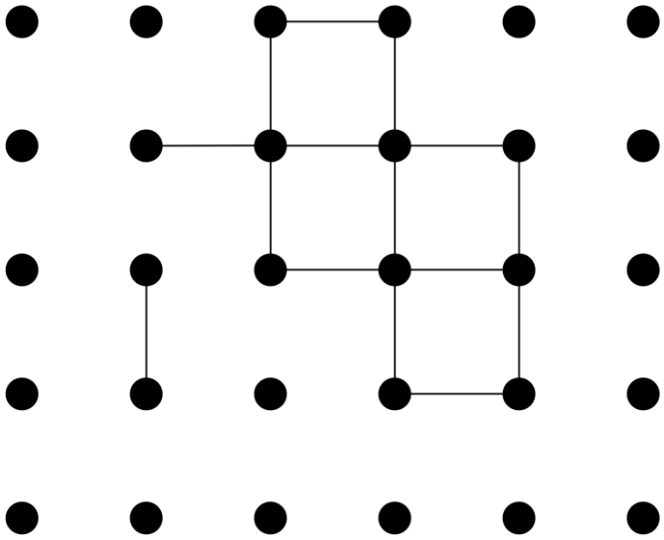
Manual Técnico



Puzzle dos Pontos e das Caixas

Docentes da disciplina

Joaquim Filipe

Cédric Grueau

Um trabalho realizado por

Artur Esteves 140221076 Engenharia Informática

Daniel Costa 120221058 Engenharia Informática

"Computers are like humans they do everything except think"

John von Neumann

Conteúdo

Introdução

No âmbito da disciplina Inteligência Artificial, do curso de Engenharia informática do Instituto Politécnico de Setúbal – Escola Superior de Tecnologia de Setúbal, foi proposto pelos docentes da disciplina o desenvolvimento de um projeto, em LISP, denominado por "Puzzle dos pontos e caixas", de modo que os alunos possam aplicar os seus conhecimentos adquiridos nas aulas práticas e laboratórios da disciplina.

Este programa visa conceder ao utilizador a hipótese de jogar o puzzle dos pontos e das caixas contra um adversário, sendo ele um humano ou uma máquina. Consiste num jogo tático para dois jogadores, e tem como objetivo de conseguir fechar o maior número de caixas através do acréscimo de arcos, entre dois pontos adjacentes tanto na horizontal como vertical, a partir de uma configuração inicial do tabuleiro. O jogador que ganha a partida é o jogador que tiver fechado o maior número de caixas.
Para mais informações sobre as características do jogo é favor consultar o enunciado do projeto de forma a integrar-se no jogo e o seu modo de procedimento.

Ambiente de desenvolvimento

Software

A Tabela 1 Softwares utilizados, contém os softwares utilizados para o desenvolvimento deste projeto:

TABELA 1 SOFTWARES UTILIZADOS

LispWorks 7.0 (x64)	Ferramenta de desenvolvimento
Notepad++	Editor de Texto
GIT	Repositório
Souce Tree	Gestão do repositório

Requisitos

A Tabela 2 Requisitos mínimos software, contém os requisitos mínimos e recomendados das máquinas de modo a suportar o LispWorks 7.0 (64bits):

TABELA 2 REQUISITOS MÍNIMOS SOFTWARE

Requisitos mínimos	
Windows (x64)	Windows (x32)
Processador AMD64|Intel 64 Memória de disco: 170MB Sistema Operativo: Windows Vista SP2 (x64)	Processador Pentium 4 Memória de disco: 170MB Sistema Operativo: Windows Vista SP2 (x64)
Macintosh (x64)	**Macintosh (x32)**
Processador Intel-based Macintosh Memória de disco: 285MB Sistema Operativo: OS X v.10.5	Processador Intel-based Macintosh Memória de disco: 265MB Sistema Operativo: OS X v.10.5

Algoritmo geral

Conteúdo da aplicação

Este projeto foi proposto com a seguinte estrutura:

- Jogo.lisp

  Ficheiro responsável pela interação com o utilizador, imprimindo menus no ecrã da máquina com as opções.

- alfabeta.lisp

  Encontra-se neste ficheiro, o algoritmo alfa-beta, as funções MAX e MIN, função utilidade, sucessores e funções auxiliares. Ficheiro responsável pela execução do algoritmo de modo a permitir mais tarde, a jogada "inteligente" do computador.

- pontosecaixas.lisp

  Ficheiro responsável pela gestão das regras do domínio da aplicação. Neste ficheiro são encontrados os operadores, validações e outras funções úteis para o seu desenvolvimento.

NOTA:
Gostaríamos de alertar para a utilização dos ficheiros nos diversos Sistemas Operativos (Mac OS X; LINUX; Elementary entre outros), na qual não tínhamos qualquer meio de utilizar nesses mesmos ambientes.

Algoritmos

Neste projeto foi proposto a utilização de um algoritmo que fosse capaz de executar um jogo entre dois jogadores (multi-jogador). Para tal foi utilizado o algoritmo alfabeta, derivado de um outro que é designado por Minimax, a grande diferença é que o Alfabeta realiza cortes na árvore e estes cortes não provocam a perda de informação relevante, sendo este mais eficiente que o algoritmo MINIMAX. Ao longo das próximas secções vamos descrever algumas das suas características.

Minimax

O algoritmo minimax tem como objetivo minimizar a perda máxima possível, onde determina a estratégia ótima para o valor MAX. Normalmente é utilizado em jogos de dois jogadores, onde começam ambos a zero, como por exemplo o jogo do galo, damas entre outros. Outros jogos como Xadrez, tem fatores que não são ideais (tempo de jogada, fator médio de ramificação, nós analisados, …).

A tabela seguinte mostra a forma sintetizada a execução do algoritmo:

TABELA 3 MINIMAX

1. Gerar toda a árvore de procura desde o nó inicial até aos nós terminais.
2. Aplicar a função de utilidade a cada nó terminal para determinar o respetivo valor.
3. Usar a utilidade dos nós terminais para determinar através de um processo de *backup* a utilidade dos nós no nível imediatamente acima na árvore de procura:
   a. Se for um lance do MIN o valor calculado é o mínimo dos nós do nível inferior;
   b. se for o MAX a jogar, o valor calculado é o máximo.
4. Continuar a usar o processo de *backup* um nível de cada vez até atingir o nó inicial.
5. Tendo chegado ao nó inicial, realizar o lance correspondente ao valor determinado para esse nó.

Em pseudocódigo o algoritmo comporta-se da seguinte forma:

TABELA 4 PSEUDOCODIGO MINIMAX

```
Determinar
        SE {
        profundidade limite atingida
        OU Nível é Minimizador
        OU Nível é Maximizador }
        ENTÃO
                SE profundidade limite
                        Calcular valor do estado corrente
                        Retornar resultado
                SE Nível Minimizador
                        Aplicar minimax aos sucessores
                        Retornar Mínimo
                SE Nível Maximizador
                         Aplicar minimax aos sucessores
                        Retornar Máximo
```

Segue um exemplo de um minimax, onde ▲ representa o MAX e ▼ representa MIN (Figura 1 Exemplo minimax):

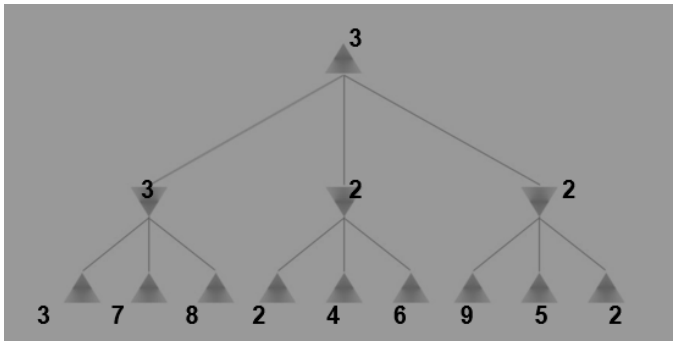


FIGURA 1 EXEMPLO MINIMAX

Algumas anotações do algoritmo minimax:

- Se a profundidade máxima da árvore for ***m*** e em cada ponto houver ***b*** lances possíveis (fator de ramificação), então a complexidade (temporal) do minimax é $O(b^m)$.
- O algoritmo pode ser implementado essencialmente como *procura* em profundidade primeiro (depth-first) embora usando recursividade em vez de uma fila de nós por isso os requisitos de espaço são lineares em ***b*** e ***m***.
- Para problemas reais o custo de tempo é geralmente inaceitável, mas este algoritmo serve de base a outros métodos mais realistas, bem como de suporte à análise matemática de jogos.

Alfa-beta

O algoritmo alfabeta é uma variação do algoritmo minimax que visa calcular a árvore recorrendo a cortes, de modo a evitar a analise de todos os nós dessa mesma. Este algoritmo pára de avaliar um nó, quando encontra um que seja pior que o previamente examinado, como tal, os movimentos posteriores não precisam de ser avaliados, ou seja, ocorre um corte. Por outras palavras, a procura pode ser descontinuada abaixo de qualquer nó MIN com um valor beta ≤ alfa de qualquer dos seus antecessores MAX (corte alfa). No caso de se encontrar abaixo de qualquer nó MAX com um valor alfa ≥ beta de qualquer dos seus antecessores MIN (corte beta).

Em pseudocódigo o algoritmo comporta-se da seguinte forma:

TABELA 5 PSEUDOCODIGO ALFABETA

AlfaBeta(n; α; β)

1. Se ***n*** no limite de profundidade ***d***, devolve AlfaBeta(***n***)=*f*(***n***), caso contrário calcula os sucessores n_1, ...,n_k, ... n_b (por ordem), faz k=1 e, se ***n*** é um nó MAX, vai p/2 c.c. vai p/ ii.
2. α ← max[α, AlfaBeta(n_k; α; β)]
3. Se α ≥ β devolve β; c.c. continua
4. Se k=b devolve α; c.c. vai para n_{k+1} i.e. k← k+1 e vai p/2
    a. β ← min[β, AlfaBeta(n_k; α; β)]
    b. Se β ≤ α devolve α; c.c. continua
    c. Se k=b devolve β; c.c. vai para n_{k+1} i.e. k k+1 e vai p/ii

Segue um exemplo de um alfabeta, onde ▲ representa o MAX e ▼ representa MIN (Figura 2 exemplo alfabeta)

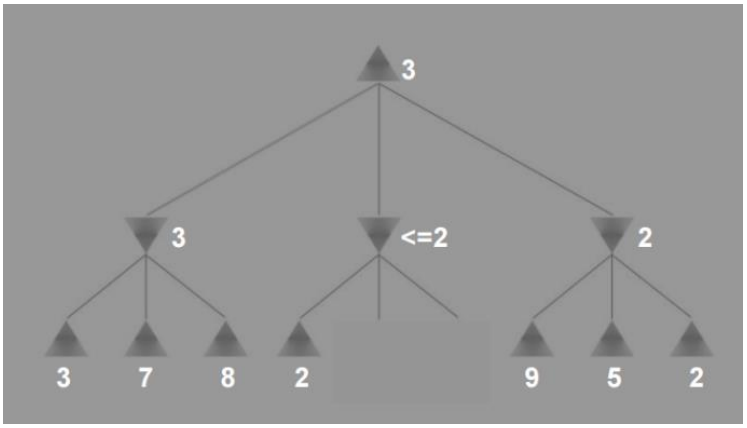


FIGURA 2 EXEMPLO ALFABETA

Algumas considerações:

- Notar que a procura alfabeta é do tipo *depth-first*, pelo que em cada instante apenas é necessário considerar os nós ao longo de um ramo da árvore de procura.
- Seja α o valor da melhor escolha encontrada até ao momento, ao longo do ramo corrente, para MAX.
- Seja β o valor da melhor escolha encontrada até ao momento, ao longo do ramo corrente, para MIN.
- O Alfabeta atualiza o valor de α e de β ao longo da procura e corta uma subárvore (terminando uma chamada recursiva) logo que se sabe ser pior que os valores correntes de α e de β.

Função Utilidade

A função utilidade, também chamada função objetivo, retorna uma estimativa da utilidade a partir da posição. Esta função vai verificar o resultado do vencedor, e se existir retorna um valor, neste caso 2000, caso contrário retorna – 2000. Caso não exista um vencedor, isto é, empate, a função utilidade verifica o resultado através do número de caixas do jogador, se o jogador 2 tiver mais caixas fechadas que o jogador 1, será retornado o valor da utilidade antiga mais 50, caso contrário retorna -50. Se nenhuma das anteriores condições forem aceites, será retornado o valor da utilidade que a função recebe como parâmetro.

Diagrama - Jogo

O diagrama de sequência que se segue, é uma representação de baixo nível, da interação que o utilizador terá quando inicia o programa na consola (Ilustração 1 DIagrama sequências da interação inicial)

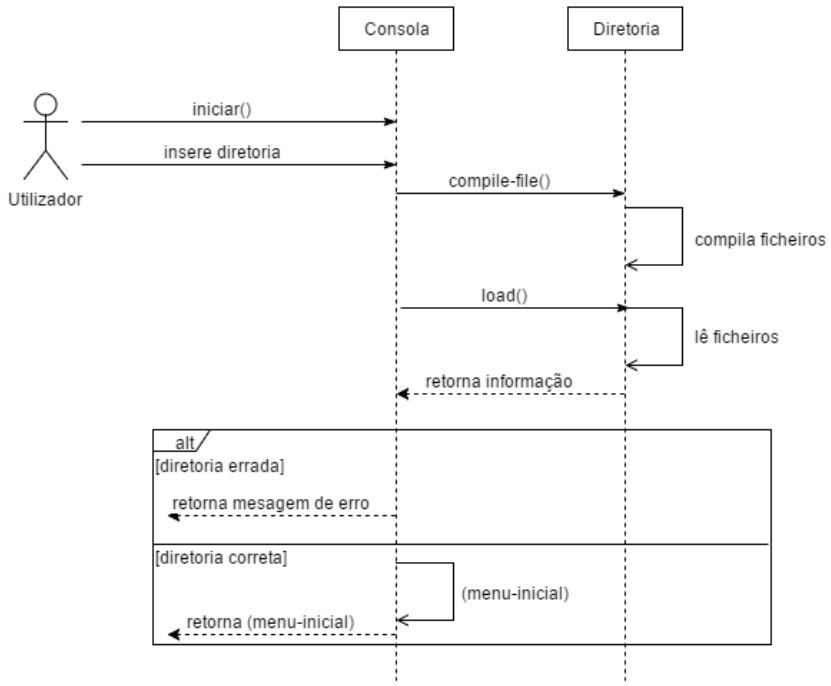


ILUSTRAÇÃO 1 DIAGRAMA SEQUÊNCIAS DA INTERAÇÃO INICIAL

O seguinte diagrama ilustra a representação de baixo nível lógico do jogo entre o humano com a máquina. Neste exemplo começamos por inicializar a jogada com o humano. (Ilustração 2 DIAGRAMA SEQUÊNCIAS Da interação humano e máquina).

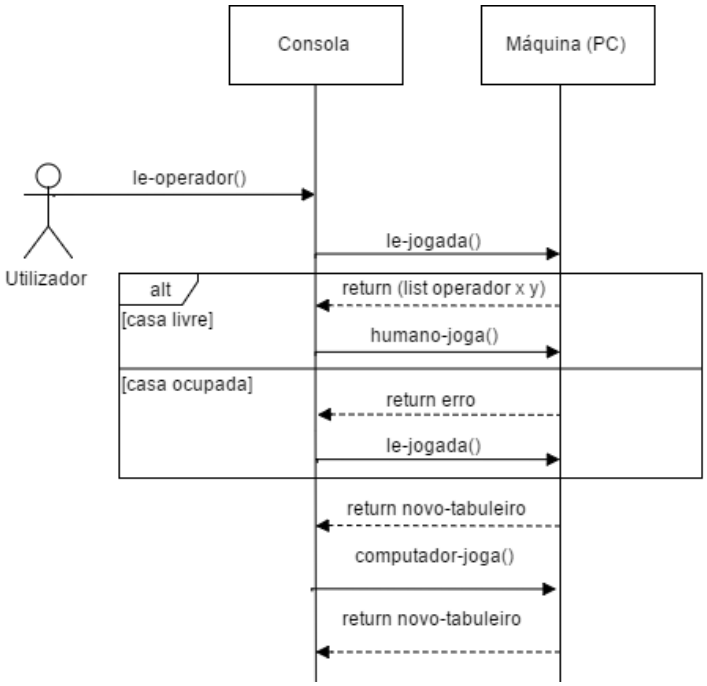


ILUSTRAÇÃO 2 DIAGRAMA SEQUÊNCIAS DA INTERAÇÃO HUMANO E MÁQUINA

Análise dos resultados

Resultados

Segue os resultados finais, de duas jogadas, onde o vencedor, em ambas, foi a máquina (Tabela 6 Resultados):

TABELA 6 RESULTADOS

		Humano vs Computador
Resultado 1	Vencedor	2
	Caixas Fechadas	25
	Nós analisados	647092
	Corte Alfa	4860
	Corte Beta	65286
	Tempo Dispensado	3
Resultado 2	Vencedor	2
	Caixas Fechadas	25
	Nós analisados	329717
	Corte Alfa	2406
	Corte Beta	36736
	Tempo Dispensado	5
Resultado 3	Vencedor	2
	Caixas Fechadas	25
	Nós analisados	353286
	Corte Alfa	3247
	Corte Beta	24816
	Tempo Dispensado	0

Em ambos os resultados o vencedor foi o computador, e verificou assim que alcançou as 25 caixas fechadas do tabuleiro.

Limitações da aplicação

Este projeto apresenta uma limitação que é a eficiência da jogada do computador. Este problema deve-se ao fato do computador analisar todas as possibilidades de colocar um arco, tento em conta a jogada do humano. A função de utilidade utilizada neste projeto também não é melhor, isto significa uma diminuição no desempenho do algoritmo alfabeta. É de notar que estamos a lidar com um tabuleiro de dimensão 7 x 7, o que dá um total de 49 caixas, que também irá influenciar as jogadas que o utilizador fazer, isto porque, não existe qualquer elemento (por exemplo cor) que faça uma distinção vistosa entre jogador 1 e jogador 2.

Conclusão

Ao longo do desenvolvimento deste segundo projeto, nestas últimas semanas do 1º semestre escolar, foi bastante enriquecedor no que toca aos assuntos abordados de LISP, nomeadamente o estudo da teoria de jogos com os algoritmos minimax e alfabeta, no entanto, e de notar que é uma linguagem que apresenta um grau de dificuldade exigente.

Inicialmente, encontramos algumas dificuldades de codificação, no desenvolvimento dos sucessores e alfabeta, mas assim que essa foi ultrapassada deparamo-nos com uma fraca eficiência do programa. Também a função utilidade que foi desenvolvido podia ser melhorada, investigando mais de modo a procurar a solução ótima.

Em suma, achamos um projeto desafiante, que apresentou alguns obstáculos, mas no fim conseguimos ultrapassar os de maior preocupação, restando apenas os apresentados nas limitações.

Bibliografia

Fontes Literárias

- Filipe, Joaquim (2015) – Teoria de Jogos, Setúbal.

Fontes de endereço

- Calado Luis, Sousa João, Campos José e Silva Rodolfo, Algoritmo Minimax. Disponível em https://web.fe.up.pt/~eol/IA/IA0809/APONTAMENTOS/Alunos_MiniMax.pdf Acesso em 27 janeiro de 2017.
- Prof. Marcos Quinet, Universidade Federal Fluminense (UFF). Disponível em http://www.professores.uff.br/mquinet/07_IA.pdf Acesso em 27 janeiro de 2017.
- Wikipédia, Minimax. Disponível em https://pt.wikipedia.org/wiki/Minimax Acesso em 27 janeiro de 2017.
- Wikipédia, Poda alfa-beta. Disponível em https://pt.wikipedia.org/wiki/Poda_(computa%C3%A7%C3%A3o) Acesso em 27 janeiro de 2017.
- AED (2002), Teoria de Jogos. Disponível em http://comp.ist.utl.pt/ec-aed/PDFs/AB.pdf Acesso em 27 janeiro de 2017.