

# Trench

## Relatório Intercalar



Mestrado Integrado em Engenharia Informática e  
Computação

Programação em Lógica

### **Grupo Trench\_1 :**

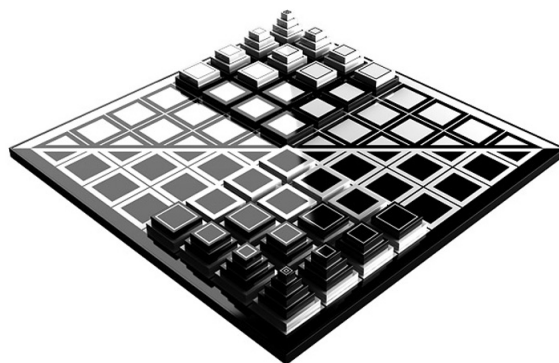
Kevin Amorim - 201207231

Luís Magalhães - 201207224

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

8 de Novembro de 2014

# 1 O Jogo



Trench é um jogo de tabuleiro criado em Portugal por Rui Alípio Monteiro, em 2013. O jogo de tabuleiro para 2 jogadores baseia-se na guerra de trincheiras da 1ª Guerra Mundial.

Os algoritmos aplicados no jogos seguem os princípios referidos no livro "The Art of the War", de Sun Tzu.

## 1.1 Objetivos

O objetivo do jogo é capturar todas as peças inimigas. No entanto, nem sempre é possível que tal aconteça (pelas limitações das peças e do tabuleiro), pelo que a vitória ou a derrota regem-se por um sistema de pontuação, explicado em baixo.

## 1.2 Modo de Jogo

O jogador que possuir as peças de cor preta inicia o jogo. Cada jogador tem direito a uma jogada por vez. O jogo termina ao fim de 25 jogadas se nenhuma peça for capturada nesse intervalo.

### 1.3 Tabuleiro de Jogo

O tabuleiro representa o campo de batalha, com as respectivas trincheiras. Este tem uma forma em diamante (losango), dividido por uma linha diagonal, que representa a linha das trincheiras (cada metade do tabuleiro tem uma cor predominante: preto para um jogador, branco para o outro). O tabuleiro é constituído por 64 casas (8x8), divididas em dois territórios opostos.

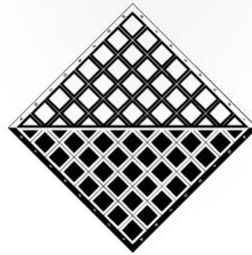


Figura 1: Tabuleiro de jogo 8x8

### 1.4 Peças do Jogo

As peças do jogo são as apresentadas na seguinte tabela:

Postos	Pretas		Brancas		Nº de Peças
General					<i>x 1</i>
Coronel					<i>x 2</i>
Capitão					<i>x 3</i>
Sargento					<i>x 4</i>
Soldado					<i>x 6</i>

Peças

Figura 2: Tabel das peças do jogo

Estas possuem uma forma em losango, inspirada nas estrelas usadas pelos soldados em batalha e simbolizam a hierarquia militar em pirâmide.



Figura 3: Peça do jogo Trench

As peças seguem a seguinte hierarquia (estando no topo o de mais alto nível):

1. General
2. Coronel
3. Capitão
4. Sargento
5. Soldado

#### 1.4.1 Disposição das Peças

As peças são dispostas no tabuleiro de forma a simular a formação de um exército Romano, em diamante, como se poder ver na Fig. 5. O general, a peça com maior ranking na hierarquia, fica no topo da metade aliada do tabuleiro, atrás de todo o exército. Na linha da frente ficam os 6 soldados (possuem o menor ranking dentro da hierarquia).

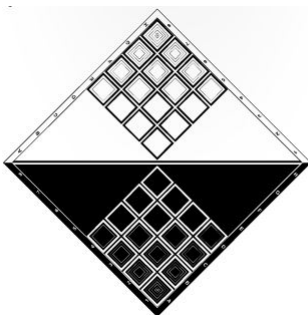


Figura 4: Disposição das peças do jogo

### 1.4.2 Movimento das Peças

- **Soldado:** 1 casa (na diagonal em qualquer direção);
- **Sargento:** 2 casas (na diagonal em qualquer direção e para a frente);
- **Capitão:** 3 casas (na diagonal em qualquer direção e tanto para a frente como para trás);
- **Coronel:** 4 casas (na diagonal em qualquer direção, para frente, esquerda e direita - mas não para trás);
- **General:** 5 casas (em todas as direções);





		<b>Soldado</b>
		<b>Sargento</b>
		<b>Capitão</b>
		<b>Coronel</b>
		<b>General</b>

Figura 5: Representação do movimento das peças do jogo

### 1.4.3 Restrições no Movimento das Peças

- O jogador é sempre obrigado a movimentar uma peça, na sua vez.
- Nenhuma das peças de jogo pode avançar sobre outra peça (seja do seu ou do exército adversário);
- Para capturar uma peça inimiga, a peça do jogador passa a ocupar a parcela quadrada da peça do adversário (terminando o movimento do jogador imediatamente).
- Nenhuma peça é obrigada a percorrer a totalidade das casas que pode percorrer. Isto é, por exemplo, o General pode-se mover apenas 1 casa ou 5 casas, conforme o jogador quiser.

## 1.5 A Trincheira

A trincheira é representada pela linha horizontal no centro do tabuleiro, como referido anteriormente. As peças nesta linha usufruem de um conjunto de vantagens e desvantagens:

- **1ª Vantagem:** Uma peça na trincheira não pode ser atacada por uma peça adversária.
- **2ª Vantagem:** Uma peça na trincheira não é obrigada a parar quando ataca uma peça adversária. Portanto, essas podem continuar o seu movimento ou até atacar outras peças, até concluir a sua totalidade de casas a movimentar, ou o jogador decidir parar.
- **1ª Restrição:** Uma peça em trincheira não pode capturar peças adversárias que se encontrem no território aliado (retaguarda). Mas podem ser atacadas por peças adversárias que se encontrem no território aliado, sendo, aliás, a única forma para capturar peças inimigas na trincheira.
- **2ª Restrição:** O Coronel e o General podem-se movimentar ao longo de toda a linha da trincheira, mas não podem capturar nenhuma peça adversária que aí se encontre, ficando com os seus movimentos limitados.

## 1.6 Fim do Jogo

O jogo termina ao fim de duas partidas. Depois de cada partida os jogadores trocam de lado, para que cada um jogue uma vez com as brancas e outra com as pretas. Uma partida termina quando algum jogador capturar todas as peças adversárias. No entanto, se após 50 jogadas ninguém capturar todas as peças adversárias o jogo termina e procede-se a contagem de pontos, segundo a seguinte tabela:













  Soldado	 1 Estrela = 2 Pontos
  Sargento	 2 Estrelas = 4 Pontos
  Capitão	 3 Estrelas = 6 Pontos
  Coronel	 4 Estrelas = 8 Pontos
  General	 5 Estrelas = 10 Pontos

Figura 6: Valor de cada peça do jogo

No final das duas partidas somam-se os pontos feitos por cada jogador em cada partida e ganha o jogo o jogador com mais pontos. Em caso de empate joga-se outra partida e ganha o jogo o jogador que conseguir capturar 40 pontos primeiro.

## 2 Representação do Estado do Jogo

A representação do estado do jogo irá ser feita através de uma lista de listas (matriz). A matriz irá ser composta por uma lista contendo 15 listas de diferentes tamanhos, para se poder representar a forma em diamante do tabuleiro. Portanto, a primeira e a última lista serão compostas por apenas 1 elemento, enquanto que a oitava lista, a do meio, correspondente à trincheira, será composta por 8 elementos.

Cada peça do jogo será representado pelo seguinte átomo:

- **Soldado:** So
- **Sargento:** Sa
- **Capitão:** Ca
- **Coronel:** Co
- **General:** G
- **Espaço Vazio:** E

A cada uma destas peças (excepto as que representam o espaço vazio) será acrescentado um sufixo com o número do jogador. Por exemplo, para o jogador 1, um soldado será representado por: 'So1'.

Para distinguir, no tabuleiro, a metade branca da metade preta, assume-se que a primeira será sempre a metade superior do tabuleiro (logo, a metade do jogador 1), sendo que a metade inferior será a preta (pertencente ao jogador 2).

Em Prolog a representação da matriz de jogo será a seguinte (vários casos):

### Estado inicial

```
gameList( [ [g1], [co1, co1], [ca1, ca1, ca1], [sa1, sa1, sa1, sa1], [e, so1, so1, so1, e], [e, e, so1, so1, e, e], [e, e, e, so1, e, e, e], [e, e, e, so2, e, e, e], [e, e, so2, so2, e, e], [e, so2, so2, so2, e], [sa2, sa2, sa2, sa2], [ca2, ca2, ca2], [co2, co2], [g2] ]).
```

### Estado final com todas as peças conquistadas (exemplo)

```
gameList( [ [e], [co1, co1], [ca1, ca1, e], [sa1, e, sa1, sa1], [e, so1, e, so1, e], [e, e, so1, so1, e, e], [e, ca1, e, e, e, e, e], [e, e, e, e, g1, e, e], [e, e, sa1, e, e, e], [e, e, e, e, e], [e, e, e, so1], [e, e, e], [e, e], [e] ]).
```

### Estado intermédio (exemplo)

```
gameList( [ [e], [co1, co1], [ca1, ca1, e], [sa1, e, sa1, sa1], [e, so1, e, so1, e], [e, e, so1, so1, e, e], [e, ca1, e, e, e, e, e], [e, so2, e, e, g1, e, e], [e, e, sa1, e, e, e], [ca2, e, e, e, e], [e, e, e, so1], [e, g2, e], [e, e], [e] ]).
```



### 3 Visualização do Tabuleiro

A representação do jogo na consola, em Prolog, está implementada da seguinte forma:

```
print_board([], _).

// print_board(GameList, LineIndex)
print_board([H|T], I) :-
    I \= 7, <— trench_line
        I2 is I * 2,
        S is abs(14 - I2),
        print_spaces(S),
        print_board_left_ref(I), <— left_rule
        write(' '),
        print_board_line(H),
        write(' '),
        print_board_right_ref(I), <— right_rule
        nl,
        I1 is I + 1,
        print_board(T, I1);
    I2 is 7 * 2,
    S is abs(14 - I2),
    print_spaces(S),
    write(' '),
    print_board_trench(H),
    nl,
    I1 is 7 + 1,
    print_board(T, I1).

// Prints the actual game line
print_board_line([]).

// Parameters: Game line to print (list)
print_board_line([H]) :-
    get_board_symbol(H, S),
    write(S).

print_board_line([H|T]) :-
    get_board_symbol(H, S),
    write(S),
    write(' '),
    print_board_line(T).
```

### 3.1 Visualização do Tabuleiro (continuação)

```

print_board_trench ([ ]).

print_board_trench ([H]) :-
    get_board_symbol(H, S),
    write(S).

print_board_trench ([H|T]) :-
    get_board_symbol(H, S),
    write(S),
    format('~c', [215]),
    print_board_trench(T).

print_board_header(_) :-
    print_spaces(16),
    write('a'),
    print_spaces(3),
    write('i'), nl.

print_board_footer(_) :-
    print_spaces(16),
    write('p'),
    print_spaces(3),
    write('h'), nl.

```

```

      a i
      b (5) j
      c (4) (4) k
      d (3) (3) (3) l
      e (2) (2) (2) (2) m
      f - (1) (1) (1) - n
      g - - (1) (1) - - o
      h - - - (1) - - - p
      - x - x - x - x - x - x -
      i - - - [1] - - - a
      j - - [1] [1] - - b
      k - [1] [1] [1] - c
      l [2] [2] [2] [2] d
      m [3] [3] [3] e
      n [4] [4] f
      o [5] g
      p h

```

Figura 7: Output para a consola em Prolog

## 4 Movimentos

```
// movePiece(GameList, [From], [To], NewGameList).
move_piece(L, [X1, Y1], [X2, Y2], NL) :-
    convert_alpha_num(X1, R1),
    convert_alpha_num(Y1, C1),
    convert_alpha_num(X2, R2),
    convert_alpha_num(Y2, C2),
    get_piece(L, [R1, C1], P),
    can_move(L, [X1, Y1], [X2, Y2]), % Verifies if the move can be done
    set_piece(L, e, [R1, C1], L1),
    set_piece(L1, P, [R2, C2], NL).

// getPiece(GameList, [Row, Column], Piece).
//      Row and Columns must be in:      [1, 2, 3, 4, 5, 6, 7, 8]
get_piece(L, [R,C], P) :-
    convert_to_grid_pos(R, C, Row, Col),
    select_elem(Row, Col, L, P).

// setPiece(GameList, Piece, [Pos], NewGameList)
set_piece(L, P, [R, C], NL) :-
    convert_to_grid_pos(R, C, Row, Col),
    nth1(Row, L, X),
    replace(X, Col, P, Res),
    replace(L, Row, Res, NL).

// Predicate that checks if a given piece can move from a pos to another.
// canMove(GameList, [From], [To])
can_move(L, [R1, C1], [R2, C2]) :-
    get_piece(L, [R1, C1], PI),
    get_distance([R1, C1], [R2, C2], DIST),
    max_distance_for(PI, MAX),
    DIST < (MAX + 1),          <— Distance verification
    get_direction(L, [R1, C1], [R2, C2], DIR),
    get_allowed_dir_for(PI, DIR).
```

## 5 Caso de fim do Jogo

```
game_over(GameList) :-
    \+ player_has_pieces(GameList, p1),
        write('Player 1 has no pieces left. '), nl,
        write('Player 2 wins '), !, nl;
    \+ player_has_pieces(GameList, p2),
        write('Player 2 has no pieces left. '), nl,
        write('Player 1 wins '), !, nl;
    \+ player_has_moves(GameList, p1, 1, 1),
        write('Player 1 has no moves left. '), nl,
        write('Player 2 wins '), !, nl;
    \+ player_has_moves(GameList, p2, 1, 1),
        write('Player 2 has no moves left. '), nl,
        write('Player 1 wins '), !, nl.

// checks if the player has any piece remaining
player_has_pieces(GameList, Player) :-
    player_pieces(Player, L),
    player_has_pieces(GameList, L).

player_has_pieces([_|_], []) :- fail.

player_has_pieces(GameList, [P|R]) :-
    member_matrix(P, GameList);
    player_has_pieces(GameList, R).
```

## 5.1 Caso de fim do Jogo (continuação)

```
// checks if the player has any pieces with a valid move
player_has_moves(GameList, Player, X, Y) :-
    X < 9, Y < 9,
    get_piece(GameList, [X,Y], P),
    check_piece_player(P, Player),
    %write('> Piece: '), write(P), nl,
    piece_has_moves(GameList, [X,Y], [1,1]).

player_has_moves(GameList, Player, X, Y) :-
    %write('> FAILED'), nl,
    X < 9, !,
        X1 is X + 1,
        player_has_moves(GameList, Player, X1, Y);
    Y < 9, !,
        Y1 is Y + 1,
        player_has_moves(GameList, Player, X, Y1);
    fail.

// checks if a piece has a valid movement to it
piece_has_moves(GameList, [X,Y], [A1,A2]) :-
    A1 < 9, A2 < 9,
    get_piece(GameList, [X,Y], P),
    can_move(GameList, [X,Y], [A1,A2]).

piece_has_moves(GameList, [X,Y], [A1,A2]) :-
    A1 < 9, !,
        B1 is A1 + 1,
        piece_has_moves(GameList, [X,Y], [B1,A2]);
    A2 < 9, !,
        B2 is A2 + 1,
        piece_has_moves(GameList, [X,Y], [A1,B2]);
    fail.
```