**Distrify**

**Relatório Final**



Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

**Grupo 02: Distrify**

André Rodrigues Barros – up201303567  
Edgar Duarte Ramos – up201305973

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

8 De Novembro de 2015

# Resumo

Foi proposto ao grupo desenvolver um jogo em PROLOG, denominado de Distrify, um jogo de tabuleiro cujo objetivo é, como na maioria dos jogos de tabuleiro, tentar vencer o nosso oponente.

O projeto teve como objetivo aplicarmos os conhecimentos transmitidos na unidade curricular, sendo de salientar que PROLOG é uma linguagem de programação diferente do que estamos habituados e por isso foi algo complicado realizar de forma eficiente o jogo, sendo que envolveu muita pesquisa para que conseguíssemos encontrar solução para os problemas que foram aparecendo, aquando do desenvolvimento do projeto. Recorrendo a diversos predicados já existentes, e também criando alguns novos, a solução aos problemas foi encontrada.

Como resultado final do projeto, temos um jogo simples mas apelativo e muito divertido de se jogar, sendo um bom jogo para amantes de jogos de tabuleiro. É, também, um jogo eficiente visto que foi desenvolvido para execução na linha de comandos.

Em suma, é de fácil perceção que o nosso conhecimento da linguagem PROLOG aumentou, sendo possível a consolidação dos conceitos aprendidos nas aulas e aplicados nas aulas práticas, sendo que, a nosso ver, o jogo apresentado é um motivo para um sentimento de realização para o grupo.

# **Índice**

[Resumo 2](#_Toc434766107)

[Índice 3](#_Toc434766108)

[1 Introdução 4](#_Toc434766109)

[2 O Jogo Distrify 5](#_Toc434766110)

[2.2 Detalhes do Jogo 5](#_Toc434766111)

[2.3 Objetivo 5](#_Toc434766112)

[2.4 Jogada 5](#_Toc434766113)

[2.5 Movimento 5](#_Toc434766114)

[2.6 Captura 7](#_Toc434766115)

[2.7 Última Linha 7](#_Toc434766116)

[3 Lógica do Jogo 8](#_Toc434766117)

[3.1 Representação do Estado do Jogo 8](#_Toc434766118)

[3.2 Visualização do Tabuleiro 8](#_Toc434766119)

[3.3 Execução de Jogadas 9](#_Toc434766120)

[3.4 Avaliação do Tabuleiro 10](#_Toc434766121)

[3.5 Final do Jogo 10](#_Toc434766122)

[3.6 Jogada do Computador 11](#_Toc434766123)

[4 Interface com o Utilizador 13](#_Toc434766124)

[5 Conclusões 18](#_Toc434766125)

[6 Bibliografia 19](#_Toc434766126)

[6.1 Livros 19](#_Toc434766127)

[6.2 Páginas Web 19](#_Toc434766128)

[6.3 Documentação 19](#_Toc434766129)

[7 Anexos 20](#_Toc434766130)

# Introdução

No âmbito da unidade curricular de Programação em Lógica, do curso Mestrado Integrado em Engenharia Informática e Computação, foi-nos sugerido o desenvolvimento de um jogo na linguagem PROLOG. Dentro de um leque de várias opções que nos foram disponibilizados pelos docentes da unidade curricular, achamos que o Distrify era o jogo com o qual nos identificamos mais, e daí ser essa a nossa escolha.

A escolha em torno de Distrify baseou-se no facto de ser um jogo simples mas divertido de se jogar, porém embora seja simples envolve toda uma estratégia e inteligência que nos apelou e nos levou a escolher o mesmo.

O jogo permite também uma boa prática mental e estratégica com o desenrolar de uma partida entre dois elementos, sendo que o nosso estilo de jogo está dependente das ações do nosso adversário.

O objetivo deste trabalho foi a aplicação dos primeiros conceitos interiorizados na unidade curricular. Este método de avaliação torna-se importante pois permite-nos avaliar os conhecimentos que adquirimos até então. Por outro lado, também permite enriquecer ainda mais o nosso conhecimento, e favorece o trabalho autónomo por parte dos alunos, estudando em simultâneo se os alunos são capazes ou não de produzir algo que é funcional e eficiente a partir destes conhecimentos.

Assim, este relatório encontra-se dividido em várias secções:

* as regras e as jogadas possíveis do jogo
* a lógica implementada no jogo, descrição do projeto e da sua implementação em PROLOG
* o modo de interação entre o utilizador e o programa, e entre a jogabilidade e a visualização
* uma conclusão final do grupo em relação ao jogo e a tudo o que o envolveu
* uma série de elementos anexados, como por exemplo secções do código do projeto

# O Jogo Distrify

## Detalhes do Jogo

Distrify é um jogo de conexão jogado por dois jogadores, constituído por peças pretas e peças brancas, num tabuleiro cujas dimensões vão desde 9x9 até 19x19.

O jogador que controla as peças pretas joga primeiro, colocando uma peça preta em qualquer local do tabuleiro. As jogadas seguintes são alternadas entre os jogadores e cada jogada consiste na colocação de uma ou duas peças em qualquer espaço livre.

## Objetivo

O objetivo do jogo é realizar uma linha de peças que una os limites superior e inferior do tabuleiro, se for o jogador que controla as peças pretas. Por outro lado, se for o jogador que controla as peças brancas, o objetivo é construir um caminho que una os limites laterais do tabuleiro.

## Jogada

Em cada jogada, um jogador pode introduzir uma ou duas peças (sendo que pode ter como objetivo uma de duas opções: bloquear o caminho do adversário ou criar o seu caminho).

## Regras de introdução de peças

No caso em que sejam jogadas duas peças:

a) só podem ser jogadas na vertical ou horizontal

b) não podem resultar na formação de um triplet ou crosscut.

Um triplet refere-se a três peças da mesma cor seguidas em qualquer direção.

Um crosscut refere-se a um bloco de 2x2 em que cantos opostos são controlados por jogadores diferentes

# Lógica do Jogo

## Representação do Estado do Jogo

Todo o código relativo à representação do estado do jogo encontra-se no ficheiro *distrify.pl*.

O nosso objetivo ao armazenar toda a informação do estado do jogo numa lista era aproximar o desenvolvimento do projeto a um paradigma semelhante ao de programação orientada a objetos. No ficheiro *gameClass.pl* podem ser observados vários predicados que realçam este paradigma: os predicados *getters* e *setters*, que obtêm e modificam um elemento do estado do jogo, por exemplo.

O primeiro elemento da lista que compõe a *classe* do estado do jogo é uma lista de listas que representam o estado atual do tabuleiro de jogo, ou seja, o conteúdo de cada posição no tabuleiro. A disposição das peças no tabuleiro é, portanto, armazenada nesta *matriz*.

O segundo elemento da lista é outra lista de dois elementos - um par - cujo conteúdo é, respetivamente, o número de peças que o jogador branco e preto têm sobre o tabuleiro.

O terceiro elemento determina qual dos jogadores deve efetuar a jogada naquele estado. Para um estado de jogo em que seja a vez do jogador branco efetuar uma jogada, o terceiro elemento será portanto: ***whitePlayer***; por sua vez, se fosse o jogador preto o próximo a jogar, o terceiro elemento seria ***blackPlayer***.

Finalmente, o quarto elemento contém o modo de jogo. Existem três modos de jogo: humano contra humano, humano contra computador e computador contra computador. Os átomos que representam estes três modos de jogo diferentes são, respetivamente: ***pvp***, ***pvb*** e ***bvb***.

## Visualização do Tabuleiro

Os predicados responsáveis pela visualização do tabuleiro na linha de comandos encontram-se no final do ficheiro *eximo.pl*. Os predicados são quase todos recursivos e foram desenvolvidos por camadas.

Para imprimir o tabuleiro basta chamar o predicado ***printBoard(+Board)***, onde ***Board*** é um tabuleiro de um estado de jogo. Por sua vez, este predicado faz uso de outros predicados com funções cada vez mais específicas para imprimir o tabuleiro de uma forma simples, formatada e concisa sempre que necessário.

## Execução de Jogadas

Em cada jogada, é pedido ao jogador que tem a vez de jogar as coordenadas da peça no tabuleiro que deseja movimentar. Logo depois de as coordenadas serem validadas, o programa verifica se as coordenadas correspondem a uma peça do jogador que tem a vez de jogar e não a uma das peças do oponente. Esta verificação é feita com recurso ao predicado ***validateChosenPieceOwnership(+SrcRow, +SrcCol, +Board, +Player)*** e caso falhe, o programa retrocede e pede ao jogador para inserir outras coordenadas.

Uma vez validadas as coordenadas da peça a mover, é pedido ao jogador que insira as coordenadas do destino da peça que selecionou. Após essas coordenadas serem validadas, verifica-se se as coordenadas de origem e destino do movimento não são as mesmas com o predicado ***validateDifferentCoordinates(+SrcRow, +SrcCol, +DestRow, +DestCol)***. Caso as coordenadas de origem e destino sejam as mesmas, o programa retrocede até ao ponto inicial.

O predicado ***validateMove(+SrcRow, +SrcCol, +DestRow, +DestCol, +Game, -TempGame)*** é chamado para tentar unificar o movimento descrito pelo jogador com um dos movimentos possíveis segundo as regras do jogo:

* ***validateOrdinaryMove(+SrcRow, +SrcCol, +DestRow, +DestCol, +Game, -ResultantGame)***;
* ***validateJumpMove(+SrcRow, +SrcCol, +DestRow, +DestCol, +Game, -ResultantGame)***;
* ***validateCaptureMove(+SrcRow, +SrcCol, +DestRow, +DestCol, +Game, -ResultantGame)***.

Cada um destes predicados valida os possíveis deslocamentos da peça consoante o tipo de movimentação e retorna o estado de jogo resultante. Se a jogada unificar com uma das regras de movimentação possíveis, a peça é efetivamente deslocada e o estado de jogo é atualizado com os respetivos efeitos secundários que essa movimentação possa ter causado.

Contudo, se a movimentação da peça acarretar movimentações extras obrigatórias, – como é o caso do movimento em salto e de captura: movimentos onde, se for possível continuar a saltar/capturar, é obrigatório fazê-lo – então o programa encarrega-se de continuar a solicitar ao jogador as coordenadas do destino da peça que está a executar o movimento complexo e de atualizar o estado de jogo devidamente e só depois devolver o estado de jogo final resultante da combinação de movimentações.

Finalmente, depois da jogada completa ter sido efetuada, o predicado ***changePlayer(+TempGame, -ResultantGame)*** responsabiliza-se por alternar os jogadores entre jogadas e obter o resultado final do estado do jogo.

## Avaliação do Tabuleiro

Apesar de esta funcionalidade não estar implementada, como ponderámos utilizar o algoritmo Minimax no modo humano contra computador, e este algoritmo precisar de uma função de avaliação de um estado de jogo, decidimos que uma boa função de avaliação para começar a testar o algoritmo seria, por exemplo:

***4 \* (16 - numPeçasHumano) + numPeçasBot***

Esta função avalia a favorabilidade de um estado de jogo para o bot de acordo com o número de peças existentes no tabuleiro, sendo que a valorização do número de peças que o humano não possui vale quatro vezes mais que o número de peças em posse pelo bot. A razão para a escolha desta função pode ser exemplificada pelo seguinte exemplo:

Um tabuleiro em que o humano tenha zero peças, e o bot tenha uma peça – 4 \* (16 – 0) + 1 = **65** –, é mais favorável do que um tabuleiro em que o humano tenha uma peça e o bot tenha quatro peças – 4 \* (16 – 1) + 4 = **64**.

## Final do Jogo

Quando o primeiro predicado ***playGame(Game)*** é chamado e falha por um dos jogadores não conseguir efetuar qualquer movimento (*stalemate*), ou por não ter mais peças no tabuleiro, os restantes predicados são chamados.

O segundo predicado ***playGame*** é bem-sucedido se o número de peças de cada jogador for superior a zero (o que significa que o jogador que tem a vez de jogar se encontra bloqueado), terminando o jogo.

O terceiro predicado sucede se um dos jogadores não tiver peças em tabuleiro e o jogo termina.

## Jogada do Computador

Tal como referido no ponto ***3.4 Avaliação do Tabuleiro***, apesar de termos pensado na possibilidade de implementar o algoritmo *Minimax*, não o fizemos por falta de tempo e limitámo-nos a implementar dois tipos de *bots*: um ***random bot***, que executa jogadas aleatórias; e um ***greedy bot***, que executa a jogada que possibilita capturar o maior número de peças do adversário.

***letRandomBotPlay(+Game, -ResultantGame)***

O ***random bot*** escolhe aleatoriamente uma das suas peças e tenta movimentá-la segundo um dos movimentos possíveis. Caso essa peça não possa ser movida, outra peça é escolhida aleatoriamente até que seja encontrada uma peça que possa ser movida.

Quando o *bot* é obrigado a continuar um movimento de salto/captura e possui mais do que uma escolha, não é feita nenhuma avaliação dos caminhos possíveis; novamente, é feita uma decisão aleatória sobre qual dos caminhos a peça deve seguir.

***letGreedyBotPlay(+Game, -ResultantGame)***

O ***greedy bot***, em cada jogada, percorre a totalidade das suas peças e para cada peça, calcula uma lista de sequência de capturas que essa peça consegue executar. No final, o *bot* opta por movimentar a peça que tiver associada uma maior lista de sequência de capturas. Se nenhuma peça tiver associada uma lista de sequência de capturas, ou seja, nenhuma peça consegue fazer uma única captura, o *bot* escolhe aleatoriamente entre fazer um movimento ordinário ou em salto.

Quando o *bot* é obrigado a continuar um movimento de captura e possui mais do que uma escolha, não é feito nenhum cálculo porque esse cálculo já foi feito quando se percorreu a totalidade das peças para calcular qual delas possuía a maior lista de sequência de capturas, pelo que o bot se limita a movimentar a peça segundo as “instruções” da lista associadas à peça.

Contudo, quando o *bot* é obrigado a continuar um movimento de salto e possui mais do que uma escolha possível, essa escolha é feita aleatoriamente.

# Interface com o Utilizador

A interface da linha de comandos foi feita de forma a proporcionar uma experiência agradável e simples ao utilizador. Os menus estão devidamente identificados e para navegar entre estes basta escolher o seu identificador e pressionar a tecla *Enter*.

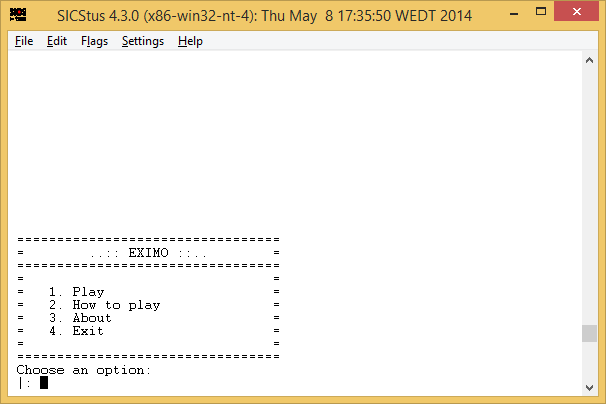


Figura 9 - Menu principal.

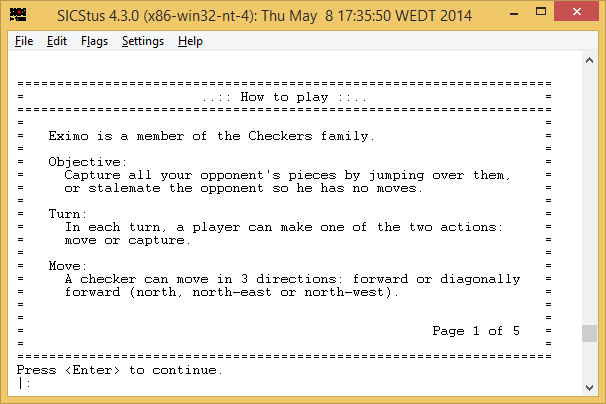


Figura 10 - Instruções do jogo.

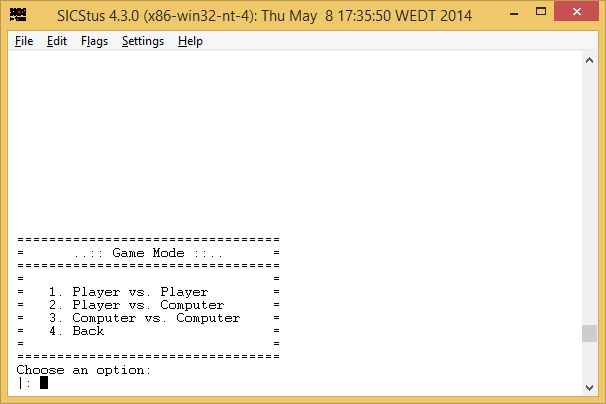


Figura 11 - Menu de escolha do modo de jogo.

Durante o jogo em si, a linha de comandos é limpa (com recurso ao predicado ***cleanConsole*** definido em *utilities.pl*), o estado atual do tabuleiro é exibido, bem como uma mensagem do jogador que tem a vez de jogar. De seguida, é solicitado ao jogador que insira as coordenadas da peça a movimentar, e as coordenadas do destino dessa peça.

Para inserir coordenadas, basta inserir a linha e a coluna da peça escolhida e confirmar com *Enter*. Por exemplo, para selecionar a peça na linha 3, coluna f, inserir-se-ia: ***3f<Enter>***. Quando um jogador tenta fazer um movimento que não é permitido, o motor do jogo encarrega-se de retroceder até um ponto conveniente para o utilizador introduzir novamente outro input para ser validado.

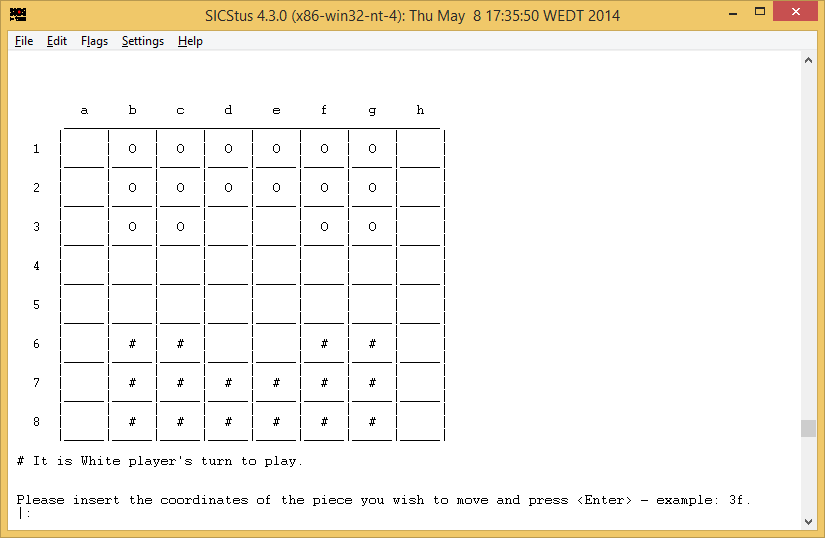


Figura 12 - Estado inicial de um jogo.

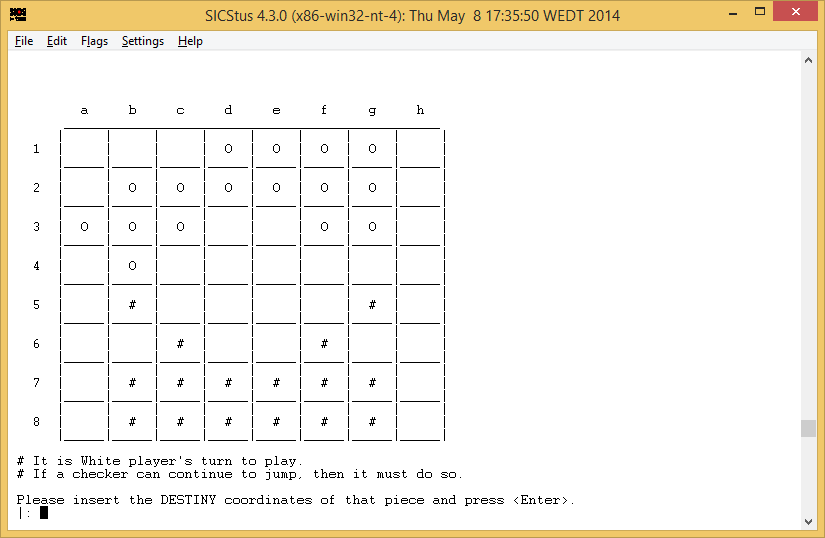


Figura 13 - Exemplo de um estado de jogo após duas jogadas.

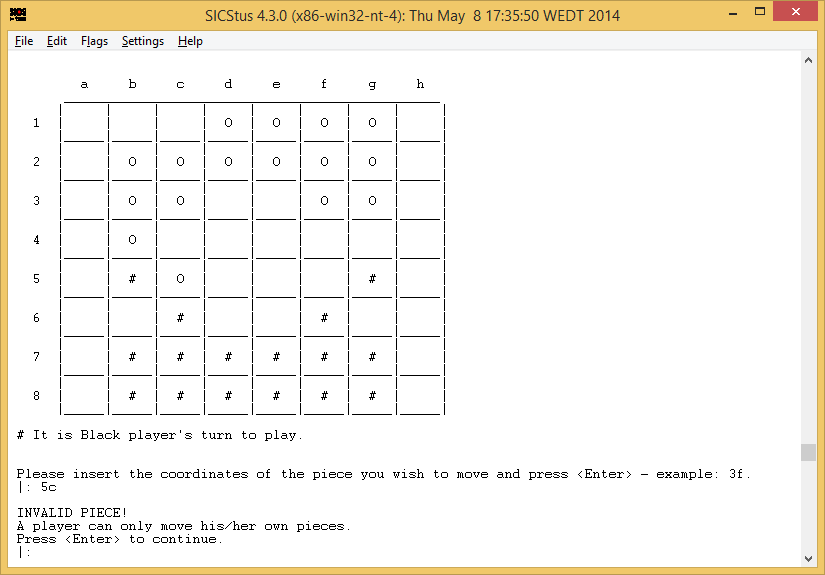


Figura 14 - Exemplo de seleção de uma coordenada cuja peça não pertence ao jogador que tem a vez de jogar.

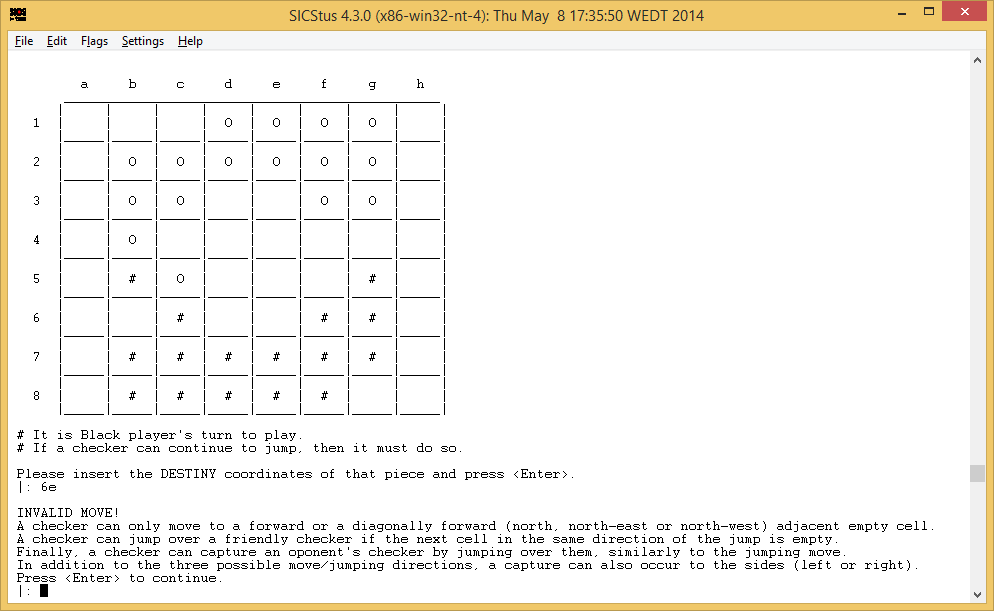


Figura 15 - Exemplo de seleção de uma coordenada de destino inválida.

# Conclusões

O jogo Eximo exigiu imenso tempo ao grupo para a sua implementação. Por fim apresentamos as nossas conclusões finais:

O grupo vê positivamente o resultado final que obteve e os conhecimentos adquiridos durante o desenvolvimento do projeto. Apesar do escasso tempo para a entrega final do projeto e as consecutivas sobreposições de trabalhos de outras unidades curriculares, conseguimos concluir o que havíamos planeado.

Eximo mostrou-se um desafio que, com esforço, dedicação e empenho se tornou um jogo muito apelativo e simples que proporciona ao jogador uma boa prática mental e é, como se esperava, um bom passatempo.

As dificuldades encontradas foram superadas, porém poderiam haver melhorias, nomeadamente na forma como o algoritmo *greedy* foi implementado.

Em suma o grupo gostou da experiência de desenvolvimento de um jogo na linguagem PROLOG. Ao contrário do que estamos habituados, este tipo de linguagem requer um pensamento lógico em cada predicado desenvolvido.

O grupo despede-se, orgulhosamente, desejando aos utilizadores um bom jogo.

# Bibliografia

## Livros

- Sterling, Leon S.; Shapiro, Ehud Y. - The Art of Prolog : Advanced Programming Techniques

## Páginas Web

- <http://cs.union.edu/~striegnk/learn-prolog-now/html/index.html>

## Documentação

- Documentos fornecidos na página da unidade curricular presentes no Moodle

# Anexos

O código fonte do projeto encontra-se na pasta *src* anexada junto deste relatório.