Eximo

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo 04: Eximo
Henrique Manuel Martins Ferrolho - 201202772
João Filipe Figueiredo Pereira - 201104203

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

11 De Novembro de 2014

Resumo

Ao longo das últimas semanas de aulas, o grupo encontrou-se a desenvolver um jogo em PROLOG, denominado de Eximo, da família das Damas.

O trabalho teve como objetivo aplicarmos os conhecimentos interiorizados na implementação do jogo, sendo este por ventura o maior problema que encontramos, pois PROLOG é uma linguagem de programação com um paradigma diferente do que estamos habituados. Com alguma pesquisa e consulta aos materiais fornecidos pelos docentes foi possível uma melhor compreensão e aplicação das novas abordagens aqui requeridas. Com recorrência a diversos predicados já existentes e novos criados pelo grupo a solução aos problemas foi encontrada.

Houve alguns fatores que são necessários de frisar: a paciência, lucidez e cooperação foram muito importantes para conseguirmos atingir os objetivos propostos.

Como resultado final do projeto temos um jogo não só simples e apelativo como também robusto e eficiente visto que foi desenvolvido para execução na linha de comandos.

Como conclusões finais a este trabalho podemos dizer que o nosso conhecimento acerca da linguagem PROLOG aumentou consideravelmente, sendo possível a consolidação dos conceitos aprendidos nas aulas, e temos orgulho no trabalho até aqui desenvolvido e apresentado.

Índice

1 Introdução

No âmbito da unidade curricular de Programação em Lógica, do curso Mestrado Integrado em Engenharia Informática e Computação, foi-nos sugerido a elaboração de um jogo em PROLOG. Esse jogo foi selecionado pelo grupo dentro de um leque de várias opções que nos foram disponibilizados pelos docentes.

A escolha em torno de Eximo baseou-se no estilo que o prescrevia. A jogabilidade simples e a combinação de jogadas possíveis tornaram-no num bom partido e a motivação por parte do grupo para a sua implementação foi forte.

Assim como nas Damas, é possível haver uma boa prática mental e estratégica com o desenrolar de uma partida entre dois elementos. O grupo tomou em consideração todos os pontos descritos em cima para a escolha do jogo final, que se viria a tornar o primeiro trabalho realizado em Programação em Lógica.

O objetivo deste trabalho foi a aplicação dos primeiros conceitos interiorizados nas aulas teóricas e desenvolvidos nas aulas práticas da cadeira. Este método de avaliação torna-se importante pois permite-nos avaliar os conhecimentos que adquirimos até então e saber se somos ou não capazes de, com uma linguagem de programação nova e um paradigma completamente diferente do que estamos habituados, produzir algo de útil para o quotidiano e futuro.

Este relatório encontra-se dividido em várias secções tais como:

- a história, regras e jogadas possíveis do jogo Eximo;
- a lógica implementada no jogo, descrição do projeto e da sua implementação em PROLOG;
- o modo de interação entre o utilizador e o programa, entre jogabilidade e visualização;
- uma conclusão final do grupo a este primeiro desafio com o ambiente PROLOG;
- uma lista de referências a livros e páginas web;
- uma série de elementos anexados, como o código do projeto e outros.

2 O Jogo Eximo

2.1 História

Eximo é um jogo de tabuleiro da família das Damas, concebido em 1 de Fevereiro de 2013.

2.2 Detalhes do Jogo

O jogo realiza-se num tabuleiro de dimensões 8x8, em que as casas têm todas cores semelhantes. Cada jogador começa com 16 peças colocadas em locais predefinidos no respetivo lado do tabuleiro, como mostra a imagem abaixo.

Figura 2 - Peça Branca

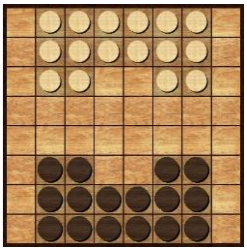

Figura 3 - Tabuleiro

Figura 1 - Peça Preta

No jogo, as movimentações e as capturas podem ser ortogonais ou diagonais. Há apenas um tipo de peça: os homens. Os homens podem saltar sem efetuar captura. Quando um homem atinge a última linha, ocorre a libertação de outro homem.

2.3 Objetivo

O objetivo do jogo é, tal como nas Damas, **capturar todas as peças** do oponente, saltando sobre elas, ou **incapacitar o adversário** de realizar qualquer movimento.

2.4 Jogada

Em cada jogada, um jogador pode fazer uma de duas ações: **mover ou capturar**.

2.5 Movimento

Uma peça pode mover-se em três direções: para a frente ou na diagonal (**norte, nordeste ou noroeste**). Numa jogada, o movimento nunca pode ser efetuado para a retaguarda.
Existem dois tipos de movimentos: **Normal e Salto**.
Movimento Normal: uma peça move-se para uma **casa adjacente e vazia**.
Movimento em Salto: **uma peça salta sobre uma peça aliada adjacente, se e só se a casa correspondente** (ao lado da peça aliada) **estiver vazia**, colocando assim a peça nessa casa. Se a mesma peça do jogador puder continuar a realizar o mesmo movimento de salto sobre outra peça amigável então terá de o fazer. **Durante um movimento de salto a peça não pode capturar peças inimigas**.
Quando existe **mais do que uma forma de saltar**, o jogador **pode escolher a peça que irá usar para executar o salto**, bem como o tipo de salto ou sequência de saltos a fazer. Não é obrigatório que a sequência de saltos escolhida pelo jogador seja aquela que possui o maior número de saltos; porém, **após escolher uma sequência, o jogador deve executar todos os saltos possíveis**.

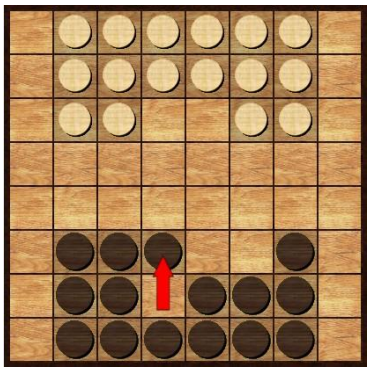

Figura 5 – Movimento Normal

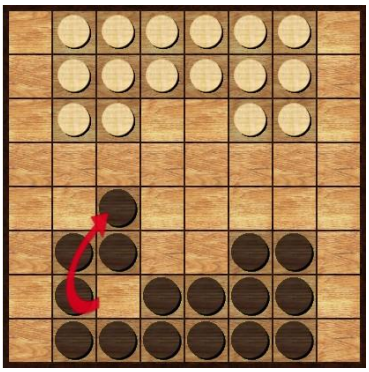

Figura 4 – Movimento em Salto

Figura 6 - Movimentos

2.6 Captura

Um **jogador pode capturar** em cinco direções: **frente**, **diagonal para a frente**, **direita** ou **esquerda** (norte, nordeste, noroeste, este ou oeste). **Captura**: um jogador **salta sobre uma peça adjacente do adversário**, se a **próxima casa**, na mesma direção, **estiver vazia**, colocando, assim, a peça sobre essa casa. A **peça do oponente é removida do tabuleiro**. Se a **peça** do mesmo jogador **puder continuar a capturar outras peças do adversário**, então **deve fazê-lo**. A **captura é obrigatória** e deve continuar enquanto for possível.

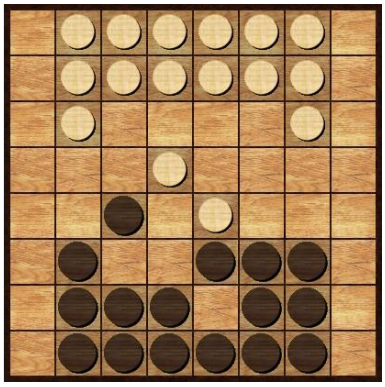

Figura 8 - Estado anterior à captura

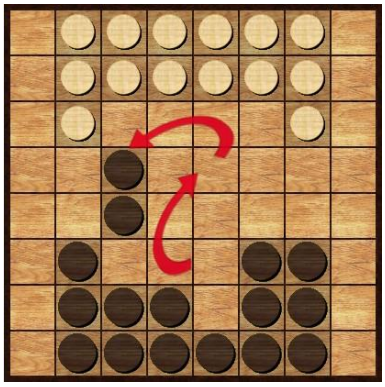

Figura 7 - Estado posterior à captura

Tal como no **Movimento de Salto**, **o jogador escolhe** livremente **qual a sequência de saltos a efetuar**.

2.7 Última Linha

Quando uma peça atinge a extremidade do tabuleiro, essa peça é removida de imediato e o jogador recebe dois movimentos-extra para efetuar nesse mesmo momento: colocar duas peças novas numa casa vazia localizada nas duas primeiras linhas, à exceção das quatro casas laterais (duas do lado esquerdo, e duas do lado direito).

3 Lógica do Jogo

3.1 Representação do Estado do Jogo

O estado do jogo é representado por uma lista de quatro elementos. Todo o código relativo à representação do estado do jogo encontra-se no ficheiro *gameClass.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***.

3.2 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.

3.3 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.

3.4 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**.

3.5 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.

3.6 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.

4 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*.

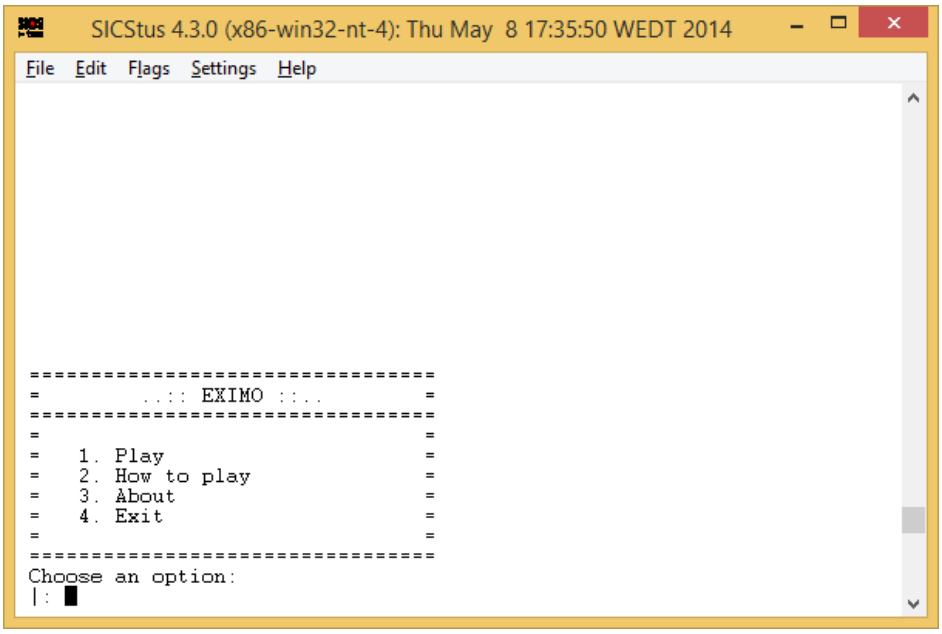

Figura 9 - Menu principal.

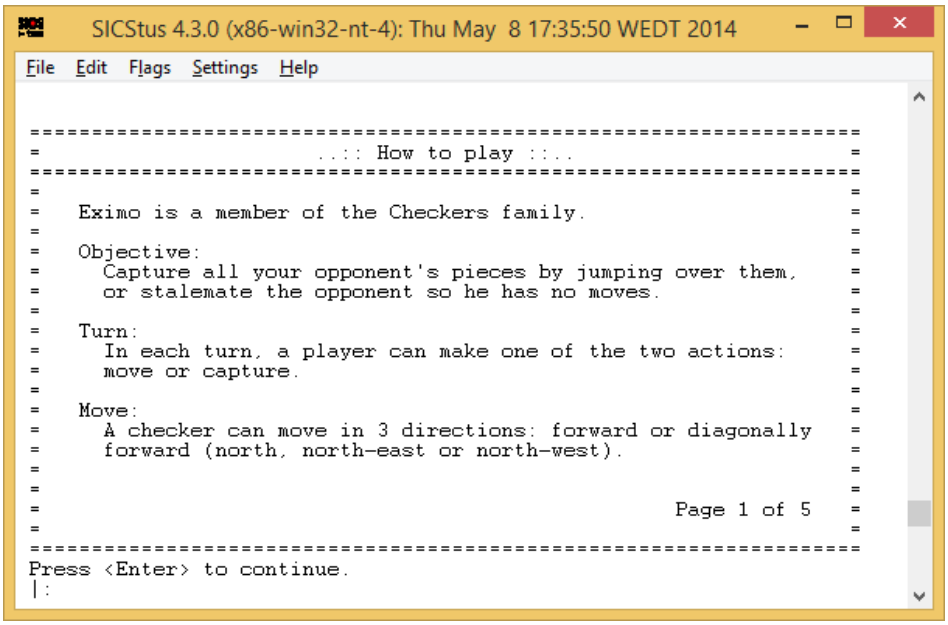

Figura 10 - Instruções do jogo.

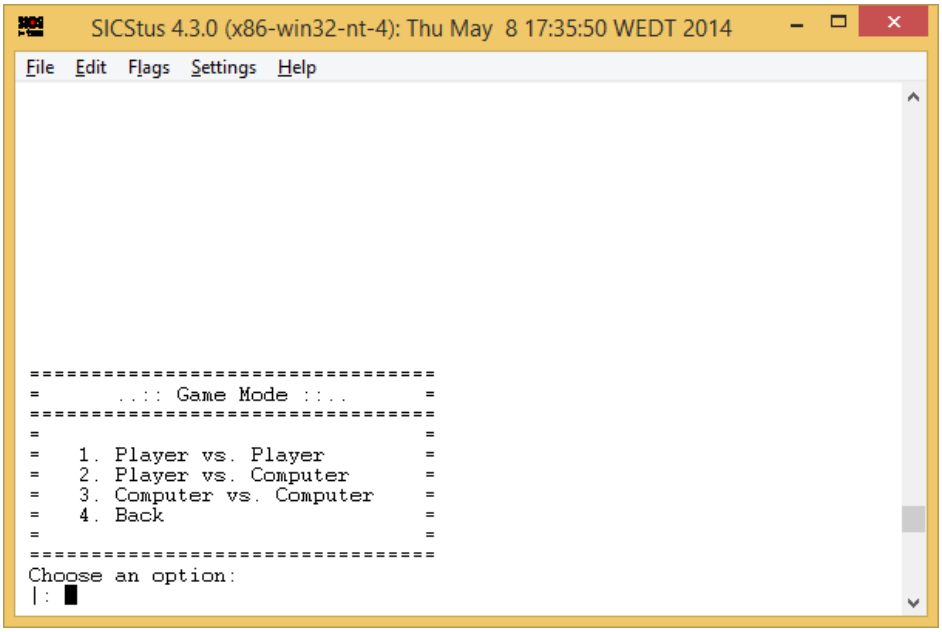

Figura 11 - Menu de escolha do modo de jogo.

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.

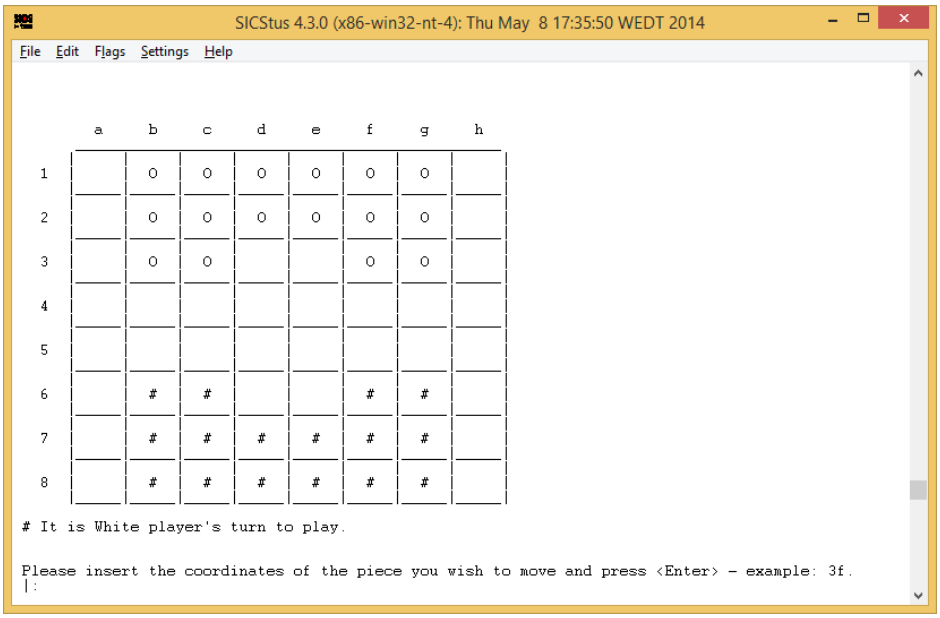

Figura 12 - Estado inicial de um jogo.

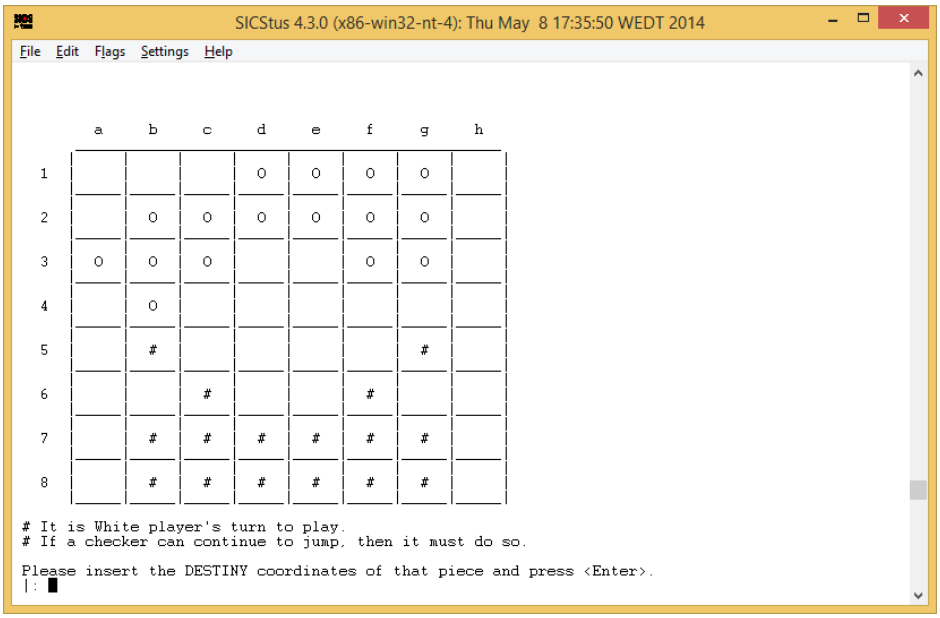

Figura 13 - Exemplo de um estado de jogo após duas jogadas.

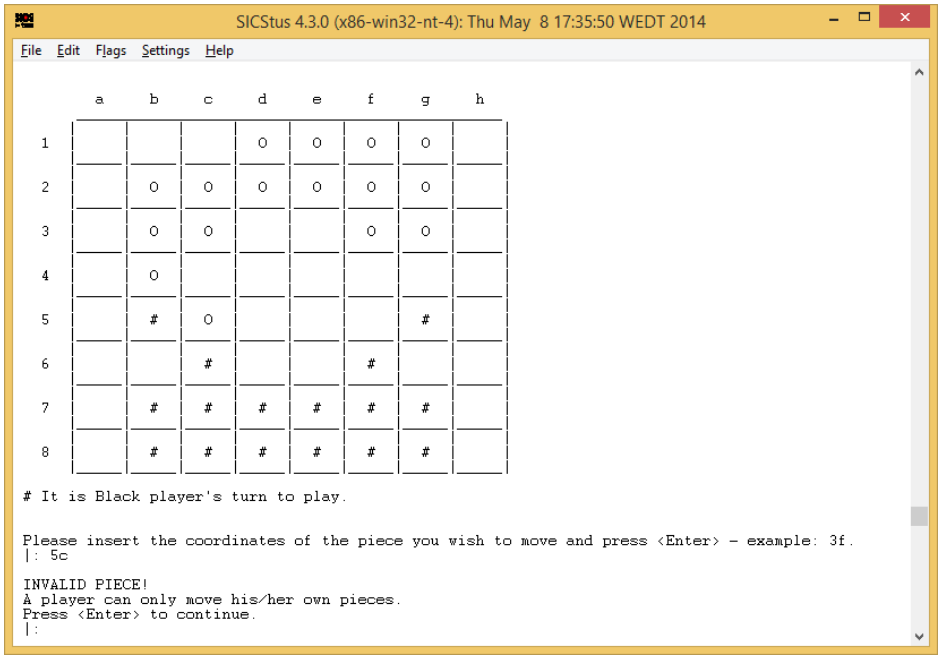

Figura 14 - Exemplo de seleção de uma coordenada cuja peça não pertence ao jogador que tem a vez de jogar.

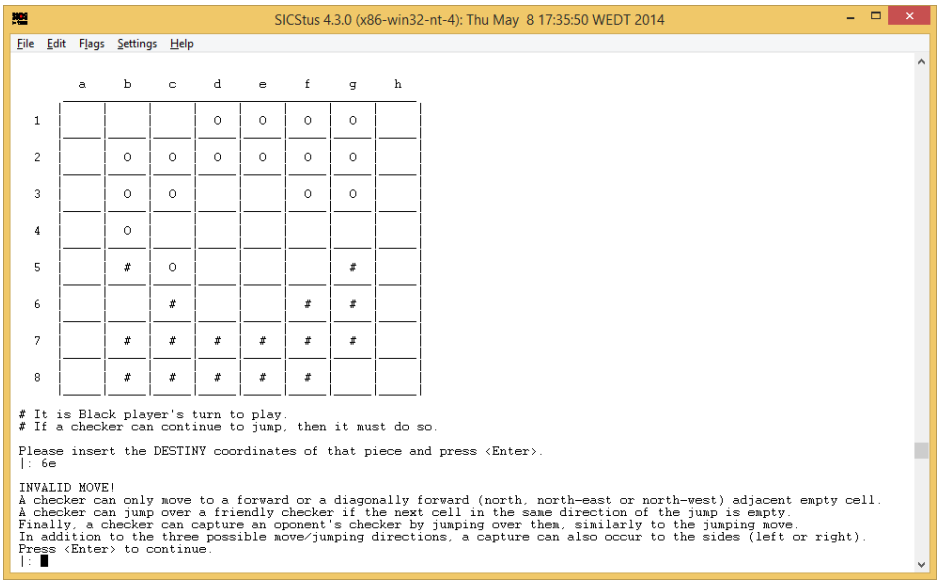

Figura 15 - Exemplo de seleção de uma coordenada de destino inválida.

5 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.

6 Bibliografia

6.1 Livros

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

6.2 Páginas Web

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

6.3 Documentação

- Documentos fornecidos na página da unidade curricular presentes no Moodle

7 Anexos

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