nível 3 (9, 8,7), o nó com utilidade 4 (nível 2) irá escolher o mínimo nó entre os nós do nível 3 (6,5,4) e por último o nó com utilidade 1 (nível 2) irá escolher o mínimo nó entre os nós do nível 3 (3,2,1).



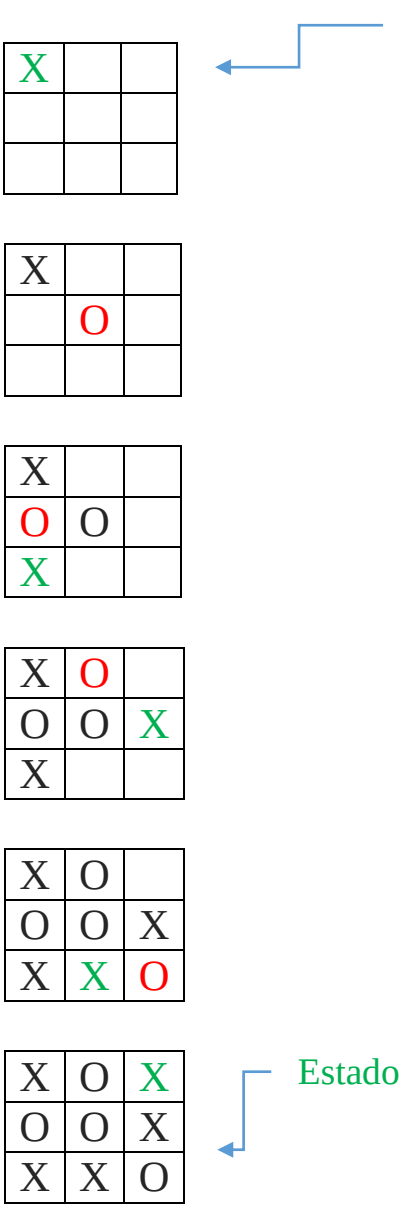
Podemos concluir então que para cada decisão de jogada é necessário conhecer a árvore de expansão.

3.Algoritmo Alpha-Beta

O algoritmo Alpha-Beta é uma variação do MiniMax que visa reduzir o número de nós que são avaliados na árvore de procura de forma a eliminar uma ramificação da árvore de pesquisa para que não seja examinada. Este algoritmo para de avaliar um movimento assim que encontra outro que tem um valor pior que o movimento previamente avaliado. Posto isto, os movimentos posteriores não necessitam de ser avaliados, visto que já foi encontrada a melhor utilidade no contexto.
Esta técnica não altera o resultado das avaliações das subárvores da árvore de pesquisa e é uma solução ótima para árvores de pesquisa com um fator de ramificação elevado no que toca à utilização da memória.

4. Discussão e comentários

Vimos que o MiniMax é um algoritmo que maximiza o ganho e minimiza a perda no entanto o tempo para determinar a decisão ótima é impraticável para jogos mais complexos devido ao número de estados que são criados na árvore de pesquisa, estados estes que constituem por vezes caminhos cuja possibilidade de serem seguidos é praticamente nula pois não constituem solução.
O Alpha-Beta é uma técnica que visa melhorar a implementação do Minimax de forma a eliminar estes caminhos que não contêm solução.



Nós Visitados	
MiniMax	Alpha-Beta
59705	2338
927	189
61	33
5	5

Podemos reparar que o Alpha-Beta visita consideravelmente menos nós do que o MiniMax devido aos cortes que faz na árvore de procura. Como é natural, os números de nós visitados pelos algoritmos vão-se aproximando um do outro pois as possibilidades de procura vão sendo cada vez menos.