Shacru

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo 02:
Mário André Macedo Ferreira - ei12105
Tiago Filipe Abreu Figueiredo - ei12069

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

14 de Novembro de 2016

Resumo

O projeto consistiu em desenvolver o jogo de tabuleiro Shacru[1], que pode ser jogado desde 2 até 4 jogadores, utilizando Prolog como linguagem de programação em lógica. A primeira abordagem à implementação foi a representação do estado de jogo, optando-se pela utilização de uma lista de listas para onde '0' representa um espaço vazio e os espaços ocupados por peças de jogadores são representados por 2 dígitos. O das dezenas (1 a 4) identifica o jogador e o das unidades (1 a 8) uma das 8 direções possíveis (4 ortogonais e 4 diagonais). O jogo tem 3 modos de jogo: humano contra humano, humano contra computador e computador contra computador. O computador apenas escolhe posições aleatórias até acertar numa jogada válida. No projeto foi utilizado SWI-Prolog[4] em detrimento de SICStus uma vez que o primeiro permite a impressão de cores (diferentes de preto) na consola, facilitando a representação e melhorando a jogabilidade. Uma das abordagens tentadas no modo difícil do computador foi a implementação do algoritmo Minimax[3] que acabou por não ser utilizado porque não ia de encontro à jogabilidade do Shacru.

Conteúdo

1 Introdução

O objetivo do trabalho consiste no desenvolvimento de um jogo de tabuleiro usando a linguagem de programação Prolog, tendo sido escolhido o jogo Shacru que faz parte de um conjunto de 3 jogos de tabuleiro semelhantes, Pacru, Shacru e Azacru[2]. O desenvolvimento do trabalho passou inicialmente pela representação do estado de jogo em Prolog, tendo-se optado por representar o tabuleiro usando uma lista de listas. Para melhorar a jogabilidade e facilitar a impressão do tabuleiro na consola optou-se por utilizar SWI-Prolog uma vez que permite mudar a cor com que os caracteres são impressos na consola. Depois deste passo, desenvolveu-se a lógica do movimento das peças dos jogadores e implementação da condição de vitória. No final passou-se à implementação de duas dificuldades de jogo contra o computador. Nas restantes secções deste relatório é explicado em detalhe da representação do estado de jogo, a implementação da visualização do tabuleiro, lista de jogadas válidas e execução das mesmas, condições de paragem e jogadas do computador. Na secção 4 é explicada a interação com o utilizador. No final é possível ler algumas conclusões do projeto.

2 O Jogo Shacru

O jogo Shacru foi criado em 2002 por Mike Wellman, mas apenas em 2004 foi publicado pela primeira vez, juntamente com o jogo que lhe deu origem, Pacru. É um jogo de tabuleiro baseado em estratégia e o principal objetivo do jogo é obter o máximo de quadrados da cor do jogador antes de o jogo terminar. Pode ser jogado por um mínimo de 2 pessoas e um máximo de 4. O jogo consiste num tabuleiro quadrangular 9x9 dividido em 9 setores de 3x3 quadrados e cada jogador é representado por uma cor. Cada jogador tem marcadores e setas. Os marcadores servem para marcar as casas onde as peças do jogador passam e as setas são as peças móveis e que para além disso indicam a direção do deslocamento do último movimento. O número de setas a que cada jogador tem direito depende do número de jogadores em jogo, isto é, se forem 2 cada jogador tem direito a 4 setas, se forem 3 ou 4 cada jogador tem direito a 3 setas.

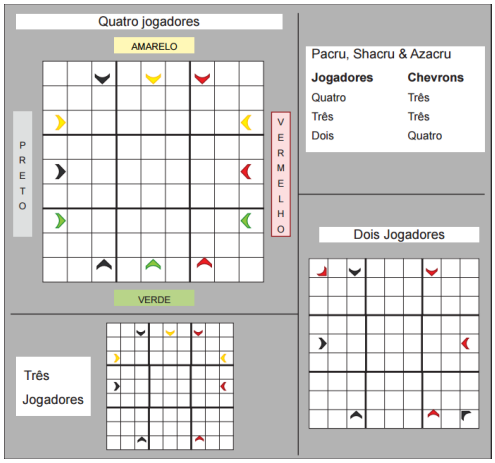


Figura 1: Distribuição inicial das setas dos jogadores, dependendo do número de jogadores Fonte: http://www.pacru.com/rulesPT.pdf

A cada turno, o jogador tem de mexer uma seta numa de 3 direções: para a frente ou num ângulo de 45º para a esquerda ou para a direita. Sempre que

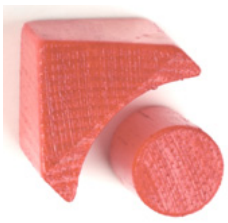


Figura 2: Exemplo de uma peça de shacru. O marcador é a peça cilíndrica e a seta representa a seta do jogo. Fonte: http://www.pacru.com

a seta do jogador muda para um novo campo (quadrícula) a seta fica virada na direção que ocorreu o movimento e o jogador põe nesse mesmo campo um marcador. Opcionalmente, quando o movimento executado resulta na mudança de sector o jogador pode rodar a seta em questão 45º. O jogador não pode mexer a seta para um campo onde já exista um marcador e/ou seta. Um jogador só pode passar o turno quando não tiver movimentos disponíveis.

O jogo termina quando nenhum jogador tiver movimentos disponíveis e/ou os movimentos disponíveis não permitirem a colocação de um novo marcador. Como referido previamente, ganha o jogo o jogador com mais marcadores da sua cor no tabuleiro.

3 Lógica do Jogo

3.1 Representação do Estado do Jogo

Como referido acima, o tabuleiro é quadrangular sendo que cada lado tem uma largura de 9 células. Assim sendo, a abordagem que considerámos mais apropriada para este caso foi criar uma lista de listas em que cada lista consiste numa linha. Assim sendo existem 9 listas de 9 elementos dentro da lista que contem o tabuleiro num total de 81 células, ou seja 81 espaços inicialmente vazios para as peças.

```
initBoard([[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],
           [0,0,0,0,0,0,0,0,0],
           [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],
           [0,0,0,0,0,0,0,0,0],
           [0,0,0,0,0,0,0,0,0]).
```

Listing 1: Representação do tabuleiro vazio em Prolog

Tendo em conta que podem existir até 4 jogadores e as peças têm 8 orientações possíveis, optamos por representar as peças com um número inteiro com 1 dígito no caso de uma casa vazia ou marcador e 2 dígitos no caso de uma seta. Em ambos os casos, o algarismo das unidades representa o jogador a que a peça corresponde (1 4 identifica o jogador e 0 um espaço vazio) e o algarismo das dezenas representa uma das 8 orientações possíveis (referência ao numpad).

```
initBoard2P([[31,22,0,0,0,0,21,0,0],
             [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],
             [62,0,0,0,0,0,0,0,41],
             [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],
             [0,0,82,0,0,0,81,0,72]]).
```

Listing 2: Representação do tabuleiro inicial para 2 jogadores em Prolog

Optamos ainda por guardar as peças de cada jogador numa lista de listas em que cada lista é composta pelas coordenadas e orientação de cada peça.

3.2 Visualização do Tabuleiro

Por forma a imprimir a lista foram declarados predicados em Prolog e chegámos a uma representação que achamos adequada e percetível pelo jogador. Optamos por usar SWI-prolog em vez de SICStus já que as bibliotecas a que recorremos não apresentam diferenças significativas e o primeiro permite o output de cor o que facilita a identificação das peças.

Principais predicados para imprimir o tabuleiro:

```
drawBoard([H|T], Y):-
        write('   +---+---+---++---+---+---++---+---+---+\n'),
        write(Y),
        write('  |'),
        drawLine(H),
        Yi is Y + 1,
        drawBoard(T, Yi).
```

Listing 3: Principal predicado para imprimir tabuleiro

```
drawLine([H|T]):-
        drawCell(H),
        write('|'),
        drawLine(T).
```

Listing 4: Principal predicado para imprimir cada linha do tabuleiro

```
drawCell(Cell):-
        parseCell(Cell, P, Di),
        translateDir(Di, D),
        colorCell(P, D).
```

Listing 5: Predicado para imprimir cada célula do tabuleiro

No predicado *drawCell(+Cell)* são chamados os predicados *parseCell(+Cell, -P, -Di)*, que devolve o identificador do jogo e o número correspondente à direção do conteúdo da célula, *translateDir(+Di, -D)*, que converte o número recebido anteriormente para caracteres mais facilmente identificáveis, e *colorCell(+P, +D)*, que finalmente imprime o conteúdo da célula com a cor adequada.

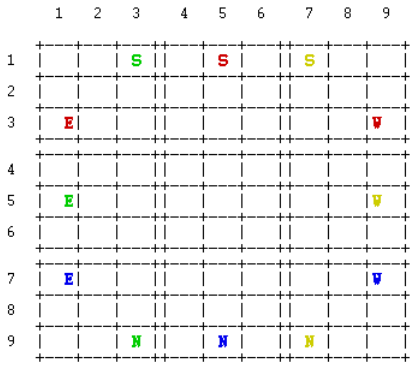


Figura 3: Tabuleiro inicial para 4 jogadores

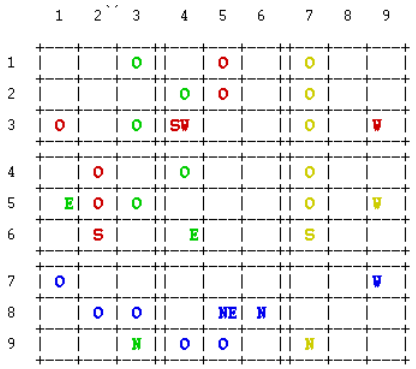


Figura 4: Possível representação do tabuleiro após 5 turnos em ASCII

3.3 Lista e Execução de Jogadas Válidas

1. Mover uma seta que pertence ao jogador.
   Restrições:

   - O movimento apenas se pode efetuar na direção em que a seta se encontra ou 45º para a direita ou esquerda.
   - A seta apenas se pode mover para uma posição vazia.

```
%validateMove(+Board, +Player, +XCoord, +YCoord, +Direction)
validateMove(B, P, X, Y, D):-
```

Listing 6: Predicado para validar o deslocamento que o jogador pretende fazer.

2. Rodar uma seta que pertence ao jogador em 45º.
   Restrição:

   - No mesmo turno, o jogador movimentou a seta entre dois sectores.

```
%validatePosition(+DirectionFromUser, +DirectionFromCell)
validateDirection(D, Di):-
```

Listing 7: Predicado para validar a rotação que o jogador pretende fazer.

Figura 5: Possiveis movimentos do jogador.
Fonte: http://www.pacru.com/rulesdiag/movingshacru.html

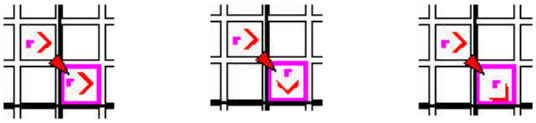


Figura 6: Possiveis rotações da seta.
Fonte: http://www.pacru.com/rulesdiag/bordertwistshacru.html

3.4 Avaliação do Tabuleiro

Depois de um jogador escolher qual a peça que pretende mover e a direção em que a pretende mexer é verificado se o movimento é válido quer no que diz respeito à posição quer à direção.

```
%validateMove(+Board, +Player, +XCoord, +YCoord, +Direction)
validateMove(B, P, X, Y, D):-
        getCell(B, X, Y, C),
        parseCell(C, Pi, Di),
        P == Pi,
        validateDirection(D, Di),
        validatePosition(B, X, Y, D).
```

Listing 8: Predicado que valida o movimento pretendido pelo jogador.

```
%validatePosition(+Board, +XCoord, +YCoord, +DirectionParsed)
validatePosition(B, X, Y, D):-
        genCoords(X, Y, D, Xn, Yn),
        Xn >= 1, Xn =< 9,
        Yn >= 1, Yn =< 9,
        getCell(B, Xn, Yn, C),
        C == 0.
```

Listing 9: Predicado que valida a posição da peça selecionada pelo jogador.

```
%validatePosition(+DirectionFromUser, +DirectionFromCell)
validateDirection(7, Di):-
        member(Di,[4,7,8]).
validateDirection(8, Di):-
        member(Di,[7,8,9]).
validateDirection(9, Di):-
        member(Di,[8,9,6]).
validateDirection(6, Di):-
        member(Di,[9,6,3]).
validateDirection(3, Di):-
        member(Di,[6,3,2]).
validateDirection(2, Di):-
        member(Di,[3,2,1]).
validateDirection(1, Di):-
        member(Di,[2,1,4]).
validateDirection(4, Di):-
        member(Di,[1,4,7]).
```

Listing 10: Predicado que valida o a direção do movimento da peça.

3.5 Final do Jogo

No fim de cada turno, verifica-se se existe algum jogador que ainda tem jogadas disponíveis. Caso ainda exista, é-lhe perguntado quais as coordenadas da peça que pretende mover. Caso não exista nenhum jogador com jogadas disponíveis então o jogo terminou.

```
%nextPlayer(+Board, +CurrentPlayer, -NextPlayer)
nextPlayer(B, 1, P1P, P2P, Pn):-
        (canPlay(B, P2P), Pn is 2);
        (canPlay(B, P1P), Pn is 1);
        Pn is 0,!.
nextPlayer(B, 2, P1P, P2P, Pn):-
        (canPlay(B, P1P), Pn is 1);
        (canPlay(B, P2P), Pn is 2);
        (not(canPlay(B, P1P)), not(canPlay(B, P2P)), Pn is 0),!.
```

Listing 11: Predicado para verificar qual o próximo jogador, verificando para isso se cada um deles tem jogadasa disponíveis.

```
%canPlay(+Board, +PlayerPieces)
canPlay(B, [PP1, PP2, PP3, PP4]):-
        hasMoves(B, PP1);
        hasMoves(B, PP2);
        hasMoves(B, PP3);
        hasMoves(B, PP4),
        !.
```

Listing 12: Predicado para verificar se os jogadores têm jogadas disponíveis.

Por fim, são avaliadas todas as células do tabuleiro de forma a calcular a pontuação de cada jogador.

```
%score(+Board)
score(B):-
        flatten(B, Bf),
        maplist(scoreHelper, Bf, S),
        subtract(S, [0], Si),
        subtract(Si, [2], S1),
        length(S1, SP1),
        subtract(Si, [1], S2),
        length(S2, SP2),
        printScore(SP1, SP2).
```

Listing 13: Predicado para calcular as pontuações finais.

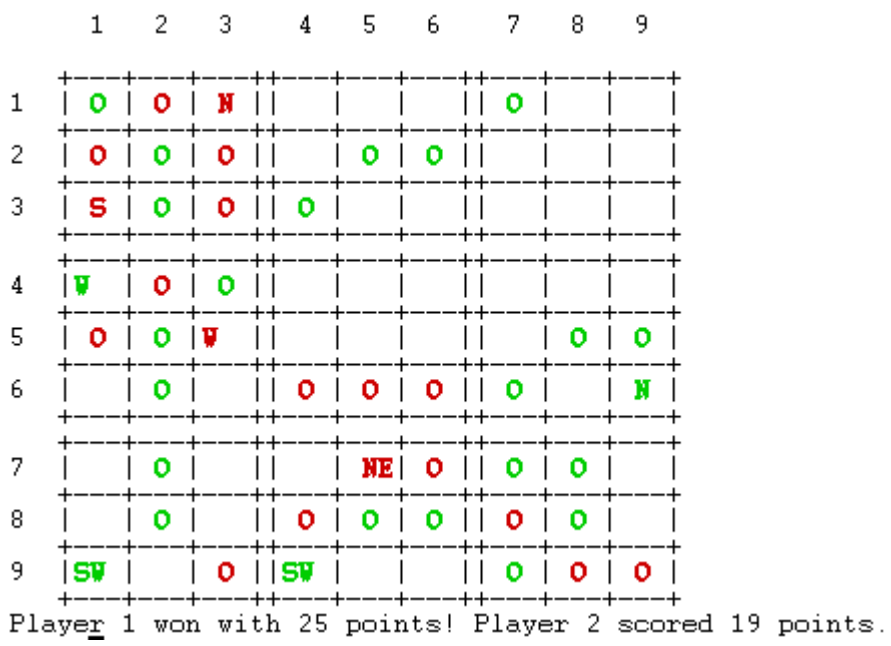


Figura 7: Final do jogo e pontuações.

3.6 Jogada do Computador

Foi implementado apenas o modo fácil, ou seja, o computador tenta posições em X e Y aleatórias até acertar numa jogada válida.

```
%move(+Board, +Player, +XCoord, +YCoord, +Direction, +PlayerPieces,
%       -BoardNew, -PlayerPiecesNew)
moveCPU(B, P, X, Y, D, PP, Bn, PPn):-
        genCoords(X, Y, D, Xn, Yn),
        setCell(B, X, Y, P, Bi),
        member([X, Y, Di], PP),
        delete(PP, [X, Y, Di], PPi),
        genCell(P, D, C),
        setCell(Bi, Xn, Yn, C, Bi2),
        append(PPi, [[Xn, Yn, D]], PPn),
        describeMoveCPU(P, X, Y, Xn, Yn),
        rotation(Bi2, X, Y, Xn, Yn, P, D, Bn).
```

Listing 14: Predicado para executar o movimento do computador.

Para o modo difícil tentou-se a adaptação de um algoritmo Minimax[3] que não resultou porque não se adequava à validação do final do jogo do Shacru. Para além disto tentou-se também, sem sucesso, que o computador tenta-se mover a peça no sentido da maior área de peças de sua cor, ou seja, o movimento não seria totalmente aleatório, pois teria em conta o resto das peças do computador no tabuleiro.

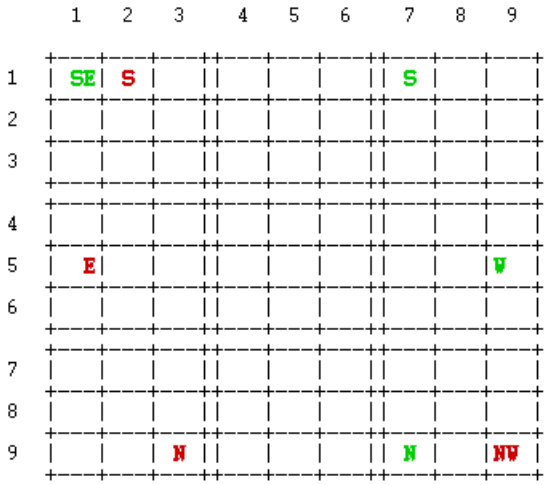
4 Interface com o Utilizador

O jogo inicia-se com a regra "start."e é apresentado um menu onde o utilizador pode escolher o modo de jogo.

```
|    start.
Please choose a game mode:
        1 Human vs Human
        2 Human vs CPU
        3 CPU vs CPU
        0 Exit
Option:
```

Figura 8: Modo de Jogo

Depois de escolher o modo de jogo, é apresentado ao jogador o estado inicial do tabuleiro e o primeiro jogador pode começar a jogar.




```
Player 1 (Green):
Input the X position of the piece you want to move [1-9]:
|: 1
Input the Y position of the piece you want to move [1-9]:
|: 1
Input the Direction in which you want to move (use numpad for reference):
|: 3
```

Figura 9: Início do Jogo. Primeiro jogador (verde) começa a jogar.

Nas jogadas seguintes a interação com o jogador é semelhante. No final do jogo é apresentado o jogador que ganhou e a respetiva pontuação.

5 Conclusões

Relativamente a este projeto percebemos que listas são realmente importantes em Prolog, pois são a forma mais viável de guardar e alterar informação nesta linguagem, especialmente no caso de desenvolvimento de jogos de tabuleiro em que se têm de representar "casas"com ou sem peças. Uma das primeiras dificuldades sentidas foi na forma como o tabuleiro iria ser apresentado ao utilizador, uma vez que o Shacru pode ser jogado por 2, 3 ou 4 jogadores todos com peças da mesma forma mas de cor diferente. Para esta dificuldade a solução encontrada foi usar SWI-Prolog pois permite alterar a cor com que os caracteres são impressos na consola. Quanto aos objetivos propostos só não foi cumprido o de o computador jogar em dois níveis de dificuldade, ou seja, o computador apenas escolhe posições aleatórias até acertar numa jogada válida. Quanto ao modo de dificuldade superior do computador tentaram-se duas soluções. A primeira foi a implementação de Minimax[3], que acabou por não ser usada porque no Shacru não existe uma possibilidade de vitória ou derrota que dependa diretamente da próxima jogada do adversário. A segunda tentativa foi a de o computador ter em conta a posição do resto das suas peças já no tabuleiro. Relativamente ao que foi feito aponta-se principalmente duas coisas a melhorar: melhorar a jogabilidade alterando a forma como as peças são mostradas na consola e implementar um algoritmo de inteligência artificial na jogada do computador.

Bibliografia

[1] Pacru. What why: Pacru, azacru: Shacru. http://www.pacru.com/shacru.html. Online em Novembro 2016.

[2] Pacru. What why: Pacru, azacru: Shacru. http://www.pacru.com/. Online em Novembro 2016.

[3] Gauthier Picard. Artificial intelligence - implementing minimax with prolog. http://www.emse.fr/ picard/cours/ai/minimax/. Online em Novembro 2016.

[4] SWI Prolog. Swi prolog. http://www.swi-prolog.org. Online em Novembro 2016.