

Blockade

Relatório Final



Mestrado Integrado em Engenharia Informática e
Computação

Programação em Lógica

Grupo:

João Barbosa - up201406241

José Martins - up201404189

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal vspace1cm

12 de Novembro de 2016

Resumo

Resumo sucinto do trabalho com 150 a 250 palavras (problema abordado, objetivo, como foi o problema resolvido/abordado, principais resultados e conclusões).

Conteúdo

1	Introdução	4
2	O Jogo Blockade	5
3	Lógica do Jogo	6
3.1	Representação do Estado do Jogo	6
3.2	Visualização do Tabuleiro	7
3.3	Lista de Jogadas Válidas	7
3.4	Execução de Jogadas	8
3.5	Avaliação do Tabuleiro	8
3.6	Final do Jogo	9
3.7	Jogada do Computador	9
4	Interface com o Utilizador	10
5	Conclusões	10
	Bibliografia	11
A	Anexo Código	11

1 Introdução

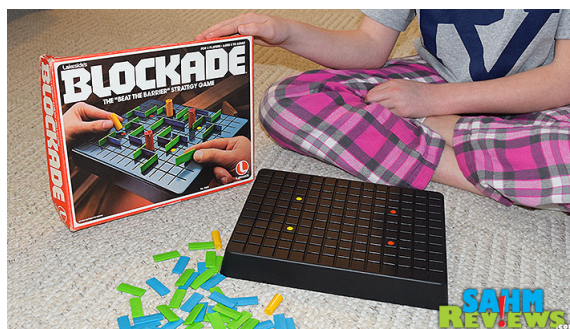
Este trabalho foi desenvolvido no âmbito da unidade curricular “Programação em Lógica” do 3º ano do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto. O seu objetivo é o de implementar em Prolog um jogo de tabuleiro de 2 jogadores de forma a possibilitar o jogo Humano vs. Humano, Humano vs. Computador e Computador vs. Computador. Neste relatório será descrito o jogo que escolhemos para a nossa implementação – o “Blockade” – assim como as suas regras. De seguida, serão detalhadas algumas funcionalidades e características da nossa implementação, desde a representação do jogo e visualização do tabuleiro até a avaliação de jogadas pelo computador e final do jogo. Por fim, serão apresentadas as conclusões que obtivemos da realização deste trabalho, bem como a sua bibliografia.

2 O Jogo Blockade

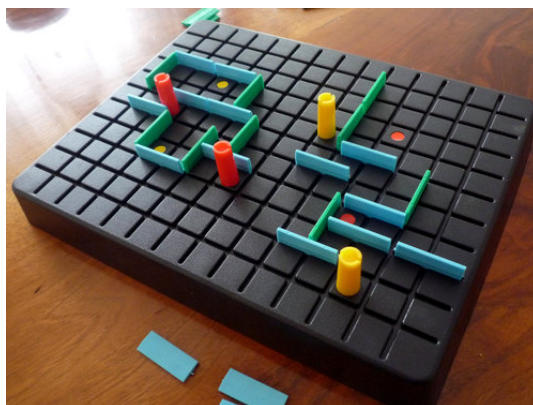
Blockade trata-se de um jogo de tabuleiro produzido pela primeira vez em 1975 pela Lakeside Games. O jogo é desenhado para 2 jogadores sendo que cada um possui:

- 2 Peões
- 9 Parede verdes (que só podem ser colocadas verticalmente)
- 9 Paredes azuis (que só podem ser colocadas horizontalmente)

O tabuleiro do jogo é um quadriculado com dimensões 11x14, com 2 pontos amarelos e dois pontos laranja que representam a base, e as posições iniciais, dos dois peões de cada jogador. Estes pontos distam 4 quadrículas de cada canto na diagonal.



Trata-se de um jogo de turnos, em que em cada turno um jogador pode mover um dos seus peões, uma ou duas quadrículas (horizontalmente, verticalmente ou uma combinação das duas), e posicionar uma parede de modo a tentar bloquear os movimentos do adversário. As paredes ocupam sempre duas quadrículas e devem ser posicionadas de acordo com a sua cor. Peões podem saltar por cima de outros peões que estejam a bloquear o seu caminho. O objetivo do jogo é levar um dos seus peões até à base de um dos peões do adversário. Quando os jogadores ficarem sem paredes para colocar, continuam a mover-se até que alguém vença o jogo.



3 Lógica do Jogo

3.1 Representação do Estado do Jogo

O jogo possui uma representação interna, utilizada para o processamento e armazenamento de informação, e uma representação externa, para tornar a visualização do jogo mais apelativa e intuitiva. A simbologia utilizada é a seguinte:

Elemento	Dimensão	Representação interna	Representação externa
Célula livre	3x3	square	
Peça 1 (J1)	3x3	[orange, 1]	$\overline{[O]}$ '''
Peça 2 (J1)	3x3	[orange, 2]	$\overline{[O]}$ ^^^
Base (J1)	3x3	[orange, base]	... O ...
Peça 1 (J2)	3x3	[yellow, 1]	$\overline{[Y]}$ '''
Peça 2 (J2)	3x3	[yellow, 2]	$\overline{[Y]}$ ^^^
Base 2 (J2)	3x3	[yellow, base]	... Y ...
Parede vertical (espaço)	3x3	[vertical,empty]	
Parede vertical (colocada)	3x3	[vertical,placed]	X X X
Parede horizontal (espaço)	6x1	[horizontal,empty]	'-----'
Parede horizontal (colocada)	6x1	[horizontal,placed]	'XXXXXX'

O tabuleiro é também representado internamente por um grafo, de modo a permitir a utilização de algoritmos de "Path Finding" na avaliação das melhores jogadas e em algumas verificações.

3.2 Visualização do Tabuleiro

O tabuleiro será visualizado através da utilização de caracteres ASCII para representar os peões, paredes e bases de cada jogador, exemplo:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
0	0
1	1
2	2
3	3
4	4
5	5
6	[0]	[0]	6
7	7
8	02	8 AAA
9	9
10	X	X	01	10 X X X XXXXX XXXXX
11	11
12	X	X	12 X X X
13	13
14	.	.	.	y1	.	.	.	X	14 X X X
15	15
16	X	16 X X X
17	17
18	18
19	XXXXX XXXXX	19
20	[Y]	y2	20 AAA
21	21
22	22
23	23
24	24
25	25
26	26

-Orange has 8 horizontal walls and 8 vertical walls
-Yellow has 8 horizontal walls and 8 vertical walls

3.3 Lista de Jogadas Válidas

As jogadas são obtidas através do input do jogador ou através dos algoritmos implementados para permitir calcular a jogada do computador, sendo posteriormente verificada a sua validade.

3.4 Execução de Jogadas

Depois de obtidas as coordenadas para a movimentação do jogador é utilizado o predicado `validPosition(+Pawn,+ Board, +X,+ Y,-Nx,-Ny)`, este predicado recebe o offset (X,Y) para onde o jogador se quer mover em relação à sua posição atual e falha quando as coordenadas finais da "futura" posição do jogador se encontram fora das dimensões do tabuleiro ou quando existe uma parede a bloquear a movimentação para as novas coordenadas.

Assim que a jogada se encontra validada é chamado o predicado `moveOneSpace(+Pawn, +X, +Y, +Board, -NewBoard)`, que move o peão num offset (X,Y) criando um novo tabuleiro.

Existe também outro predicado para validar e posicionar as paredes. Este denomina-se `placeWall(+Player,+X,+ Y,+O,+Board, -NewBoard)`, o predicado é bem sucedido quando as coordenadas da parede são validas, criando assim um novo tabuleiro.

No entanto o predicado falha quando as coordenadas são invalidas devido a um dos motivos:

- A parede está para lá dos limites do tabuleiro
- A parede está cruzada com outra parede
- A parede bloqueia completamente o jogador (ou seja quando o posicionamento da parede impossibilita que um dos peões deixe de ter um caminho para as bases adversárias)

Estes predicados podem ser encontrados no ficheiro `board/logic.pl`.

Validação e execução de uma jogada num tabuleiro, obtendo o novo estado do jogo. Exemplo: `move(+Move, +Board, -NewBoard)`.

3.5 Avaliação do Tabuleiro

Na nossa implementação, não existe uma avaliação direta do tabuleiro, para comparar as diferentes jogadas possíveis, pois:

- Quando o jogador é um humano, a direção do movimento é pedida via linha de comandos. Neste contexto, a avaliação da jogada é ignorada, pois o jogador deve ter liberdade para a fazer, independentemente da sua qualidade.
- Quando o jogador é um computador em nível fácil, os movimentos são aleatórios, não havendo qualquer avaliação do tabuleiro.
- Quando o jogador é um computador em nível difícil, a movimentação das peças é feita com recurso a *pathfinding*, que nos devolve a melhor jogada possível.

Este algoritmo avalia internamente o grafo, e determina qual o caminho de menor custo para o jogador. Esta avaliação do tabuleiro é indireta, visto que os algoritmos sobre grafos utilizados estão implementados nas bibliotecas do *Prolog*.

Avaliação do estado do jogo, que permitirá comparar a aplicação das diversas jogadas disponíveis. Exemplo: `value(+Board, +Player, -Value)`.

3.6 Final do Jogo

A verificação de que o jogo chegou ao fim é testada usando o predicado *checkEnd*. Este predicado não precisa de argumentos, pois as posições dos jogadores são extraídas diretamente da base de dados, utilizando o predicado auxiliar *position(+Player, -X, -Y)*.

Se alguns dos jogadores chegar ao seu objetivo, i.e. a uma das bases do oponente, então o predicado retorna *true*, e, conseqüentemente, termina o ciclo de jogo.

3.7 Jogada do Computador

Foram implementados dois níveis de dificuldade para a jogada do computador:

Nível Fácil

- Este modo é totalmente aleatório, desta forma não realiza nenhuma consideração sobre o estado de jogo. Para criar a jogada são utilizados os predicados:

- *randomMove(-X, -Y)* - para calcular uma movimentação aleatória.
- *randomWall(-X, -Y, -O)* - para calcular uma parede aleatória.

Quando as coordenadas devolvidas não são válidas, os predicados voltam a ser executados, até ser encontrada uma jogada possível.

Nível Difícil

- Este modo procura a melhor jogada possível no momento, utilizando "path finding" para encontrar o caminho mais curto tendo em consideração todas as paredes já posicionadas, assim é escolhido de entre os dois peões do jogador o melhor a ser movido e a melhor direção para o mover. No que diz respeito às paredes, é feito o processo contrário ou seja o predicado determina o melhor peão do oponente e a melhor direção para ele se movimentar tentando depois colocar uma parede a bloqueá-lo, se não for possível colocar a parede a bloquear a melhor direção do melhor peão do oponente é escolhida outra parede, de modo a bloqueá-lo noutra direção. Para criar a jogada são utilizados os predicados:

- *evaluateBestPawn(+Player,-N)* - para calcular o melhor peão a movimentar.
- *evaluateBestDirectionPro(+Player,+Id, -Direction)*- para calcular a melhor direção para o peão.
- *evaluateBestWall(+Player,-Walls)*- para calcular uma lista ordenada com as melhores paredes a posicionar para bloquear o oponente de *Player*.

Escolha da jogada a efetuar pelo computador, dependendo do nível de dificuldade. Por exemplo: *choose_move(+Level, +Board, -Move)*.

4 Interface com o Utilizador

Foram criados diversos menus para permitir uma fácil interação do utilizador com o programa.

As opções possíveis estão sempre explícitas nos menus, o utilizador deverá colocar um '.' a seguir à opção pretendida (ex. '1.'). Aquando da chamada do predicado 'blockade' o jogo iniciasse sendo mostrado ao utilizador o seguinte menu.

```

Blockade
Made by:
Joao Barbosa - up201406241
Jose Martins - up201404189

1. Player vs Player
2. Player vs Bot
3. Bot vs Bot
4. Rules
5. Exit
|: █
```

Se a escolha do utilizador passar pela opção 2 ou 3 é apresentado o seguinte menu, para permitir a seleção da dificuldade pretendida.

```

Blockade
Made by:
Joao Barbosa - up201406241
Jose Martins - up201404189

Difficulty:
1. Hard
2. Easy
3. Back
|: █
```

Descrever o módulo de interface com o utilizador em modo de texto.

5 Conclusões

Que conclui deste projecto? Como poderia melhorar o trabalho desenvolvido?

A Anexo Código

Código Prolog implementado devidamente comentado e outros elementos úteis que não sejam essenciais ao relatório.