



Universidade do Porto

**FEUP** Faculdade de Engenharia

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

*Trabalho Prático nº1*

# *Fields of Action*

Relatório Final

**3MIEIC05 – Fields of Action 1**

Ângelo Daniel Pereira Mendes Moura ...up201303828@fe.up.pt

César Alexandre de Costa Pinho ...up201604039@fe.up.pt

## ***Fields Of Action***

Introdução .....	- 2 -
Descrição do jogo .....	- 3 -
História .....	- 3 -
Detalhes do Jogo .....	- 3 -
Número de Jogadores.....	- 3 -
Tabuleiro.....	- 3 -
Objetivo .....	- 3 -
Ritmo de Jogo .....	- 4 -
Regras .....	- 4 -
Lógica do Jogo.....	- 5 -
Representação do Estado do Jogo .....	- 5 -
Visualização do tabuleiro.....	- 6 -
Predicados: .....	- 6 -
Lista de Jogadas Válidas.....	- 7 -
Predicados: .....	- 7 -
Execução de Jogadas .....	- 8 -
Predicados: .....	- 8 -
Final do Jogo .....	- 9 -
Predicados: .....	- 9 -
Avaliação do Tabuleiro .....	- 9 -
Predicados: .....	- 9 -
Jogada do Computador .....	- 9 -
Predicados: .....	- 9 -
Conclusões.....	- 10 -
Bibliografia.....	- 11 -

# Introdução

Este projeto tem como objetivo a implementação do jogo de tabuleiro “*Fields of Action*” usando a linguagem de programação PROLOG (Programação Lógica).

O código desenvolvido concentrase unicamente na vertente logica do jogo, como tal a sua *interface* é bastante simplificada, usando texto imprimido na consola como forma de comunicar o estado do tabuleiro e do jogo ao jogador.

Sendo este projeto realizado unicamente usando PROLOG, apresenta um desafio único, pois não é muito comum o uso de uma linguagem puramente logica para implementar um jogo de tabuleiro.

# Descrição do jogo

## História

“*Fields of Action*” é um jogo de tabuleiro de estratégia abstrata criado por Sid Sackson em 1982.

Sid Sackson foi um *designer* e colecionador Americano mais conhecido por ser o criador do jogo “*Acquire*”, que desde 2011 foi introduzido, juntamente com o seu criador, na “*Academy of Adventure Gaming Arts & Design's Hall of Fame*”.

“*Fields of Action*” foi inspirado num jogo da autoria de Claude Soucie chamado “*Lines of Action*”, jogo este que Sid Sackson publicitou no seu livro “*A Gamut of Games*” (1969).

## Detalhes do Jogo

### Número de Jogadores

“*Fields of Action*” é jogado por dois jogadores que competem entre si.

### Tabuleiro

“*Fields of Action*” joga-se num tabuleiro 8x8.

Neste tabuleiro há um conjunto de 12 peças para cada jogador, com cores diferentes para distinguir as peças de cada um (por exemplo: Preto e Branco). As peças de cada conjunto estão numeradas de 1 a 12.

O jogo começa com o tabuleiro no estado representado na Figura 1.

### Objetivo

O objetivo do jogo é capturar cinco peças inimigas que estão numeradas sequencialmente, e.g. 7-8-9-10-11 (a ordem da captura não é valorizada). Um jogador também ganha se o seu oponente não conseguir realizar nenhum movimento considerado legal.

	A	B	C	D	E	F	G	H	
8	8	7	6	5					8
7					12	11	10	9	7
6	4	3	2	1					6
5									5
4									4
3					1	2	3	4	3
2	9	10	11	12					2
1					5	6	7	8	1
	A	B	C	D	E	F	G	H	

Figura 1 - Tabuleiro inicial

## Ritmo de Jogo

Começando com o jogador que controla as peças de cor Preta, os jogadores alternam turnos até um completar o objetivo, sendo esse jogador considerado o vencedor. Ou até um jogador não conseguir realizar um movimento válido, sendo considerado o oponente desse jogador o vencedor.

## Regras

1. Apenas uma peça pode ser movida por turno.
2. As peças podem mover-se horizontalmente, verticalmente ou diagonalmente.
3. Uma peça pode ser movida o número de casas igual ao número de peças, de qualquer cor, adjacentes a essa peça, antes do movimento (Figura 2).
4. A peça escolhida pode saltar por cima de peças de qualquer cor.
5. A peça escolhida não pode pousar sobre uma peça da mesma cor.
6. A peça escolhida pousar sobre uma peça de cor inimiga. A peça inimiga é então removida do tabuleiro. Isto é referido como uma captura.
7. Uma peça sem outras adjacentes não tem número limite de casas de movimento (3ª regra), no entanto tem que pousar numa casa onde fique adjacente a pelo menos duas peças de qualquer cor (Figura 3).

# Lógica do Jogo

O jogo é, em termos de código, um ciclo.

Cada iteração do ciclo representa uma jogada, ou seja, o ciclo refere-se alternadamente aos jogadores.

O ciclo começa por verificar se a condição de vitória foi alcançada, ou seja, se o jogo acabou. De seguida calcula o tamanho do tabuleiro, e representa-o no ecrã. Com o tabuleiro visível, o ciclo passa a fase onde é determinada a jogada a realizar, isto pode ser introduzido, caso seja o turno do utilizador, ou calculado caso seja o turno do computador. Com a próxima jogada estabelecida, o ciclo entra na fase onde de facto realiza o movimento. Com o movimento do turno realizado, o ciclo troca de jogador e recomeça.

O ciclo descrito repete-se até a condição de vitória ser alcançada.

## Representação do Estado do Jogo

Para a implementação do jogo em PROLOG, foi necessário escolher a representação de dados a utilizar para o tabuleiro e peças. Assim, foi escolhida uma representação matricial, isto é uma lista de listas, de dimensão 8x8, cujos elementos são elementos termos compostos. As células vazias do tabuleiro são representadas por um elemento composto vazio (0-0), as peças são representadas por termos compostos onde o lado esquerdo indica o número do jogador, 1 ou 2, e o lado direito indica o número da peça em questão, entre 1 e 12, por exemplo 1-6 é a peça número 6 do jogador 1.

O tabuleiro inicial começa com as 12 peças de cada jogador dispostas, como já foi referido.

```
tab([ [1-8, 1-7 ,1-6 ,1-5 ,0-0 ,0-0 ,0-0 ,0-0],
      [0-0, 0-0 ,0-0 ,0-0 ,1-12,1-11,1-10,1-9],
      [1-4, 1-3 ,1-2 ,1-1 ,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, 0-0 ,0-0 ,0-0 ,2-1 ,2-2 ,2-3 ,2-4],
      [2-9, 2-10,2-11,2-12,0-0 ,0-0 ,0-0 ,0-0],
      [0-0, 0-0 ,0-0 ,0-0 ,2-5 ,2-6 ,2-7 ,2-8]]).
```

Figura 4 - Representação do tabuleiro inicial

```
tab([ [0-0, 1-7 ,1-8 ,1-5 ,0-0 ,0-0 ,0-0 ,0-0],
      [0-0, 1-3 ,0-0 ,0-0 ,0-0 ,1-11,1-10,0-0],
      [0-0, 0-0 ,0-0 ,0-0 ,0-0 ,0-0 ,1-6 ,0-0],
      [0-0, 1-4 ,1-2 ,0-0 ,0-0 ,0-0 ,0-0 ,0-0],
      [0-0, 2-10,0-0 ,0-0 ,0-0 ,0-0 ,2-5 ,0-0],
      [0-0, 0-0 ,0-0 ,0-0 ,0-0 ,2-2 ,2-6 ,2-4],
      [0-0, 0-0 ,0-0 ,0-0 ,0-0 ,0-0 ,0-0 ,2-7],
      [0-0, 0-0 ,0-0 ,2-11,0-0 ,0-0 ,0-0 ,2-8]]).
```

Figura 5 - Representação de um possível tabuleiro intermédio

```
tab([ [1-8, 2-12,1-6 ,0-0 ,0-0 ,0-0 ,0-0 ,0-0],
      [0-0, 0-0 ,1-5 ,0-0 ,0-0 ,0-0 ,0-0 ,0-0],
      [0-0, 0-0 ,0-0 ,0-0 ,1-1 ,0-0 ,0-0 ,0-0],
      [1-4, 0-0 ,0-0 ,0-0 ,0-0 ,0-0 ,1-11,0-0],
      [0-0, 0-0 ,2-9 ,0-0 ,2-8 ,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 ,2-11,2-10,0-0 ,0-0 ,0-0 ,0-0],
      [0-0, 1-9 ,0-0 ,0-0 ,0-0 ,2-6 ,2-7 ,0-0]]).
```

Figura 6 - Representação de um possível tabuleiro final

## Visualização do tabuleiro

O *display* do tabuleiro de jogo na consola é da responsabilidade de um predicado que percorre a estrutura de dados que armazena o tabuleiro e imprime-o. Esse predicado, baseado na recursividade, percorre cada linha do tabuleiro invocando outro predicado que percorrerá todas as células de uma linha imprimindo um símbolo de jogo por célula, dependendo do valor da mesma.

### Predicados:

- `display_game(Board, Player, Lines, Columns)`

```

*****
****                                ****
****          FIELDS OF ACTION          ****
****                                ****
*****

```

	A	B	C	D	E	F	G	H	
8	B <sub>8</sub>	B <sub>7</sub>	B <sub>6</sub>	B <sub>5</sub>					8
7					B <sub>12</sub>	B <sub>11</sub>	B <sub>10</sub>	B <sub>9</sub>	7
6	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>					6
5									5
4									4
3					W <sub>1</sub>	W <sub>2</sub>	W <sub>3</sub>	W <sub>4</sub>	3
2	W <sub>9</sub>	W <sub>10</sub>	W <sub>11</sub>	W <sub>12</sub>					2
1					W <sub>5</sub>	W <sub>6</sub>	W <sub>7</sub>	W <sub>8</sub>	1
	A	B	C	D	E	F	G	H	

Figura 7 - Tabuleiro inicial

```

--> Jogador 1
    PECAS CAPTURADAS :
Move : C6 -> E8

```

```

*****
****                                ****
****          FIELDS OF ACTION          ****
****                                ****
*****

```

	A	B	C	D	E	F	G	H	
8	B <sub>8</sub>		B <sub>6</sub>			B <sub>5</sub>			8
7						B <sub>11</sub>		B <sub>9</sub>	7
6	B <sub>4</sub>		W <sub>11</sub>						6
5									5
4				B <sub>12</sub>					4
3							W <sub>3</sub>	W <sub>4</sub>	3
2	W <sub>9</sub>								2
1					W <sub>5</sub>	W <sub>6</sub>		W <sub>8</sub>	1
	A	B	C	D	E	F	G	H	

```

--> Jogador 1
    PECAS CAPTURADAS :
    W1
    W2
    W7
    W10
    W12
Move : H7 -> A7

```

Figura 8 - Tabuleiro Intermédio

```

*****
***** FIELDS OF ACTION *****
*****
*****

```

	A	B	C	D	E	F	G	H	
8			B <sub>6</sub>						8
7		B <sub>9</sub>							7
6	B <sub>7</sub>								6
5									5
4									4
3									3
2		W <sub>12</sub>				W <sub>4</sub>		W <sub>8</sub>	2
1						W <sub>6</sub>	W <sub>5</sub>	B <sub>1</sub>	1
	A	B	C	D	E	F	G	H	

```

--> Jogador 2
PECAS CAPTURADAS :
B 2
B 3
B 4
B 5
B 8
B 10
B 11
B 12

Move : F1 -> H1
* Peca 1 capturada *

```

Figura 9 – Tabuleiro Final

## Lista de Jogadas Válidas

O cálculo da lista das jogadas válidas é ligeiramente diferente se for a vez do jogador ou a vez do computador.

Quando é o turno do jogador, é pedido previamente a peça que se deseja mover, calculando-se apenas os movimentos válidos que essa peça pode realizar e apresenta-se a lista ao jogador para este prosseguir a seleção do movimento.

Quando é a vez de o computador realizar um movimento, são calculadas todas as jogadas possíveis de serem realizadas nesse momento.

Outra possível divergência no cálculo de jogadas válidas acontece quando a peça a ser avaliada se encontra isolada no tabuleiro. Como neste caso o movimento das peças é diferente do normal é necessário calcular as jogadas válidas para esta peça de maneira diferente também.

### Predicados:

- `valid_moves(Board, Player, Lines, Columns, ListOfMoves)`  
Predicado que calcula todos os movimentos possíveis para o computador num dado turno chamando o predicado `valid_moves/6` que calcula os movimentos de uma peça individual.
- `valid_moves(Board, Player, Piece, Lines, Columns, ListOfMoves)`  
Predicado que calcula a lista de movimentos possíveis de apenas uma peça (Piece).
- `possible_plays(Player, Board, Piece, Lines, Columns, Plays)`  
Predicado que calcula todos os movimentos possíveis referentes à peça selecionada pelo jogador.

### Principais Predicados Auxiliares:

- `adjacent_pieces(Board, L, C, Lines, Columns, NumAdj):-`  
Calcula o número de peças adjacentes à peça na posição L-C.



- `move_horizontal(Player, Board, LinP, ColP, Columns, NumAdj, MovH) :-`  
Responsável por calcular os movimentos na horizontal.
- `move_vertical(Player, Board, LinP, ColP, Lines, NumAdj, MovV) :-`  
Responsável por calcular os movimentos na vertical.
- `move_diagonal(Player, Board, LinP, ColP, Lines, Columns, NumAdj, MovD) :-`  
Responsável por calcular os movimentos na diagonal.

## Execução de Jogadas

A execução das jogadas é realizada depois de ter sido selecionada uma peça valida e um movimento valido que esta pode realizar. Neste ponto da execução do jogo, é conhecido o valor da peça, onde ela se encontra inicialmente e onde a peça ficará. Sabendo isto, a célula do tabuleiro onde ela se encontra é preenchida de maneira a simbolizar uma célula vazia e a célula onde a peça acaba é preenchida com a informação revelante da peça, nomeadamente o seu controlador e o seu valor.

Caso exista outra peça na posição final, é realizada uma captura. Isto é feito colocando a informação da peça capturada numa lista.

### Predicados:

- `move(Jog, Board, Lines, Columns, L-C-NewL-NewC-Play, NewBoard) :-`

## Final do Jogo

O jogo “*Fields of Action*” acaba quando se encontra uma de duas situações: um dos jogadores capturou cinco peças de valor sequencial ou não existem mais jogadas possíveis.

O primeiro caso é identificado verificando uma lista que contém a informação das peças capturadas até esse ponto do jogo. Caso se verifique que um jogador capturou cinco peças de valor sequencial, o jogo acaba e esse jogador é considerado o vencedor.

O segundo é verificado vendo se existem movimentos válidos no estado de jogo atual, caso não existam o jogo acaba e o jogador que realizou o último movimento é considerado o vencedor.

### Predicados:

- `game_over(Player, Pieces) :-`

## Avaliação do Tabuleiro

A avaliação do tabuleiro é obtida avaliando o número de peças necessárias para obter uma sequência de cinco peças.

Cada possível movimento é passível de ser atribuído um valor, esse valor depende do quanto esse movimento aproxima o jogo de um estado de vitória para o jogador atual. Ou seja, o valor de um movimento é máximo quando esse movimento completa uma sequência de cinco peças capturadas.

### Predicados:

- `value(Board, Player, FirstSeqPiece-LastSeqPiece, Value) :-`

### Principais Predicados Auxiliares:

- `pieces_out_of_game(Board, Player, CaughtPieces) :-`

Retorna as peças que já foram capturadas.

- `check_sequence([X|Pieces], Quant, FirstP-LastP, FirstGreatSeqPiece-LastGreatSeqPiece, GreaterSeq) :-`

Verifica se as peças capturadas formam uma sequência.

## Jogada do Computador

O computador pode seguir dois diferentes métodos de escolha de movimentos.

O primeiro envolve a escolha de um movimento aleatório dos possíveis no momento.

O segundo é uma escolha informada. Neste modo o computador seleciona as jogadas que resultam em captura das peças inimigas. Tendo múltiplos movimentos que resultam em captura o computador escolhe o movimento que melhor lhe garante atingir uma sequência.

### Predicados:

- `choose_move(Board, Player, computer2, ListOfMoves, Move) :-`

Predicado que escolhe aleatoriamente um movimento da lista dos possíveis.

- `choose_move(Board, Player, computer2, ListOfMoves, Move) :-`

Predicado que escolhe o movimento da lista de maneira gananciosa.

### Principais Predicados Auxiliares:

- `choose_best_move(Board, Player, Lines, Columns, [L-C-NewL-NewC-Play|ListOfMoves], Value, Move) :-`

# Conclusões

Sabíamos logo de início que a dificuldade deste projeto não originaria da complexidade do jogo em si, mas sim da linguagem usada para a implementação. PROLOG é uma linguagem muito diferente do que estamos habituados a trabalhar e certas noções e técnicas tiveram de ser reaprendidas, ao mesmo tempo novas precisaram de ser aprendidas.

Um dos maiores contribuidores para a duração do tempo de desenvolvimento do trabalho foi o facto de ser necessário a programação individual de cada caso e situação possível, isto tornou o código bastante extenso e um pouco repetitivo de escrever.

Avaliado o código desenvolvido, sentimo-nos confiantes no nosso trabalho, no entanto temos noção que é possível que haja maneiras melhores de implementar um jogo em PROLOG, maneiras que devido á nossa inexperiência com a linguagem não são visíveis para nos de momento.

No geral, programar em PROLOG mostrou-se um desafio maior do que estávamos á espera inicialmente, mas acreditamos que no geral realizamos um bom trabalho.

# Bibliografia

- <http://www.iggamecenter.com/info/en/foa.html>
- <https://boardgamegeek.com/boardgame/18352/fields-action>
- <https://sicstus.sics.se/sicstus/docs/latest4/html/sicstus.html/index.html#Top>