Projeto Nº1: Época Normal - Fase nº 2

Inteligência Artificial 19/20

Prof. Joaquim Filipe

Eng. Filipe Mariano

Jogo do Cavalo (Knight Game)

Manual Técnico

Realizado por:

João Gomes - 150221001

André Gastão - 130221037

28 de Dezembro de 2020

Indice

1- Introdução

Este documento é escrito recorrendo á linguagem de marcação markdown, servindo como relatório do manual técnico do projeto Jogo do Cavalo (Knight Game) que é uma variante do problema matemático conhecido como o Passeio do Cavalo (Knight's Tour)

No âmbito da unidade curricular de Inteligência Artificial, foi nos proposto o projecto do jogo “Jogo do Cavalo”, onde esta versão do jogo consiste num tabuleiro com 10 linhas e 10 colunas.

O objectivo deste projeto é um Jogo com dois jogadores, usando um enquadramento teórico e prático adquiridos no âmbito da Teoria de Jogos.

Pretende−se que o programa permita ao computador vencer o jogador humano ou um outro computador,pelo que deverá funcionar em dois modos:

1. Humano vs Computador
2. Computador vs Computador

A resolução dos problemas mencionados será implementada na linguagem de programação funcional Common Lisp, utilizando todos os conhecimentos adquiridos na unidade curricular até ao momento, a fim de dar uma solução apropriada do problema.

Neste documento serão descritas detalhadamente todas as metricas de desenvolvimento usadas e funções implementadas.

2- Teoria de Jogos

Caracteristicas do Knight Game:

- Jogo de 2 adversários
- Jogo simétrico
- Jogo sequencial
- Jogo de soma nula, não cooperativo
- Jogo com explosão combinatória

Jogo de 2 Adversarios

Este jogo caracteriza-se por ter dois adversarios ou seja consiste em ter dois jogadores, inicialmente existe varias possiveis de jogadas tendo um Max contra o Min respectivamente com seu valor da função de utilidade. normalmente o jogador com valor Max tenta maximizar e tomar a decisão para iniciar o jogo. e quanto ao jogador com valor MIN, irá representar o adversario do jogador Max.

Jogo Simetrico

O jogo que permite os jogadores aplicaram as suas estrategias, em que cada jogador escolhe a sua estratégia, assim respectivamente e não a estratégia de quem está jogando, neste caso cada jogador desse jogo representa a sua estratégia diferente de outro para obter o resultado.

Jogo Sequencial

Este jogo trata-se normalmente por jogo dinamico, onde o proximo jogador tem o conhecimento da jogada do seu antecessor. o caso do nosso jogo, o jagador um ou o primeiro jogador poderá ter o conhecimento da jogada do antecessor, podendo necessario realizar a acção que condiciona o antecessor a não poder realizar a acção em particular.

Jogo de soma nula e não cooperativa

o jogo nula ou de soma-zero permite que os jogadores beneficiar para cada combinação de estratégia usada. cada jogador ganha a pontuação com base na falha ou pejuizo do adeversário em cada jogada, considerando o vencedor com a soma das perdas do seu oponente. o jogo não cooperativo não existem formas positivos de suportar alianças, são jogos essencialmente estabelecidos por ameaças crediveis.

Jogo com Explosão combinatória

este jogo permite encontrar uma estratégia otima para certo problema da explosão combinatória atraves das combinações utilizadas por jogador.

3- Arquitetura do Sistema

O sistema do Jogo Do Cavalo foi implementado em linguagem LISP, utilizando o IDE LispWorks. A estrutura do projeto é composta por 4 ficheiros:

- interact.lisp - Interação com o utilizador e escrita em ficheiros
- jogo.lisp - Implementação da resolução do problema incluindo seletores, operadores e outras funcões auxiliares.
- algoritmo.lisp - Implementação do algoritmo alfa-beta e suas funções auxiliares.
- log.dat - ficheiro de historico das execuções do algoritmo alfa-beta cada vez que o CPU joga. Guarda as jogadas realizadas e informação sobre a profundidade e tempo de execução.

4. Entidades e sua implementação

Tipos Abstratos de Dados

É importante referir a captação dos conceitos do domínio do problema em questão e o mapeamento para o tipo abstrato de dados que se implementaram.

Desenvolveu-se dois tipos fundamentais:

- Tabuleiro
- Nó

Tabuleiro

É constituído basicamente por uma matriz 10x10 (lista de listas)10 linhas e 10 colunas em que cada posição do tabuleiro é mapeada com um conjunto (Letra ; Numero) => (Linha ; Coluna).

```
tabuleiro

 '((94 25 54 89 21 08 36 14 41 96)
   (78 47 56 23 05 49 13 12 26 60)
   (00 27 17 83 34 93 74 52 45 80)
```

```
    (69 09 77 95 55 39 91 73 57 30)
    (24 15 22 86 01 11 68 79 76 72)
    (81 48 32 02 64 16 50 37 29 71)
    (99 51 06 18 53 28 07 63 10 88)
    (59 42 46 85 90 75 87 43 20 31)
    (03 61 58 44 65 82 19 04 35 62)
    (33 70 84 40 66 38 92 67 98 97))
```

Nó

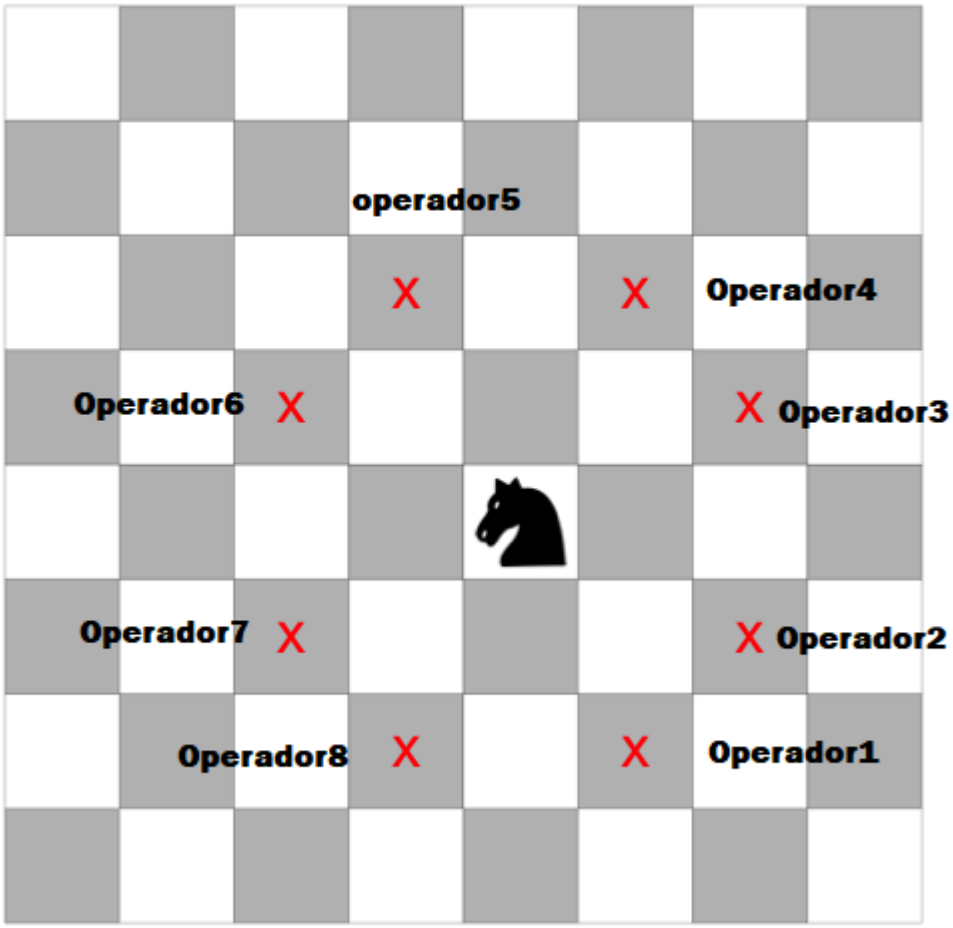
O conceito de nó, usado nos algoritmos de procura, foi captado tendo em conta as necessidades do problema. Inclui o estado do problema, neste caso um tabuleiro , o nó pai (tabuleiro anterior) , o valor (utilidade) do nó, os pontos dos dois jogadores o jogador do movimento.

```
<Nó>::= (<Estado> <Jogador> <PontosJogador1> <PontosJogador2> <Profundidade> <Pai>
<Avaliacao>)

<Estado>            ::= <Tabuleiro>
<Jogador>           ::= <Jogador Actual>
<PontosJogador1>    ::= <Pontos Jogador 1>
<PontosJogador2>    ::= <Pontos Jogador 2>
<Pai>               ::= <Nó Pai>
<Avaliacao>*        ::= <Avaliação do Nó>

*Diferença de Pontos entre Jogadores
```

Operadores

Para a segunda fase do projeto tivemos de realizar alterações nos operados nomendamente estes rceberem o jogador que os irá realizar.

Algumas das alterações foram realizadas sobre as funções validar movimento e a posição do cavalo.



5. Algoritmos e sua implementação

Neste projeto foi implementado o algoritmo Minimax com corte Alfa-Beta

Minimax

Minimax é um algotitmo de decisão que simula a decisão de um jogador em encontrar a jogada ideal para o para ele. Assume que o oponente também jogará com a melhor escolha possivel.

Existem dois Jogadores no Minimax.

- O maximizador procurará a pontuação mais alta possível.
- O minimizador procurará a pontuação mais baixa possível.
- Cada jogador verá a melhor jogada do oponente e conseguirá a melhor para ele (Informação perfeita) .

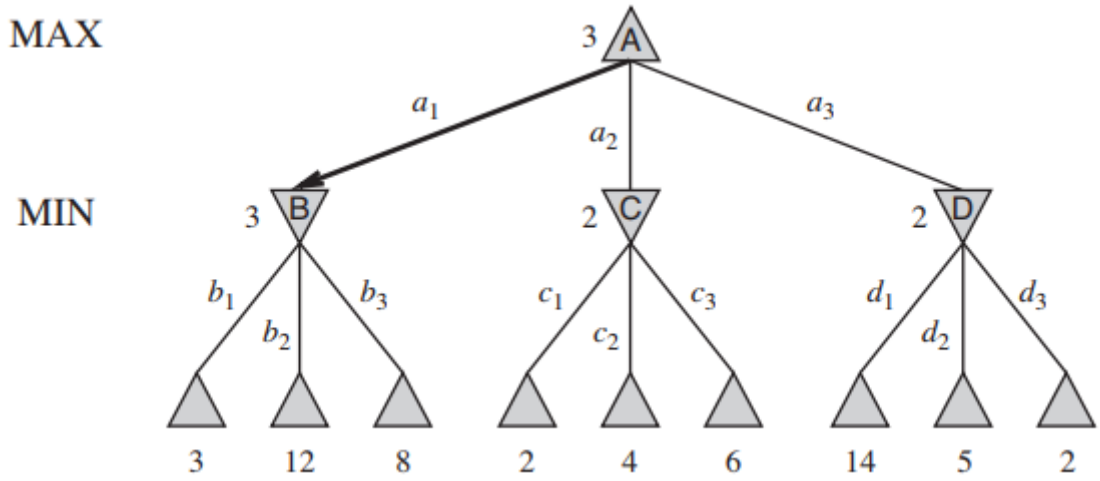
Geralmente, a pesquisa é representada nos dados da estrutura da árvore.

O Minimax consiste determinar a melhor estratégia ao percorrer a ávore com nós folhas desde do inicio ate o fim.(Depth) O Minimax aplica a função de utilidade dos nós de cada nivel de uma arvore para obter o valor, começando de baixo da arvore para acima a cada nível ou do nó final a nó inicial.

para obter o valor do min e max com corte de árvore na procura da profundidade a cada nivel foi utilizado as seguintes formulas:

Nó min é igual a Alfa<= Beta, substituindo o menor valor nó min.

Nó max é igual a Alfa>= Beta, substituindo o maior valor do nó max.



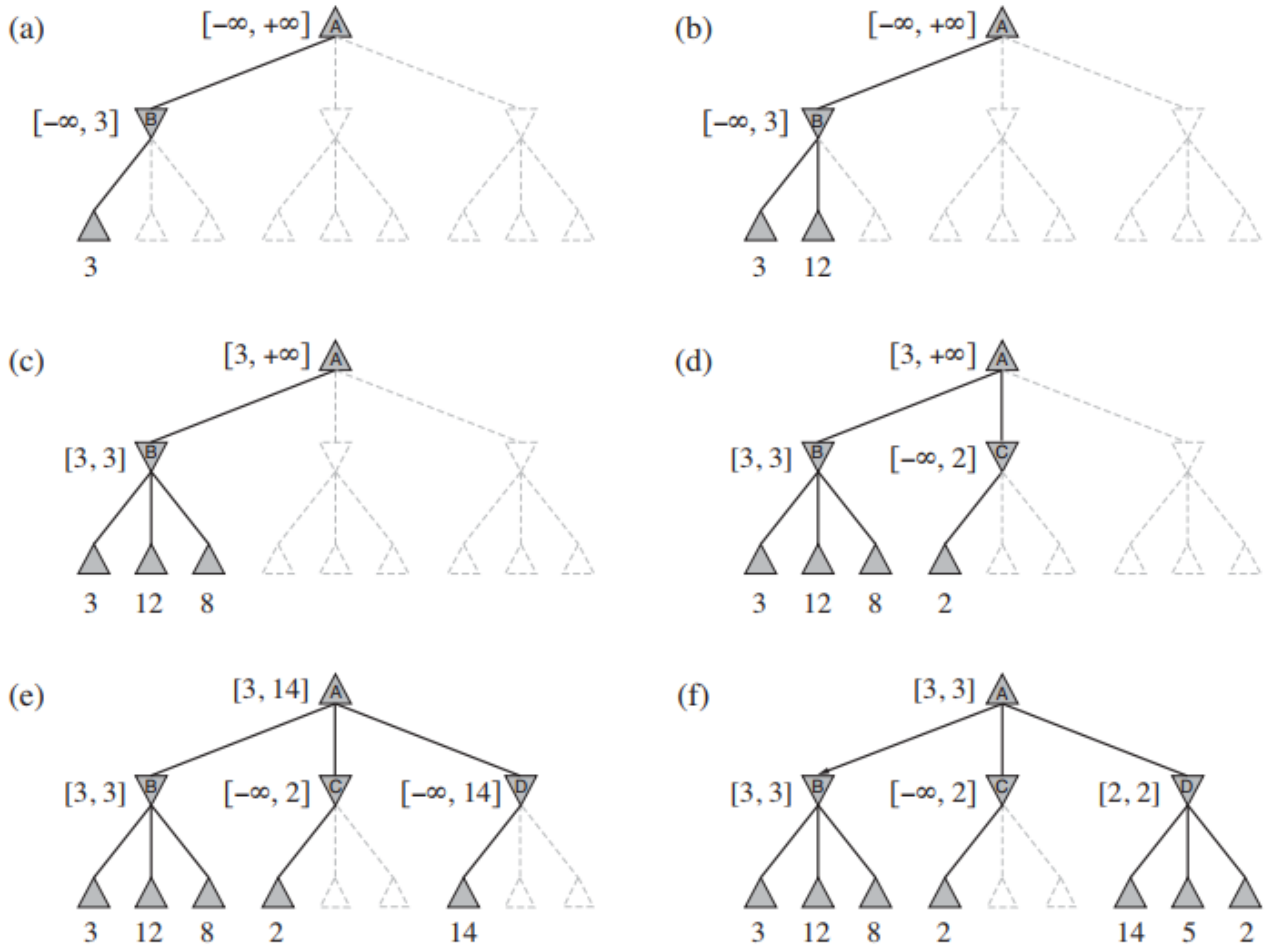
Alfabeta

O alfa-beta com cortes não é realmente um algoritmo, mas uma técnica de otimização para o algoritmo minimax. Reduzindo o tempo de computação por um grande fator. Isso permite-nos pesquisar muito mais rápido e até entrar em níveis mais profundos na árvore do jogo. Ele corta os Nós da árvore do jogo que não precisam ser pesquisados porque já existe um movimento melhor disponível. É chamado de alfa-beta com cortes minimax porque passa 2 parâmetros extras na função minimax (Alpha e Beta).

Quando aplicado ao algoritmo Minimax, ele retornará os mesmos valores que o Minimax, mas será mais rápido.

Vamos definir os parâmetros alfa e beta.

- Alfa é o melhor valor que o maximizador atualmente pode garantir nesse nível ou acima.
- Beta é o melhor valor que o minimizador atualmente pode garantir nesse nível ou acima.
- Nó max com Alfa >= Beta, então faz corte os nós do nivel acima dos nós terminais e devolve o Beta
- Nó min com Beta <= Alfa, então faz corte os nós do nivel acima dos nós iniciais e devolve o Alfa.



Ordenação dos nós

Ordenação dos nós Sucessores pode ser melhorias adicionais, usando heurísticas/função de avaliação para pesquisar partes anteriores da árvore que provavelmente forçam cortes alfa-beta.

A reordenação de nós consiste em calcular o valor heurístico de cada nó filho antes de verificá-los recursivamente. Em seguida, classificar os valores desses estados e invocar a função de ordenação (Quiksort ...) na lista classificada(sucessores).

A ideia é simples: se um estado é bom em profundidade inferior, é mais provável que seja bom em estado profundo também, e se for verdade existirá mais cortes.

Por exemplo, no xadrez, os movimentos que capturam peças podem ser examinados antes dos movimentos que não o fazem, e os movimentos que obtiveram alta pontuação em passes anteriores pela análise da árvore de jogo podem ser avaliados antes dos outros.

A principal razão de ordenarmos os nós sucessores é para a eficiencia Eficiência do minimax com cortes Alfa-Beta

A ordem pode ser estimada com base na função de avaliação estática. A eficácia depende da heurística.

Memoização

Chave	Valor

Chave	Valor
Estado tabuleiro 1	26
Estado tabuleiro 2	44
Estado tabuleiro 3	55
...	...

6. Resultados

a) Número de nós gerados;

b) Número de nós expandidos;

c) Número de nós avaliados;

d) Número de cortes alfa e beta;

e) Tempo que o algoritmo dispendeu a devolver a solução

7. Limitações técnicas e ideias para desenvolvimento futuro

Ao longo do desenvolvimento desta fase do projeto tivemos varias dificuldades em entender e implementar o algoritmo Minimax com cortes Alfa-Beta, em que por um lado podemos dizer que a implementação deste algoritmo foi o maior desafio do projeto. e por outro lado devido ao tempo faltou nos melhorar os operadores, implementar outras funcionalidades adicionais.

Outra das limitações que tivemos foi em realizar a memoização com uma hash-table, pois o LispWorks não tinha memória suficiente para realizar um jogo completo e guardar os valores dos estados, isto sem realizar uma closure.

8 Glossário

Neste glossario mostramos os acrónimos com suas convenções utilizadas

- Atomos: palavras para todos tipos de dados que não são sequenciais, em que neste projeto utilizamos simbolos, numeros(inteiros,reais), Boolean (T ou Nil) e strings
- Listas: tipos de dados dinamicos, usamos diversas listas com listas por exemplo: tabuleiro com 10 linhas e 10 colunas, lita de operadores (operadres no seu todo)
- Selectores: são listas do tipo abstratos, permitindo assim a construção e obter elementos, exemplo: cons,car,cdr etc.
- Macros: consistem em difinir a estrutura sintatica da linguagem para substituir as funções, neste projeto usamos first, second, third etc. nas funções com operadores, estados e pontos.

- Append: é uma função de maipulação de dados que permite acrescentar um valor a uma lista.
- Reverse: é uma função que permite inverter os caracteres num lista
- Concatenate: permite juntar duas ou mais palavras de uma lista.
- Operações matematicas: são funções aritmeticas que permitem efetuar as operações matematicas ou calculos matematicos, nomeadadmente sinais de mais,menos, multiplicação e divisão, função modular , Raiz quadrada entre outros.
- os tipos Bolaeanos ou Predicados: utilzamos null, listp, zerop, etc para verificar os dados e serem validados.
- Equal e eq: são operação de igualidade e tem a sua diferença na forma como são apresentadas.
- variaveis: permite guardar e inicizalar os valores, por exmeplo:
    - Let: avalia as expressões feitas em paralelo.
    - Let*: avalia as sequencias das expressões.
    - defparameter: são variaveis que estão fora do ambiente Léxico, pois são visiveis em todo programa, logo corre-se o risco de provocar efeitos laterais imprestiveis e por convenção tem de se nomear com astériscos(*) à esquerda.
- Funções: permitem construir e avaliar um determinado program, em Lisp o caso não é diferente e neste projeto existem funções para construi tabuleiro e avaliar as espressões.