

Projeto Nº1: Jogo do Cavalo

Manual Técnico



No âmbito da Unidade Curricular de Inteligência Artificial

Docente: Prof. Joaquim Filipe

Elaborado por:

João Quintiliano nº 201900287

Conteúdo

| | |
|---|---|
| Introdução..... | 3 |
| Composição do Projeto..... | 4 |
| Implementação | 4 |
| Operadores..... | 5 |
| Composição do nó | 6 |
| Funções seletoras do nó..... | 6 |
| Algoritmos | 6 |
| Breadth First Search (BFS) | 6 |
| Depth First Search (DFS)..... | 7 |
| A* (A-Star) | 8 |
| Resultados..... | 8 |
| BFS | 8 |
| DFS | 8 |
| Limitações técnicas e Melhorias futuras | 9 |
| Requisitos não implementados | 9 |

Introdução

Este Manual Técnico tem como finalidade descrever os diferentes objetos que compõem o projeto, a sua implementação, os diferentes algoritmos, as limitações encontradas e os resultados obtidos.

Este projeto foi realizado com o objetivo de aplicar os conhecimentos adquiridos nas aulas da unidade curricular de Inteligência artificial através da aplicação e programação dos algoritmos Breadth First Search (BFS), Depth First Search (DFS) e A* (A-star) em Lisp puro. Para a aplicação destes algoritmos foi programado o jogo do cavalo e usado os algoritmos supramencionados de forma a procurar as melhores jogadas.

O jogo do cavalo usa um tabuleiro de 10x10 como o de xadrez. Normalmente este é um jogo que pode ser jogado a dois, mas para este contexto vamos ter apenas um jogador. O cavalo pode realizar movimentos em L como o cavalo no xadrez podendo fazer em todas as direções totalizando a possibilidade de 8 movimentos. No contexto do projeto foram propostos os desafios de A a F anexados. Para a procura das melhores jogadas de forma a cumprir com o desafio dos problemas foram usados os algoritmos BFS, DFS e A*.

Composição do Projeto

O projeto foi implementado em Lisp Puro através do software LispWorks.

O Projeto foi dividido em 3 ficheiros distintos de forma a manter o projeto organizado:

- projeto.lisp → Carrega os outros ficheiros de código, escreve e lê ficheiros, e trata da interação com o utilizador.
- puzzle.lisp → Código relacionado com o problema.
- procura.lisp → Implementação dos algoritmos
- problemas.dat → Apresenta-se neste ficheiro os problemas propostos de A a F.

Implementação

Tabuleiro

O tabuleiro é representado sob a forma de uma lista composta por 10 outras listas, cada uma delas com 10 átomos. Cada uma das listas representa uma linha do tabuleiro, enquanto cada um dos átomos possui o valor da respetiva casa. Cada linha do tabuleiro será representada por um número de 1 a 10 e cada coluna será representada por uma letra de A a J tal como num tabuleiro de xadrez. Cada átomo pode tomar o valor de 00 a 99. As casas que já foram visitadas deixaram de ter um valor atribuído impossibilitando que o cavalo possa cair nesta casa novamente.


| | A | B | C | D | E | F | G | H | I | J | |
|----|----|----|----|----|----|----|----|----|----|---|----|
| 1 | | | | | | | | | |  | 1 |
| 2 | | | | | | | | | | | 2 |
| 3 | | | | | | | | | | | 3 |
| 4 | | 88 | | | | | | | | | 4 |
| 5 | 44 | | | | | | | | | 79 | 5 |
| 6 | 32 | | 34 | | 27 | | | | | | 6 |
| 7 | 55 | 87 | 72 | 58 | 81 | 40 | 17 | | 38 | 95 | 7 |
| 8 | 00 | 29 | 70 | 42 | 13 | 10 | | 78 | 14 | 24 | 8 |
| 9 | 56 | 01 | 71 | 39 | 84 | 77 | 03 | 25 | 63 | 66 | 9 |
| 10 | 65 | 90 | 89 | 45 | 46 | 02 | 57 | 31 | 86 | 53 | 10 |
| | A | B | C | D | E | F | G | H | I | J | |

Figura 1: Exemplo de tabuleiro

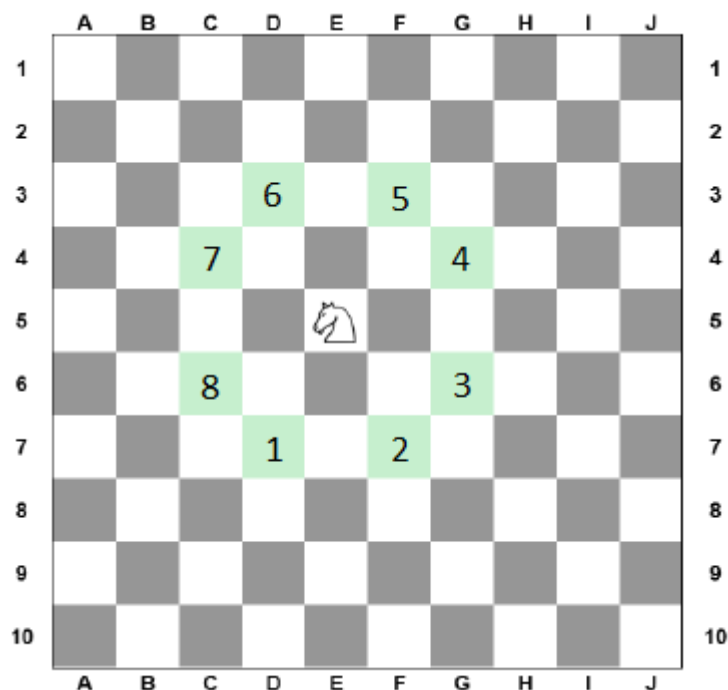
```
(  
  (NIL NIL NIL NIL NIL NIL NIL NIL NIL T)  
  (NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL)  
  (NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL)  
  (NIL 88 NIL NIL NIL NIL NIL NIL NIL NIL)  
  (44 NIL NIL NIL NIL NIL NIL NIL NIL 79)  
  (32 NIL 34 NIL 27 NIL NIL NIL NIL NIL)  
  (55 87 72 58 81 40 17 NIL 38 95)  
  (0 29 70 42 13 10 NIL 78 14 24)  
  (56 1 71 39 84 77 3 25 63 66)  
  (65 90 89 45 46 2 57 31 86 53)  
)
```

Figura 2: Representação do tabuleiro

Operadores

Os operadores representam os movimentos possíveis num determinado estado. Para o Problema do Cavalo o máximo de movimentos possíveis serão 8, o que consequentemente faz a necessidade da implementação de 8 operadores (linha, coluna):

- Operador 1 \rightarrow (2, -1)
- Operador 2 \rightarrow (2, 1)
- Operador 3 \rightarrow (1, 2)
- Operador 4 \rightarrow (-1, 2)
- Operador 5 \rightarrow (-2, 1)
- Operador 6 \rightarrow (-2, -1)
- Operador 7 \rightarrow (-1, -2)
- Operador 8 \rightarrow (1, -2)



Regras de Movimentação

Regra da Colocação do Cavalo \rightarrow para iniciar o jogo, o cavalo só pode ser colocado numa casa da 1ª linha (A1 – J1)

Regra do Simétrico \rightarrow é atribuído NIL ao valor simétrico da casa visitada (exemplo: 67 \rightarrow 76)

Regra do Duplo \rightarrow é atribuído NIL ao valor duplo mais alto (ex: 99) se a casa visitada for um duplo (ex: 33, 55, 77)

Composição do nó

O nó é composto pelo tabuleiro atual com todas as jogadas realizadas, a profundidade do nó na árvore de procura, a pontuação atual total, os operadores/jogadas realizadas que levaram ao estado atual do tabuleiro, e por fim a pontuação objetivo que se pretende alcançar.

Nó → Tabuleiro | Profundidade | Pontuação | Operadores | Objetivo

Funções seletoras do nó

As funções seletoras do nó implementadas são as seguintes:

- no-tabuleiro
- no-profundidade
- no-pontuacao
- no-operadores
- no-solucao

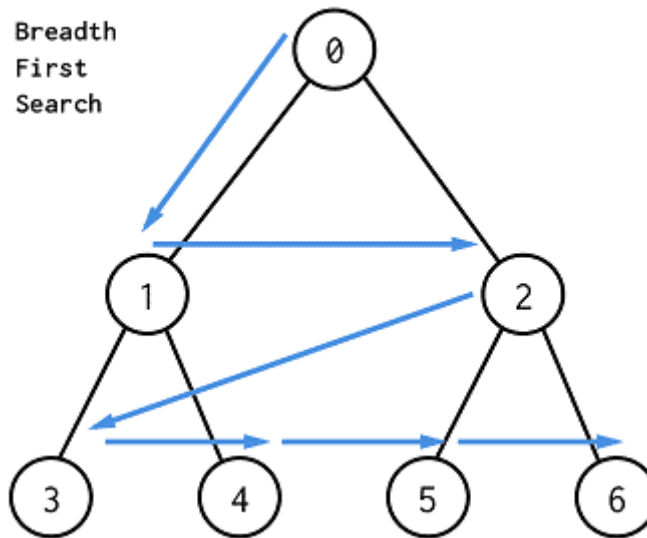
Algoritmos

Um dos objetivos do projeto é conseguir alcançar os pontos objetivos estipulados para cada desafio com o menor número de jogadas possíveis e para isso vamos usar os algoritmos já mencionados. O objetivo destes algoritmos é encontrar o número mínimo de movimentos que permita alcançar o objetivo ou até que não seja mais possível realizar movimentos.

O critério de paragem vai ser o nó que está a ser testado ser um nó objetivo, ou seja, tem ou ultrapassou os pontos desejados, ou o cavalo já não conseguir realizar qualquer movimento no tabuleiro.

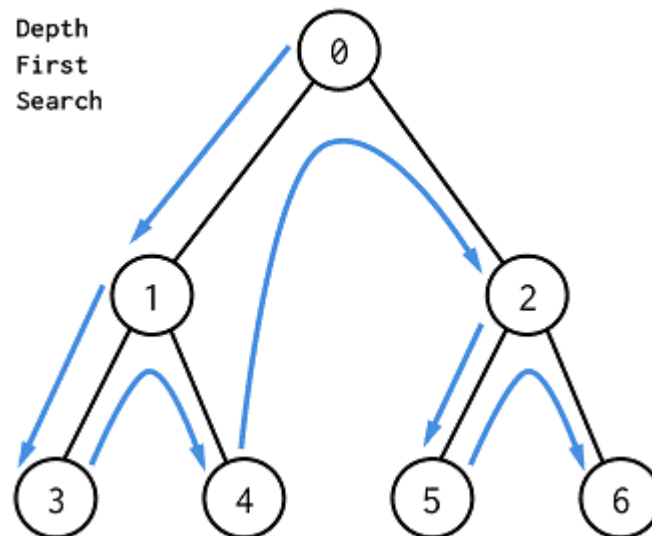
Breadth First Search (BFS)

O algoritmo BFS é um algoritmo de pesquisa em largura. Este começa pelo nó principal e visita todos os nós de cada nível de profundidade passando depois a visitar todos os nós do nível seguinte. Na sua implementação foram usadas duas listas para guardar e organizar estes nós, a lista de abertos onde ficam os nós aos quais ainda não foram expandidos os nós filhos, e a lista de fechados que guarda os nós que já foram expandidos. Para começar o nó principal é expandido e gerado todos os seus nós sucessores que são assim postos no fim da lista de abertos. É depois visitado o próximo nó da lista de abertos, é testado se este é um nó solução, caso não seja expande-se os nós sucessores deste e colocados no fim da lista de abertos. Este último nó que foi expandido é então colocado na lista de fechados. De seguida é visitado o próximo nó da lista de abertos e repetido o mesmo processo.



Depth First Search (DFS)

Este algoritmo é muito parecido com o BFS, no entanto este realiza a procura em profundidade. Aqui são também usadas a lista de abertos e fechados ambos com o mesmo propósito que no BFS. Este começa pelo nó principal, gera todos os seus sucessores e vai percorrer primeiro todos os nós de cada ramo até atingir um limite de profundidade antes de retroceder e fazer o mesmo processo no ramo seguinte. A sua implementação é bastante idêntica à do BFS com a diferença que os nós sucessores serão postos no início da lista de abertos em vez de no fim.



A* (A-Star)

O algoritmo A* é um dos mais utilizados na busca de caminhos. Este otimiza o algoritmo de Dijkstra de duas formas:

- Utiliza uma fila de prioridade para organizar os melhores vértices para explorar
- Utiliza heurísticas que permitem estimar em cada ponto o que ainda falta até chegar ao final de forma a escolher o próximo vértice.

Este algoritmo consegue convergir para o menor caminho mais rápido que os algoritmos BFS e DFS.

Resultados

BFS

FMR → Fator médio de ramificação

| Problema | Profundidade | Nós em abertos | Nós em fechados | Pontos | Tempo de execução (em ms) | Penetrância |
|----------|--------------|----------------|-----------------|--------|---------------------------|-------------|
| A | 2 | 6 | 2 | 72 | 1000 | 0.33 |
| B | 9 | 6 | 9 | 65 | 3000 | 1.50 |
| C | 5 | 15 | 12 | 272 | 3928000 | 0.33 |
| D | 12 | 12 | 35 | 665 | 4198000 | 1 |
| E | - | - | - | - | - | - |
| F | - | - | - | - | - | - |

Os problemas E e F retorna stack overflow e não possível verificar nenhum resultado.

DFS

FMR → Fator médio de ramificação

| Problema | Profundidade | Nós em abertos | Nós em fechados | Pontos | Tempo de execução (em ms) | Penetrância |
|----------|--------------|----------------|-----------------|--------|---------------------------|-------------|
| A | 2 | 12 | 2 | 72 | 1000 | 0.17 |
| B | 9 | 42 | 9 | 65 | 2000 | 0.21 |
| C | 5 | 23 | 7 | 272 | 3000 | 0.22 |
| D | 12 | 38 | 34 | 665 | 16000 | 0.32 |
| E | - | - | - | - | - | - |
| F | - | - | - | - | - | - |

Os problemas E e F retorna stack overflow e não possível verificar nenhum resultado.

Limitações técnicas e Melhorias futuras

No desenvolvimento do projeto foram encontradas várias barreiras e desafios que tive de enfrentar nomeadamente houve uma curva de aprendizagem um pouco acentuada na utilização da linguagem Lisp. Outra dificuldade encontrada foi conter a memória heap visto que a versão do LispWorks usada na implementação e testagem do código tem memória heap limitada pelo que não permite obter resultados em alguns dos desafios.

Estes desafios foram importantes para melhorar o conhecimento da linguagem Lisp.

Foram identificadas as seguintes melhorias:

- Na escolha da coluna ao posicionar o cavalo no tabuleiro, poder inserir a letra da coluna em vez do número

Requisitos não implementados

Não foram implementados os seguintes requisitos:

- Algoritmo A*
- Implementação da heurística dada e nova heurística