Breakthru

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Breakthru_1:
João David Gonçalves Baião - 201305195
Pedro Vieira de Castro - 201305337

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

8 de Novembro de 2015

Indice

1 Introdução

Como 1º trabalho prático da Unidade Curricular de Programação em Lógica do MIEIC foi nos proposto implementar um jogo de tabuleiro em PROLOG.

Para além desta implementação foi também proposta a criação de uma ferramenta que permitisse o computador jogar com certas regras com o objetivo de criar uma inteligência artificial que conseguisse ser competitiva com um utilizador humano.

O jogo que nós escolhemos foi o *Breakthru*. O *Breakthru* é um jogo estratégico para dois jogadores. O que distingue este jogo é o facto dos dois jogadores terem objetivos no jogo diferentes. Isto permite ao jogo ser mais vezes jogado sem se tornar repetitivo.

A principal dificuldade do trabalho foi a implementação de uma inteligência que conseguisse calcular a melhor jogada possível num turno tendo em conta que seria possível fazer dois movimentos. Este problema será mais aprofundado na secção da avaliação ao tabuleiro.

2 O Jogo

Como foi dito anteriormente, o Breakthru é um jogo de tabuleiro estratégico para dois jogadores. O tabuleiro tem 10x10 de dimensão.

Cada jogador vai ter o seu objetivo no jogo. O jogador *Yellow* têm que mover a sua peça mais importante (*MotherShip)* para fora do tabuleiro enquanto o outro jogador (*Grey)* tenta impedir que isso aconteça e tenta capturar a *MotherShip.*

Cada jogador vai ter à sua disposição uma frota de 12 +*MotherShip* ou 20 embarcações, para os jogadores *Yellow* e *Grey* respectivamente. Às embarcações é permitido que elas se movimentem em linha recta em qualquer direção o número de posições que o jogador desejar desde que não ocupe um espaço já ocupado ou ultrapasse outra embarcação, quer seja amigável ou não.

Ainda é possível também que as embarcações consigam capturar embarcações inimigas. Só o poderão fazer se o movimento de captura for na direção obliqua e num quadrado adjacente ao da embarcação que irá fazer a captura. Se for feita a captura, a embarcação capturada irá ser retirada do tabuleiro e a embarcação que fez a captura irá tomar o seu lugar. Esta técnica de captura é semelhante à captura do peão no Xadrez no entanto é possível em qualquer direção desde que seja obliqua.

No entanto o número de jogadas possíveis de fazer num turno vai depender das jogadas que forem feitas. De forma a facilitar tanto a compreensão destas regras como a implementação do jogo, decidimos dar custo a certas jogadas e definir que cada jogador tem que gastar **2 pontos de jogadas por turno.** Com isto é possível perceber podem haver até duas movimentações por turno.

As regras são as seguintes:

1. Movimentar qualquer embarcação (**Excepto a MotherShip**) custa **1 pontos de jogada.**
2. Usar o movimento de captura efetuado por qualquer embarcação custa **2 pontos de jogada.**
3. Movimentar a *MotherShip* custa **1 pontos de jogada.**

	1	2	3	4	5	6	7	8	9	10	11
1	.	.	.	.	.	.	.	.	.	.	.
2	.	.	.	D	D	D	D	D	.	.	.
3	.	.	.	.	.	.	.	.	.	.	.
4	.	D	.	.	A	A	A	.	.	D	.
5	.	D	.	A	.	.	.	A	.	D	.
6	.	D	.	A	.	M	.	A	.	D	.
7	.	D	.	A	.	.	.	A	.	D	.
8	.	D	.	.	A	A	A	.	.	D	.
9	.	.	.	.	.	.	.	.	.	.	.
10	.	.	.	D	D	D	D	D	.	.	.
11	.	.	.	.	.	.	.	.	.	.	.

Todos os jogos começam com a mesma disposição de embarcações e *MotherShip.*

3 Representação do Estado de Jogo

Sendo o tabuleiro de 11x11 e as listas a melhor estrutura de dados a ser usada em PROLOG escolhemos uma lista de listas para representar o tabuleiro.

Cada elemento dessa lista será um espaço vazio ou uma peça. Para distinguir as peças dos dois jogadores, estas têm valores diferentes, sendo que as peças do jogador cinzento correspondem a "1" no tabuleiro, as peças do jogador amarelo são "2" e a *MotherShip*, também esta do jogador amarelo corresponde ao "5". Os espaços vazios são "0".

Portanto, a representação inicial do tabuleiro em forma de lista de listas será da seguinte forma:

```
initial_board([[0,0,0,0,0,0,0,0,0,0,0],
               [0,0,0,1,1,1,1,1,0,0,0],
               [0,0,0,0,0,0,0,0,0,0,0],
               [0,1,0,0,2,2,2,0,0,1,0],
               [0,1,0,2,0,0,0,2,0,1,0],
               [0,1,0,2,0,5,0,2,0,1,0],
               [0,1,0,2,0,0,0,2,0,1,0],
               [0,1,0,0,2,2,2,0,0,1,0],
               [0,0,0,0,0,0,0,0,0,0,0],
               [0,0,0,1,1,1,1,1,0,0,0],
               [0,0,0,0,0,0,0,0,0,0,0]]).
```

Durante a execução do jogo também que terá que ser mantida uma variável que defina qual dos jogadores é a jogar porque os jogadores têm jogadas e restrições diferentes.

4 Visualização do Tabuleiro

Para mostrar o tabuleiro aos jogadores irá ser necessário haver uma impressão do tabuleiro na consola. O tabuleiro impresso na consola irá numerar as colunas e linhas para facilitar as jogadas dos jogadores.

Quanto à simbologia usada a *MotherShip* irá ser simbolizada com um "M", as embarcações do jogador *Yellow* com "A" e as embarcações do jogador *Gray* com "D".

Quando houver uma jogada, a peca que foi movida irá ser impressa com parenteses à sua volta para facilitar a perceção da jogada feita.

Dependendo de quem está a jogar vão ser impressas diferentes informações. Quando o computador joga vão ser impressas as coordenadas do movimento feito. Se o computador estiver a jogar será impresso também o tempo que demorou a calcular a jogada

```
    1   2   3   4   5   6   7   8   9   10  11
1 | . | . | . | . | . | . | . | . | . | . | . |
-------------------------------------------------
2 | . | . | . | D | D | D | D | . | . | . | . |
-------------------------------------------------
3 | . | . | . | . | . | . | . | . | . | . | . |
-------------------------------------------------
4 | . | D | . | . | A | A | A | D | . | . | . |
-------------------------------------------------
5 | . | D | . | A | . | . | . | A | A | D | . |
-------------------------------------------------
6 | . | D | . | A | . | . | . | . | . | . | D |
-------------------------------------------------
7 | . | . | D | A | . | . | . | A | . | . |(M)|
-------------------------------------------------
8 | . | D | . | . | A | A | A | . | . | D | . |
-------------------------------------------------
9 | . | . | . | . | . | . | . | . | . | . | . |
-------------------------------------------------
10| . | . | . | D | D | D | D | D | . | . | . |
-------------------------------------------------
11| . | . | . | . | . | . | . | . | . | . | . |
-------------------------------------------------
```

5 Validação de Jogadas

A validação de jogadas é feita pelo predicado **validMove**. Antes desta chamada as coordenadas que são passadas já foram revistas e declaradas validas. Além disso também já foi confirmado que tanto existe uma peça nas coordenadas inicias como essa peça pertence ao jogador que está a fazer a movimentação da peça. Este predicado vai fazer as seguintes principais confirmações:

- O movimento é em linha reta.
- O movimento é diagonal.

Se nenhuma destas condições estiver correta, então o predicado devolve imediatamente *fail*. Aproveitamos para realçar o facto de ser impossível as duas condições serem confirmadas.

Se o movimento for **em linha reta** é ainda confirmado que não há nenhuma peça entre a coordenada original e a coordenada de destino. Se houver o predicado dá *fail*. Esta confirmação é feita através do predicado **inLine**.

Este predicado vai ser mais importante um pouco à frente porque vai ser com ele que o computador vai compilar uma lista de todas as jogadas possíveis.

```
    1   2   3   4   5   6   7   8   9   10  11
1 | . | . | . | . | . | . | . | . | . | . | . |
-------------------------------------------------
2 | . | . | . | D | D | D | D | D | . | . | . |
-------------------------------------------------
3 | . | . | . | . | . | . | . | . | . | . | . |
-------------------------------------------------
4 | . | D | . | . | . | A | A | A | . | . | D | . |
-------------------------------------------------
5 | . | D | . | A | . | . | . | . | A | . | D | . |
-------------------------------------------------
6 | . | D | . | A | . | M | . | . | A | . | D | . |
-------------------------------------------------
7 | . | D | . | A | . | . | . | . | A | . | D | . |
-------------------------------------------------
8 | . | D | . | . | . | A | A | A | . | . | D | . |
-------------------------------------------------
9 | . | . | . | . | . | . | . | . | . | . | . | . |
-------------------------------------------------
10| . | . | . | D | D | D | D | D | . | . | . | . |
-------------------------------------------------
11| . | . | . | . | . | . | . | . | . | . | . | . |
-------------------------------------------------

What piece do you want to move? Jogador Yellow with 2 Plays:
X = |: 5.

Y = |: 5.

Impossivle Play

What piece do you want to move? Jogador Yellow with 2 Plays:
X = |: 6.

Y = |: 6.

New X = |: 9.

New Y = |: 7.

Impossivle Play

What piece do you want to move? Jogador Yellow with 2 Plays:
X = |:
```

6 Execução de Jogadas

A execução de jogadas por parte tanto dos jogadores como pelo computador foi implementada de forma diferente da que foi descrita no relatório intercalar.

No início de cada turno, o jogador vai ter **2 pontos de jogada por turno**. Vai ser pedido a cada jogador, quer ele seja humano ou computador, as coordenadas originais da peça e as coordenadas do seu destino. Depois de fazer as confirmações faladas anteriormente, é calculado o **custo da jogada.** Se o custo da jogada for superior aos restantes **pontos de jogada por turno,** o predicado responsável por chamar o **movePiece** vai dar *fail* e terá que ser feito um novo input de coordenadas (a falar mais à frente).

Se não for, então os **pontos de jogada por turno** do jogador atual vai ser decrementado e vai-lhe ser permitido jogar outra vez. Quando os **pontos de jogada por turno** ficarem a 0 o turno será passado para o outro jogador.

O **movePiece** vai ser responsável por mover uma peça de uma coordenada para outra. Na prática, o **movePiece** nunca vai dar *fail* porque antes de ser chamado as condições de falha já foram testadas e passadas com sucesso.

O **movePiece** vai portanto simplesmente guardar o valor que está na posição das coordenadas originais, definir essa posição como vazia e definir a posição das coordenadas finais com o valor que estava na posição inicial.

7 Final do Jogo

Como já foi falado anteriormente o objetivo dos dois jogadores é diferente. Consequentemente o necessário para terminar o jogo vai ser obviamente diferente. Vamos aproveitar este tópico para introduzir também os predicados adicionais que criamos para a inteligência que são os predicados **checkMateYellow** e **checkMateGray.**

Começando por estes últimos, os predicados têm como objetivo determinar se é possível que o jogador *Yellow* ou o *Gray* podem de alguma forma concluir o jogo com uma jogada. Como função adicional deste predicado, vai também devolver que movimento precisa de fazer para concluir o jogo.

No entanto estes predicados só são uteis quando estão a ser utilizados comportamentos do computador com a mínima inteligência artificial.

Estes predicados não substituem portanto um que permita saber se o jogo acabou. Para isso nós usamos o **continueGame.** Este predicado retorna falso se a *MotherShip* tiver sido capturada ou se se encontrar na borda do tabuleiro.

Em todos os ciclos do jogo esta função é chamada garantindo assim que é possível continuar a ser jogado o jogo. Quando retorna falso, o jogo acaba sendo o ultimo jogador a ter efetuado uma jogada o vencedor.

8 Avaliação do Tabuleiro

A avaliação do tabuleiro foi uma das principais dificuldades ao realizar o trabalho. Como não estávamos muito familiarizados com o jogo foi difícil perceber o que era ou não uma boa jogada. Mas concluímos rapidamente que as avaliações do tabuleiro para cada jogador teriam de ser diferentes devido aos seus diferentes objetivos.

Ao avaliar o tabuleiro, é lhe atribuída uma pontuação. Isto vai ajudar o computador a determinar mais tarde qual a melhor jogada. Quanto maior essa pontuação melhor esse tabuleiro é para o jogador para qual está a ser feita a avaliação, ou seja, mais perto estará teoricamente da sua vitória. Teoricamente está sublinhado porque o jogo não foi estudado muito a fundo, principalmente porque não era o objetivo do projeto. O predicado responsável por avaliar o tabuleiro é o **evaluateBoard**. Este predicado recebe a matriz e o jogador que fez a ultima jogada e devolve um valor que resulta da situação em que o tabuleiro.

A nossa solução é portanto heurística. As avaliações que nós fazemos ao tabuleiro são as seguintes (estão por ordem de importância e divididas em a favor do jogador ou contra):

A Favor	Contra
Jogo Vitorioso	É impossível fazer uma jogada que resulte no jogador adversário se tornar vitorioso
Nº de CheckMates a favor do jogador atual	Nº de CheckMates a favor do jogador adversário
Nº de Embarcações Aliadas	Nº de Embarcações Inimigas
Distância da MotherShip ao centro (só para o jogador *Yellow*)	-

A razão pela qual nós usamos Nº de CheckMates é porque quanto mais situações de CheckMate existirem mais facilmente é possível ganhar. Logo se com uma jogada for possível haver um maior numero de acabar o jogo essa jogada vai ser mais valiosa.

Depois de fazer todas estas avaliações, vai ser obtido um valor numérico. Este valor vai ser usado para comparar tabuleiros entre si, importante ao incorporar a inteligência artificial.

9 Jogada do Computador

Ao desenvolver o projeto começamos por desenvolver inteligências muito simples, começando aliás por uma cujo próximo movimento era unicamente um aleatório retirado da lista de movimentos possíveis. No entanto, ao longo da realização fomos ganhando conhecimento nesta área e podemos implementar duas inteligências mais avançadas.

Acabamos por decidir manter todas as dificuldades no projeto, portanto explicaremos brevemente cada uma delas.

Random

A inteligência **Random** não passa disso. Através do predicado **validMove** e da utilização de **findall** é possível obter uma lista de todas as jogadas possíveis. Desta lista é retirado um elemento aleatório. Os detalhes deste movimento são depois passados ao predicado por realizar o movimento (**movePiece**) e atualização do restantes **pontos de jogada por turno**.

Random with CheckMate

Esta dificuldade é muito parecida com anterior. A diferença é que nesta dificuldade quando o computador se apercebe que pode ganhar o jogo faz essa jogada em vez de uma jogada aleatória retirada da lista. Aqui é onde é dado o primeiro uso aos predicados **checkMateYellow** e **checkMateGray** anteriormente falados**.** Este pormenor vai permitir que esta dificuldade ganhe facilmente os jogos à primeira dificuldade.

Hard

Esta é a primeira verdadeira inteligência artificial porque é a primeira a dar uso ao predicado **evaluateBoard.** Esta inteligência vai pegar na mesma lista que **Random** e vai simular todos os movimentos. Esta lista tem que se baralhada. A razão para isto é o facto de as jogadas iniciais possíveis serem sempre as mesma e se não houver um baralhamento da lista de jogadas possíveis, os jogos começariam sempre da mesma forma. Depois de os simular vai avalia-los com o **evaluateBoard**. Depois de ser avaliados vai ser jogado o melhor movimento possível.

O algoritmo desenvolvido foi baseado no **evaluate_and_choose** descrito no livro Art of PROLOG. Foi feita uma pequena alteração ao algoritmo para o manter mais eficiente. Quando é avaliado um tabuleiro que resulta na vitória do jogador, então o resto das jogadas não são avaliadas porque terão no máximo o mesmo valor que esta. Poupa-se assim tempo computacional.

Hardest (Dev Mode)

Esta dificuldade requer um pouco mais de explicação. A dificuldade anterior ia simulando jogadas e avaliando os tabuleiros. Sublinhamos jogadas porque é essa a grande diferença entre esta dificuldade e a anterior.

Esta dificuldade faz a sua decisão avaliando o **turno.** Como já foi dito anteriormente é possível fazer até duas jogadas por turno. Portanto nesta dificuldade o que vai ser feito é pegar na original lista de possíveis jogadas, uma a uma ir simulando e se ainda houver **pontos de jogada por turno,** criar uma nova lista de jogadas a partir do novo tabuleiro e as simular como na dificuldade **Hard**.

Depois de tudo isto o melhor turno é jogado, quer seja ele um ou dois movimentos.

A grande necessidade de criar esta dificuldade saíu de uma situação bastante particular em que nós víamos a dificuldade **Hard** perder quando não havia razão para isso acontecer. O que acontecia é que a dificuldade não entendia que podia mover a mesma peça duas vezes levando a para uma posição mais vantajosa. Como não conseguia perceber isto achava que ao movê-la uma vez na direção correta era o mesmo que não o fazer porque a avaliação do tabuleiro assim não o permitia. Daqui surgiu a necessidade de criar uma dificuldade que conseguisse ver além de um só movimento.

Para concluir gostaríamos de explicar a razão pela qual não implementamos o algoritmo minimax e derivados deste. A razão prende-se no facto de que a nossa ultima dificuldade desenvolvida já demorava a calcular uma resposta até 6 segundos. Por esta razão ao prever que com uma profundidade decente o tempo necessário para calcular uma jogada seria absurdo decidimos não o fazer.

10 Interface com o Utilizador

A interface com o utilizador é bastante similar ao que estamos todos habituados a ver em jogos.

O menu principal tem 3 opções:

1. **Play ->** Começa um novo jogo.
2. **Credits->** Apresenta os créditos do jogo.
3. **Exit->** Acaba com o jogo.

A partir do **Play** é possível escolher que jogador vai ser jogado, se vai ser o computador a jogar ou humanos.

1. **HUM(Y) - HUM(G)->** Humano: *Yellow* / Human : *Gray*
2. **HUM(Y) - PC(G)->** Humano: *Yellow* / Computador : *Gray*
3. **PC(Y) - HUM(G)->** Computador: *Yellow* / Human : *Gray*
4. **PC(Y) - PC(G)->** Computador: *Yellow* / Human : *Gray*
5. **Back->** Volta ao menu principal.

```
Bem Vindo ao BreakThru

1 - Play
2 - Credits
3 - Exit
|: 1.


1 - HUM(Y) - HUM(G)
2 - HUM(Y) - PC(G)
3 -  PC(Y) - HUM(G)
4 -  PC(Y) - PC(G)
5 - Back
|:
```

Se para um ou para os dois dos jogadores for selecionado o computador, uma nova pergunta será feita a pedir a dificuldade do computador. No caso de ambos os jogadores serem computador é possível até escolher diferentes dificuldades para cada jogador.

Durante o jogo o display vai ser este:

```
Choose Difficulty for Yellow Bot :
1 - Random
2 - Random with CheckMate
3 - Hard
4 - Hardest (Dev Mode)
|: 4.
Number of Possible Plays : 60
RunTime : [1765,1375]
Yellow Bot played X:6|Y:6 to XF 6|YF:7

    1   2   3   4   5   6   7   8   9   10  11
1 | . | . | . | . | . | . | . | . | . | . | . |
-----------------------------------------------
2 | . | . | . | D | D | D | D | D | . | . | . |
-----------------------------------------------
3 | . | . | . | . | . | . | . | . | . | . | . |
-----------------------------------------------
4 | . | D | . | . | A | A | A | . | . | D | . |
-----------------------------------------------
5 | . | D | . | A | . | . | . | A | . | D | . |
-----------------------------------------------
6 | . | D | . | A | . | . | . | A | . | D | . |
-----------------------------------------------
7 | . | D | . | A | . |(M)| . | A | . | D | . |
-----------------------------------------------
8 | . | D | . | . | A | A | A | . | . | D | . |
-----------------------------------------------
9 | . | . | . | . | . | . | . | . | . | . | . |
-----------------------------------------------
10| . | . | . | D | D | D | D | D | . | . | . |
-----------------------------------------------
11| . | . | . | . | . | . | . | . | . | . | . |
-----------------------------------------------

What piece do you want to move? Jogador Gray with 2 Plays:
X = |:
```

Podemos ver a jogada feita pelo computador tanto em texto como no próprio tabuleiro (peça com parenteses à volta).

Quando um jogador ganha é apresentada uma mensagem de vitória e o jogo volta ao menu principal.

11 Conclusão

A realização do trabalho foi bastante trabalhosa. No entanto ambos os elementos do grupo podem afirmar que quando acabaram o projeto se sentiram bastante mais aptos a realizar outros trabalhos neste paradigma de programação.

Alem do paradigma em si, o grupo também sente que ganhou conhecimento na área da inteligência artificial tanto por prática como pesquisa para realizar o trabalho.

Para sumariar, nós gostamos de realizar o trabalho. O raciocínio lógico por detrás de cada predicado implementado acabou por nos cativar e assim foi possível entregar um jogo que tanto requer pensamento a faze-lo como a jogar e nos deu tanto gosto a fazer como agora a joga-lo.

12 Bibliografia

Sterling, Leon S.; Shapiro, Ehud Y. - **The Art of Prolog**: Advanced Programming Techniques

https://en.wikipedia.org/wiki/Breakthru_(board_game)

13 Anexos

O código fonte encontra-se no ficheiro “Breakthru.pl” que se encontra anexado a este relatório.