# Rastros

## Trabalho realizado por:

João Figueiredo Martins Peixe dos Santos

Luís Filipe Cruz Sobral

Paulo Silva Sousa

A89520
João Santos

A89474
Luís Sobral

A89465
Paulo Sousa

Projeto LI2
2019/2020
Universidade do Minho

# Índice

# 1 Introdução

Nesta unidade curricular foi-nos proposta a implementação de um jogo de tabuleiro chamado **RASTROS**

Inicialmente, o projeto consistiu no desafio de implementar este JOGO na linguagem C. Apesar da intenção de tornar rápida a execução das funcionalidades do programa a ser desenvolvido, alguns dos focos prioritários deste projeto foram a modularidade e o encapsulamento das estruturas de dados por nós utilizadas.

Ao longo do desenvolvimento deste projeto, consideramos que o maior desafio foi a implementação do módulo genérico listas ligadas.

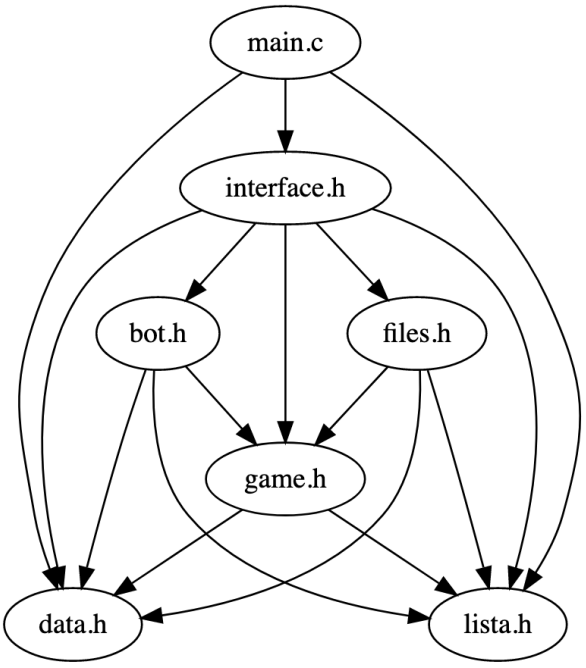
# 2 Estrutura do Projeto e Grafo de Dependências

A divisão do projeto foi feita em três camadas: a camada lógica, a camada dos dados e a camada da interface.

A camada da interface inclui os módulos interface e files. No primeiro, incluímos o interpretador de comandos, bem como todas as funções que devolvessem resultados visuais. Já no segundo, integrámos as funções que permitem gravar e ler a partir de um ficheiro.

A camada lógica inclui os módulos game e bot. No primeiro, integrámos as funções responsáveis pela execução dos comandos coordenada e pos e a função que verifica se o jogo já terminou. No outro, incorporámos as funções relativas ao algoritmo minmax, utilizado pelo comando jog e pelo bot.

A camada de dados inclui os módulos data e lista. No primeiro, incluímos as estruturas necessárias para o estado e todas as funções que permitem o acesso indireto a este a partir dos outros módulos. No segundo, integrámos a estrutura genérica da lista ligada e funções que, tal como as funções do módulo data, permitem o acesso indireto ao módulo lista a partir de outros módulos.



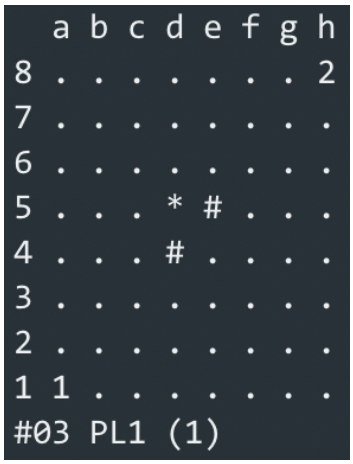

Grafo de dependências

# 3 Interação com o Utilizador


Quando o utilizador corre o programa, é-lhe apresentado o menu com os comandos possíveis de utilizar.

Durante a execução do programa, o menu poderá voltar a ser apresentado utilizando o comando menu.

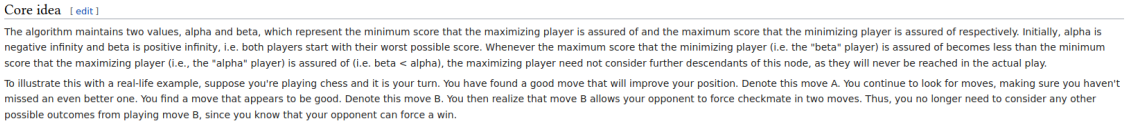
Os comandos que necessitem de argumentos, como está na figura, têm de ser introduzidos, para que o comando em questão seja executado.




A figura representa o **estado** do jogo. O número de comandos introduzidos até ao momento está presente no canto inferior esquerdo(neste caso seriam três comandos introduzidos), seguido do jogador atual(PL1) e, por último, o número de jogadas efetuadas((1)).

# 4 Bot

Como estratégia para a construção do bot, o grupo utilizou o algoritmo Alpha-Beta Pruning.



Core idea [ edit ]

The algorithm maintains two values, alpha and beta, which represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. Initially, alpha is negative infinity and beta is positive infinity, i.e. both players start with their worst possible score. Whenever the maximum score that the minimizing player (i.e. the "beta" player) is assured of becomes less than the minimum score that the maximizing player (i.e., the "alpha" player) is assured of (i.e. beta < alpha), the maximizing player need not consider further descendants of this node, as they will never be reached in the actual play.

To illustrate this with a real-life example, suppose you're playing chess and it is your turn. You have found a good move that will improve your position. Denote this move A. You continue to look for moves, making sure you haven't missed an even better one. You find a move that appears to be good. Denote this move B. You then realize that move B allows your opponent to force checkmate in two moves. Thus, you no longer need to consider any other possible outcomes from playing move B, since you know that your opponent can force a win.

Alpha-Beta pruning e Minmax: Ao utilizarmos o primeiro algoritmo reduzimos o número de tabuleiros avaliadas pelo algoritmo minmax, tornando o segundo algoritmo mais eficiente. Promovemos assim também a eficiência do programa. Para isso foi necessário estabelecer funções que avaliassem a jogada, escolhessem o mínimo, o máximo e ainda definir a estratégia do bot.

## 4.1 Estratégia

Enquanto as jogadas forem menores do que cinco o bot tenta aproximar-se da casa adversária. Quando o número de jogadas for superior a cinco o bot tenta aproximar-se da sua casa.

A profundidade é definida de acordo com o número de jogadas e vai aumentando com o número de jogadas.Para um número de jogadas menor do que 3 a profundidade do algoritmo minmax é zero. Entre três e cinco a profundidade é nove. A profundidade é onze entre seis e onze. Entre doze e dezoito é treze. Para maior que dezoito é quinze.

## 4.2 Avaliar Jogada

Na função avaliarJogada atribuímos uma pontuação a uma jogada válida. Esta pontuação era de 1 ou 8 caso o jogo acabasse (1 se perdeu o jogo, 8 se ganhou) e atribuímos uma pontuação de 2 a 7 para as restantes jogadas de acordo com a estratégia do bot.

## 4.3 Min e Max

O próximo passo foi definir funções que escolhessem sucessivamente o max e depois o min. Para isso criamos duas funções: “min” e “max”. Estas funções simulam todas as jogadas, escolhendo a que tem maior pontuação para o bot e menor para o jogador.

# 5 Conclusão e Reflexão Crítica

Desde logo, os objetivos deste projeto foram atingidos e durante a sua realização foram de reter determinados pontos.

Tais como, a necessidade de encapsular os dados para ter um código seguro, protegido do utilizador, como também a relação entre o número de jogadas analisadas pelo algoritmo minmax e o tempo de execução deste, de forma a não exceder um segundo.

Para concluir, estamos satisfeitos com a solução do nosso trabalho, pois conseguimos implementar um programa seguro e de fácil utilização. Ainda, conseguimos a implementação de um bot competente.