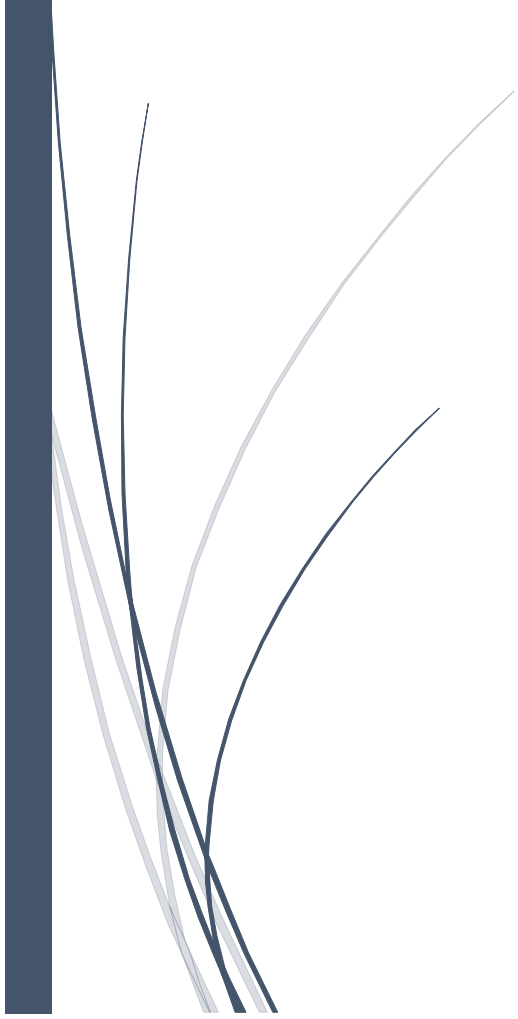
Morelli

Programação Em Lógica



Mariana Gaspar Oliveira: up201207835
Ricardo Dantas Cerqueira: up201304000

Resumo

Foi implementado na linguagem Prolog o jogo de tabuleiro Morelli.

O objectivo deste projecto era criar este jogo com os modos Humano vs Humano, Humano vs Computador e Computador. Para o modo de jogo do computador foram implementados três graus de dificuldade, sendo eles: 1- jogadas aleatórias, 2- método ganancioso com uma jogada de consideração, 3- algoritmo MiniMax com podagem alfa-beta e duas jogadas de consideração.

Uma das dificuldades encontradas foi a implementação do algoritmo MiniMax de modo a manter o jogo eficiente. No entanto, essa dificuldade e outras menores foram ultrapassadas e os objectivos foram cumpridos.

Em suma, o projecto foi bem sucedido e todos os modos de jogo são funcionais. Na realização deste mesmo projecto foi possível a melhor compreensão dos conteúdos leccionados nas aulas tais como outros que achámos necessário incluir para melhorar o jogo por nós implementado.

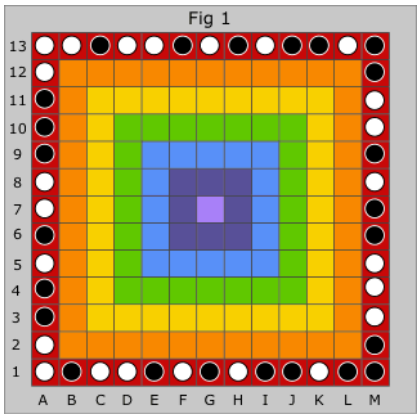
História e regras do jogo

O jogo Morelli cujo nome deriva de uma personagem do livro Hopscotch do autor Julio Cortázar, é jogado por dois jogadores num tabuleiro dividido em 13x13 quadrados e colorido em bandas concêntricas com as sucessivas cores do arco-íris: do vermelho ao violeta. O quadrado do centro é denominado *O Trono*.

Existem 48 peças reversíveis pretas/brancas, lisas e circulares.

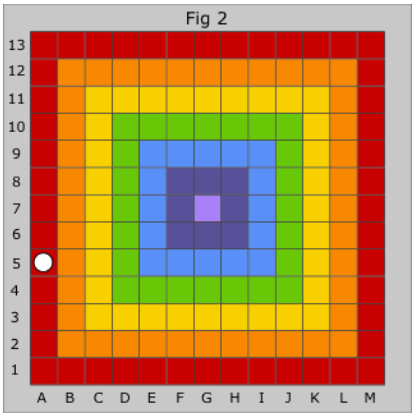
No início do jogo, as equipas preta e branca têm 24 peças cada uma. Estas são dispostas no tabuleiro de forma aleatória com a única condição que cada peça tem de ter uma peça de cor contrária na sua diagonal oposta, como mostra na Fig. 1 abaixo.

Fig 1



Inicialmente, o Trono está vazio mas este pode vir a ser ocupado por um Rei de qualquer uma das duas equipas. Uma vez que o Trono é ocupado, este pode alternar um número ilimitado de vezes de Branco para Preto e vice-versa. O vencedor é o jogador cujo Rei está no Trono no momento em que o jogo acaba. Se o Trono permanecer vazio do inicio ao fim do jogo, é um empate.

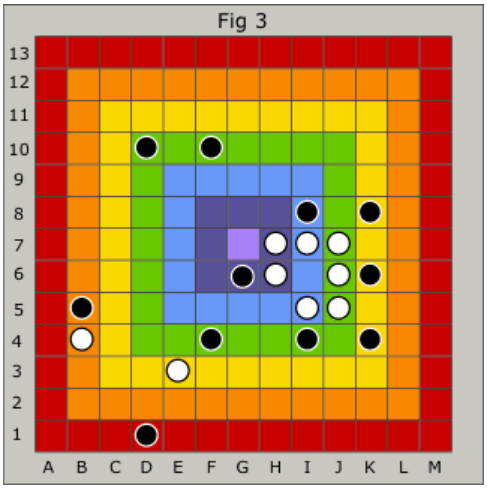
A equipa preta é a primeira a jogar e, após essa jogada, a equipa branca e preta jogam alternadamente. Cada peça pode ser movida qualquer distância e em qualquer direção ao longo de linhas perpendiculares e diagonais à sua posição, obedecendo às seguintes condições:



- Deve calhar num quadrado de cor mais próxima do Trono relativamente à posição onde se encontrava antes da jogada, por exemplo: a peça branca localizada em A5(como se pode observar na Fig. 2 ao lado), pode mover-se até B5, L5 e qualquer quadrado nessa linha; assim como também pode mover-se para B6 até H12 ou para B4 a D2. Mas os quadrados I13, M5 e E1 não estão disponíveis, tal como qualquer quadrado da linha A.
- Não pode cruzar-se com ou ocupar o mesmo quadrado que outra peça, quer seja da sua equipa ou da equipa contrária.
- Pode cruzar o Trono desde que este esteja vazio.
- Não pode, sob qualquer circunstância, ocupar o Trono.
- Os Reis não se movem, apenas ocupam o Trono.

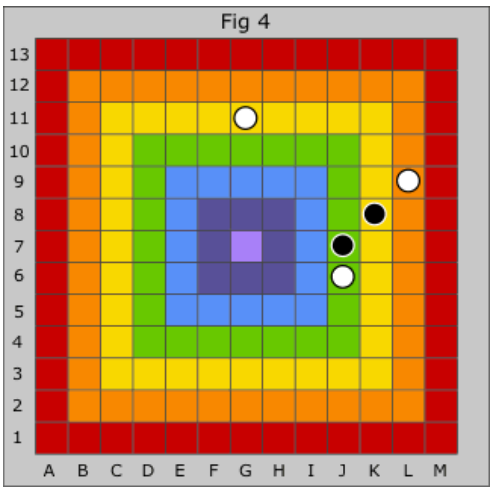
A captura no jogo é feita quando, durante uma jogada, a peça é movida para um quadrado ao lado de uma peça da equipa contrária e cujo "vizinho" no sentido oposto, ou perpendicularmente ou diagonalmente, pertence à mesma equipa da peça que está a ser movida.

Na Fig. 3 a peça preta em D1 poderia capturar a peça branca em B4 se se movesse para B3, ou a peça em E3 se movesse para D2. Se a peça se movesse para I6, seis peças brancas iriam ser capturadas de uma só vez!



- As peças capturadas mudam para a cor da equipa que a capturou.
- A captura deve ser o efeito imediato de um movimento ativo por parte do captor, por isso as peças localizadas na faixa vermelha são sempre imunes à captura.

- Uma peça pode ser colocada, sem perigo de ser capturada, entre duas peças da equipa contrária que já lá se encontravam. Por exemplo, na Fig.3, a peça branca em E3 poderia, se o jogador assim o quisesse, mover-se para E10, em segurança.
- As capturas são realizadas como parte da jogada que lhe deu origem e qualquer captura decorrente de um movimento deve ser efetuada.
- Mesmo que uma peça capturada, em virtude da mudança de cor, o que numa situação normal seria uma captura, aqui não pode acontecer. Capturas secundárias não são permitidas. Na Fig. 4 o movimento da peça branca de G11 para J8, leva à captura da peça preta em J7, no entanto essa mudança de cor na peça em J7 não permite a captura da peça preta em K8.



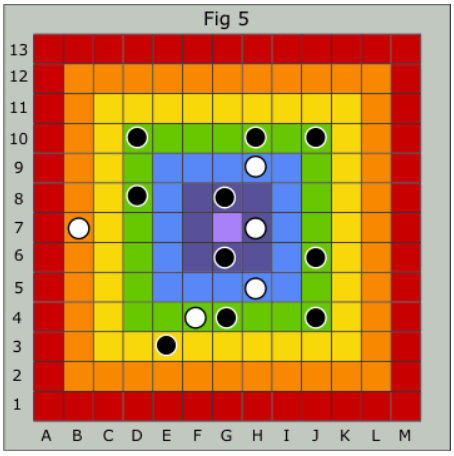
Um quadro é uma combinação de quatro peças, todas pertencentes à mesma equipa, que formam os vértices de um quadrado perfeito centrado no Trono.

Um quadro pode ser completado numa jogada de várias maneiras:

- Quando o vértice que falta é preenchido pela peça que está a ser movida nessa jogada, ou
- Quando o vértice de cor contrária é convertido devido à sua captura, ou
- Ambas as situações acima descritas.

Assim, na Fig. 5, a peça preta em E3 move-se para D4 e um quadro fica completo no canto verde. No entanto, se o movimento fosse de E3 para E4, completa também, através da captura, um quadro diferente e menos óbvio também na cor verde. E ainda, se a peça branca em B7 for movida para F7, forma-se um quadro na cor Índigo, adicionando três vértices de uma só vez.

Como na captura, o completar um quadro deve ser o resultado imediato de uma jogada, assim, não podem ser formados quadros na faixa vermelha.



Quando uma jogada completa um quadro, o jogador que a fez deve, como parte da mesma jogada, ocupar o Trono, retirando de lá o Rei da equipa oposta, se este lá estiver.

Quando o Rei do jogador que completa um quadro já se encontra no Trono, não há alterações.

O jogo acaba quando nenhum dos jogadores, no seu respetivo turno, tem jogadas possíveis ou por acordo entre ambos os jogadores.

Representação do estado do jogo

Internamente, o tabuleiro de jogo será representado por uma lista de listas. Cada peça é representada por um átomo:

'.'- se não existir peça.

'o', ou 'x' – peça normal consoante o jogador.

'O' ou 'X' – rei, consoante o jogador.

Movimentos

gen_board(+Matrix, -Board) – geração aleatória das peças

valid_move(+Matrix, ?Origin, ?Destination, +Team) – geração ou verificação de jogadas possíveis

move_pattern(+Origin, ?Destination, +Size, ?Direction) – jogadas possíveis na direcção Direction, sem contar com outras peças nem proximidade ao centro

unblocked(+Matrix, +Origin, +Destination, +Direction) – verificação que o caminho de Origin até Destination está livre.

distance_to_center(+Size, +Position, -Level) – cálculo da camada em que se encontra uma célula.

commit_move(+Matrix, +Origin, +Destination, +Team, -NewMatrix) - efetua uma jogada no tabuleiro

calulate_captures (+Matrix, +Size, +MoveDestination, +MoveTeam, +ModifiedAccumulator, -ModifiedCells,) – calcula as capturas associadas a uma jogada

swap_pieces(+Matrix, +ToSwap,+Team, -NewMatrix) – efetua as capturas calculadas

throne(+Matrix, +ModifiedCells, -NewMatrix) – se tiver sido formado um quadrado, é posto o rei do jogador no centro

process_pos(+Pos, -X, -Y) – processa um átomo para coordenadas x e y (ex: 'b11' -> x=1, y=10)

Visualização do tabuleiro de jogo em modo texto

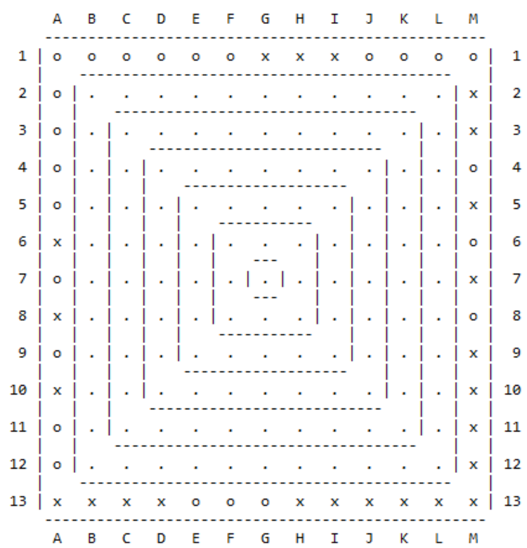
print_board(+Board) – impressão do tabuleiro no ecrã.

Aqui fica o seu predicado que imprime cada caracter da string até encontrar um "."e aí imprime a próxima peça:

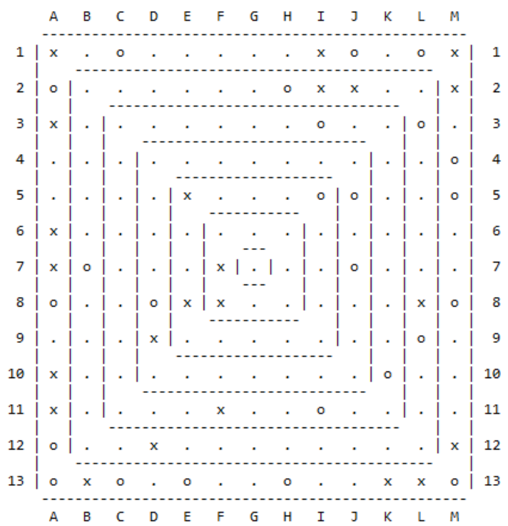
```
board_format("    A  B  C  D  E  F  G  H  I  J  K  L  M\n  ---------------------------------------------
----- \n 1 | .  .  .  .  .  .  .  .  .  .  .  .  . | 1\n  |   ----------------------------------------    |\n 2 | . |
.  .  .  .  .  .  .  .  .  .  . | . |  2\n  |  |   ----------------------------------   |   |\n 3 | . | . | .  .  .  .  .  .
.  .  .  . | . | . |  3\n  |   |   |   ---------------------------   |   |   |\n 4 | . | . | . | .  .  .  .  .  .  .  . | . | .
| . |  4\n  |   |   |   |   -------------------   |   |   |   |\n 5 | . | . | . | . | .  .  .  .  .  . | . | . | . | . |  5\n
|   |   |   |   |   -----------   |   |   |   |   |\n 6 | . | . | . | . | . | .  .  . | . | . | . | . | . |  6\n   |   |   |   |
|   |   ---   |   |   |   |   |   |\n 7 | . | . | . | . | . | . | . | . | . | . | . | . | . |  7\n   |   |   |   |   |   |   ---
|   |   |   |   |   |\n 8 | . | . | . | . | . | .  .  . | . | . | . | . | . |  8\n   |   |   |   |   |   -----------   |   |   |
|   |\n 9 | . | . | . | . | .  .  .  .  .  . | . | . | . | . |  9\n   |   |   |   |   -------------------   |   |   |   |\n10
| . | . | . | .  .  .  .  .  .  .  . | . | . | . | 10\n   |   |   |   ---------------------------   |   |   |\n11 | . | . | .
.  .  .  .  .  .  .  .  . | . | . | 11\n   |   |   ----------------------------------   |   |\n12 | . | .  .  .  .  .  .  .  .
.  .  .  . | . | 12\n   |   ----------------------------------------   |\n13 | .  .  .  .  .  .  .  .  .  .  .  .  . |
13\n   ---------------------------------------------------\n    A  B  C  D  E  F  G  H  I  J  K  L  M    ").
```

```
print_board_3([],[]):- print('\n'),print('\n').

print_board_3(Format, [[]|Rest]):-  !, print_board_3(Format, Rest).

print_board_3([46|FormatRest], [[E | RestOfLine]|Rest]):-  print_cell(E),
print_board_3(FormatRest, [RestOfLine|Rest]).


print_board_3([FormatFirst| FormatRest], Board):- FormatFirst \= 46, format("~1c",
[FormatFirst]), print_board_3(FormatRest, Board).

print_board(Board):- board_format( F ), print_board_3(Format, Board).
```

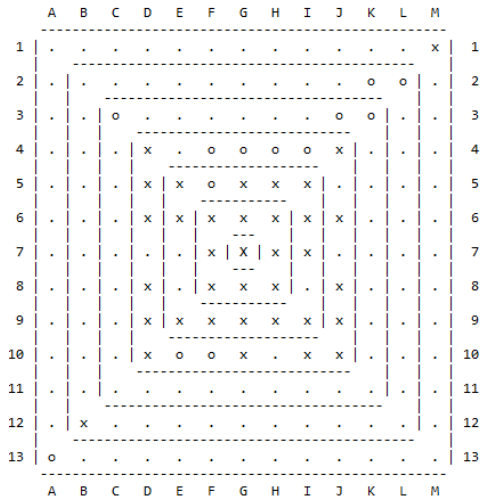
Estado Inicial:



Estado Intermédio:



Estado Final:



Lista de jogadas válidas

possible_moves(**Board**, **Team**, **List**)**:-** findall([**I**,**F**], (matrix_get(Board, **Xi**, **Yi**, Team), I = [Xi, Yi], valid_move(Board, I, F, Team), I \= F), List)**.**

valid_move(**Matrix**, [**X1**, **Y1**], [**X2**, **Y2**], **Team**)**:-** matrix_get(Matrix, X1, Y1, Team),
matrix_height(Matrix, **Size**),
move_pattern([X1, Y1],[X2, Y2], Size, **P**),
unblocked(Matrix, [X1, Y1], [X2, Y2], P),
distance_to_center(Size, [X1, Y1], **L1**),
distance_to_center(Size, [X2, Y2], **L2**),
L1 > L2, L2 =\= *0***.**

Uma jogada é válida para um determinado jogador se este tiver uma das suas peças na posição de origem, se o movimento tiver num dos padrões possíveis (vertical, horizontal e diagonais), se o caminho até a posição de destino estiver livre e se a camada de fim for mais interior que a de início.

Execução de jogadas

commit_move(**Matrix**, **Move**, **Team**, **NewM**)**:-** no_piece(**None**),
Move = [[**X1**, **Y1**], [**X2**, **Y2**]],
matrix_set(Matrix, X1, Y1, None, **Temp1**),
matrix_set(Temp1, X2, Y2, Team, **Temp2**),
matrix_height(Matrix, **Size**),
calculate_captures(Temp2, Size, [X2, Y2], Team, [], **Modified**),
swap_pieces(Temp2, Modified, Team, **Temp3**),
throne(Temp3, Size, [[X2, Y2]|Modified], Team, NewM)**.**

Para executar uma jogada, assumindo que esta é válida, é alterada a posição da peça da posção inicial para a final, são efetuadas as capturas e finalmente é posto o rei no meio do tabuleiro, caso tenha sida cumprida essa condição.

Avaliação do Tabuleiro

evaluate_board(**Board**, **Team**, **Score**)**:-** findall([**Xi**,**Yi**], (matrix_get(Board, Xi, Yi, Team)), **List**),
length(List, **N**), current_winner(Board, **Winner**),
swap_piece(Team, **Opponent**),
if(Winner=Team, **WinScore** = *1*, if(Winner = Opponent, **WinScore** = *-1*, **WinScore** = *0*)),
count_2_3(Board, Team, List, _, **Count2**, **Count3**),
count_2_3(Board, Opponent, List, _, **OCount2**, **OCount3**),
C2 is Count2 - OCount2,
C3 is Count3 - OCount3,
Score is (N + *50**WinScore + C2 * *5* + C3 * *10*)**.**

Na avaliação do tabuleiro, é tida conta o número de peças de cada jogador, o rei que está atualmente no tabuleiro, e a contagem de quadros parciais já formados por cada jogador.

Final do jogo

game_not_over(**Board**, **Team**)**:-** matrix_get(Board, **Xi**, **Yi**, Team), valid_move(Board, [Xi, Yi], [_Xf, _Yf], Team)**.**

O jogo termina quando o jogador cuja vez é de jogar não tem mais jogadas possível.

current_winner(**Matrix**, **Team**)**:-** matrix_height(Matrix, **S**), **S2** is S // *2*, matrix_get(Matrix, S2, S2, **Piece**), if(no_piece(Piece), Team='none', king_of(Piece, Team))**.**

Nessa situação, o vencedor é aquele que tem o rei no meio do tabuleiro.

Jogadas do Computador

decide_move('Random AI', **Board**, **Team**, **Move**)**:-** possible_moves(Board, Team, **List**), list_length(List, **N**), random(*0*, N, **R**), list_get(List, R, Move)**.**

decide_move('Greedy AI', **Board**, **Team**, **Move**)**:-** min_max_move(Board, Team, *1*, Move)**.**
decide_move('Min Max With Alpha Beta Pruning', **Board**, **Team**, **Move**)**:-** min_max_move_alpha_beta(Board, Team, *2*, Move)**.**

Foram implementadas três dificuldades para o computador. Na primeira, são gerados todas as jogadas possíveis e é escolhida uma aleatoriamente. Na segunda, usando o algoritmo de min max com profundidade de 1, é escolhida a jogada que resulta num tabuleiro com melhor pontuação. Finalmente, as otimizações inerentes à podagem alfa-beta permitem atingir uma profundidade maior na árvore, o que resulta numa melhor consideração dos vários aspetos do jogo num intervalo de tempo igual e numa inteligência artificial com maior probabilidade de vencer.

Interface com o utilizador

O utilizador pode iniciar o jogo escrevendo “morelli.” na consola. Seguidamente é pedido para introduzir o modo de jogo para cada equipa.

```
Select Mode for Team x
1 :- Human
2 :- Random AI
3 :- Greedy AI
4 :- Min Max With Alpha Beta Pruning
|:
```

O utilizador deve escolher cada opção consoante o número que é impresso na consola.

No modo de jogador humano, cada jogada deve ser introduzida do seguinte modo:

```
Playing: x

|: a1.
|: b2.
```

Indicando que A1 é a posição da peça a mover e B2 é a posição para a qual a peça deve ser movida. Caso a jogada seja inválida, é pedido ao utilizador que a reintroduza.

Conclusões

Este trabalho permitiu adquirir um conhecimento aprofundado da linguagem Prolog que não seria possível unicamente com os exercícios das aulas. Serviu também como motivo para aprendizagem de alguns algoritmos de inteligência artificial como o MiniMax e a podagem alfa-beta. O trabalho poderia ser melhorado, permitindo ao utilizador escolher a profundidade da árvore de jogo a considerar, já que tal não foi feito devido aos extraordinários tempos de processamento que tal obriga. Outro aspeto de melhoramento seria permitir ao utilizador mudar o tamanho do tabuleiro de jogo, no entanto foi dada prioridade à representação clara do tabuleiro com o tamanho *default*.

Bibliografia

- http://mindsports.nl/index.php/arena/morelli/
- https://boardgamegeek.com/boardgame/103830/morelli
- https://chessprogramming.wikispaces.com/Alpha-Beta