Blockade

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Blockade-4:
João Barbosa - up201406241
José Martins - up201404189

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

12 de Novembro de 2016

Resumo

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, foi proposto o desenvolvimento de um jogo de tabuleiro, utilizando a linguagem *Prolog*. O jogo escolhido denomina-se **Blockade**.

A utilização de *Prolog*, uma linguagem do paradigma da programação lógica, mostrou-se não só interessante, pois foi possível perceber as grandes potencialidades e vantagens da linguagem, como também desafiante, devido à inexperiência neste tipo de paradigma.

Para o desenvolvimento do projeto, foi necessário perceber e estruturar os conceitos-chave do jogo, de forma a obter a melhor forma de proceder à implementação do jogo. Além disso, foi necessário aprofundar o conhecimento sobre as bibliotecas disponibilizadas pela linguagem, para garantir qualidade e segurança no código.

O resultado final da implementação do jogo mostra-se como uma boa reprodução do **Blockade**. Adicionalmente, permite que os jogadores sejam **humanos** (que escolhem as jogadas que pretendem fazer), ou ***bots*** (que avaliam a melhor jogada que podem executar).

Conteúdo

1 Introdução

Este projeto 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 implementar em *Prolog* um jogo de tabuleiro de 2 jogadores, possibilitando os modos de 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 à avaliação de jogadas pelo computador e final do jogo.

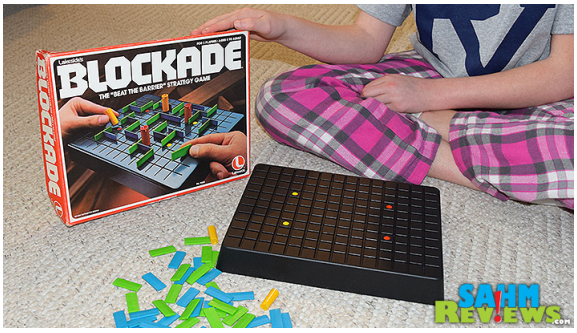
Por fim, serão apresentadas as conclusões que obtivemos da realização deste trabalho, bem como a sua bibliografia.

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:

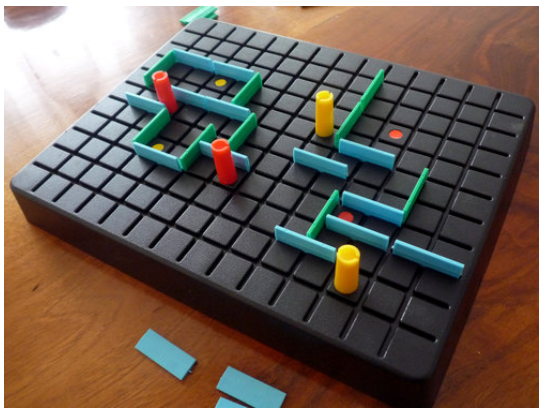
- 2 Peões
- 9 Parede verdes (que só podem ser colocadas verticalmente)
- 9 Paredes azuis (que só podem ser colocadas horizontalmente)

O tabuleiro do jogo é um quadriculado com dimensões 11x14, com 2 pontos amarelos e dois pontos laranja que representam as bases, 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

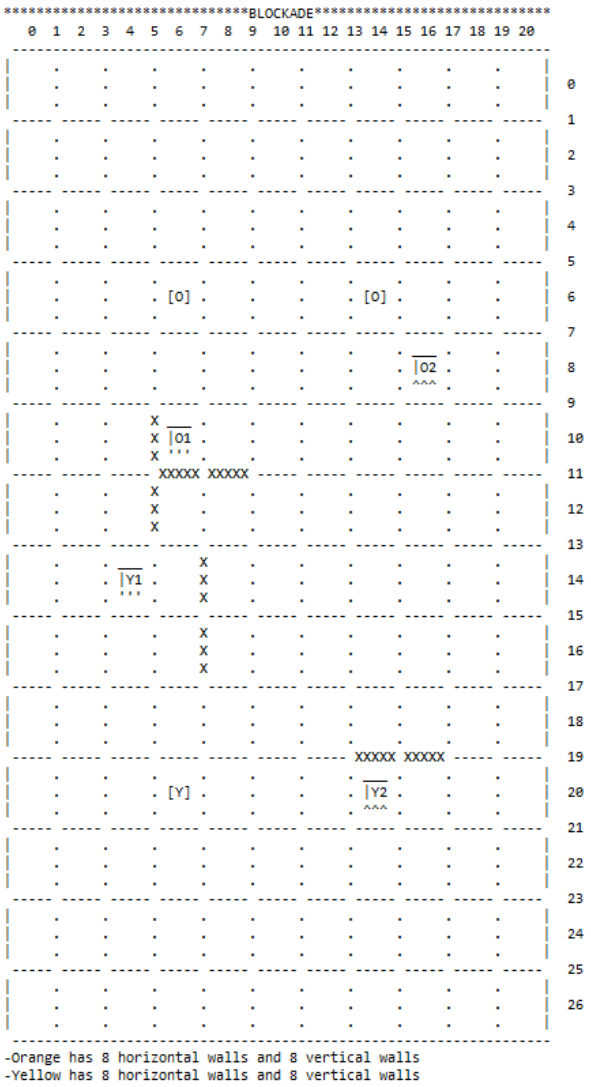
O jogo possui uma representação interna, utilizada para o processamento e armazenamento de informação, e uma representação externa, para tornar a visualização do jogo mais apelativa e intuitiva. A simbologia utilizada é a seguinte:

Elemento	Dimensão	Representação interna	Representação externa
Célula livre	3x3	square	
Peça 1 (J1)	3x3	[orange, 1]	‾‾‾ |O1 ‘‘‘
Peça 2 (J1)	3x3	[orange, 2]	‾‾‾ |O2 ^^^
Base (J1)	3x3	[orange, base]	... O ...
Peça 1 (J2)	3x3	[yellow, 1]	‾‾‾ |Y1 ‘‘‘
Peça 2 (J2)	3x3	[yellow, 2]	‾‾‾ |Y2 ^^^
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]	‘XXXXX ‘

O tabuleiro é tambem representado internamente por um grafo, de modo a permitir a utilização de algoritmos de *pathfinding* 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 peãos, paredes e bases de cada jogagor, exemplo:



3.3 Lista de Jogadas Válidas

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

Note-se que, por norma, não é necessária a utilização uma **lista** de jogadas válidas, devido à natureza do jogo. Os predicados que retornam listas de possibilidades são:

- *evaluateBestDirection(+Player,+Id,-Directions)*
- *evaluateBestWall(+Player,-Walls)*

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.

Depois de a jogada ser validada, o predicado *moveOneSpace(+Pawn, +X, +Y, +Board, -NewBoard)* move o peão num offset (X,Y), criando um novo tabuleiro.

Existe tambem outro predicado para validar e posicionar as paredes, denominado *placeWall(+Player,+X,+ Y,+O,+Board, -NewBoard)*. O predicado é bem sucedido quando as coordenadas da parede são validas, i.e. quando o seguinte não acontece:

- 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 caminho para as bases adversárias)

Se *placeWall* for invocado com sucesso, cria um novo tabuleiro com as informações atualizadas. Além disso, este predicado vai também atualizar o grafo do tabuleiro.

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 biliotecas do *Prolog*.

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

3.7 Jogada do Computador

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

Nivel Fácil

- Este modo é totalmente aleatório, e 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 validas, os predicados voltam a ser executados, até ser encontrada uma jogada possivel.

Nivel Difícil

- Este modo procura a melhor jogada possivel 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.

  Utilizando estas informações, o computador calcula a lista de jogadas que melhor bloqueiam o movimento do oponente, ordenadas por ordem de crescente de qualidade. Se uma parede não pode ser colocada, é escolhida a seguinte, num processo recursivo. Se nenhuma parede pode ser colocada, então o computador decide não colocar nenhuma parede nessa jogada.

  Para criar a jogada neste nível 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.

4 Interface com o Utilizador

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

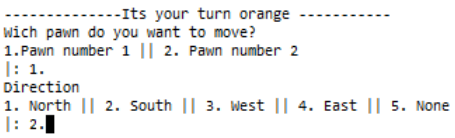
As opções possiveis estão sempre explicitas nos menus, o utilizador deverá colocar um '.' a seguir á opção pretendida (ex. '1.').

Aquando da chamada do predicado *blockade* (sem argumentos) o jogo inicia-asse, sendo mostrado ao utilizador o seguinte menu.

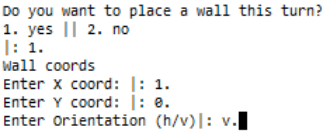
Se a escolha do utilizador passar pela opção 2 ou 3 é apresentado o seguinte menu, para permitir a seleção da diculdade pretendida.

Em ambiente de jogo é representado o tabuleiro atual e no fim deste são apresentados os dialogos correspondentes á ação a ser tomada no momento.

- Diálogo para a movimentação de um peão do jogador.



- Diálogo para a colocação de uma parede.



5 Conclusões

Tendo em conta resultado final da implementação, é possível afirmar que os principais objetivos deste projeto foram cumpridos:

- Reprodução completa do jogo **Blockade**.
- Modos de jogo `Humano vs Humano`, `Humano vs Computador`, `Computador vs Computador`.
- Dois níveis de inteligência para o ***bot***.
- Interface com o utilizador

Durante o desenvolvimento, foi possível perceber as potencialidades do *Prolog*, e as vantagens de utilizar programação em lógica: a codificação das operações de natureza puramente lógica tornam-se mais intuitivas de realizar, e a implementação de certos algoritmos é facilitada pela natureza da linguagem.
É também de notar que o *Prolog* possui muitas bibliotecas que podem ajudar a construir código com qualidade e segurança e, por esse motivo, foram utilizadas sempre que fossem convenientes.

No entanto, é importante reconhecer que, devido à inexperiência na linguagem, algumas implementações poderiam ser melhoradas. Especificamente, o tabuleiro de jogo possui 2 representações internas: uma em lista de listas, e outra em grafo. Isto acontece porque a representação inicial utilizada foi lista de listas, mas devido ao *pathfinding*, foi também necessário a criação de um grafo.

As duas representações foram mantidas porque:

- A maioria dos predicados utiliza a representação em lista de listas.
- O *refactor* dos mesmos predicados seria custoso, em termos de tempo.
- A utilização única de representação em grafo provocaria algum *overhead* em certas componentes (p.ex no *display* do tabuleiro).

No entanto, esta solução apresenta algumas desvantagens:

- Ocupa mais espaço em memória.
- A atualização do tabuleiro ocorre em dois sítios.

Um maior conhecimento prévio do *Prolog*, e das bibliotecas que oferece, preveniria este acontecimento.

Concluímos, portanto, que este projeto possibilitou a aprendizagem e consolidação de um novo tipo de paradigma de programação, abrindo novos horizontes e novas possibilidades, na definição e implementação de soluções para novos problemas.

Bibliografia

[1] SICStus. Sicstus prolog, 2006. [Online; accesso 5-Novembro-2016].

[2] SICStus. Unweighted graph operations, 2006. [Online; accesso 9-Novembro-2016].

[3] Leon S. Sterling. *The Art of Prolog.* MIT Press, 1994.

A Anexo Código

O código encontra-se disponível no ficheiro *blockade.zip*.