Oolong

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Oolong_3:
Ricardo Miguel Oliveira Rodrigues de Carvalho - up201503717
Tiago Lascasas dos Santos - up201503616

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

12 de Novembro de 2017

Resumo

Este trabalho teve como objetivo a implementação do jogo de tabuleiro Oolong usando Prolog. Tendo três modos de jogo principais, esta implementação é interativa, permitindo ao(s) jogador(es) escolher as suas jogadas com base no estado atual do jogo e à realização de jogadas por parte do computador que, com base em dois níveis de dificuldade, realiza jogadas cujo objetivo é causar problemas ao adversário e usando um critério de distinção que nós, como jogadores humanos, usámos ao jogar um contra o outro. Por fim, foram tiradas várias conclusões relacionadas com as capacidades do Prolog na resolução deste problema, das quais se destaca, pela positiva, a sua capacidade de representação de regras e respetiva interação, enquanto que pela negativa se critica as suas capacidades de I/O rudimentares.

Conteúdo

1 Introdução

Este trabalho teve como objetivo a implementação em Prolog do jogo de tabuleiro Oolong e no qual é possível jogar um jogador contra outro, um jogador contra o computador e ainda o computador contra si mesmo. Neste relatório encontra-se primeiramente uma descrição do jogo e das suas regras, ao que se segue uma demonstração da interface com o utilizador e finalmente uma explicação detalhada dos vários elementos da lógica do jogo e da sua implementação. No fim, procede-se à inferência de várias conclusões sobre as capacidades do Prolog na implementação deste sistema.

2 O Jogo Oolong

Oolong é um jogo de tabuleiro para 2 jogadores baseado num paradigma de controlo de área, e foi criado em 2017 pelo estúdio independente Black Straw Games. O tema do jogo é uma competição dentro de uma casa de chá em que o objetivo de cada jogador é, jogando à vez, servir o seu chá (verde ou preto) à maioria das pessoas na maioria das mesas.

O tabuleiro é composto por 9 mesas com 9 lugares, dispostas como indicado na figura 1.

Figura 1: Tabuleiro de Oolong.

Cada mesa, exceto a central, tem um marcador especial, que serão descritos posteriormente. As peças são compostas por 40 de cada cor (preto e verde) mais uma peça vermelha, o empregado. O empregado marca qual a mesa em que a próxima jogada será feita e em que lugar o jogador não pode colocar a sua peça.

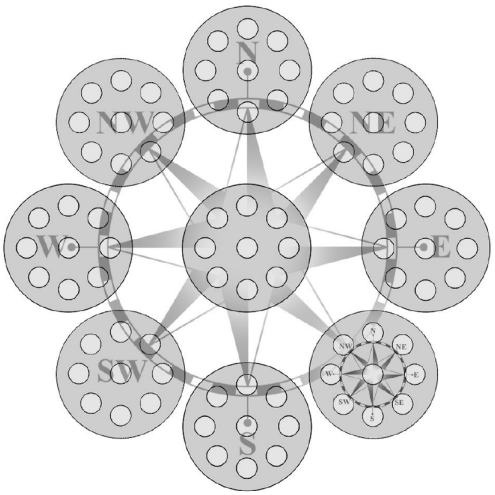

Figura 2: Definição do tabuleiro com pontos cardeais.

O jogo é começado pelo jogador com as peças pretas e com o empregado na posição central da mesa central. Assim, o primeiro jogador tem de jogar numa das 8 posições livres da mesa central e a posição escolhida indicará onde a primeira peça verde será colocada. As mesas e as suas posições estão organizadas segundo os pontos cardeais, com um ponto adicional para sinalizar o centro (Figura 2).

Assim sendo, a jogada seguinte é sempre feita na mesa correspondente ao ponto cardeal onde foi feita a jogada anterior e o empregado é colocado no ponto correspondente à mesa da jogada anterior. Um exemplo do fluxo do jogo pode ser o seguinte:

1. Empregado na mesa Central e casa Central. Jogador preto coloca a peça na casa NW da mesa Central;
2. Empregado na mesa NW e na casa Central. Jogador verde coloca a peça na casa E da mesa NW;
3. Empregado na mesa E e na casa NW. Jogador preto coloca a peça na casa S da mesa E;
4. ...e assim sucessivamente.

O jogo é continuado, à vez, até que um jogador conquiste 5 das 9 mesas. Para conquistar uma mesa, um jogador tem de ocupar pelo menos 5 posições nessa mesa.

Algumas jogadas podem ativar ações especiais, dadas por marcadores que são atribuídos aleatoriamente às mesas, no máximo de um por mesa. As ações possíveis são:

- Um jogador pode mover uma das suas peças de uma mesa para outra sendo que nenhuma destas foi conquistadas.
  - Existem dois marcadores para esta ação, cada um referente a uma das cores das peças dos jogadores.

  - Esta ação é desencadeada com 3 peças da mesma cor na mesa correspondente.

- Um jogador pode mover o empregado da mesa atual para a mesma posição numa mesa à escolha, mudando a mesa em que a próxima jogada é feita.
  - Existem dois marcadores para esta ação, cada um referente a uma das cores das peças dos jogadores.
  - Esta ação é desencadeada com 5 peças da mesma cor na mesa correspondente.

- O jogador que ativa esta ação pode rodar a mesa em qualquer direção (rodando o empregado com ela).
  - Existem dois marcadores para esta ação.
  - Esta ação é desencadeada com 4 peças da mesma cor na mesa correspondente.

- O jogador que ativa esta ação pode trocar quaisquer duas mesas não conquistadas (sem as rodar), movendo também o empregado caso este esteja numa das mesas.
  - Esta ação é desencadeada com 4 peças da mesma cor na mesa correspondente.

- O jogador que ativa esta ação pode trocar qualquer mesa conquistada com uma não conquistada (mantendo a orientação destas), movendo o empregado caso este esteja em qualquer das mesas afetadas.
  - Esta ação é desencadeada com 5 peças da mesma cor na mesa correspondente.

No caso particular de uma jogada (incluindo ações especiais) implicar que o jogador coloque a sua peça numa mesa completa, o jogador pode escolher colocar a peça em qualquer lugar vazio em qualquer das mesas.

3 Interface com o Utilizador

A visualização textual do tabuleiro foi feita representando as nove mesas como matrizes de 3x3 caracteres (sem considerar espaços), sendo que cada caracter representa uma posição nessa mesa. Tanto os caracteres dentro da matriz como as próprias matrizes seguem o modelo de pontos cardeais apresentado anteriormente. As posições tomam o caracter o se estiverem vazias, g se pertencerem ao jogador verde e b se pertencerem ao preto. A cada matriz está associada uma descrição da jogada especial associada, e por fim existe ainda a indicação sobre qual a posição atual do waiter. Na figura 3 podemos ver uma representação esquemática desta interface, e na figura 4 podemos ver a interface em si, neste caso representada no SICStus Prolog.

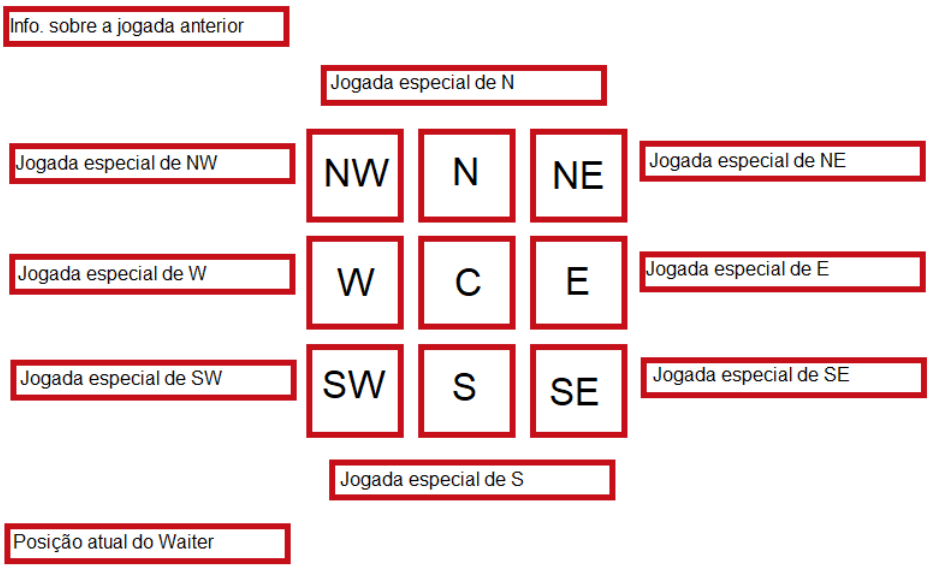

Figura 3: Esquema da interface

```
Player g placed a piece on table ne, position s

                                Green moves waiter
                               o o o   o g o   o o o
 Black moves waiter            o o o   o o o   o o o Move 1 black piece
                               o o o   o b o   o g o

                               o o o   o b b   o o o
 Player rotates table          o o o   o o o   o o o Move 1 green piece
                               o o o   o o o   o o o

                               o o o   o o o   o o o
 Switch 2 unconquered          o o o   o g o   o o o Switch unconquered w/ conquered
                               o o o   o o o   o o o
                               Player rotates table

o - Empty g - Green  b - Black
Waiter in table s and position ne

Position of piece b |: 
```

Figura 4: Exemplo da interface após 3 jogadas de cada jogador.

A nível de input, pode-se iniciar o jogo chamando o átomo *play*. Cada partida começa por perguntar ao jogador qual o tipo de jogo que quer: **'1vs1'**, **'1vsAI'** ou **'AIvsAI'**, entrando num ciclo de processamento que aceita apenas uma destas três alternativas. É também requisitado um nível de dificuldade, com dois valores possíveis (**easy** e **hard**), caso o tipo de jogo seja '1vsAI' ou 'AIvsAI'. No tipo de jogo '1vsAI', o jogador humano será sempre o que usa as peças pretas.

Dependendo das escolhas feitas, caso haja pelo menos um jogador humano é sempre pedida uma jogada em que normalmente é necessário indicar uma posição dentro de uma mesa pré-estabelecida pelas regras do jogo. Prompts adicionais ocorrem para as situações de jogada especial e de quando a mesa está cheia, sendo que todos os inputs, tal como os de início de jogo, estão num ciclo de processamento que aceita apenas escolhas válidas.

Caso o jogo seja terminado a meio através de uma interrupção do Prolog é necessário reconsultar o ficheiro *oolong.pl* de modo a limpar a base de dados. Caso termine normalmente, contudo, a base de dados é normalizada e uma nova partida pode ser iniciada normalmente e sem consequências.

4 Lógica do Jogo

4.1 Representação do Estado do Jogo

Para representar o estado do jogo recorremos à base de dados interna do Prolog, criando um predicado para representar uma posição, *pos(Table, Position, Content)*, e usando factos desse predicado para representar cada posição. Table indica a mesa, Position a posição dentro da mesa e Content o conteúdo dessa posição. O estado inicial contém todas as posições vazias, o que é internamente representado por um o. Table e Position tomam valores referentes a pontos cardeais (n, s, e, w, nw, ne, sw, se), mais o valor c para representar o centro, seguindo exatamente a mesma distribuição da figura 2, e Content toma o valor de o, b ou g (vazio, peça preta ou peça verde, respetivamente). Existe também um predicado, *waiterPos(T, P)*, que indica qual a posição do waiter. A razão pela qual o waiter não poder ser representado pelos predicado *pos/3* prende-se com o facto de o waiter poder partilhar uma posição com uma casa ocupada.

Existem ainda outros predicados relacionados com o estado do jogo: *gameType/1* indica qual o tipo de jogo ('1vs1', '1vsAI' ou 'AIvsAI'), *currentPiece/1* qual o jogador atual (b ou g) e outros predicados relacionados com jogadas especiais.

4.2 Visualização do Tabuleiro

Visto que o estado do jogo está contido na própria base de dados do Prolog sobre a forma de factos, não é necessário passar o estado do jogo ao predicado de visualização. Como tal, em vez de um predicado recorreu-se a um átomo, *printBoard*, que realiza a impressão das nove mesas linha a linha, usando um novo átomo para cada (*printRow1*, *printRow2*, etc). Estes átomos, por sua vez, mostram cada um três linhas, uma de cada mesa, recorrendo a três predicados (*printTableTop(Table), printTableMiddle(Table)* e *printTableBottom(Table)*), cada um com aridade 1 e cuja variável é a mesa a que se referem. Finalmente, estes predicados usam o predicado *printPos(Table, Pos)*, que imprime o conteúdo da

posição Pos da mesa Table. É realizada ainda a impressão das jogadas especiais por mesa, da posição do waiter e da última jogada feita.

4.3 Lista de Jogadas Válidas

A lista de jogadas válidas é dada pelo predicado *getValidPositions(Table, Positions, ValidPositions)* que, para uma determinada mesa Table, cria uma lista, ValidPositions, com as jogadas válidas entre todas as Positions. De modo a determinar se uma jogada é válida faz uso do predicado *validPosition(Table, Position)*, que sucede se a posição Position da mesa Table estiver vazia, isto é, sem nenhuma peça preta ou verde ou com o waiter.

4.4 Execução de Jogadas

Uma jogada regular é, como anteriormente descrito, direcionada pelo empregado, sendo geralmente apenas necessário ao utilizador indicar a posição pretendida dentro da mesa onde este se encontra. No entanto, se o empregado estiver numa mesa cheia, verificado com recurso a *checkFullTable/1*, o jogador pode escolher qualquer posição em qualquer mesa para colocar a peça.

Para descobrir a posição pretendida foi criado o predicado *getMove(Table, Position)*, que unifica nos seus argumentos a mesa e posição para colocar a próxima peça. Esta função faz a separação e o devido tratamento caso a jogada seja feita por um jogador humano ou pelo computador. Nesta secção apenas será tratada a validação e execução da jogada, visto que o método usado para gerar uma jogada por parte do computador é descrito na secção 4.6.

A validação de cada tentativa de jogada é feita com recurso aos predicados *validInput/1* e *validPosition/2*. O predicado *validInput/1* assegura que o valor introduzido é uma coordenada, correspondendo a uma mesa ou uma posição ou ainda a 'stop', opção que permite sair do jogo. O predicado *validPosition/2*, descrito anteriormente, falha caso a posição já esteja ocupada.

Para a execução da jogada é usado o predicado *move/2*, que atualiza o tabuleiro, coloca a peça no lugar pretendido, faz a chamada a *checkSpecialMove* (descrita na secção seguinte), atualiza a posição do empregado de acordo com as regras anteriormente descritas e chama o átomo *flipCurrentPiece*, que faz a troca do jogador.

4.5 Execução de Jogadas Especiais

Para além das jogadas regulares, acima descritas, este jogo inclui ainda um conjunto de jogadas especiais, tal como descrito nas regras do jogo previamente esclarecidas.

Com vista à implementação correta destas jogadas foram tidos em conta os seguintes aspetos: a distribuição das jogadas pelas mesas, a verificação se alguma jogada especial foi ativada, assegurar que cada jogada só é ativada uma vez e a implementação das jogadas em si.

Seguindo as regras do jogo, a distribuição das jogadas especiais pelas mesas é feita de forma aleatória no início da partida. Há oito jogadas especiais a serem distribuídas por todas as mesas exceto a central, e para tal foi criado um átomo, *initSpecMoves*, que faz chamadas aos predicados *initSpecMoves/3* e *activateSpecMoves/0*, seguidamente descritos.

Na chamada a *initSpecMoves/3* é usada uma lista com as mesas possíveis ([n, s, e, w, nw, ne, sw, se]), a quantidade de mesas possíveis e uma lista com

as jogadas especiais em que cada elemento é uma lista respeitando o formato: *[predicado da jogada especial, numero de peças para ativar a jogada, jogador alvo da jogada]*. O predicado *initSpecMoves/3* é recursivo, registando na base de dados interna, em predicados do tipo *specMovePos/4*, os elementos consecutivos da lista de jogadas e elementos da lista de mesas escolhidos aleatoriamente, seguidamente chamando-se com os restantes elementos de cada lista até estas ficarem vazias.

O átomo *activateSpecMoves* regista na base de dados interna do Prolog, em factos *specialMoveActive/1*, as mesas nas quais ainda podem ser ativadas jogadas especiais, que inicialmente são todas exceto a central.

Para verificar se uma jogada ativou uma jogada especial foi criado o átomo *checkSpecialMove*, chamado depois de ser finalizada uma nova jogada mas antes de ser atualizada a posição do empregado ou de ser atualizado o jogador. Assim, o predicado faz a verificação baseando-se na mesa do empregado para saber onde foi feita a jogada e, portanto, onde pode ter sido ativada a jogada especial. Caso a mesa em questão seja a central o predicado passa e o jogo é resumido, caso contrário, é verificada a jogada especial atribuida à mesa, registando o nome do predicado, a quantidade de peças para ativação e o jogador alvo da jogada e é feita a contagem, com recurso a *getPieceCountOnTable/3* do número de peças do ultimo jogador a jogar na mesa em questão. Caso o número de peças seja inferior ao requerido para ativar a jogada ou a jogada já não esteja ativa, o jogo é resumido, e caso contrário a jogada é ativada. Ao ativar uma jogada especial, a mesa é retirada das jogadas ativas e, com recurso ao operador *univ*, é chamado o predicado para a jogada especial ativada que pergunta ao utilizador os parâmetros a ser usados na jogada em questão.

Na preparação da jogada especial é também tido em conta se o jogador que vai ser alvo da jogada é um jogador humano ou virtual. Essa distinção é possível através do modo de jogo e da peça alvo, isto é, caso o jogo seja 1vsAI e a peça alvo seja verde (g) ou AIvsAI, o alvo da jogada é AI, caso contrário, caso o jogo seja 1vsAI e a peça alvo seja preta (b) ou 1vs1, então o alvo é um jogador humano.

- Para a jogada especial **'mover uma peça de um jogador'** é chamado pelo *checkSpecialMove* o predicado *specialMovePiece/2*, que obtém a informação necessária para se efetuar a jogada com o predicado *specialMovePiece/4*.
  - No caso de a jogada ter como jogador alvo o computador, *specialMovePiece/2* seleciona aleatoriamente uma peça sua e uma posição livre para fazer a troca.
  - Caso a jogada tenha como jogador alvo um jogador humano, é perguntado ao utilizador qual a posição de origem e a posição alvo, verificando sempre se a opção é válida.
  - Após um dos dois casos anteriores suceder, é chamado o predicado *specialMovePiece/4* com as coordenadas da peça e do lugar livre escolhido e é feita a troca, mudando o conteúdo da base de dados interna.
- Para a jogada especial **'um jogador muda a mesa do empregado, mantendo a sua posição'** é chamado pelo *checkSpecialMove/0* o predicado specialMoveWaiter/2, que obtém a informação necessária para se efetuar a jogada com o predicado *specialMoveWaiter/1*.

  - No caso de a jogada ter como jogador alvo o computador, *specialMoveWaiter/2* seleciona aleatoriamente uma mesa diferente da atual mesa do empregado.
  - Caso a jogada tenha como jogador alvo um jogador humano, é perguntado ao utilizador qual a mesa de destino do empregado, verificando sempre se a opção é válida.
  - Após um dos dois casos anteriores suceder, é chamado o predicado *specialMoveWaiter/1* com a mesa de destino e é feita a troca, mudando a mesa do empregado e mantendo a posição.

- Para a jogada especial **'jogador atual pode rodar a mesa atual'** é chamado pelo *checkSpecialMove* o predicado *specialMoveRotate/2*, que obtém a informação necessária para se efetuar a jogada com o predicado *specialMoveRotate/1*.

  - No caso de a jogada ter como jogador alvo o computador, *specialMoveRotate/2* seleciona aleatoriamente um valor entre 0 e 8 para fazer a rotação à direita.
  - Caso a jogada tenha como jogador alvo um jogador humano, é perguntado ao utilizador quantas posições a mesa será rodada para a direita, sendo sempre verificado que o número está entre 0 e 8.
  - Após um dos dois casos anteriores suceder, é chamado o predicado *specialMoveRotate/1* com a quantidade desejada e é feita a rotação, convertendo o conteúdo da mesa numa lista cuja cabeça é o centro (que não é rodado) e cujo conteúdo da cauda resto tem uma ordem bem definida, rodando a cauda da lista para a direita tantas unidades como especifivado, com recurso ao predicado *append/3*, e reconstruindo a mesa a partir da nova lista.

- Para a jogada especial **'jogador atual pode trocar quaisquer duas mesas não conquistadas'**, bem como para a jogada '**jogador atual pode trocar qualquer mesa não conquistada por uma mesa conquistada'**, é chamado pelo *checkSpecialMove* o predicado *specialMoveSwitch/2*, que obtém a informação necessária para se efetuar a jogada com um predicado semelhante. Neste caso, a distinção feita é entre o número de peças para ativar a jogada, no primeiro caso é 4 e no segundo é 5, este aspeto também permite distinguir estes predicados do que efetiva a jogada porque onde estes predicados recebem um numero (4 ou 5) o outro predicado recebe a coordenada de uma mesa.

  - Caso o caso seja troca de duas mesas não conquistadas:
    * No caso de a jogada ter como jogador alvo o computador. *specialMoveSwitch/2* seleciona aleatoriamente duas mesas em que a contagem de peças de nenhum jogador exceda 4.
    * Caso a jogada tenha como jogador alvo um jogador humano, é perguntado ao utilizador quais as mesas a trocar, verificando sempre que estas estão, de facto, não conquistadas.
  - Caso contrário, para troca de uma mesa não conquistada por uma mesa conquistada:

    - No caso de a jogada ter como jogador alvo o computador, *specialMoveSwitch/2* seleciona aleatoriamente duas mesas em que na primeira a contagem peças de pelo um jogador exceda 4 e outra em que esta contagem não exceda 4 para qualquer jogador.
    - Caso a jogada tenha como jogador alvo um jogador humano, é perguntado ao utilizador quais as mesas a trocar, verificando sempre que a primeira está conquistada e a segunda não.

- Após um dos dois casos anteriores suceder, é chamado o predicado *specialMoveSwitch/2* com as coordenadas das mesas a trocar e são trocados os conteúdos das mesas bem como a posição do empregado caso a próxima posição deste esteja em qualquer das mesas a trocar, neste caso é sinalizado que, ao retomar o jogo normal, o empregado já não tem de ser atualizado.

4.6 Avaliação do Tabuleiro e Jogada do Computador

Como indicado previamente, o jogador artificial possui dois níveis de dificuldade, easy e hard, sendo que a escolha da jogada a realizar é feita de forma muito diferente para cada dificuldade. Para esse efeito é usado o predicado *getMoveAI/1*.

Para o caso da dificuldade easy este predicado limita-se a escolher aleatoriamente uma das jogadas possíveis, sem qualquer consideração sobre as consequências dessa jogada. No caso em que uma mesa está cheia, a nova mesa a escolher é também gerada aleatoriamente. Esta última jogada, dada a sua raridade, é realizada da mesma forma independentemente da dificuldade.

Para o caso da dificuldade hard é realizada uma avaliação das jogadas possíveis: cada jogada possível determina qual a mesa em que o adversário vai jogar, sendo que se escolhe a posição que aponta para a mesa em que o adversário tem menos peças. Esta condição faz com que o adversário seja obrigado a espalhar mais as suas peças, dificultando a aglomeração de muitas peças numa mesma mesa, o que, por sua vez, dificulta a conquista desse tabuleiro. Esta avaliação e escolha é feita usando os predicados *getBestPosition(Table, Position)*, que primeiramente verifica quais as jogadas válidas na mesa atual, Table, e de seguida determina a posição Position a jogar usando o predicado *getMinimalTable(validMoves)*, que determina, a partir das jogadas válidas, qual a jogada a escolher com base nas condições acima estabelecidas.

4.7 Final do Jogo

A verificação da condição de terminação é feita pelo átomo *checkVictory*, que verifica se um dos jogadores cumpre a condição de vitória chamando o predicado *checkVictoryPlayer(Player)*, e falhando se nenhum dos jogadores a cumprir.

O predicado *checkVictoryPlayer(Player*) conta o número de mesas em que o jogador Player tem 5 ou mais peças, e tem sucesso caso essa contagem for igual ou maior que 5. Esta contagem é feita pelo predicado *countTablePieces(Player, Tables, Cnt)* que, para cada mesa em Tables (que é uma lista com as coordenadas de todas as mesas) conta o número de peças pertencentes a Player aí existentes, incrementando Cnt caso essa contagem seja igual ou maior que 5, usando o predicado *getPieceCountOnTable(Player, Positions, Cnt)* para esse efeito.

O predicado *getPieceCountOnTable(Player, Table, Cnt)* conta o número de peças pertencentes a player existentes na mesa Table. Esta contagem é feita

usando o predicado *findall/3*, que agrupa todos os factos de *pos/3* com mesa e jogador comuns numa lista, sendo que de seguida se unifica Cnt com o tamanho dessa lista.

5 Conclusões

Com este trabalho foi possível concluir que o Prolog é ideal para a representação e execução de regras complexas usando escassas linhas de código, o que não seria possível noutros paradigmas de programação e tornando-o numa ferramenta ideal para a representação e aplicação das regras de um jogo de tabuleiro, nos quais o Oolong se insere. Contudo, foi também notada uma certa rudimentariedade no que diz respeito à manipulação de input e output, o que levou à criação de uma interface não muito agradável e intuitiva. Uma das maiores dificuldades encontradas prendeu-se com as jogadas do computador e da quantificação da qualidade dessa jogada, tendo-se encontrado uma solução (escolher a jogada que faz com que o adversário tenha de jogar na mesa em que tenha menos peças, tal como descrito previamente) que tenta imitar a estratégia que nós, como humanos, usámos ao jogar o jogo um contra o outro. Para finalizar, foi possível também aferir certas conclusões no que diz respeito à representação do estado do jogo usando factos: enquanto que é uma representação elegante, intuitiva e simples, inserir e retirar factos da base de dados é um processo que deve ser feito com cuidado, de modo a que não se deixem factos desnecessariamente instanciados depois da terminação do predicado principal, visto que estes podem influenciar partidas futuras caso ainda existam. Uma representação alternativa poderia ser realizada com listas, o que decerto traria vantagens e desvantagens diferentes. Seja como for, fizemos amplo uso de listas em tudo o que não estivesse relacionado diretamente com alterações ao estado interno do jogo, nomeadamente nas jogadas especiais e nas jogadas do computador, o que nos permitiu concluir que, apesar de serem uma forma menos legível e intuitiva de representar conhecimento, possuem bastante flexibilidade na sua manipulação, o que se provou ser extremamente útil.

6 Bibliografia

- Regras básicas e tabuleiro do Oolong (consultado 2017-11-12):
  `http://blackstrawgames.com/portfolio/oolong/`
- História do Oolong (consultado 2017-11-12):
  `https://boardgamegeek.com/boardgame/176212/oolong`
- Regras detalhadas do Oolong (consultado 2017-11-12):
  `http://unpub.net/games/detail/?proto=740&o=636`
- Documentação dos predicados built-in e bibliotecas do Prolog (consultados 2017-11-12):
  `http://www.swi-prolog.org/pldoc/doc_for?object=root`
  `https://sicstus.sics.se/sicstus/docs/latest4/pdf/sicstus.pdf`