MOD X

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Mod X_3:
António Manuel Vieira Ramadas - 201303568
Rui Miguel Teixeira Vilares - 201207046

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

8 de Novembro de 2015

Resumo

No âmbito da unidade curricular Programação em Lógica, foi-nos proposto desenvolver um jogo em linguagem PROLOG. Esse jogo, denomina-se MOD X e é bastante recente, tendo sido criado este ano.

O PROLOG é uma linguagem funcional e não uma linguagem procedimental. Assim, trabalhar nesta linguagem de programação foi um maiores desafios, tendo em conta a diferença que apresenta em relação com outras, tipicamente mais utilizadas.

O esforço e o trabalho em equipa foram essenciais para resolver os problemas que foram surgindo. No entanto, reforçamos a importância das aulas teóricas, onde a matéria era exposta com clareza e permite uma mais fácil adaptação ao funcionamento e à essência dos objetivos propostos.

Através da cooperação e da forma objetiva que o trabalho foi elaborado, foi possível chegar a um resultado final simples, robusto, eficiente e apelativo. A execução em linha de comandos consegue ser clara e facilmente interpretada pelo utilizador comum.

Em suma, considera-se o trabalho realizado de extrema importância para consolidar, mais facilmente, os conceitos abordados na aula. Os trabalhos práticos são sempre uma mais valia para adquirir conhecimento sólido, de uma forma direta e apelativa.

Conteúdo

1 Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Programação em Lógica, do curso Mestrado Integrado em Engenharia Informática e Computação. Inicialmente, foram disponibilizados vários jogos com o intuito de serem desenvolvidos em Prolog.

A escolha recaiu no MOD X, essencialmente porque o seu mecanismo de jogo é compreendido com facilidade, utiliza um tabuleiro típico e apenas possui dois tipos de peças de jogo. Entenda-se por tabuleiro típico, um tabuleiro quadrado, semelhante ao utilizado nas damas.

Uma partida disputada entre dois jogadores, exige um bom conhecimentos sobre as regras de jogo e uma estratégia definida, adequada às próprias regras. Assim, este é um típico jogo de tabuleiro, onde a prática mental é exigida e testada em cada movimento.

O principal objetivo da realização deste trabalho prende-se na aplicação dos conceitos abordados nas aulas, tanto teóricas como práticas. A realização de um trabalho prático é sempre mais estimulante e adequada para aprender uma linguagem de programação.

Ao longo do relatório, procura-se contextualizar este jogo, abordado as suas regras, objetivos e componentes. É importante também descrever a implementação lógica do jogo em Prolog, seja ao nível de representação do estado do jogo, da visualização do tabuleiro, da lista de jogadas válidas, da execução de jogadas, da avaliação do tabuleiro, das condições finais de jogo e da forma de jogar do computador. A interface com o utilizador é descrita detalhadamente, facilitando a sua utilização e compreensão. A conclusão, a bibliografia e os anexos têm também uma presença chave neste relatório.

2 O Jogo MOD X

2.1 Contextualização

O MOD X é um jogo de tabuleiro, criado em 2015, recomendado para jogadores com mais de 15 anos de idade. É um jogo de estratégia, divertido e fácil de aprender. Cada partida reúne 2 a 4 jogadores e tem a duração prevista de 20 a 30 minutos.

2.2 Componentes do jogo

- 1 Tabuleiro 8x8;
- 56 peças de jogo em formato X (14 em cada cor);
- 72 marcadores de pontuação (18 em cada cor);
- 5 peças *Joker* em formato X (brancas);

Figura 1: Caixa do MOD X

2.3 Objetivo do jogo

O objetivo deste jogo é criar padrões com peças coloridas (vermelho, preto, amarelo ou laranja). Pretende-se assim, formar o maior número de padrões possível, de forma a conseguir a melhor pontuação. O vencedor é o jogador com mais pontos. Evidentemente, é também suposto bloquear os adversários de modo que não consigam construir esses padrões.

2.4 Regras

- Define-se a ordem dos jogadores, cada um escolhe a cor das suas peças e define-se o limite de pontos;
- Cada jogador inicia o jogo com 14 peças e 18 marcadores;
- Os *Jokers* são dispostos inicialmente de forma aleatória no tabuleiro;

- No seu turno, cada jogador coloca uma peça no tabuleiro, em qualquer posição livre;
- Os padrões utilizados para ganhar pontos são o "X", o "+" e o "cinco em linha";

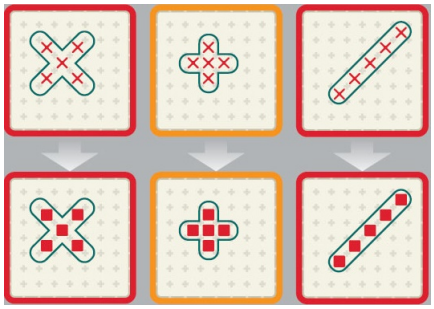
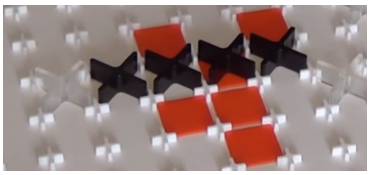
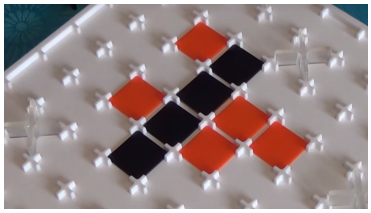


Figura 2: Padrões usados

- O jogador pode usar os *Jokers* para formar padrões, como se das suas próprias peças se tratassem;
- Quando um padrão é formado, retiram-se as peças de jogo e introduzem-se marcadores nessas posições. As peças de jogo podem agora voltar a ser utilizadas e as casas com marcadores também;
- Cada marcador colocado corresponde a um ponto;
- Caso tenha sido usado um *Joker* para formar um padrão, nessa posição não é introduzido um marcador. O *Joker* é agora colocado numa posição ao critério do jogador. Atenção, o *Joker* não pode ser usado para formar um novo padrão de imediato;



(a) Peças



(b) Marcadores

Figura 3: Substituição das peças por marcadores

- O primeiro jogador a atingir o número de pontos, determinado inicialmente, é o vencedor;
- O jogo pode também acabar quando um jogador já não dispõe de peças ou marcadores. Nesse caso, ganha o jogador com mais pontos até ao momento;

3 Lógica do Jogo

3.1 Representação do Estado do Jogo

O jogo é representado por uma lista de seis elementos. O objetivo desta estratégia é armazenar a informação do estado de jogo numa única lista. O primeiro elemento dessa lista corresponde ao estado atual do tabuleiro. O segundo elemento da lista corresponde a uma lista com a informação à cerca das peças disponíveis pelos jogadores. O terceiro elemento determina qual o jogador do turno atual. O quarto elemento da lista corresponde ao modo de jogo, do desafio em curso. O quinto elemento corresponde ao número de pontos com que o jogo deve terminar. Finalmente, o último elemento, corresponde ao nível de dificuldade do computador, quando é necessário.

Na representação exterior, ou seja, quando o tabuleiro é visualizado pelo utilizador, usamos as seguintes propriedades:

- Peças de jogo:
  - espaço em branco = livre (vazio)
  - 0 = *Joker* (transparente)
  - 1 = jogador 1
  - 2 = jogador 2
- Marcadores:
  - espaço em branco = vazio
  - * = marcador do jogador 1
  - : = marcador do jogador 2

Na representação interna, ou seja, nos dados armazenados pelo computador, usamos a seguinte sintaxe:

- Peças de jogo:
  - -1 = livre (vazio)
  - 0 = *joker* (transparente)
  - 1 = jogador 1
  - 2 = jogador 2
- Marcadores:
  - -1 = vazio
  - 11 = base do jogador 1
  - 22 = base do jogador 2

3.2 Visualização do Tabuleiro

A visualização do tabuleiro em modo de texto é feita através da composição de caracteres ASCII, como visto anteriormente. Para este efeito, foram desenvolvidos um conjunto de predicados.

O predicado principal é:

- **printBoard(Board)**: recebe uma lista descritiva das peças que constituem o tabuleiro e imprime-o;

Os predicados auxiliares são:

- **printBoardAux(Board, CurrentNumberVertical)**: chamada auxiliar de *printBoard* que desenha *(Board.length mod 8)* vezes;
- **printHorizontalLine(NumberOfDashes)**: imprime o número de travessões horizontais pretendido;

- **printBlock(Line)**: imprime todos os blocos de uma determinada linha (horizontal);
  - **printLine1(' ', Tile)**: imprime a primeira linha do bloco;
  - **print1(' ', Tile)**: predicado auxiliar de *printLine1*, usada para a chamada recursiva;
  - **printLine2(' ', Tile)**: imprime a segunda linha do bloco;
  - **print2(' ', Tile)**: predicado auxiliar de *printLine2*, usada para a chamada recursiva;
  - **printLine3(' ', Tile)**: imprime a terceira linha do bloco;
  - **print3(' ', Tile)**: predicado auxiliar de *printLine3*, usada para a chamada recursiva;
  - **toPrintMiddle(N)**: verifica se $N > 0 \wedge N < 8$;
  - **printBeginning(_)**: imprime o travessão vertical da primeira linha do bloco;
  - **printMiddle(_)**: imprime o travessão vertical da segunda linha do bloco;
  - **printEnd(_)**: imprime o travessão vertical da terceira linha do bloco;
- **convertCode([Cross|Tile], X, Y)**: traduz o átomo em peça de jogo;
  - **convertCodeAux(Cross, [Tile|_], X, Y)**: predicado auxiliar de *convertCode*;
  - **translateCodeToChar(X, Y)**: traduz o código X para o código Y;
- **printInfo(_)**: apresenta informação à cerca da representação do jogo;

```
Information About How ModX is displayed:
        Cross:
                [whitespace] -> empty
                0 -> joker (transparent)
                1 -> player 1
                2 -> player 2
        Tile:
                [whitespace] -> empty
                * -> tile of player 1
                : -> tile of player 2
        Board Game Example:
                -----------------
                | 1:::1 | 2   2 |
                | ::1:: |   2   |
                | 1:::1 | 2   2 |
                |---------------|
                |       | ***** |
                |       | ***** |
                |       | ***** |
                -----------------
```

Figura 4: Informação geral apresentada

3.3 Lista de Jogadas Válidas

No MOD X não faz sentido ser apresentada uma lista de jogadas válidas, porque uma jogada é sempre válida desde que a célula esteja vazia. Assim, caso esta funcionalidade fosse implementada, limitaria-se a apresentar uma lista com

todas as células vazias. No entanto, essa condição é perfeitamente visível através da observação do tabuleiro e por isso é perfeitamente dispensável.

Apesar de existirem jogos de tabuleiro em que uma peça só pode ser movida mediante algumas restrições, isso não acontece no MOD X. Aqui, o jogador tem toda a liberdade de colocar uma nova peça numa posição vazia do tabuleiro.

O predicado que verifica se uma determinada célula está vazia é o **checkValidPosition(Board, Row, Col)**.

3.4 Execução de Jogadas

Em cada turno, o jogador introduz as coordenadas do tabuleiro onde pretende introduzir a peça através do predicado **putPlayerPiece(Board, Player, NewBoard)**. Esse predicado verifica se o estado da célula do tabuleiro, correspondente às coordenadas introduzidas, está vazia. Essa verificação é efetuada pelo predicado **checkValidPosition(Board, Row, Col)** e caso falhe, é pedido ao jogador para introduzir outras coordenadas. Assim que sejam introduzidas umas coordenadas que correspondam a uma célula vazia, a peça é colocada.

A peça *Joker* merece especial enfoque, porque em caso de se formar um padrão com ela, têm que se recolocadas. No entanto, em caso de recolocação, as coordenadas de destino têm que corresponder a uma posição vazia e o **Joker** não pode ser utilizado para formar novamente um padrão. Assim, o predicado **putJoker(Board, NewBoard)** é responsável por colocar o *Joker* numa nova posição e os predicados **checkValidPosition(Board, Row, Col)** e **checkPatern(Board, Row, Col)** responsáveis pelas respetivas verificações, mencionadas anteriormente.

Finalmente, o predicado changePlayer(Game, NewGame) responsabiliza-se por alternar os turnos entre jogadores, depois de cada jogada estar completa.

3.5 Avaliação do Tabuleiro

A avaliação do tabuleiro é efetuada no final de cada turno. Assim, depois de cada peça colocada no tabuleiro, ele é analisado completamente, verificando-se a existência de algum padrão. O predicado **checkAll(Board, Piece, NewBoard)** é responsável por essa verificação.

Caso algum padrão tenha sido formado, o devido ajuste ao tabuleiro é efetuado pelo predicado **fixBoard(Board, NewBoard, Game, NewGame)**. Atenção que o número de *Jokers* é sempre analisado depois da formação de um padrão, para isso, recorremos ao predicado **assertNumJokers(Board, NewBoard)**.

3.6 Final do Jogo

O predicado **playGame(Game)** é usado várias vezes para o normal funcionamento do programa. Associado à primeira declaração deste predicado, está o predicado **checkEndConditions(Game)**, que falha caso tenha sido encontrada uma condição de fim.

O jogo termina quando o número final de pontos, estabelecido inicialmente, é atingido ou quando o número de peças ou marcadores chega a zero. Para

verificar essas condições, são usados os predicados: **checkPointsEnd(Game)**, **checkPiecesEnd(Game)** e **checkMarkersEnd(Game)**.

O segundo predicado **playGame(Game)**, termina o jogo e declara que é o vencedor.

3.7 Jogada do Computador

A forma mais adequada para proceder à implementação desta funcionalidade, seria através do algoritmo Minimax. No entanto, como isso ultrapassa os objetivos da unidade curricular, decidimos criar um algoritmo *smart* que é responsável pelas jogadas do computador.

O predicado **pcSmartMoveAlgorithm(Board, Player, NewBoard, Game)** cria 20 tabuleiros e avalia-os, escolhendo a melhor opção, ou seja, a que gera mais pontos. Cada um desses tabuleiros é avaliado individualmente, sendo calculada a pontuação resultante de cada um deles. A melhor opção é escolhida.

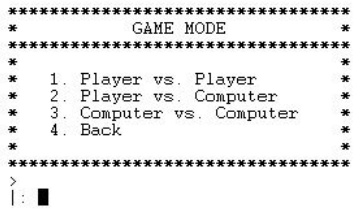
Além deste nível de jogo, está também disponível o modo *Random*. Este nível de jogo, usado em turnos do computador, simplesmente coloca a peça numa posição aleatória.

Dependendo do nível de jogo definido pelo utilizador, o predicado **pcRandomMove(Game, NewGame)** ou **pcSmartMove(Game, NewGame)** são utilizados.

4 Interface com o Utilizador

A interface de linha de comandos permite uma fácil utilização do programa. Os menus são simples e as opções são claras, permitindo ao utilizador selecionar a opção pretendida, tendo atenção a possíveis erros de *input*.

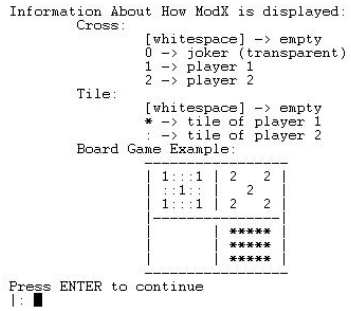
(a) Main Menu



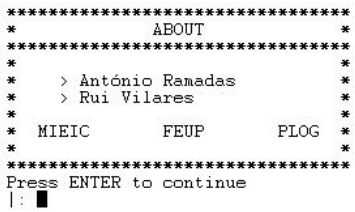
(b) Game Mode

Figura 5: Escolha de modo de jogo

Ao longo da execução do programa, o ecrã vai sendo "limpo", através do predicado **cleanConsole**. Na prática o ecrã não é apagado, apenas são introduzidos espaços em branco para tornar a apresentação do jogo mais agradável.



(a) Help Menu



(b) About Menu

Figura 6: Outras opções do menu principal

Em cada turno, é apresentado o tabuleiro e a informação associada ao jogador atual. Caso seja o turno do jogador, é pedido para serem introduzidas as coordenadas para inserir a peça. Caso seja o turno do computador, pede-se para pressionar ENTER e a peça é adicionada.

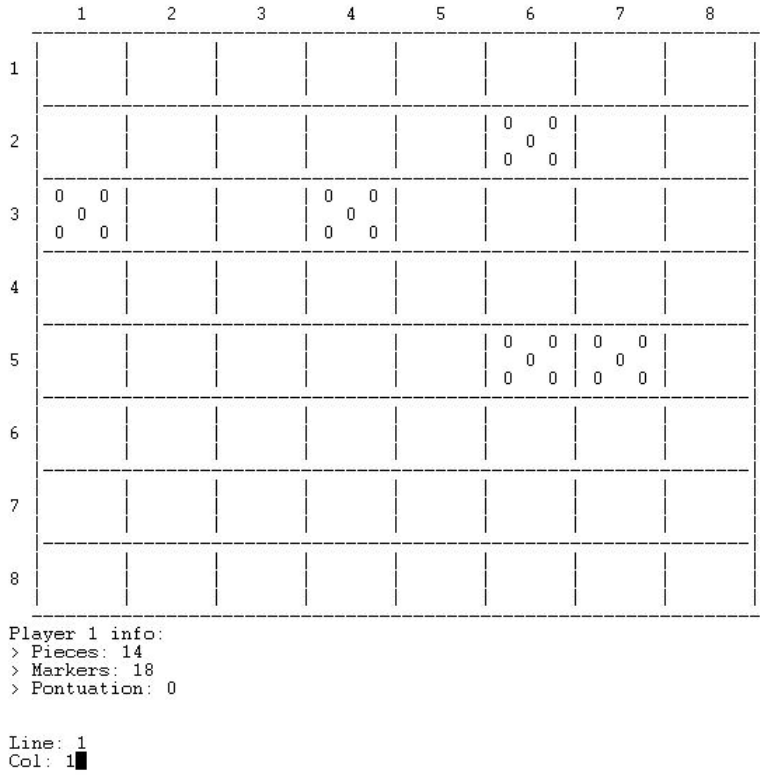


Figura 7: Exemplo de jogo

5 Conclusões

Apesar do exaustivo trabalho que a implementação do jogo MOD X exigiu, o resultado final e os conhecimentos adquiridos ao longo da sua implementação são considerados bastante positivos. Considera-se que todos os objetivos propostos inicialmente foram cumpridos, bem como os critérios estabelecidos inicialmente pelos docentes.

A implementação do MOD X foi um desafio encarado com frontalidade. Importa referir que algumas partes do processo de desenvolvimento mostraram-se complicadas de realizar. Nem sempre a linguagem foi um problema, como seria de esperar. Em alguns casos, as regras próprias do jogo foram a verdadeira entrave.

Como dito anteriormente, todos os objetivos e requisitos propostos foram cumpridos. No entanto, há sempre espaço para melhorias, nomeadamente na implementação do algoritmo *smart*. Poderia ser implementado uma forma anti-jogo, impossibilitando o adversário de formar padrão.

Concluindo, o Prolog é uma linguagem que requer um pensamento lógico. A utilização desta linguagem acabou por ser uma agradável surpresa e por isso, consideramos que os conceitos adquiridos serão benéficos no futuro.

Bibliografia

- https://boardgamegeek.com/boardgame/131387/mod-x
- https://www.cryptozoic.com/games/mod-x
- https://youtu.be/pto-7O618rI
- Apresentações utilizadas nas aulas teóricas
- Livro: Sterling, Leon; The Art of Prolog

Anexos

Código fonte do projeto encontra-se na pasta MODX anexado junto deste relatório.