FEUP – Programação Funcional e em Lógica 2021/2022

TP2 – BreakthroughTanks_1

Diogo Luís Henriques Costa - 50%
up201906731

Francisco José Barbosa Marques Colino - 50%
up201905405

13 de janeiro de 2022

1 Instalação e execução

Para a execução deste jogo é necessário o *software SICStus Prolog 4.7*. Neste, é necessário incluir o ficheiro *main.pl* que se encontra na pasta *src* e executar *play.*, a partir deste ponto o menu do jogo fornecerá as instruções de uso necessárias.

2 Descrição do jogo

BreakthroughTanks é um jogo de tabuleiro de estratégia por turnos. É jogado por 2 jogadores oponentes, cada um controlando um conjunto de peças sobre um tabuleiro quadrangular, de dimensões pares, que podem variar de 6x6 até 26x26. O objetivo do jogo é ser o primeiro a ter uma peça na linha mais distante, ou seja, a casa do oponente.

2.1 Peças

Existem 3 tipos de peças:

1. *Medium tank*
2. *Heavy tank*
3. *Tank destroyer*

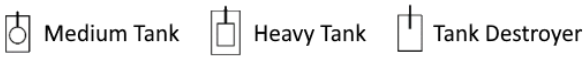


Figura 1: Tipo de peças.

Num tabuleiro 8x8, cada jogador começa com 2 *heavy tanks*, 4 *tank destroyers* e 10 *medium tanks*. A sua disposição inicial no tabuleiro é a seguinte:

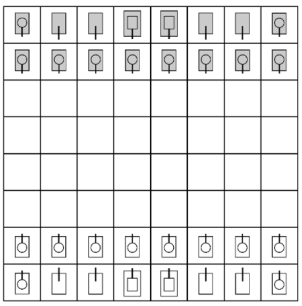


Figura 2: Disposição inicial do tabuleiro.

Em todas as dimensões de tabuleiro, cada jogador fica com as linhas mais próximas de si completas com *tanks*. A 2ª linha mais próxima fica completa com *medium tanks* e a linha mais próxima varia no número de *tank destroyers*, sendo que eles se posicionam em ambos os lados dos *heavy tanks*. Em todas as dimensões o posicionamento e quantidade (2) de *heavy tanks* mantém-se.

2.2 Movimentação simples

Em cada turno uma peça faz uma movimentação simples ou executa uma captura. No caso da movimentação, esta é igual para todas as peças: uma unidade para a frente, ou uma unidade para uma das diagonais da frente.

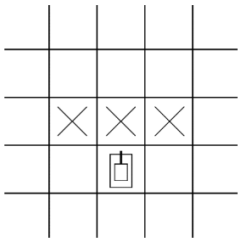


Figura 3: Movimentação.

2.3 Captura

Quando uma peça **A** captura uma peça inimiga **B**, a peça **B** é retirada do tabuleiro e a peça **A** passa a ocupar o lugar anteriormente ocupado por **B**.

1. O *Medium tank* captura da mesma forma que se move.
2. O *Heavy tank* captura 2 casas tanto para a frente como nas diagonais da frente.
3. O *Tank destroyer* captura 2 casas para a frente.

Neste jogo, o jogador não é obrigado a capturar caso seja possível. Ele pode escolher capturar, ou mover a peça em causa de forma normal, ou até mesmo mover outra peça.

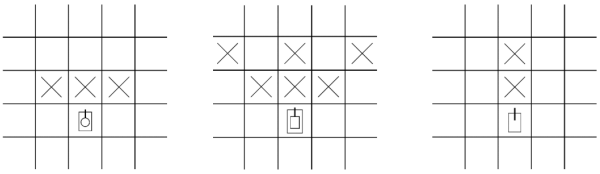


Figura 4: Captura de peças.

2.4 Fim do jogo

O jogo acaba quando um jogador consegue chegar com uma peça sua à linha mais afastada, ou seja, à linha mais próxima do oponente. De notar que, caso um jogador fique sem peças, o seu adversário vence também o jogo.

3 Lógica do jogo

3.1 Representação interna do estado do jogo

O estado do jogo, *GameState*, é representado por: Turn-Board. Por sua vez, Turn pode ser "top"ou "bot"e Board é uma lista de listas de inteiros representando o tabuleiro.

O board é preenchido da seguinte forma:

- 0 – Espaço vazio
- 1 – *Medium Tank* do *bot player*
- 2 – *Heavy Tank* do *bot player*
- 3 – *Tank destroyer* do *bot player*
- -1 – *Medium Tank* do *top player*
- -2 – *Heavy Tank* do *top player*
- -3 – *Tank destroyer* do *top player*

Desta forma, cada tipo de *tank*: *medium tank, heavy tank, tank destroyer*, pode ser obtido recorrendo ao valor absoluto. Por outro lado, as peças de cada jogador distinguem-se pelo sinal: as peças do *top player* têm representação interna negativa enquanto que as peças do *bot player* têm representação positiva.

Exemplos de estados de jogo:

3.1.1 Estado de jogo inicial, 8x8

```
bot-[
        [ -1, -3, -3, -2, -2, -3, -3, -1],
        [ -1, -1, -1, -1, -1, -1, -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],
        [  1,  1,  1,  1,  1,  1,  1,  1],
        [  1,  3,  3,  2,  2,  3,  3,  1]
    ]
```

De modo a obter este estado, recorre-se ao predicado *initial_state(+Size, -GameState)* definido no ficheiro *representation.pl*:

```
% initial_state(+Size, -GameState)
initial_state(Size, bot-Board):-
    between(6, 26, Size),
    even(Size),
    get_initial_board(Size, Board).
```

3.1.2 Estado de jogo intermédio, 8x8

```
top-[
        [  0,  0,  0,  0,  0, -3, -3, -1],
        [  0,  0,  0,  0, -1, -1, -1, -1],
        [ -1,  2,  0,  0,  0,  0,  0,  0],
        [  0,  0,  0,  0, -2,  0,  0,  0],
        [  0,  0,  0,  0,  0,  0,  0,  0],
        [  0,  0,  0,  1,  0,  0,  0,  0],
        [  0,  0,  0,  0,  1,  1,  1,  1],
```

```
        [  1,  0,  0,  0,  0,  3,  3,  1]
]
```

3.1.3 Estado de jogo final, 8x8

```
top-[
        [  2,  0,  0,  0,  0, -3, -3, -1],
        [  0,  0,  0,  0,  0, -1, -1, -1],
        [ -1,  0,  0, -1,  0,  0,  0,  0],
        [  0,  0,  0,  0,  0,  0,  0,  0],
        [  0,  0,  0,  1,  0,  0,  0,  0],
        [  0,  0,  0,  0,  0,  0,  0,  0],
        [  0,  0,  0,  0,  1,  1,  1,  1],
        [  1,  0,  0,  0,  0,  3,  3,  1]
    ]
```

Neste caso o vencedor foi o *bot player* dado que conseguiu alcançar a linha mais próxima do oponente com um dos seus *tanks*.

3.2 Visualização do estado do jogo

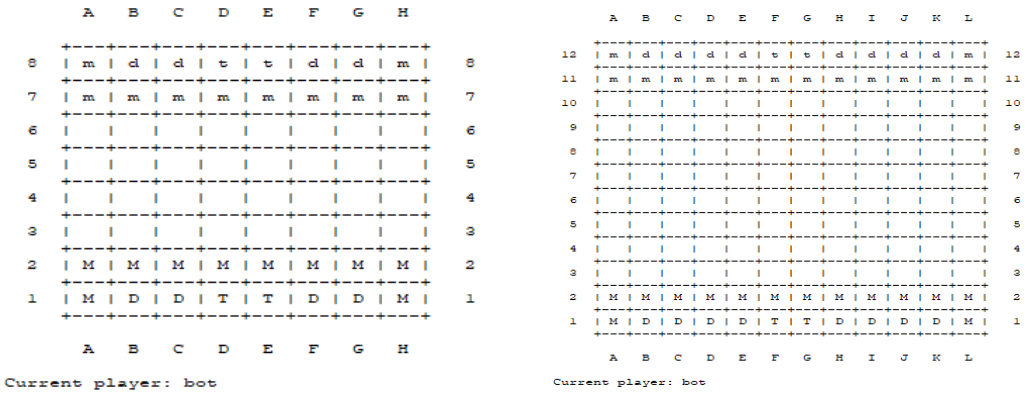
Os predicados de visualização do estado de jogo estão definidos no ficheiro *display.pl*.
As peças são representadas da seguinte forma:

- ' ' – Espaço vazio
- 'M' – *Medium Tank* do *bot player*
- 'T' – *Heavy Tank* do *bot player*
- 'D' – *Tank destroyer* do *bot player*
- 'm' – *Medium Tank* do *top player*
- 't' – *Heavy Tank* do *top player*
- 'd' – *Tank destroyer* do *top player*

O predicado *display_game(+GameState)* é usado para visualizar o estado do jogo, descrito na secção anterior:

```
% display_game(+GameState)
display_game(Turn-Board):-
    print_board(Board),
    write('Current player: '),
    write(Turn),
    nl.
```

Este predicado é flexível uma vez que permite visualizar tabuleiros de diversos tamanhos:



(a) Tabuleiro 8x8.

(b) Tabuleiro 12x12.

Figura 5: Visualização do estado jogo.

O predicado *print_board(+Board)* é usado para visualizar o tabuleiro do jogo sendo a sua definição modular já que ele recorre a predicados tais como:

- *print_board_line(+Line, +Number)*
- *print_board_horizontal_separator(+Size)*
- *print_nav_horizontal(+Size)*

Esta separação de responsabilidades parametrizável com *Size*, permite a flexibilidade quanto ao tamanho.
Quanto ao *input*, os predicados estão definidos no ficheiro *input.pl*, sendo estes os principais:

- *read_move(+Size, -Move)*
- *read_coords(+Size, -ColumnIndex, -RowIndex)*
- *read_valid_column_index(+Size, -ColumnIndex)*
- *read_valid_row_index(+Size, -RowIndex)*
- *read_letter(-LetterCode)*
- *read_number(-Number)*

De notar que a este nível, a validação de *input* é feita quanto ao tipo de caracteres prentendido (letras ou dígitos) e tamanho do *Board*, ou seja, que não se seleciona uma posição fora do tabuleiro. A verificação de validade da jogada é feita ao nível do módulo lógico que valida as jogadas e caso não seja válida, o *backtracking* inerente à execução de *Prolog* faz com que o *user* volte a ter que inserir uma jogada.

De notar que *read_number(-Number)* lê e valida um número, com número indefinido de dígitos, seguido de um *enter* enquanto que *read_letter(-LetterCode)* lê e valida uma letra, seja ela maiúscula ou minúscula, seguida de um *enter*. Isto permite que o utilizador, quando pretende selecionar uma coluna tanto possa escrever com maiúsculas como mínusculas.

3.3 Execução de jogadas

O predicado *move(+GameState, +Move, -NewGameState)* , definido no ficheiro *logic.pl* é o responsável pela validação e execução de uma jogada:

```
% move(+GameState, +Move, -NewGameState)
move(Turn-Board, Move, NewTurn-NewBoard):-
    valid_move(Turn-Board, Move),
```

```
    do_valid_move(Board, Move, NewBoard),
    switch_turn(Turn, NewTurn).
```

A validação de uma jogada, efetuada por *valid_move(+GameState, ?Move)* faz-se da seguinte forma:

1. Obtém-se o elemento na posição inicial
2. Verfica-se se é um *tank* do jogador que está a jogar
3. Obtém-se o elemento na posição final
4. Faz-se a subtração das coordenadas para obter a translação associada
5. Verifica-se se a posição final não tem um *tank* do jogador que está a jogar
6. Verifica-se se a translação do movimento é válida dado o *tank* da posição inicial e o elemnto (*tank* inimigo ou espaço vazio) da posição final

De notar que, com o objetivo de reutilizar código, normalizaram-se as jogadas para o *bot player*. Exemplo disso é o facto de, caso seja o *top player* a jogar, a translação associada ao movimento é refletida horizontalente de modo a utilizar os mesmos predicados para ambos os jogadores. Também os *tanks* são tomados pelo seu valor absoluto (o *bot player* tem os seus *tanks* representados por números positivos).

Após saber que a jogada é válida, para executar uma jogada basta colocar um *0* na posição inicial e colocar na posição final o número (*tank*) que estava na posição inicial recorrendo-se ao predicado *matrix_put_at(+Matrix, +[C,L], +Elem, -ModifiedMatrix)* definido no ficheiro *utils.pl*.

3.4 Final de jogo

O predicado *game_over(+GameState, -Winner)* é usado para a deteção do final do jogo e atribuição do vencedor. Um jogador ganha se o seu oponente não tiver mais peças para jogar, *not(matrix_has_range(Board, positive))* e *not(matrix_has_range(Board, negative))*, ou, no caso do *top*, se tiver uma peça sua na última linha do *board*, a linha mais próxima do inimigo, *min_member(Min, BotRow)*, e no caso do *bot* se tiver uma peça sua na primeira linha do *board*, a linha mais próxima do inimigo, *max_member(Max, TopRow)*.

3.5 Lista de jogadas válidas

O predicado *valid_moves(+GameState, -ListOfMoves)* é usado para obter a lista de jogadas válidas e está definido no ficheiro *logic.pl*:

```
% valid_moves(+GameState, -ListOfMoves)
valid_moves(GameState, ListOfMoves):-
    findall(Move, valid_move(GameState, Move), ListOfMoves).
```

Este predicado tem uma definição curta dado que faz uso do predicado *valid_move(+GameState, ?Move)* (com definição descrita na secção de *Execução de jogadas*) e recorre a *findall(Term, Goal, List)* para obter uma listagem de todas as jogadas válidas.

3.6 Avaliação do Estado do Jogo

A avaliação do estado de jogo, *value(+GameState, +Player, -Value)*, é feita para cada jogador tendo em conta o tabuleiro todo usando os seguintes fatores:

- Tipo de *tank*
  - o *medium tank* vale 100 pontos

  - o *tank destroyer* vale 125 pontos
  - o *heavy tank* vale 150 pontos

- Distância à sua base

O valor atribuído a cada *tank* é calculado com a multiplicação do valor do seu tipo pelo fator da distância que é $2^{dist\_to\_home}$, sendo que a *dist_to_home* é o número de casas a que o *tank* se encontra do jogador ao qual pertence. O valor final do tabuleiro é calculado através da soma do valor de todos os *tanks* do jogador para o qual está a ser calculada a jogada (*top* ou *bot*) e é depois subtraído o valor de todos os *tanks* que pertencem ao seu oponente.

3.7 Jogada do Computador

O jogo oferece dois níveis de dificuldade do computador como inimigo. No nível 1 o computador faz uma jogada aleatória que seja válida, no nível 2 o computador escolhe a jogada que lhe dá mais vantagem, fazendo as previsões numa distância de uma jogada (algoritmo míope). Para este efeito são simuladas todas as jogadas possíveis que o computador pode fazer e é calculado o valor de cada, através do método de avaliação mencionado na secção anterior, e é então escolhida a jogada que obtém um estado de jogo com maior valor, *choose_move(+GameState, +Level, -Move)*, no caso de empate é escolhido um dos melhores resultados aleatoriamente.

4 Conclusões

Foi implementado em *Prolog* um jogo de tabuleiro a 2 jogadores com tamanho de tabuleiro variável. Os jogadores tanto podem ser humanos como *PC* sendo que neste último caso existem dois níveis associados. Assim, todos os objetivos deste trabalho foram alcançados.

Caso se pretendesse desenvolver e adicionar complexidade ao jogo, duas possíveis alterações seriam a possíbilidade de configurar a disposição inicial do tabuleiro e adicionar mais níveis ao jogador *PC* fazendo uso, por exemplo, do algorítmo *minimax*.

A realização deste trabalho permitiu fazer um estudo mais imersivo de *Prolog* que não seria possível realizando apenas os exercícios das aulas práticas.

5 Bibliografia

5.1 Página do jogo

https://boardgamegeek.com/boardgame/321224/breakthrough-tanks