Morelli

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo 3:
Francisco Rodrigues - 201305627
Marta Lopes - 201208067

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

8 de Novembro de 2015

Resumo

Este trabalho teve como finalidade consolidar todo o conhecimento que fomos adquirindo na cadeira de PLOG e aplica-lo para criar um produto final.

Como grupo podemos dizer que cooperamos bastante e trabalhamos sempre lado-a-lado, tornando todo o desenvolvimento do jogo mais fácil e fluido.

Prolog é uma linguagem de programação que se enquadra do paradigma da Programação em Lógica Matemática o que a torna um pouco diferente das linguagens a que estávamos habituados o que tornou mais difícil a resolução de pequenos problemas que felizmente foram sendo resolvidos com recurso à consulta dos materiais fornecidos pelos professores. Com isso conseguimos alcançar um jogo final não tão completo como o que gostaríamos que fosse no inicio, mas estamos contentes com o que conseguimos desenvolver apesar de tudo.

Conteúdo

1 Introdução **4**

2 Morelli **4**

3 Lógica do Jogo **8**
3.1 Representação do Estado do Jogo e Visualização do Tabuleiro . . 8
3.2 Lista de Jogadas Válidas 12
3.3 Execução de Jogadas 12
3.4 Final de Jogo 13

4 Jogada do Computador **13**

5 Interface com o Utilizador **14**

6 Conclusões **17**

Bibliografia **18**

A Pasta *code* **19**

1 Introdução

Para este projeto tínhamos como objetivo desenvolver um jogo em linha de comandos, como tal, decidimos escolher o *Morelli* porque achamos que seria um jogo apelativo e interessante para desenvolver. De inicio o jogo parecia bastante simples, mas à medida que foi sendo desenvolvido notamos um certo aumento da dificuldade, no entanto nunca deixamos de perceber a dinâmica de jogo.

Reparamos em certas semelhanças com o jogo *Othello* e *Ming Mang* mas no geral o jogo que nós desenvolvemos é muito mais abstracto que todos os outros jogos de tabuleiro que conhecemos. Este trabalho serviu então para avaliar todos os nossos conhecimentos nesta linguagem nova para nós, e desenvolvendo este jogo que é completamente diferente daquilo a que estamos habituados.

Este relatório encontra-se dividido nas seguintes secções:

- História e regras de jogo;
- Implementação da lógica de jogo;
- Descrição da interface;
- Conclusões sobre este projeto;
- Bibliografia;
- Anexos, como o código do projeto.

2 Morelli

Morelli é um jogo de tabuleiro criado por *Richar Moxham* em 2011. É jogado por apenas 2 jogadores num tabuleiro de 13x13 quadrados em faixas concêntricas. A faixa de fora será a maior, com 48 casas, terminando na casa central.

- O jogo começa com **24 peças pretas** e **24 peças brancas**, ambas reversíveis, posicionadas nos quadrados de fora, de forma diametralmente oposta. Para além disso cada jogador vai ter **uma torre da cor respectiva**, que não vai entrar no tabuleiro no inicio do jogo.

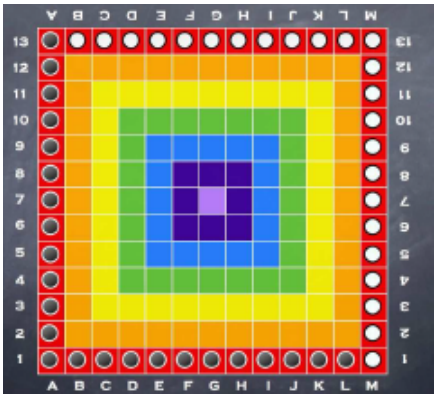

Figura 1: Inicio de Jogo

- Cada jogador joga na sua vez, começando primeiro o jogador que tiver as peças da cor preta. Um **movimento legal** (Fig.2) consiste em mover a peça para um quadrado desocupado numa linha ortogonal ou diagonal desde que seja para uma faixa mais próxima do centro do que aquela em que se encontra no momento, não se podendo manter na mesma faixa ou voltar para trás (Fig.3).

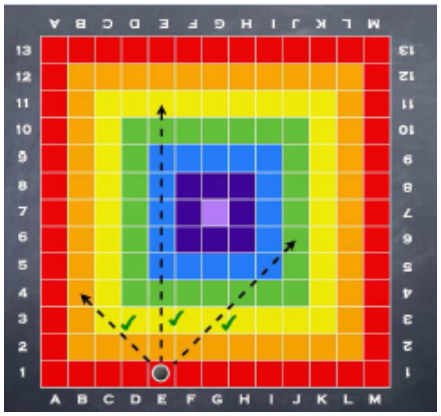

Figura 2: Movimento legal

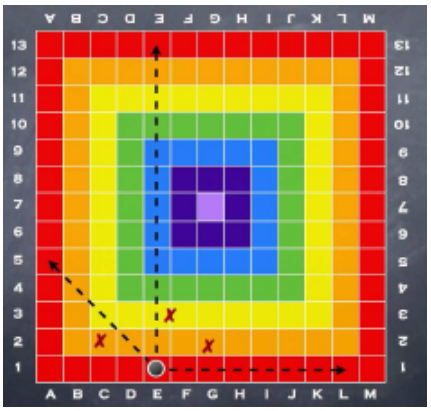

Figura 3: Movimento Ilegal

- Apenas será possivel passar pelo centro se este estiver vazio, não podendo parar nele. (Fig.4, Fig.5).

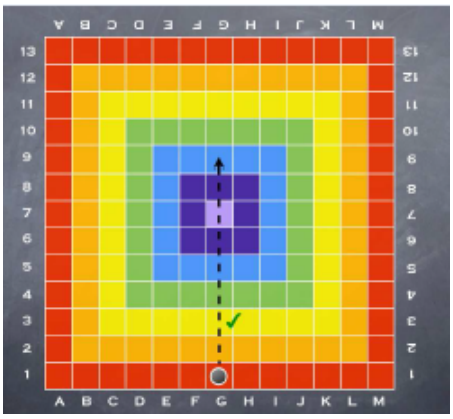

Figura 4: Movimento Legal

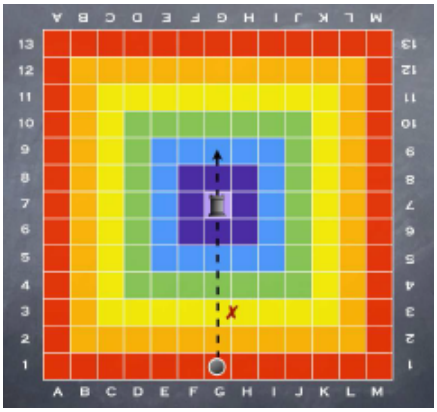

Figura 5: Movimento Ilegal

- A **captura do centro** é feita quando o jogador cria um quadrado de qualquer tamanho com as suas peças, centrado na célula central (Fig.4 e Fig.5). Quando o centro está capturado é colocada a torre da cor respectiva no centro do tabuleiro, removendo a torre adversária se se aplicar.

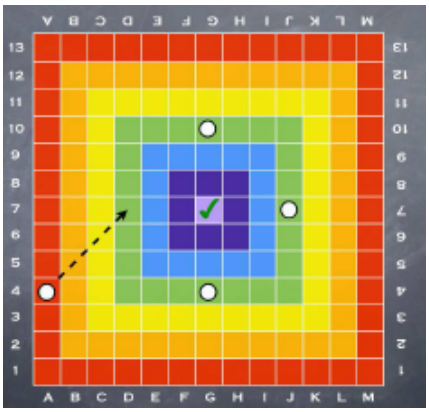

Figura 6: Captura do centro

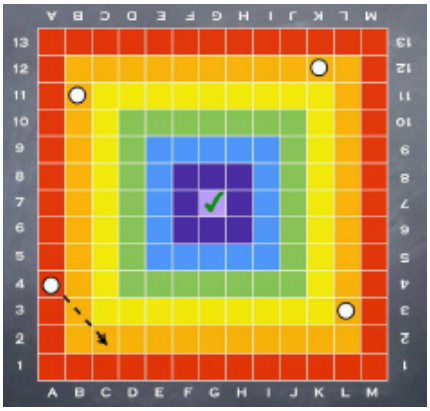

Figura 7: Captura do centro

- A cada movimento o jogador poderá **capturar uma peça adversária** quando conseguir rodea-la por duas peças da sua cor seja ortogonal ou diagonalmente (Fig.6). Uma peça capturada é revertida passando para a cor contrária.

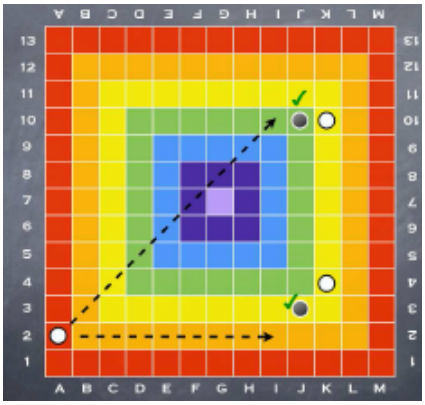

Figura 8: Captura de peça adversária

- O **vencedor** será o jogador que terá a sua torre no centro no final do jogo. O jogo acaba quando não existirem mais jogadas possíveis. Se o centro se mantiver sem nenhuma torre até ao final do jogo, é um empate.

3 Lógica do Jogo

3.1 Representação do Estado do Jogo e Visualização do Tabuleiro

Sendo o tabuleiro de 13x13 o estado de jogo é representado numa matriz com esse tamanho, onde a partir dos predicados que obtêm e modificam o elemento vão substitui-lo para a visualização no tabuleiro. Cada elemento dessa lista será um espaço vazio, uma peça ou o centro vazio ou ocupado por uma torre.

Para distinguir as peças dos dois jogadores, estas vão ter um valor diferente. As peças do jogador preto vão ter valor 1 e a sua torre o valor 4 e as peças do jogador branco vão ter valor 2 sendo que a sua torre terá o valor 5. O centro vazio terá o valor de -1 e o espaço vazio de 0. A visualização do tabuleiro é feita com o predicado **startDrawingBoard(_, +BoardSize, -Board1)**.

Na **visualização do tabuleiro**, as peças pretas serão um 'x' e a torre preta um 'X', já as peças brancas serão um 'o' e a sua torre um 'O'. O centro vazio será representado por um 'N'.

A representação de vários estados do tabuleiro será então apresentada em baixo com a respectiva visualização do tabuleiro na consola.

```
gameExampleStart(K) :-
    K =
    [
    [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 2],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2 ]
    ].
```

Figura 9: Exemplo de um estado inicial do tabuleiro

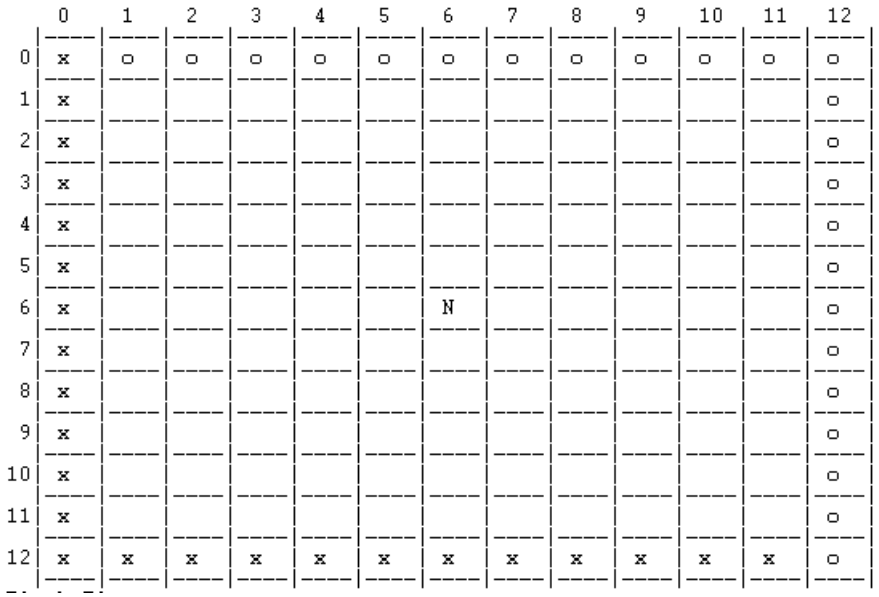

Figura 10: Visualização de um estado inicial na consola

```
gameExampleTest(K) :-
    K =
    [
    [1, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 2, 0, 0, 2, 0, 0, 0, 2, 0, 2 ],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 2],
    [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 0, 2 ],
    [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2 ],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
    [0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 2 ]
    ].
```

Figura 11: Exemplo de um estado intermédio do tabuleiro

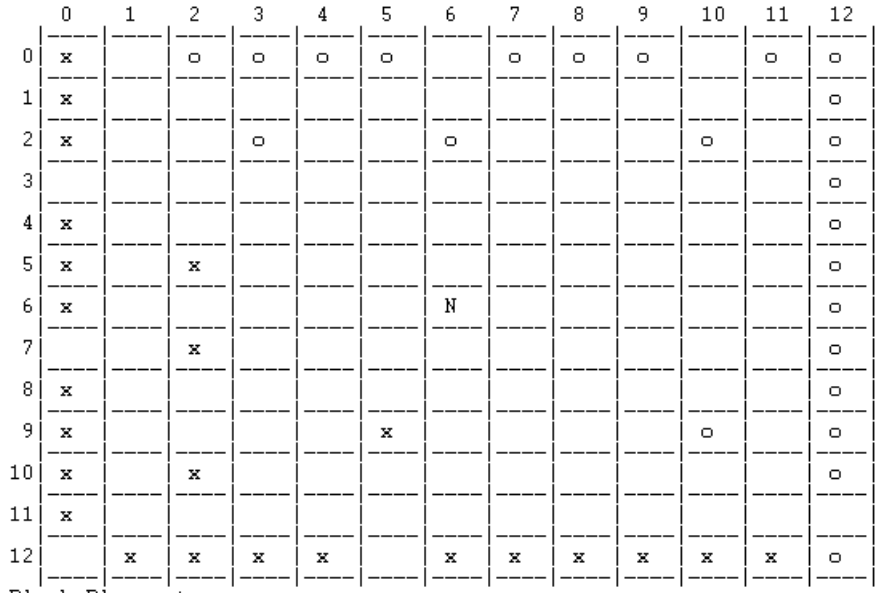

Figura 12: Visualização de um estado intermédio na consola

```
gameExampleEnd(K) :-
    K =
    [
    [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, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0],
    [0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
    [0, 0, 0, 1, 2, 2, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 2, 1, 2, 2, 4, 1, 1, 0, 0, 0, 0],
    [0, 0, 2, 1, 2, 2, 2, 1, 1, 0, 0, 0, 0],
    [0, 0, 2, 2, 2, 2, 2, 2, 1, 2, 0, 0, 0],
    [0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 0, 0, 0],
    [0, 0, 0, 2, 0, 0, 0, 0, 2, 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]
    ].
```

Figura 13: Exemplo de um estado final do tabuleiro

```
GAME OVER
The winner is: White Player!
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0													
1													
2													
3				x	x	x	x	x	x	x			
4			x	x	x	x	x	x	x				
5				x	o	o	x	o	x				
6			o	x	o	o	O	x	x				
7			o	x	o	o	o	x	x				
8			o	o	o	o	o	o	x	o			
9				x	o	o	o	o	o	o			
10				o					o				
11													
12													

Figura 14: Visualização de um estado final na consola

3.2 Lista de Jogadas Válidas

A lista de listas jogadas válidas vai ser obtida a partir do predicado **validMoves(0,0,13, +Piece, +Board, -ListOfMoves)**. A lista de cada jogada vai conter no inicio as coordenadas de origem da peça pretendida e de seguida todas as coordenadas possiveis de destino para essa peça, por exemplo:
[[0, 0, [1,1], [2,2], [3,3], [4,4], [5,5], [7,7], [8,8], [9,9], [10,10], [11,11]].

```
validMoves(A, Max, Max, Piece, Board, ListOfMoves):-
    A2 is A + 1,

    validMoves(A2, 0, Max, Piece, Board, ListOfMoves).



validMoves(A, B, Max, Piece, Board, ListOfMoves):-
    B2 is B + 1,

    getMatrixElemAt(A, B, Board, Elem),
    ((Elem \= Piece) -> validMoves(A, B2, Max, Piece, Board, ListOfMoves);
                        findall([DestRow,DestCol],
                        validInput(A, B ,DestRow,DestCol,Board),Bag),
                        append([A,B], Bag, Res),
                        append(ListOfMoves, [Res], List),
                        validMoves(A, B2, Max, Piece, Board, List)).


validMoves(Max2, Max, Max,_, _,ListOfMoves):-
    Max2 is Max - 1,
    write(ListOfMoves).
```

Figura 15: Código para gerar a lista de jogadas válidas

3.3 Execução de Jogadas

A cada jogada, é pedido ao jogador as coordenadas da peça que deseja mover e as coordenadas de destino da respectiva peça. Ao longo deste processo, vão ser feitas as seguintes verificações:

1. **getPieceCoords(+Board, +Player,+CurrRow, +CurrCol)** vai verificar se as coordenadas de origem correspondem à peça do jogador respectivo.

2. **getDestCoords(+Board, +Player, +CurrRow, +CurrCol, +DestRow, +DestCol)** vai ter os predicados de verificação e execução do *move*

   2.1. O predicado de verificação, **validInput(+CurrRow, +CurrCol, +DestRow, +DestCol, +Board)**, vai fazer todas as verificações de acordo com as regras para saber se o movimento é legal ou não. Se for legal a matriz vai ser alterada.

   2.2. Após a alteração da matriz vão ser usados predicados para verificar capturas (**checkCapture(+DestRow, +DestCol, +Piece,+**

Board2, -Board3)) e a captura do centro (**checkCenter(+DestRow, +DestCol,+Piece, +Board3, -Board4)**). A captura do centro é verificada não só para a peça que foi movimentada, mas também para as peças que possam ser capturadas na sequência da jogada.

Depois de todas as verificações e alterações na matriz de jogo, é chamado o predicado **switchPlayer(-NextPlayer, +Player)** que vai alternar os jogadores no fim de cada jogada.

3.4 Final de Jogo

Quando o predicado de jogo é chamado, é sempre verificado o final de jogo com o predicado **checkEnd(+Board, 1, 1, 13, +Piece)**, se não houver mais movimentos possiveis para o jogador que está a jogar este predicado chama o predicado **gameOver(+Board)** que vai terminar o jogo dizendo o vencedor ou, se for o caso, devolver um empate e regressar ao menu inicial.

4 Jogada do Computador

Devido à falta de tempo apenas nos foi possivel implementar o **random bot**. É feito no predicado **getRandomPlay(+Board, +ListOfMoves,+ Player)** que antes já passou pelo predicado **validMoves** que vai gerar as jogadas possiveis para um dado jogador.

O *bot* vai escolher da lista de listas de jogadas, uma lista aleatória, obtendo nas primeiras duas posições a posição original da peça. De seguida, vai escolher da lista uma jogada possível e vai implementa-la. As verificações de captura de peças adversárias, de centro e de fim de jogo também são implementadas neste caso.

5 Interface com o Utilizador

A interface da linha de comandos foi feita para ser simples de perceber e para facilitar a experiência de jogo. Os menus de navegação estão identificados e para navegar entre eles é necessário escolher o número correspondente e pressionar *Enter*. O jogador para além do menu de jogo, poderá aceder a um pequeno resumo das regras, e um *About*.

Figura 16: Menu inicial

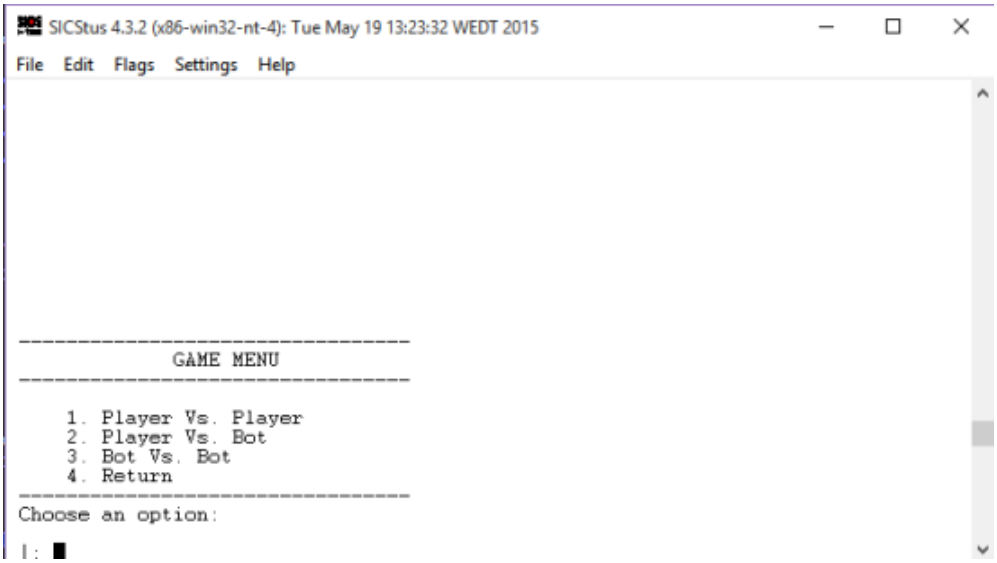

Figura 17: Menu de Jogo

Durante o jogo a linha de comandos é limpa, é exibida uma mensagem para o jogador respectivo iniciar a jogada, e o estado atual do tabuleiro é imprimido na consola. É então solicitado ao jogador as coordenadas da peça que quer movimentar e as coordenadas de destino.

Para a inserção de coordenadas o jogador terá que inserir primeiro a *Row* e de seguida a *Column*. Quando o jogador tenta jogar com uma peça que não seja a dele, ou mover a sua peça para um destino inválido, são exibidas mensagens de erro e o jogador terá de recomeçar a inserção das coordenadas de origem e destino.

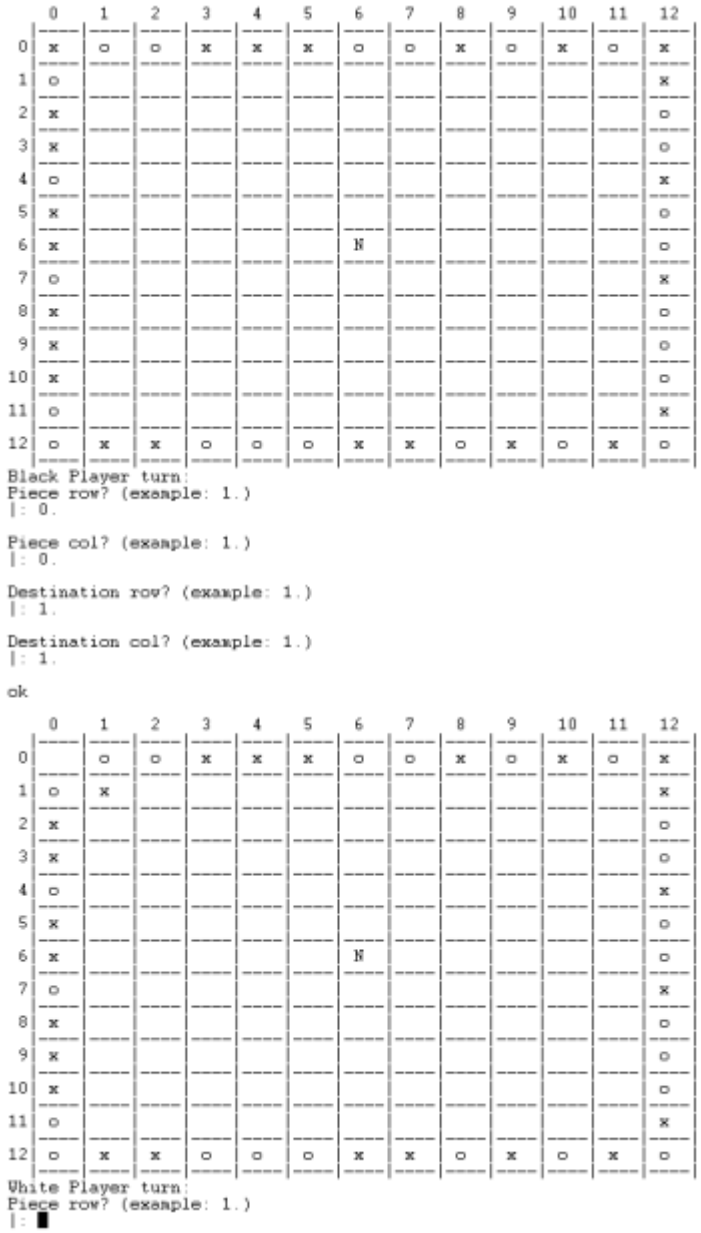

Figura 18: Realização de uma jogada

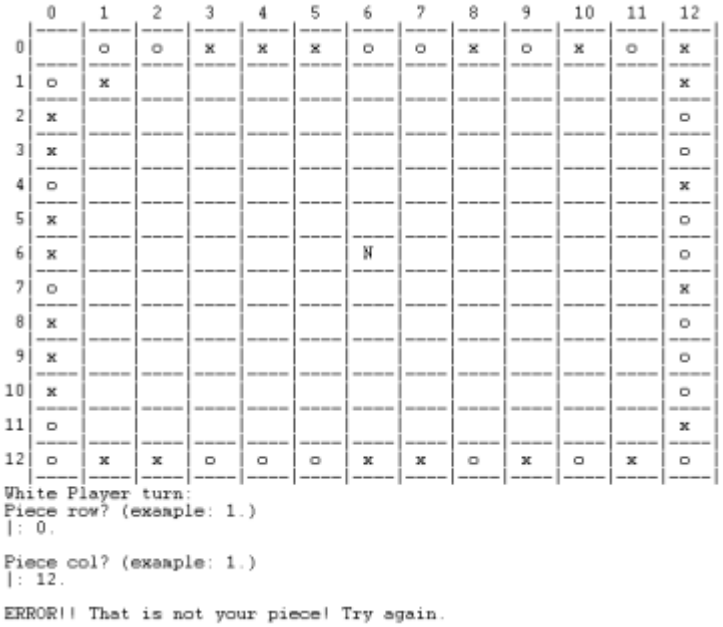

Figura 19: Exemplo de uma seleção de peça inválida

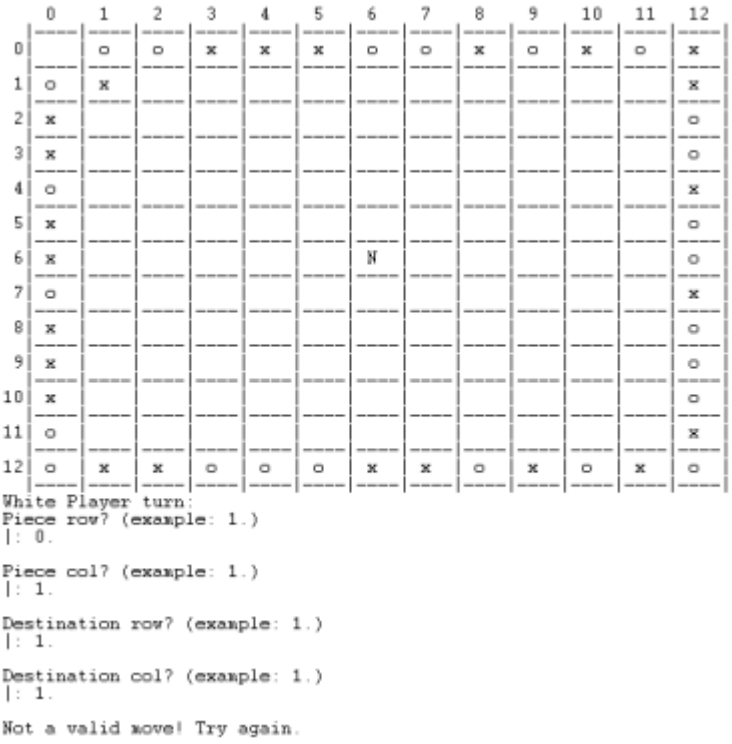

Figura 20: Exemplo de uma escolha de coordenadas de destino inválidas

6 Conclusões

Para a realização do Morelli foi dispendido bastante tempo mas achamos que o resultado final foi positivo e conseguimos com este projecto adquirir bastantes conhecimentos. Apesar de acharmos que o tempo para a entrega final poderia ter sido maior, foi possivel concluir uma grande parte do projecto.

As dificuldades que fomos encontrando ao longo do desenvolvimento foram superadas, apesar de que a implementação do *bot* não foi implementada totalmente e também poderiamos ter melhorado algumas partes do código para não ficar tão extenso.

Bibliografia

[1] "Morelli."BoardGameGeeks. N.p., 2011. Web. 2015.
[2] "Morelli at Boardspace.net"Boardspace.net. N.p., 2011. Web. 2015.

A Pasta *code*

O código *Prolog* do projeto encontra-se na pasta *code* que está anexada junto deste relatório.