FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO
DEPARTAMENTO DE CIÊNCIAS
DOS COMPUTADORES

RELATÓRIO DE TRABALHO

JOGO_DO_GALO(COMPUTER VS PLAYER)

Disciplina

Inteligência Artificial

CC2006

Docente

Inês Dutra

dutra@fc.up.pt

Grupo

André Meira

up201404877@fc.up.pt

Dinis Costa

up201406364@fc.up.pt

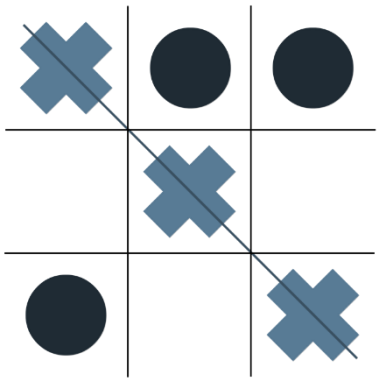
Conteúdo

Introdução

Desde sempre que os jogos proporcionam, para além do convívio, a competição entre o ser humano. Há os que requerem a vertente física, os que estimulam as capacidades cognitivas e ainda os que combinam ambas as componentes. Focando a competência cognitiva, podemos realçar a capacidade de memorização e o raciocínio estritamente lógico como características fundamentais para se obter bons resultados. Jogos que requerem essas capacidades são, por exemplo, os de tabuleiro como xadrez, damas ou jogo do galo. Por este motivo, este tipo de jogos torna-se um alvo para a investigação e aprendizagem de inteligência artificial. Esta é uma área que quando aplicada a jogos tem como objetivo encontrar, para as diferentes situações, a melhor ação a realizar num determinado momento.

Relativamente aos problemas de busca vistos anteriormente a introdução de um oponente complica o processo de escolha da jogada a realizar. Isto porque há a introdução de incerteza, pois não é possível saber qual a jogada que o adversário irá fazer. No entanto, o que distingue realmente a verdadeira dificuldade dos diferentes tipos de jogos com oponentes é o grau de dificuldade em encontrar a solução. Isto acontece porque o número de jogadas possíveis varia consoante o jogo. Ao invés do xadrez, o jogo do galo é considerado um jogo aborrecido porque é bastante fácil determinar qual a jogada correta. O mesmo acontece quando se constrói inteligência artificial, uma vez que há menos incerteza nas jogadas possíveis, a árvore de pesquisa é menor.

Neste relatório são apresentados e descritos dois algoritmos para jogos com dois oponentes, o MiniMax e Alpha-Beta, que foram neste trabalho aplicados ao jogo do galo. Este é jogado alternadamente por dois jogadores, que preenchem os espaços em branco de uma matriz 3x3, utilizando símbolos diferentes um do outro. O jogador que conseguir com sucesso formar uma linha quer seja na horizontal, vertical ou diagonal, apenas com o seu símbolo vence a partida. A imagem seguinte representa uma partida do jogo do galo vencida pelo jogador “X”.



Algoritmos

Os algoritmos seguintes tendem a ser aplicados a jogos com dois oponentes. Isso significa que os ganhos de um jogador traduzem-se em perdas do adversário, pelo que apenas um pode vencer cada partida. Além disso, estes jogos devem ter toda a informação disponível para poder ser analisada, isto é, a variável sorte não deve estar presente. Isto permite que todas as jogadas futuras sejam previstas.

MiniMax

1. Descrição

Minimax é uma regra de conhecida por minimizar as possíveis perdas no pior dos cenários. Neste algoritmo, os jogadores são representados de forma diferente em que um deles procura maximizar a sua possibilidade de vitória e o outro procura minimizar a possibilidade de vitória do primeiro. Desta forma, pode-se dizer que o algoritmo procura determinar a estratégia ótima para o primeiro, independentemente daquilo que o segundo faça.

Para cada nó da árvore de pesquisa, este algoritmo gera os seus sucessores e, para cada filho, utiliza uma função heurística para determinar o valor de utilidade de todos os sucessores. Em todas as jogadas do primeiro jogador, o algoritmo procura pelo nó com maior valor de utilidade, enquanto que nas jogadas do segundo, o algoritmo escolhe o nó com menor valor de utilidade.

Se em jogos simples é possível expandir a árvore de procura até nós terminais, ao aplicar o Minimax a jogos mais complexos, torna-se praticamente impossível expandir a árvore de pesquisa até estes nós que são conhecidos por nós-folha. A estratégia que se deve adotar nessas situações consiste em percorrer a árvore de pesquisa até um número de níveis predefinido, determinado pelos recursos de tempo e de memória disponíveis.

Quando se percorre a árvore de busca até aos nós-folha é possível atribuir-lhes valores que traduzam uma situação de vitória, empate ou derrota. No entanto quando isso não acontece, é atribuído a cada nó um valor determinado por uma função de utilidade diferente. Assim, o valor que volta até ao nó-raiz não é uma indicação sobre se é ou não possível alcançar a vitória, mas sim uma estimativa determinada por esta função.

Desta forma, o algoritmo Minimax é essencialmente composto por duas funções:

1. A função MAX_VALUE, que simula todas as jogadas possíveis do computador em resposta às jogadas do oponente, dadas pela função MIN_VALUE;
2. A função MIN_VALUE, que simula todas as jogadas possíveis do oponente em resposta às jogadas do computador, dadas pela função MAX_VALUE.

2. Pseudo-Código

```
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 a, 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 a, s in SUCCESSORS(state) do
     v <- MIN(v, MAX-VALUE(s))
   return v
```

Alpha-Beta

1. Descrição

O algoritmo Alpha-Beta é um aperfeiçoamento do algoritmo Minimax, na medida em que, quando é aplicado a uma árvore Minimax, retorna a mesma jogada que o algoritmo Minimax retornaria, porém "poda" os ramos da árvore que não têm influência na decisão final. Sabendo que o algoritmo tem um crescimento exponencial do número de estados a analisar em cada nível da árvore de pesquisa, desta forma, é possível reduzir esse número.

Relativamente ao algoritmo Minimax, o algoritmo Alpha-Beta tem como novidade a introdução de duas variáveis: alpha e beta, onde alpha corresponde ao maior valor encontrado na jogada do Max, e beta corresponde ao menor valor encontrado na escolha do caminho de Min. O algoritmo vai atualizando os valores de alpha e de beta, que são utilizados para podar a árvore de pesquisa. Assim que é determinado um sucessor e conhecido o seu valor de utilidade, superior ou inferior do que o valor de alpha ou beta, a arvore é ou não podada.

2. Pseudo-Código

```
function ALPHA-BETA-SEARCH(state) returns an action
   inputs: state -> estado corrente no jogo
   v <- MAX-VALUE(state,-inf,+inf) % alfa inicia com -inf,
                                   % beta inicia com +inf
   return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state,alfa,beta) returns a utility value
   inputs: state -> estado corrente no jogo
           alfa  -> valor da melhor alternativa para MAX
           beta  -> valor da melhor alternativa para MIN
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- -infinito
   for a, s in SUCCESSORS(state) do
     v <- MAX(v, MIN-VALUE(s,alfa,beta))
     if (v >= beta ) then return v % momento da poda
     alfa <- MAX(alfa,v)
   return v

function MIN-VALUE(state,alfa,beta) returns a utility value
   inputs: state -> estado corrente no jogo
           alfa  -> valor da melhor alternativa para MAX
           beta  -> valor da melhor alternativa para MIN
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- +infinito
   for a, s in SUCCESSORS(state) do
     v <- MIN(v, MAX-VALUE(s,alfa,beta))
     if (v <= alfa ) then return v % momento da poda
     beta <- MIN(beta,v)
   return v
```

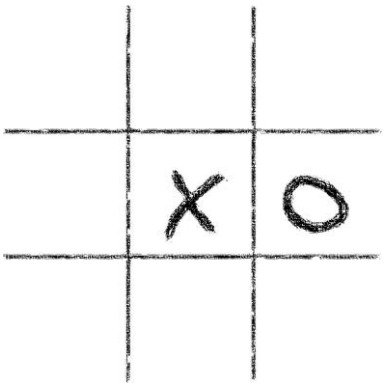
Propriedades

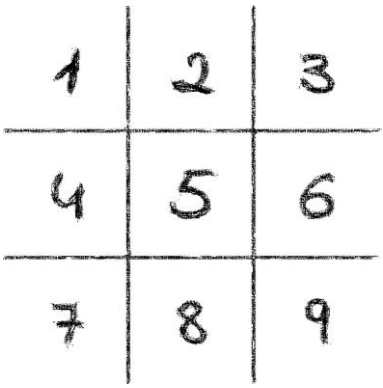
Algoritmo	Complexidade Temporal	Complexidade Espacial
Minimax	$O(b^m)$	O(b x m) - pesquisa até encontrar um nó-folha O(m) - pesquisa limitada em profundidade
Alpha-Beta	$O(b^{m/2})$ *	

Onde, m – profundidade máxima da árvore de pesquisa, b - número de jogadas possíveis em cada estado.

* Quando ordenado.

Jogo do Galo





1. O tabuleiro contêm três linhas e três colunas.
2. Dois jogadores escolhem um símbolo cada um, normalmente um (O) e um (X).
3. Os jogadores jogam alternadamente numa das nove posições, desde que vazia.
4. O jogador atinge o objetivo quando coloca o seu símbolo nas três posições de um dos seguintes conjuntos :

LINHAS	COLUNAS	DIAGONAIS
{1,2,3}	{1,4,7}	{1,5,9}
{4,5,6}	{2,5,8}	{3,5,7}
{7,8,9}	{3,6,9}	

Jogador (X) ganha.

Nenhum dos jogadores ganha.

Implementação

Os algoritmos foram implementados na linguagem C++.

Estrutura de dados

Os estados do jogo são guardados num *std::array* de c*hars* com 9 posições, uma para cada casa do tabuleiro. Cada uma das 9 posições de jogo contém o valor 0, 1 ou 2, que significam que a posição está vazia, contêm um (X) ou um (O), respetivamente. A utilização de *chars* acontece porque não era necessário um inteiro de 16 bits visto que utilizamos no máximo 2 bits para guardar a informação de cada posição.

Função Utilidade

Na implementação dos algoritmos aplicados ao Jogo do Galo foram utilizadas duas funções utilidade. Uma delas é utilizada quando estamos na presença de nós-folha, a outra é utilizada para calcular a utilidade de todos os outros nós.

Nós Finais

Esta função é extremamente simples. Quando encontramos estes nós estamos na presença de uma de três situações possíveis. Ou nenhum dos jogadores atingiu o objetivo, e por isso há um retorno do valor zero a indicar empate, ou um deles conseguiu vencer. Neste caso será retornado um valor positivo ou negativo em caso de vitória ou derrota, respetivamente. Esse valor terá de ser maior do que todos os possíveis valores retornados pela outra função utilidade de modo a garantir que estes casos são os melhores e os piores casos que se pode atingir no jogo. Note que o zero indica uma situação neutra.

Nós Não Finais

Para se determinar o valor de utilidade, no caso de a configuração não ser terminal, basta subtrair o número de hipóteses que o computador tem de vitória, pelo número de hipóteses que o jogador tem. Este cálculo é feito avaliando a configuração atual do tabuleiro. A avaliação consiste na contagem de linhas, colunas e diagonais que só contem um dos símbolos ou casas em branco. Isto é, um jogador só pode atingir o objetivo usando uma determinada linha se esta ainda não contiver o símbolo do adversário.

Verificação Estado Final

Esta função tem como objetivo saber se uma determinada configuração é final ou não. Para ser final é preciso que se verifiquem uma de duas coisas. É claro que quando todas as posições já se encontram ocupadas não é possível haver uma nova jogada. A outra condição que também indica se o estado atual é final é o objetivo ter sido atingido. Isto é, caso exista um conjunto do grupo de conjuntos vistos anteriormente, que contenha nas três posições o mesmo símbolo.

Limitação Profundidade

Inicialmente, e tendo em conta o baixo valor do fator de ramificação deste jogo, não demos importância até onde se expandia a árvore de busca. Para além disso, a profundidade máxima da árvore é apenas 9. No entanto, após a implementação deste jogo sem limitar a profundidade, decidimos melhorar os resultados obtidos, tendo então, decidido limitar a profundidade, o que seria obrigatório em jogos com maior nível de complexidade.

Em vez de este limite estar presente em todos os jogos, optamos por dar ao utilizador a oportunidade de tomar essa decisão no início de cada jogo. O limite definido assume o valor 4, uma vez que o valor 3 não dá as garantias necessárias. Isto é, em algumas situações em que o utilizador inicie o jogo pode-se dar o caso de o computador perder o jogo.

Jogada do Computador

Esta é a função que determina a jogada computador, de nome *computerMove*, que retorna o tabuleiro após a jogada ter sido efetuada para ser impresso ao jogador. Esta função cria os nós-filho do tabuleiro que lhe é passado como argumento e chama a função min_value para cada um deles. O máximo retorno de todas as chamadas à min_value determina o retorno da função. Note-se que na primeira jogada do jogo, em caso de ser o computador a jogar, apenas são criados 3 filhos porque todos os outros são simétricos. Assim, melhoramos significativamente os recursos gastos no pior caso.

Jogada do Utilizador

Função muito simples que apenas necessita de duas ações. A leitura de um número inteiro enquanto este não cumpra dois requisitos. Um deles é estar presente no domínio [1-9], uma vez que são estas as posições da tabela. Depois de verificado o primeiro é necessário que a posição onde jogou não tenha sido ocupada anteriormente, isto porque as regras só permitem jogadas em casas desocupadas.

Resultados

Explicação

Durante a avaliação dos resultados foi realizado um jogo igual em cada um dos algoritmos. Isto é, para todas as escolhas foram tomadas as mesmas decisões. Deste modo existe mais precisão ao comparar os diferentes algoritmos. Atribuímos ao computador o símbolo (O) dando-lhe depois a oportunidade de ser o primeiro a jogar. Assim foi possível observar o seu comportamento na escolha que requer mais análise.

```
Movimento do
Computador: 1

     |     |
  O  |  2  |  3
     |     |
-----+-----+-----
     |     |
  4  |  5  |  6
     |     |
-----+-----+-----
     |     |
  7  |  8  |  9
     |     |
```

```
Insira o numero da
casa em que quer
jogar: 3

     |     |
  O  |  O  |  X
     |     |
-----+-----+-----
     |     |
  4  |  X  |  6
     |     |
-----+-----+-----
     |     |
  7  |  8  |  9
     |     |
```

```
Movimento do
Computador: 6

     |     |
  O  |  O  |  X
     |     |
-----+-----+-----
     |     |
  X  |  X  |  O
     |     |
-----+-----+-----
     |     |
  O  |  8  |  9
     |     |
```

```
Insira o numero da
casa em que quer
jogar: 5

     |     |
  O  |  2  |  3
     |     |
-----+-----+-----
     |     |
  4  |  X  |  6
     |     |
-----+-----+-----
     |     |
  7  |  8  |  9
     |     |
```

```
Movimento do
Computador: 7

     |     |
  O  |  O  |  X
     |     |
-----+-----+-----
     |     |
  4  |  X  |  6
     |     |
-----+-----+-----
     |     |
  O  |  8  |  9
     |     |
```

```
Insira o numero da
casa em que quer
jogar: 8

     |     |
  O  |  O  |  X
     |     |
-----+-----+-----
     |     |
  X  |  X  |  O
     |     |
-----+-----+-----
     |     |
  O  |  X  |  9
     |     |
```

```
Movimento do
Computador: 2

     |     |
  O  |  O  |  3
     |     |
-----+-----+-----
     |     |
  4  |  X  |  6
     |     |
-----+-----+-----
     |     |
  7  |  8  |  9
     |     |
```

```
Insira o numero da
casa em que quer
jogar: 4

     |     |
  O  |  O  |  X
     |     |
-----+-----+-----
     |     |
  X  |  X  |  6
     |     |
-----+-----+-----
     |     |
  O  |  8  |  9
     |     |
```

```
Movimento do
Computador: 9

     |     |
  O  |  O  |  X
     |     |
-----+-----+-----
     |     |
  X  |  X  |  O
     |     |
-----+-----+-----
     |     |
  O  |  X  |  O
     |     |
```

Apresentação de Números

Minimax, não limitado

```
Empate.
               Nos Vistos    Tempo
Escolha 1:     179115        0.027843017
Escolha 2:     7331          0.000962306
Escolha 3:     197           0.000030808
Escolha 4:     13            0.000004376
Escolha 5:     1             0.000002798
```

Alpha-Beta, não limitado

```
Empate.
               Nos Vistos    Tempo
Escolha 1:     24865         0.011064187
Escolha 2:     3143          0.000429939
Escolha 3:     161           0.000024775
Escolha 4:     13            0.000004417
Escolha 5:     1             0.000002788
```

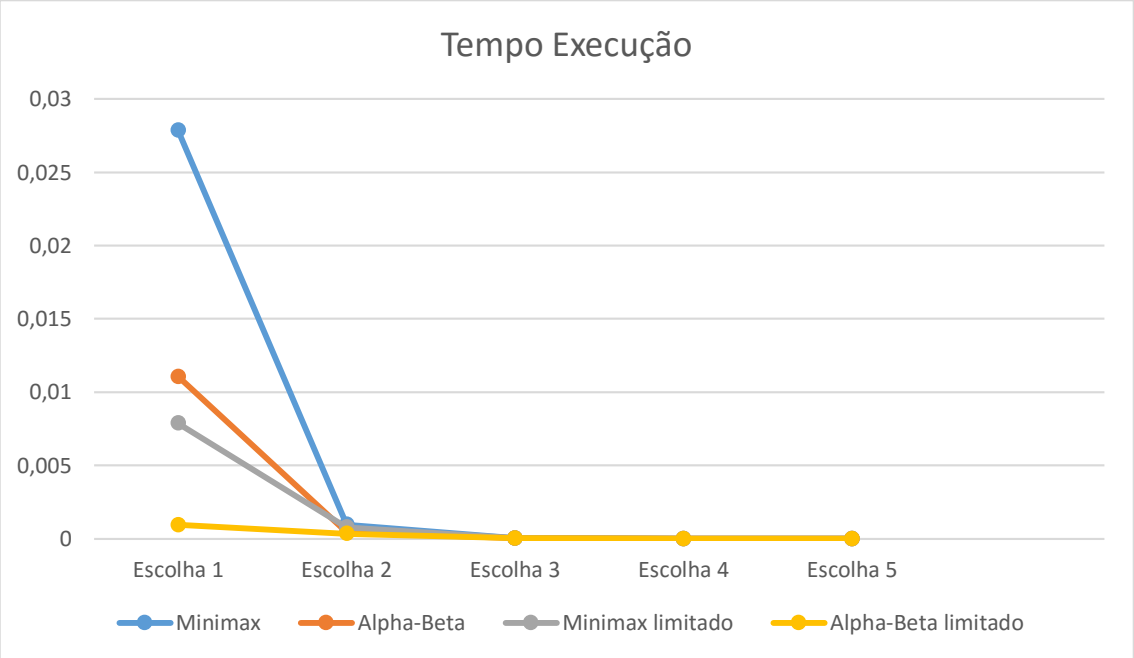
Minimax, limitado

```
Empate.
               Nos Vistos    Tempo
Escolha 1:     24243         0.007869339
Escolha 2:     5747          0.000772567
Escolha 3:     197           0.000031235
Escolha 4:     13            0.000004323
Escolha 5:     1             0.000002826
```

Alpha-Beta, limitado

```
Empate.
               Nos Vistos    Tempo
Escolha 1:     5756          0.000933876
Escolha 2:     2521          0.000339488
Escolha 3:     161           0.000025900
Escolha 4:     13            0.000004361
Escolha 5:     1             0.000002783
```

Gráficos



Os resultados foram obtidos utilizando uma máquina com sistema de operação Ubuntu 14.04, processador Intel Core i7 4710HQ 2.5GHz e 12 GB de memória RAM.

Comentários

Como é possível observar nas tabelas e nos gráficos apresentados anteriormente, o desempenho dos algoritmos é diferente. No entanto a diferença só é muito significativa nas primeiras duas jogadas, sendo que a partir desse ponto são mínimas. O motivo para tal, é existirem poucas ou nenhuma posição da tabela preenchidas inicialmente, sendo a quantidade de jogadas possíveis a analisar maior.

O algoritmo *Minimax* é o que tem pior desempenho. Este era um resultado esperado e facilmente compreendido porque os outros são um aperfeiçoamento seu. Isto é, a árvore de pesquisa criada por este algoritmo é a maior de todas porque não sofre nenhuma poda e é expandida até aos nós-folha.

O algoritmo *Alpha-Beta* apresenta um desempenho bastante satisfatório. A grande diferença no número de nós expandidos na primeira jogada do computador, constatada já anteriormente, acontece porque o algoritmo "poda" nós com utilidades inferiores ou iguais ao valor alfa e os superiores ou iguais a beta.

Quando limitamos o algoritmo *Minimax* o desempenho aumenta de forma significativa, aproximando-se mesmo dos valores do *Alpha-Beta*. Assim conseguimos ter uma ideia do nível onde correm as podas do algoritmo anterior. Isto é, a média dos níveis onde ocorrem as mesmas será semelhante ao nosso limite.

A melhor performance acontece indiscutivelmente quando se limita o algoritmo *Alpha-Beta.* Um resultado também esperado pelo facto de não existirem nós gerados depois do limite e, para além disso, continuar a ocorrer a poda da árvore de busca em níveis inferiores.

Relembramos que estas análises de dados são relativas essencialmente às primeiras escolhas. Nas jogadas seguintes, a diferença já não é tão acentuada, porque o número de jogadas possíveis diminui, fazendo com o que nível não seja atingido e que a diferença entre o número de nós expandidos e "podados" seja menor.

Bibliografia

https://www.dcc.fc.up.pt/~ines/aulas/1516/IA/IA.html