Manual Técnico

Unidade Curricular: Inteligência Artificial 2017/2018

Andreia Pereira nº 150221021
Lucas Fischer nº 140221004

Tabela de Conteudos

1- Introdução

Blokus-Duo é uma aplicação desenvolvida em Common Lisp para a unidade curricular de Inteligência Artificial. A aplicação consiste num programa que tem como objetivo simular um jogo de blokus entre um humano e um computador, ou entre dois comptadores.

Para o contexto da aplicação, o problema a solucionar enquadra-se no jogo do Blokus, mais específicamente em dar a possíbilidade ao computador de fazer jogadas mais informadas.

2- Estrutura do Projeto

De modo a facilitar a compreensão e a manutenção do código desenvolvido para esta aplicação, a mesma está dividida em três ficheiros ***.lisp***, estes são:

- ***puzzle.lisp*** - Neste ficheiro estão situadas todas as funções que modelam o dominio problema a solucionar, é neste ficheiro que está implementado o código dependente do dominio do problema. O ficheiro puzzle.lisp consta com funções que permitem: **estruturar um tipo de dados para o problema**, **definir os operadores do problema**, **identificar o nó objetivo**, **calcular o valor heurístico de um nó**, entre outras funções.

- ***algortimo.lisp*** - O ficheiro algortimo.lisp é o ficheiro onde estão situadas as funções que implementam o **algoritmo AlfaBeta**,

os sucessores, e outras funções auxiliares. As funções presentes neste ficheiro são independentes do dominio do problema, podendo ser reutilizadas para outros problemas. O ficheiro algortimo.lisp é o ficheiro que apresenta um maior teor teórico relevante para a unidade curricular de **Inteligência Artificial**

- ***jogo.lisp*** - Este ficheiro é responsável por orquestrar os restantes ficheiros, carregando as funções presentes nos outros ficheiros para memória e compilando-as de modo a que se tornem executáveis. É também neste ficheiro que decorre a implementação do código responsável pela interação com o utilizador e com a leitura e escrita de ficheiros.

- ***log.dat*** - O ficheiro log.dat é onde podem ser consultadas todas as jogadas realizadas durante os jogos. Toda a informação apresentada no ecrã durante a execução do jogo é guardada neste ficheiro para consulta posterior.

- ***problemas.dat*** - Neste ficheiro estão presentes todos os tabuleiros iniciais possíveis para a execução da aplicação. Consiste numa série de listas que representam um tabuleiro separadas por um separador legal.

```
blokus-duo/
        ├── jogo.lisp ; Ficheiro onde é iniciada a
aplicação
        ├── algoritmo.lisp ; Contem a implementação
```

```
do algoritmo alfabeta
        ├── puzzle.lisp ; Contem as funções do
domínio do problema
        ├── log.dat ; Ficheiro que contem logs das
jogadas realizadas
        └── problemas.dat ; Ficheiro com os
tabuleiros iniciais do problema
```

Detalhes da Implementação

Dado que Common Lisp é uma linguagem de programação de natureza funcional o desenvolvimento da aplicação consistiu em desenvolver código num paradigma funcional, deste modo algumas técnicas como sequênciação, ciclos e atribuição não foram utilizadas para poder-se focar na recursividade e desenvolvimento de funções. Embora tenham sido permitidas algumas utilizações especiais de ciclos, estes não foram utilizados, sendo que apenas foram utilizados atribuições a variáveis globais quando necessárias de maneira a facilitar a compreensão e manutenção do código.

Função de Utilidade

```
;; funcao-utilidade

(defun funcao-utilidade (no)
  "Funcao de utilidade que tem em conta o numero de quadrados que um jogador possui, premiand
o as jogadas que levem a uma maior reducao de quadrados"
  (let (
        (pecas (cond
                ((= (get-valor-jogador-no no) 1) (get-pecas-jogador1-no no))
                (T (get-pecas-jogador2-no no))))
        )
    (-
     (+ (* 5 15) (* 4 10) 10) ;Total de quadrados iniciais
     (+ (* 5 (third pecas)) (* 4 (second pecas)) (first pecas)) ;Quadrados que o jogador tem
    )
  )
)
```

Neste capítulo iremos detalhar a implementação e a razão da função de utilidade desenvolvida pelo grupo.
Tendo em conta que o vencedor do jogo é o jogador que possui menor número de quadrados quando já mais nenhum jogador tem jogadas possiveis, então achamos por bem desenvolver uma função de utilidade que tivesse em conta o número de quadrados presentes num determinado tabuleiro em que quanto **mais** peças um jogador tiver jogada **maior** será a **diferença entre o total de quadrados num tabuleiro vazio e o total de quadrados que um determinado jogador ainda possui no estado passado por argumento**.
A implementação desta função de utilidade é relativamente simples, passa simplesmente por descobrir qual é o jogador que estamos a verificar num determinado estado passado por argumento, para que consigamos ter acesso às suas peças, e depois simplesmente **subtrair o número total de quadrados num tabuleiro vazio pelo número de quadrados que esse jogador possui na sua "mão"**.
Ex.: Caso o jogador ainda só tenha jogado uma peça pequena -> ***f(n) = 255 - 254 = 1*** (Baixa utilidade para este nó)
Caso o jogador já só possua uma peça pequena na sua mão -> ***f(n) == 255 - 1 = 254*** (Alta utilidade para este nó)

Estudo de eficiência

Um dos objetivos principais no desenvolvimento deste projeto é

construir uma ferramenta que não só podesse simular o jogo entre um computador e um humano (ou computador vs computador), mas que também nos proporcionasse alguma informação teórica sobre os cálculos e o algoritmo utilizado pelo computador para determinar a melhor jogada possível.
Para este efeito foram implementadas algumas funcionalidades na aplicação que premitem obter informações sobre a sua eficiência nomeadamente: **O número de nós analisádos**; **O número de cortes alfa e beta**; **O tempo que o computador levou para decidir sobre uma jogada**.

- **Estudo Prático**

De modo a termos um caso de exemplo para melhor identificar estas situações, vamos considerar um jogo **Computador vs Computador**, com um **tempo limite** de **5 segundos** e uma profundidade máxima para a árvore de procura de **4 níveis**. Para este exemplo iremos realçar algumas das jogadas mais relevantes.

- **Jogada Inicial**

```
(1 1 0 0 0 0 0 0 0 0 0 0 0 0)
(1 1 0 0 0 0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0 0 0 0 0 0)

-Jogador que fez a jogada: Jogador 1
-Pecas Jogador 1: (10 9 15) | Pecas Jogador 2: (10 10 15)

-Nos analisados: 39
-Cortes Alfa: 2
-Cortes Beta: 6
-Tempo gasto na Jogada: 0 segundos
```

A primeira jogada que o computador faz é bastante rápida sendo não existe muitas possibilidades de jogo para um tabuleiro vazio.
O computador para um tabuleiro vazio analisou **39 nós** onde fez **8 cortes alfa-beta** para determinar que colocar uma **peça média** seria a melhor jogada possível. Esta decisão advém da função de utilidade desenvolvida que, como referido, priveligia tabuleiros que possuam mais quadrados do jogador. Ao jogar a peça média o Computador consegue jogar **4 quadrados** (comparando com apenas **1 quadrado** ao jogar uma peça pequena) e consegue também possibilitar um maior número de jogadas possíveis do que se jogasse uma peça pequena.

- **Jogada Inicial-Intermédia**

```
(1 1 0 0 0 0 0 0 0 0 0 0 0 0)
(1 1 0 1 0 0 0 0 0 0 0 0 0 0)
(0 0 1 1 1 0 0 0 0 0 0 0 0 0)
(0 1 0 1 0 0 0 0 0 0 0 0 0 0)
(1 1 1 0 0 0 0 0 0 0 0 0 0 0)
(0 1 0 1 0 0 0 0 0 0 0 0 0 0)
(0 0 1 1 1 0 0 0 0 0 0 0 0 0)
(0 1 0 1 0 0 0 0 0 0 0 0 0 0)
(1 1 1 0 0 0 0 0 0 0 0 0 2 0)
(0 1 0 0 0 0 0 2 0 0 0 2 2 2)
(0 0 0 0 0 0 2 2 2 0 2 0 2 0)
(0 0 0 0 0 0 0 0 2 0 2 2 2 0 0)
(0 0 0 0 0 0 0 0 0 0 0 2 0 2 2)
(0 0 0 0 0 0 0 0 0 0 0 0 0 2 2)

-Jogador que fez a jogada: Jogador 1
-Pecas Jogador 1: (10 9 11) | Pecas Jogador 2: (10 9 12)

-Nos analisados: 1507
-Cortes Alfa: 34
-Cortes Beta: 126
-Tempo gasto na Jogada: 5 segundos
```

Nesta jogada podemos observar que o estado do jogo já se encontra mais avançado, o tabuleiro está mais preenchido e o número de jogadas possíveis já é bastante elevado. Devido a este facto o algoritmo possui vários nós para analisar (**1507 nós** mais concretamente), o que faz com que o computador necessite de todo o tempo estipulado (**5 segundos**) para determinar qual a melhor

jogada para este estado.
Podemos também observar que o **número de cortes alfa e beta** aumentou em grande nível ajudando a uma melhor exploração da árvore no mesmo espaço de tempo.

- **Jogada Intermédia**

```
(1 1 0 0 0 0 0 0 0 0 1 0 0 0)
(1 1 0 1 0 0 0 1 0 1 1 1 0 0)
(0 0 1 1 1 0 1 1 1 0 1 0 2 0)
(0 1 0 1 0 0 0 1 0 1 0 2 2 2)
(1 1 1 0 0 0 1 0 1 1 1 0 2 0)
(0 1 0 1 0 1 1 1 0 1 0 2 0 0)
(0 0 1 1 1 0 1 0 0 0 2 2 2 0)
(0 1 0 1 0 2 0 0 0 2 0 2 0 0)
(1 1 1 0 2 2 2 0 2 2 2 0 2 0)
(0 1 0 1 0 2 0 2 0 2 0 2 2 2)
(0 0 1 1 1 0 2 2 2 0 2 0 2 0)
(0 1 0 1 0 2 0 2 0 2 2 2 0 0)
(1 1 1 0 2 2 2 0 2 0 2 0 2 2)
(0 1 0 1 0 2 0 0 0 2 0 0 2 2)

-Jogador que fez a jogada: Jogador 1
-Pecas Jogador 1: (9 9 5) | Pecas Jogador 2: (8 9 7)

-Nos analisados: 974
-Cortes Alfa: 9
-Cortes Beta: 433
-Tempo gasto na Jogada: 2 segundos
```

Esta jogada representa em geral uma jogada executada a meio do jogo, i.e., quando os jogadores possuem relativamente metade do número de peças iniciais que tinham. Neste tipo de jogada podemos observar que começam a diminuir drásticamente o número de jogadas possiveis para uma **peça em cruz** e também para uma **peça média** o que explica a diminuição de nós analisádos pelo algoritmo. Podemos também observar um grande número de **cortes beta** que permitem melhorar a eficiência deste algoritmo.

- **Jogada Final**

```
(1 1 0 0 1 0 1 0 1 0 1 0 2 0)
(1 1 0 1 0 1 0 1 0 1 1 1 0 2)
(0 0 1 1 1 0 1 1 1 0 1 0 2 0)
(0 1 0 1 0 1 0 1 0 1 0 2 2 2)
(1 1 1 0 1 0 1 0 1 1 1 0 2 0)
(0 1 0 1 0 1 1 1 0 1 0 2 0 2)
(1 0 1 1 1 0 1 0 1 0 2 2 2 0)
(0 1 0 1 0 2 0 2 0 2 0 2 0 2)
(1 1 1 0 2 2 2 0 2 2 2 0 2 0)
(0 1 0 1 0 2 0 2 0 2 0 2 2 2)
(1 0 1 1 1 0 2 2 2 0 2 0 2 0)
(0 1 0 1 0 2 0 2 0 2 2 2 0 0)
(1 1 1 0 2 2 2 0 2 0 2 0 2 2)
(0 1 0 1 0 2 0 2 0 2 0 0 2 2)

-Jogador que fez a jogada: Jogador 1
-Pecas Jogador 1: (0 9 5) | Pecas Jogador 2: (2 9 7)

-Nos analisados: 3
-Cortes Alfa: 0
-Cortes Beta: 0
-Tempo gasto na Jogada: 0 segundos
```

Esta jogada representa a última jogada realizada pelo **Jogador1**, nela podemos observar que não existem mais jogadas possiveis para nenhum dos jogadores o que leva ao final do jogo. Sendo esta a última jogada realizada o algoritmo analisa apenas **3 nós** sem fazer qualquer corte alfa-beta pois o número de nós que o algoritmo tem para analisar é muito pequeno.

Limitações

Devido à inexistência de um compilador de LISP incorporado na maioria dos Sistemas Operativos, é necessário recorrer a software de terceiros para conseguir complar código LISP. Neste projeto foi utilizado o software LispWorks para este objétivo, e dado ser a versão gratuita deste software cadece de limitações de memória Stack e Heap o que compromete a execução de alguns algoritmos para alguns problemas.

A componente de memoização do projeto, embora implementada na função **alfabeta-memo** do ficheiro **algoritmo.lisp**, não foi incorporada no produto final por possuir algumas falhas que levassem à imperfeição do jogo. As falhas consistiam em algumas jogadas não serem consideradas o que levaria a resultados diferentes no jogo.