Ordo

Programação em Lógica

Mestrado Integrado em Engenharia Informática e Computação

Ordo_2:
Artur Sousa Ferreira - 201204899
Pedro Filipe Agrela Faria - 201406992

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

13 de Novembro de 2016

Resumo

O projeto tem o objetivo de pôr em prática os conceitos lecionados na unidade curricular **Programação em Lógica**, com foco na linguagem **Prolog**. Consiste num jogo de tabuleiro para dois jogadores que permite três modos de utilização: Humano/Humano, Humano/Computador e Computador/Computador.

Decidimos desenvolver o jogo **Ordo** e utilizamos **SICStus Prolog**.

Neste documento apresentamos, em várias secções, o que consideramos relevante de modo a permitir adquirir um conhecimento global sobre o resultado final do projeto bem como uma reflexão.

A solução que obtemos cumpre todos a maioria dos requisitos do projeto, apenas ficando a faltar o segundo nível de dificuldade do computador. Conseguimos implementar o programa de forma eficiente, recorrendo sobretudo à base de dados do Prolog e fazendo uso de listas.

A realização deste projeto foi essencial para pôr em prática e compreender melhor os conceitos teóricos inerentes à unidade curricular.

Conteúdo

1- Introdução

Em resposta ao repto lançado pelos docentes, da unidade curricular de Programação em Lógica, o grupo decidiu desenvolver o jogo Ordo, na linguagem de programação Prolog, seguindo os paradigmas da unidade curricular.

O projeto tem o objetivo de pôr em prática os conceitos lecionados na unidade curricular, com foco na linguagem Prolog. Consiste num jogo de tabuleiro para dois jogadores que permite três modos de utilização: Humano/Humano, Humano/Computador e Computador/Computador e dois níveis de inteligência.

O grupo propõe-se a fazer a nossa "própria" versão deste jogo com a preocupação de ir ao encontro dos objetivos da unidade curricular.

O código fonte está estruturado em 4 ficheiros: auxiliar.pl, display.pl, main.pl e logic.pl - cada um tem as suas funções relacionadas.

A elaboração deste relatório final tem como objetivo a consolidação do trabalho realizado ao longo da primeira metade do semestre.

Assim, o relatório está organizado e subdividido nas seguintes partes:

- **O jogo**: onde é explicado o funcionamento e principais regras do jogo.
- **Lógica do Jogo**: contendo a descrição do projeto e implementação da lógica do jogo, verificações de regras, visualização do tabuleiro e seu estado e a determinação do final jogo.
- **Interface**: fazendo referência à interface em modo de texto com o utilizador.
- **Conclusões**: contendo o comentário e análise final do grupo relativamente ao projeto.

2- O Jogo ORDO

O ordo é jogado num tabuleiro 10 por 8, composto por 2 jogadores: branco e azul, cada jogador dispõe de 20 peças da correspondente cor. O objetivo do jogo é chegar à casa do oponente, sem que as suas peças se separem.

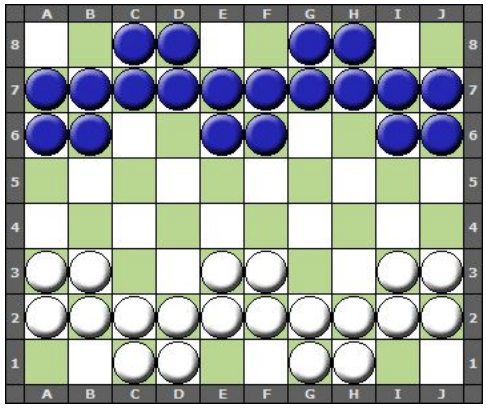

Fig.1 - Jogo inicial.

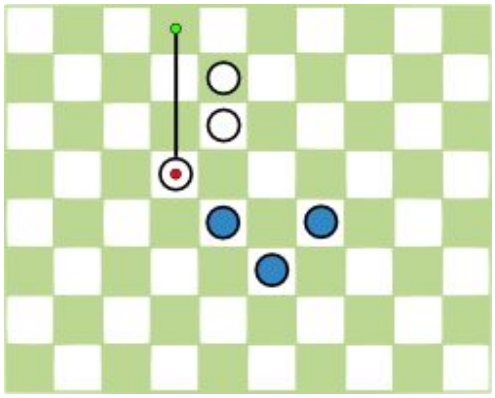
Fig.2 - Fim de jogo. Branco é vencedor.

O jogador pode também vencer o jogo se capturar todas as peças do oponente, ou se causar separação das peças do adversário sem que este consiga reagrupa-las na próxima jogada.

O jogador a iniciar a partida é o que possui as damas brancas, jogando depois alternadamente entre os dois jogadores.

Durante o jogo, cada jogador terá de manter as suas peças conectadas, verticalmente, horizontalmente, na diagonal ou na ortogonal. Caso ocorra alguma separação das peças, devido a uma jogada do oponente, o jogador terá de conecta-las na próxima jogada. Caso não o consiga, este perde o jogo.

Existem dois tipos de movimentos, singular ou ordo.

O movimento singular pode ser feito verticalmente, horizontalmente ou diagonalmente, em qualquer número de casa vazias. Se o jogador ocupar uma casa do adversário, esta é capturada e removida do tabuleiro. A única vez possível de o jogador recuar uma peça é quando este tem de reconectar o seu grupo.

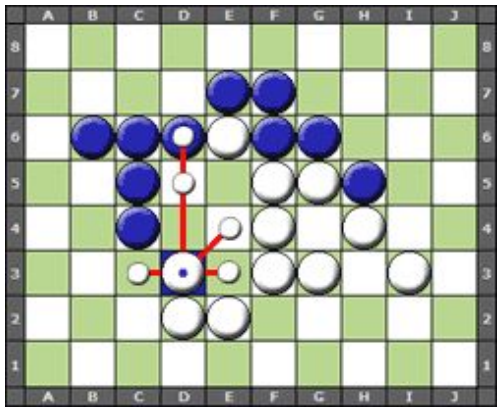

Fig.3 - 5 Movimentos possíveis nesta jogada.

O movimento ordo, é semelhante ao movimento singular, mas realizado com duas ou mais peças do grupo, que estejam conectadas em linha reta. Este movimento só é possível se houver casas vazias, não é possível capturar peças do adversário neste movimento. É obrigatório que todas as peças escolhidas para esta jogada percorram o mesmo número de casas. Semelhante ao movimento singular, o jogador pode mover-se para trás na necessidade de agrupar as peças.

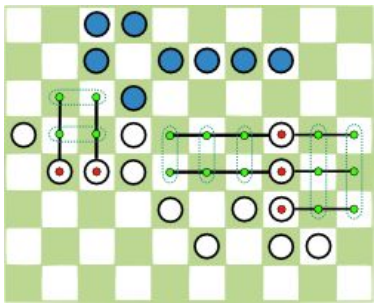
Fig.4 - 7 movimentos possíveis nesta jogada

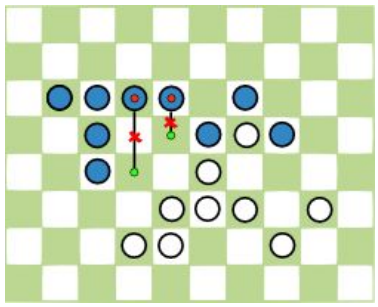
Fig.5 - Movimento impossível.

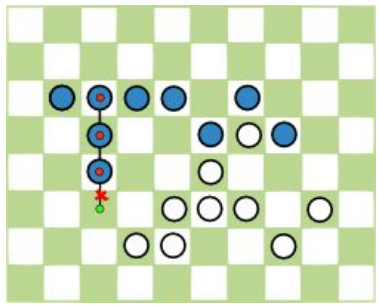
Fig.6 - Movimento em fila impossível.

Caso o jogador se encontre em modo de desconexão, este é obrigado a reconectar as peças e pode mover-se para trás. Se for impossível a reconexão o jogador perde o jogo.

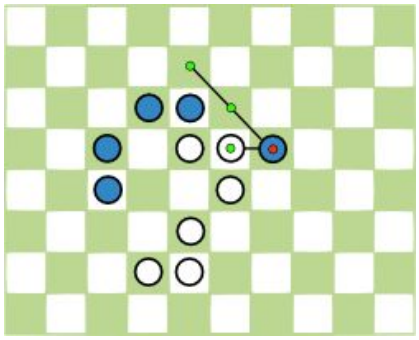

Fig.7 - 3 movimentos singular de reconecção possíveis.

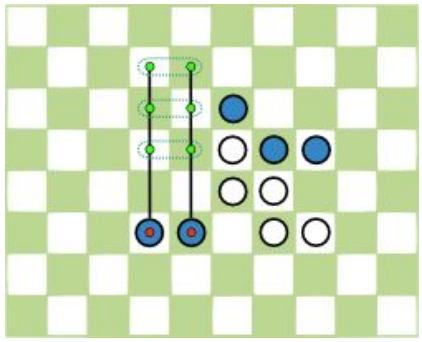

Fig.8 - 3 movimentos ordo de reconecção possíveis.

3- Lógica do Jogo

As secções seguintes descrevem o projeto e a implementação da lógica do jogo em Prolog, incluindo a forma de representação do estado do tabuleiro e sua visualização, a execução de movimentos, verificação do cumprimento das regras do jogo, determinação do final do jogo e das jogadas a realizar pelo computador.

3.1- Representação do Estado do Jogo

-Início:

```
   A   B   C   D   E   F   G   H   I   J
1-     |     | X | X |     |     | X | X |     |     |
  - - - - - - - - - - - - - - - - - - - -
2- X | X | X | X | X | X | X | X | X | X |
  - - - - - - - - - - - - - - - - - - - -
3- X | X |     |     | X | X |     |     | X | X |
  - - - - - - - - - - - - - - - - - - - -
4-     |     |     |     |     |     |     |     |     |
  - - - - - - - - - - - - - - - - - - - -
5-     |     |     |     |     |     |     |     |     |
  - - - - - - - - - - - - - - - - - - - -
6- O | O |     |     | O | O |     |     | O | O |
  - - - - - - - - - - - - - - - - - - - -
7- O | O | O | O | O | O | O | O | O | O |
  - - - - - - - - - - - - - - - - - - - -
8-     |     | O | O |     |     | O | O |     |     |
  - - - - - - - - - - - - - - - - - - - -
```

Fig.9 - Tabuleiro inicial.

-Decorrer do jogo:

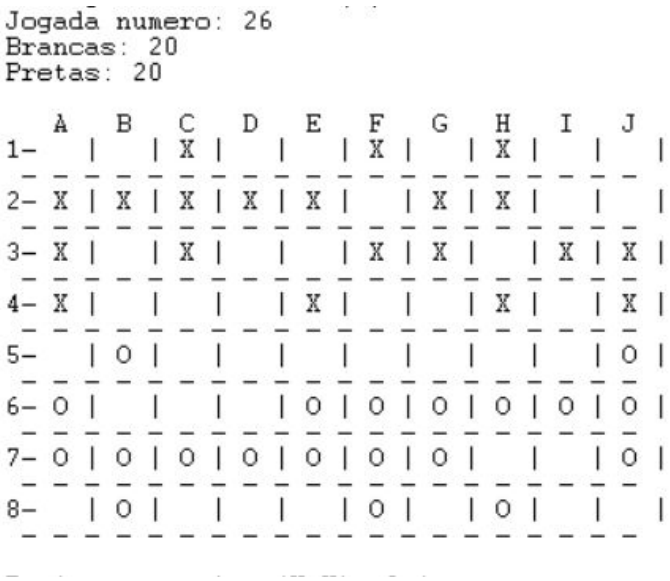

Fig.10 - Exemplo do tabuleiro após 26 jogadas.

-Fim:

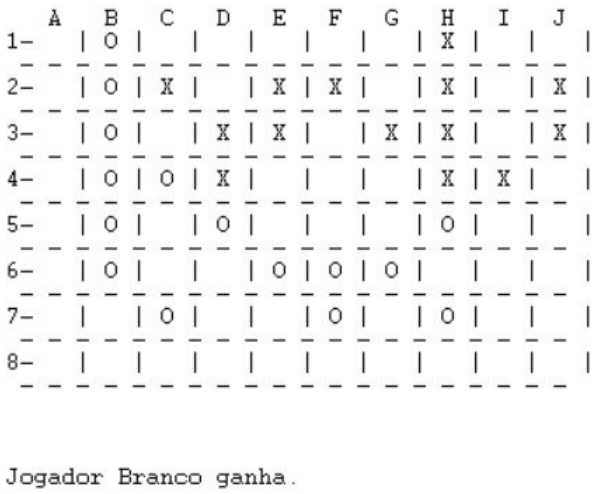

Fig.11 -Exemplo de fim do jogo

3.2- Visualização do Tabuleiro

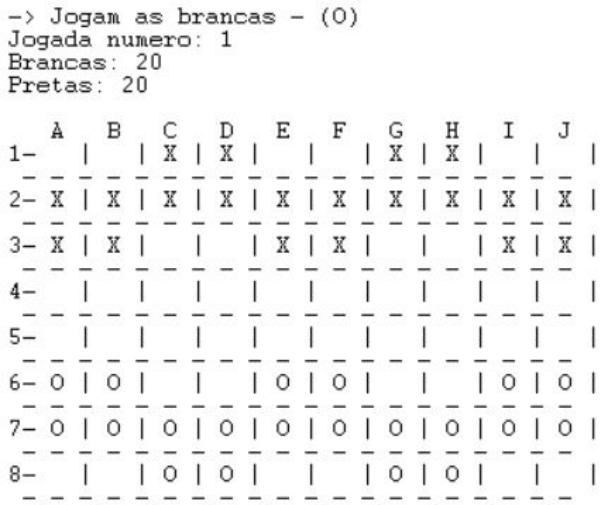

Fig.12 -Tabuleiro de Jogo

O tabuleiro é pela função **gameArea**(Jogada,Board) que invoca separadamente funções para contar o número de elementos de cada jogador, imprimir essa informação e de quem é a vez de jogar, imprime a primeira linha de A-J e depois imprime finalmente a o resto da board com a numeração.

3.3- Movimentos

Existem dois tipos de movimentos possíveis no jogo do Ordo, o movimento simples e ordo, para realizar cada movimento iremos utilizar os seguintes predicados:

- Movimento simples manual:

% Predicado principal do movimento simples das peças pretas

simpleBlackMove(TipoJogo, NumeroJogada, AtualBoard, NewBoard)

% Predicado principal do movimento simples das peças brancas

simpleWhiteMove(TipoJogo, NumeroJogada, AtualBoard, NewBoard)

- Movimento ordo manual:

% Predicado principal do movimento ordo do jogador das peças brancas

ordoWhiteMovement(TipoJogo, NumeroJogada, AtualBoard, NewBoard)

% Predicado principal do movimento ordo do jogador das peças pretas

ordoBlackMovement(TipoJogo, NumeroJogada, AtualBoard, NewBoard)

- Movimento simples automático e aleatório:

% Predicado principal do movimento simples no modo Computador para as peças pretas

simpleRandoomBlackMove(TipoJogo, NumeroJogada, AtualBoard, NewBoard)

% Predicado principal do movimento simples no modo Computador para as peças brancas

simpleRandoomWhiteMove(TipoJogo, NumeroJogada, AtualBoard, NewBoard)

Após cada movimento, será chamado um predicado, que verifica se este movimento é possível de ser efetuado ou não.

3.3- Listas de Jogadas Válidas

O grupo em vez de uma lista de jogadas válidas, optou por verificar se a jogada pretendida é válida numa board auxiliar, rejeitando as alterações caso não respeitasse as regras.

3.4- Execução de Jogadas

A execução das jogadas no tabuleiro só é efetivamente executada se for validade as seguintes funções:

% Conjunto de algumas regras do jogador das peças brancas

verifySimpleWhiteMove(TipoJogo, STATUS_CONECTION, NewElement, OldElement, NumeroJogada, AtualBoard, NewElement2, OldY, NewY)

% Conjunto de algumas regras do jogador das peças pretas

verifySimpleBlackMove(TipoJogo, STATUS_CONECTION, NewElement, OldElement, NumeroJogada, AtualBoard, NewElement2, OldY, NewY)

% Verifica se no modo ordo o jogado tenta tirar uma peça do adversário

cantEatOponentBlackOrdo(TipoJogo, NumeroJogada, AtualBoard, Element)

% Verifica se no modo ordo o jogado tenta tirar uma peça do adversário

cantEatOponentWhiteOrdo(TipoJogo, NumeroJogada, AtualBoard, Element)

% Verifica se a peça a escolher para mover é a do jogador adversário

verifyElementNonBlack(TipoJogo, Element, NumeroJogada, AtualBoard)

% Verifica se a peça a escolher para mover é a do jogador adversário

verifyElementNonWhite(TipoJogo, Element, NumeroJogada, AtualBoard)

% Verifica se a peça a escolher para mover é uma peca da board vazia

verifyElementNonNone(TipoJogo, Element,NumeroJogada,AtualBoard)

% Verifica se a peça a escolher como destino e uma das suas próprias peças - Jogador X

verifyPieceX(TipoJogo, NewElement, NumeroJogada, AtualBoard, NewElement2)

% Verifica se a peça a escolher como destino e uma das suas próprias peças - Jogador O

verifyPieceO(TipoJogo, NewElement, NumeroJogada, AtualBoard, NewElement2)

% Verifica se o jogador X pode andar para trás

verifyBlackNonBackMove(TipoJogo, NewY,OldY , NumeroJogada, AtualBoard)

% Verifica se o jogador O pode andar para trás

verifyWhiteNonBackMove(TipoJogo, NewY,OldY , NumeroJogada, AtualBoard)

% Verifica a conexão do jogador no final da jogada dependendo se estava conectado ou não ao início da jogada

verifyConnectionByPlayerOrOther(TipoJogo, NumeroJogada, AtualBoard, NewBoard, STATUS_PLAYER1, STATUS_PLAYER2, STATUS_CONECTION2)

% Função auxiliar da verifyElementConnection() - limpa o elemento e verifica os seus vizinhos, limpando estes até não haver mais elementos vizinhos

verifyConnection(X, Y, Element)

% Verifica a conectividade das peças de um jogador, retorna 1 se conectado e 0 se caso contrário

verifyElementConnection(BoardToTest, Element, Return)

% Verifica se está conectado a uma outra peça após uma jogada

connected(TipoJogo, Board, Backup, X, Y, Element, NrJogada)

Estes predicados invocam outros que não estão aqui representados, mas os comentários ajudam a entender. De qualquer forma, no código fonte está completa a sua definição. O código fonte está estruturado, e esta parte da lógica encontra-se no ficheiro logic.pl

3.5- Avaliação do Tabuleiro

% Verifica a conectividade das peças de um jogador, retorna 1 se conectado e 0 se caso contrário

verifyElementConnection(BoardToTest, Element, Return):-

```
asserta(dyBoard(BoardToTest)),
getPositionElement(Element,BoardToTest,Xval,Yval),
verifyConnection(Xval, Yval, Element),
dyBoard(List),retract(dyBoard(_)),
contaListaDeLista(Element, List, NrElements),
((NrElements > 0, Return is 0);
(NrElements == 0, Return is 1)).
```

3.6- Final do Jogo

O jogo termina se um jogador conseguir chegar à outra margem do tabuleiro ou se o jogador que se encontra em modo de desconexão não se reconecta na jogada seguinte.

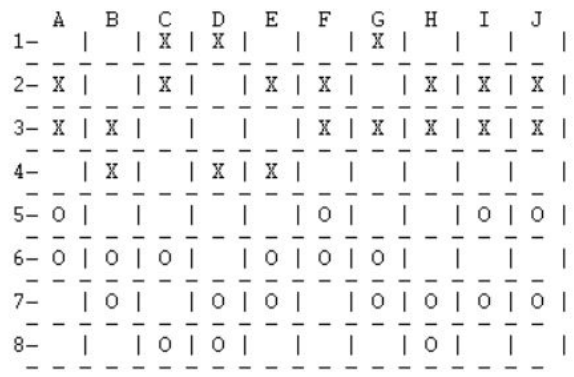

Fig.13 - Exemplo de fim do jogo - Jogador branco desconectado.

```
Nao estas conenctado!!!
Jogador Preto ganha.
```

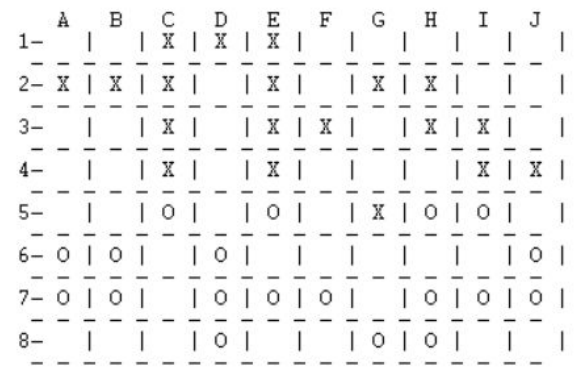

Fig.14 - Exemplo de fim do jogo - Jogador preto desconectado.

```
Nao estas conenctado!!!
Jogador Branco ganha.
```

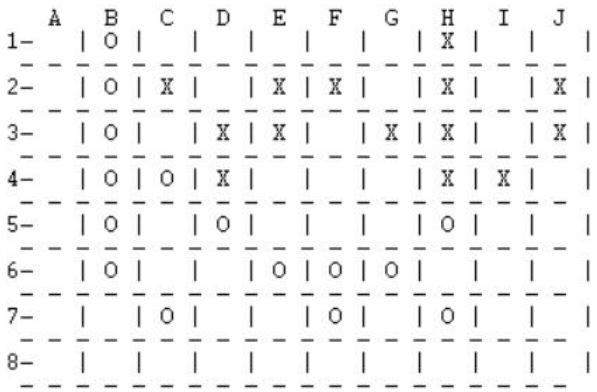

Fig.15 - Exemplo de fim do jogo - Jogador branco alcança a margem do jogador preto.

```
Jogador Branco ganha.
```

3.7- Jogada do Computador

```
%Jogada Par - Jogam as Pretas - ' X ' - COMPUTADOR

computadorvscomputador(NumeroJogada,AtualBoard) :-
        par(NumeroJogada),
        gameArea(NumeroJogada, AtualBoard),
        simpleRandoomBlackMove(3, NumeroJogada, AtualBoard, NewBoard),
        Y is NumeroJogada + 1,
        computadorvscomputador(Y, NewBoard).
```

```
%Jogada Impar - Jogam as Brancas - ' O ' - COMPUTADOR

computadorvscomputador(NumeroJogada,AtualBoard) :-
        impar(NumeroJogada),
        gameArea(NumeroJogada, AtualBoard),
        simpleRandoomWhiteMove(3, NumeroJogada, AtualBoard, NewBoard),
        Y is NumeroJogada + 1,
        computadorvscomputador(Y, NewBoard).
```

4- Interface com o Utilizador

O utilizador apenas necessita de digitar alguns caracteres no teclado, seguidos da tecla Enter, interagindo dessa forma com o jogo.

No ecrã inicial o jogador tem de escolher se pretende iniciar um novo jogo (jogador vs jogador), ou modo de jogo que tem todos os modos de jogo (jogador vs jogador, jogador vs computador e computador vs computador).

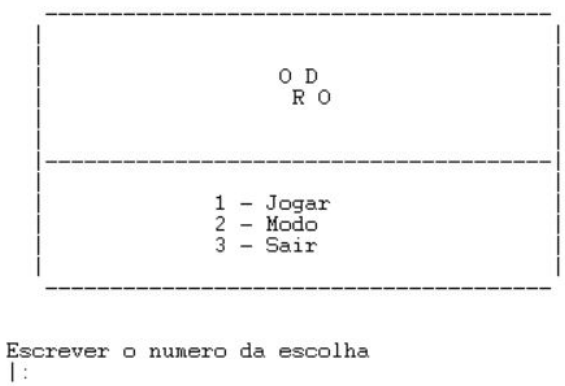

Fig.16 - Menu inicial do Jogo.

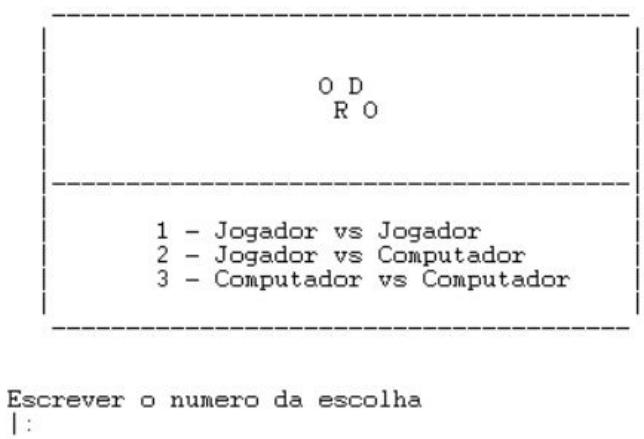

Fig.17 - Menu de escolha do Modo de jogo.

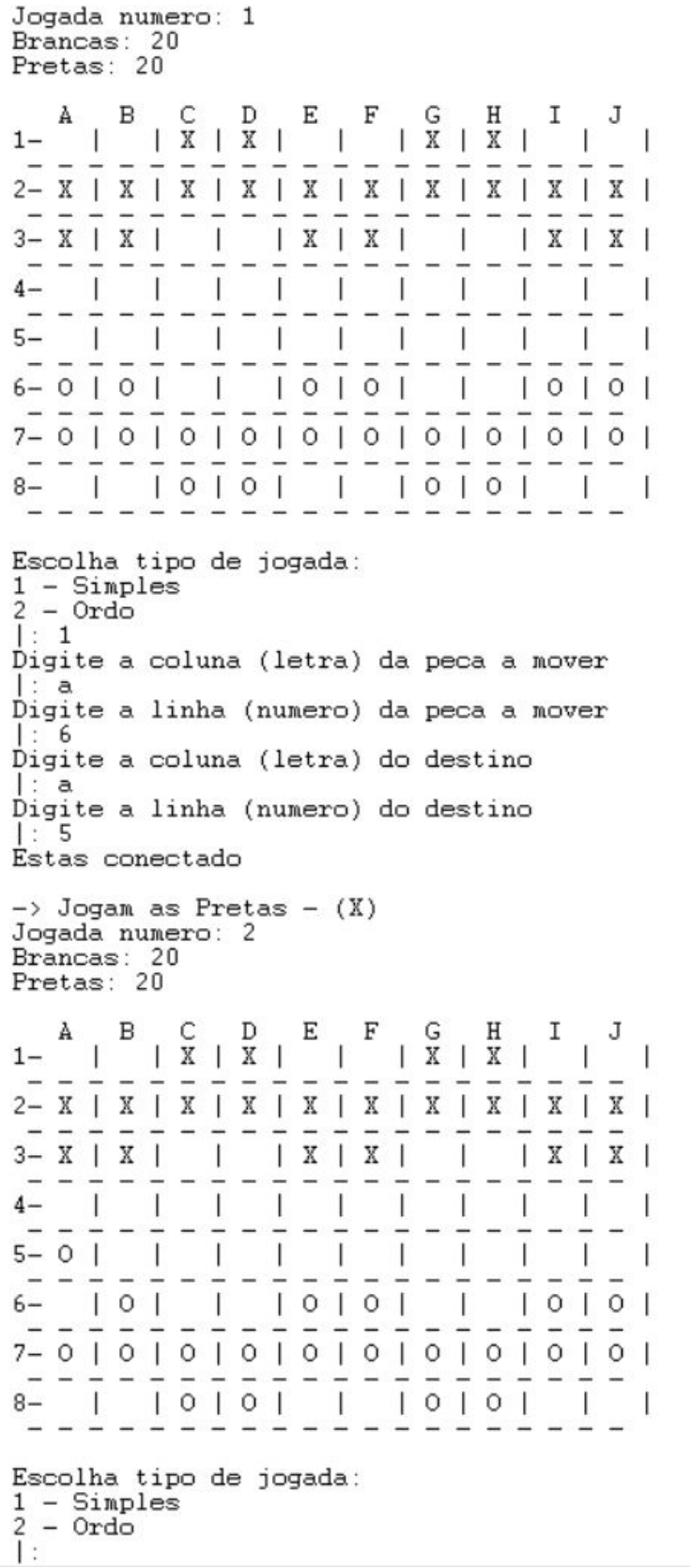

Fig.16 - Exemplo de Jogo - Jogador vs Jogador .

5- Conclusões

O grupo utilizou o trabalho da melhor maneira para pôr em prática os conceitos leccionados nas aulas. Trabalhamos desde a primeira hora e todos os prazos foram cumpridos.

Em traços muito gerais o projeto desenvolvido por nós simula o jogo Ordo, um jogo de tabuleiro para dois jogadores, e permite três modos de utilização: Humano/Humano, Humano/Computador e Computador/Computador. No entanto não conseguimos a implementação do segundo nivel de inteligencia.

A maior dificuldade sentida foi na implementação da lógica de jogo, nomeadamente a regra de as peças de cada jogador estarem sempre todas conectadas e o movimento ordo.

Concluímos que atingimos quase todos os objetivos que nos tínhamos proposto, que a contribuição foi equitativa e trabalhamos bem como grupo.

Bibliografia

- http://www.iggamecenter.com/info/pt/ordo.html#classic
- https://spielstein.com/games/ordo/rules/official
- https://sicstus.sics.se/
- Apontamentos das aulas teóricas