Dots & Boxes - Manual Técnico

Projeto 2 - IA 2022/2023 (Época Normal)

Docente

Joaquim Filipe

Alunos

202001310, Leandro Francisco

201801899, Daniel Lachkeev

1. Arquitetura do Sistema

- `procura.lisp` - ficheiro que contém a implementação dos algoritmos de procura utilizados pelo programa (utiliza diversas funções de `puzzle.lisp`);
- `jogo.lisp` - ficheiro que contém os menus de interação com o utilizador e que lhe permite jogar (utiliza diversas funções de `procura.lisp` e `puzzle.lisp`);
- `puzzle.lisp` - ficheiro que contém a definição do estado e as operações (utiliza funções ligeiras de `procura.lisp`).

2. Entidades e sua Implementação

- `Tabuleiro` - composto por duas sublistas (arcos horizontais e arcos verticais).
- `Pontuação` - composto por dois valores (caixas fechadas pelo Jogador 1 e caixas fechadas pelo Jogador 2).
- `Jogador` - utilizador do programa ou o próprio computador (representado por "1" ou "2").
- `Estado` - composto pelo tabuleiro de jogo e pelas pontuações dos jogadores.
- `Nó (Alfabeta)` - composto pelo estado, pelos valores de `alfa` e de `beta`, pela sua profundidade e pelo seu nó-pai.

3. Algoritmos e sua Implementação

Em todos os ficheiros, qualquer função que acaba em "-teste" devolve uma estrutura iniciliazada com valores pre-definidos.

3.1 puzzle.lisp

- `tabuleiro` - função que retorna um lista composta por duas sublistas de arcos.
- `arcos` - função auxiliar de `tabuleiro`, que retorna um conjunto de linhas de arcos.
- `linha-arcos` - função auxiliar de `arcos`, que retorna uma linha de arcos.
- `tabuleiro-30-caixas` - função que retorna um tabuleiro com 30 caixas.
- `get-arcos-horizontais` - função que retorna os arcos horizontais de um tabuleiro.
- `get-arcos-verticais` - função que retorna os arcos verticais de um tabuleiro.
- `get-arco-na-posicao2` - função que retorna um arco numa determinada posição, que normaliza os valores do tabuleiro para um só valor.
- `substituir` - função que substitui um arco.
- `arco-na-posicao` - função para inserção de um arco, numa determinada posição.
- `verificar-arcop` - predicado que verifica se é possível inserir um arco, numa determinada posição.
- `arco-horizontal` - operador - inserção de um arco horizontal.
- `arco-vertical` - operador - inserção de um arco vertical.
- `criar-posicoes` - função auxiliar que retorna todas as posições possíveis, a partir dos arcos horizontais, no formato:

```
((1 1) (1 2) (1 3) (2 1) [...])
```

- `posicoes-caixas` - função auxiliar que retorna todas as posições preparadas para a contagem de caixas.
- `caixa-fechada` - função que determina se uma caixa está fechada, segundo uma determinada posição.
- `caixas-fechadas` - função que determina as caixas fechadas de um tabuleiro, começando numa determinada posição.
- `n-caixas-fechadas` - função que retorna as caixas fechadas de um tabuleiro.
- `contar-caixas` - função auxiliar de `n-contar-caixas`.
- `n-contar-caixas` - função que retorna o número de caixas fechadas de um tabuleiro.
- `n-caixas-quase-fechadas` - função que retorna o número de caixas com um arco em falta.
- `n-caixas-semi-fechadas` - função que retorna o número de caixas com dois arcos em falta.
- `n-caixas-pouco-fechadas` - função que retorna o número de caixas todos ou três dos arcos em falta.
- `estado-solucao` - verifica se o tabuleiro corresponde a uma solução.
- `n-linhas` - conta nº de linhas de um tabuleiro
- `operadores` - lista de operadores ao estado.
- `no-teste` - função que torna um Nó Alfabeta de teste.
- `no-estado` - função que retorna o estado de um Nó.
- `no-tabuleiro` - função que retorna o tabuleiro de um nó.
- `gerar-jogadas` - função que calcula as posições de jogadas possíveis horizontais e verticais.
- `novo-sucessor` - função auxiliar de `sucessores-alphabeta`.

- `sucessores-operador` - função auxiliar intermédia de `sucessores-alphabeta`.
- `sucessores-alphabeta` - função que retorna os sucessores de um nó fornecido.
- `sucessores-jogadas` - função auxilia de `jogada-aleatoria` de `jogo.lisp` (lista de jogadas).
- `criar-estado` - função que retorna um estado vazio, para representação do progresso de um jogo.
- `estado-tabuleiro` - função que retorna o tabuleiro de um estado fornecido.
- `estado-caixas-jogador1` - função que retorna o número de caixas fechadas pelo Jogador 1.
- `estado-caixas-jogador2` - função que retorna o número de caixas fechadas pelo Jogador 2.
- `estado-incrementar-caixas-jogador1` - função que incrementa o número de caixas fechadas pelo Jogador 1.
- `estado-incrementar-caixas-jogador2` - função que incrementa o número de caixas fechadas pelo Jogador 2.
- `estado-incrementar-caixas` - função que incrementa o número de caixas fechadas pelo Jogador 1 e pelo Jogador 2.
- `estado-resultado` - função que indica o vencedor, avaliando as pontuações.
- `limpar-estado` - função que limpa o estado.
- `tabuleiro-preenchidop` - função que verifica se um tabuleiro de jogo encontra totalmente preenchido.
- `estado-solucao` - função que verifica se um jogo terminou e indica o vencedor.
- `linhas-max` - função que retorna o número máximo de linhas que um tabuleiro pode ter.
- `n-linhas` - função que retorna o número de linhas preenchidas, num tabuleiro.
- `avaliacao` - função que avalia o estado de um tabuleiro, dando importância ao número de caixas possíveis de serem fechadas no estado atual.

3.2 procura.lisp

- `cria-no-alphabeta` - função que cria um nó, com uma estrutura adequada, a ser usado pelo algoritmo Alfabeta.
- `no-pai` - função que retorna o nó-pai de um nó.
- `no-alpha` - função que retorna o valor de alfa de um nó.
- `no-beta` - função que retorna o valor de beta de um nó.
- `no-profundidade` - função que retorna a profundidade de um nó.
- `alpha-beta` - função que implementa o algoritmo Alfabeta. Foram definidas 3 funções locais que executam o algoritmo principal iter. Em caso de se alcançar a profundidade máxima ou condição de vitória (preenchimento do tabuleiro) é devolvida uma avalição que representa quão bom é o estado atual comparado com o outro jogador. Para os cortes alfa-beta são chamadas novamente as funções respetivas para o resto dos sucessores ignorando o atual. As avaliações parecem ser iguais independentemente do jogador selecionado.

Em profundidade 3 de um nó inicial usando como base a figura 1 isto é as estatísticas.

```
 CL-USER 2 > (time (alpha-beta (no-teste) 3 1))
Timing the evaluation of (ALPHA-BETA (NO-TESTE) 3 1)

User time    =        0.218
System time  =        0.000
Elapsed time =        0.215
Allocation   = 88625536 bytes
0 Page faults
GC time      =        0.000
70
```

Com profundidade 4 já se começa a sentir o tempo que decorre.

```
 CL-USER 1 > (time (alpha-beta (no-teste) 4 1))
Timing the evaluation of (ALPHA-BETA (NO-TESTE) 4 1)

User time    =        2.828
System time  =        0.031
Elapsed time =        2.822
Allocation   = 1469000120 bytes
0 Page faults
GC time      =        0.078
92
```

Dado que o tempo máximo é de 20ms, não nos enche com confiança que esta implementação do algoritmo seja eficiente.

3.3 jogo.lisp

- `main-menu` - função responsável por gerar o menu inicial da aplicação.
- `player-vs-cpu` - função responsável por gerar o menu para indicação da duração de um turno e de quem joga primeiro.
- `cpu-vs-cpu` - função não implementada responsável por fazer um jogo entre 2 computadores.
- `ler-input` - função para leitura do teclado.
- `trocar-jogador` - função auxiliar de `proximo-jogador`.
- `proximo-jogador` - função que alterna entre o Jogador 1 e o Jogador 2.
- `ler-joagada` - função que pede um arco ao jogador, para o inserir no tabuleiro de jogo.
- `atualizar-estado` - função que a atualiza o estado, ao mesmo tempo que incrementa a pontuação de um jogador, caso o mesmo tenha fechado caixas.
- `escrever-log` - função que escreve no ficheiro log.dat de um caminho pre-definido uma string de texto.
- `jogada-humano` - função que lê e aplica a jogada de um utilizador.
- `aplicar-jogada` - função que retorna um estado, depois de uma jogada.

- `jogada-aleatória` - função que aplica a jogada do computador.
- `jogar` - função que retorna uma lista cujos primeiro e segundo elemento são, respetivamente, uma jogada e um estado resultante da sua aplicação.
- `fim-do-jogo` - função que retorna o resultado de um jogo, quando o mesmo termina.
- `imprimir-estado` - função que retorno o estado de um jogo.
- `format-tabuleiro` - função que retorna um tabuleiro preparado para ser facilmente imprimido (simples de ser lido).
- `imprimir-tabuleiro` - função que mostra o tabuleiro de jogo.
- `imprimir-tabuleiro` - função que mostra o tabuleiro de jogo.
- `imprimir-tabuleiro-caixas` - função que mostra o tabuleiro de jogo e as caixas fechadas que o mesmo contém.
- `imprimir-linha-horizontal` - função que retorna as linhas horizontais de uma lista de arcos.
- `imprimir-linha-vertical` - função que retorna as linhas verticais de uma lista de arcos.
- `convert-to-ones` - função auxiliar para preparar cada linha do tanuleiro para ser imprimida.
- `boxes-to-pattern` - função auxiliar para determinar que caixas devem ser imprimidas.
- `zigzag-pattern` - função auxiliar para combinar duas listas, em padrão zig-zag, para ajudar a imprimir linhas com caixas:

```
"| | | |   |" , "X X X X   " -> "|X|X|X|X|   |"
```

4. Descrição das Opções Tomadas

O foco primário foi garantir que todas as operações e funções auxiliares para o procedimento do jogo estivessem implementadas de forma correta. De seguida foi desenvolvida a codificação das regras e processos do jogo de modo que possamos ter um ciclo de jogadas até a condição de vitória. Por último quando o jogo conseguia minimamente correr até o final, procedemos a implementação do algoritmo alfabeta.

Em muitas funções de alto nível, há a utilização de várias funções auxiliares encadeadas. A invocação de funções recursivas exige uma função de maior nível quando se pretende obter o resultado final. Isto é porque a utilização de scopes de funções não era algo evidente a nos até o final do desenvolvimento.

Para a escrita de ficheiros, optou-se por usar um caminho hardcoded visto que não é possível de uma maneira fácil trabalhar pelo caminho relativo.

5. Limitações

- Sem possibilidade de devolver a próxima jogada ótima do computador apesar de alfabeta estar implementado.
- Sem otimizações de alfabeta como por exemplo pesquisa quiscente, ordenação de nós e memoização.
- Não há limite de tempo implementado visto que não há jogadas de computador que usem alfa-beta.
- Utilização de variáveis globais em detrimento de scopes locais.