Dominup em Prolog

Relatório Intercalar

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Dominup4:
Ângela Filipa Pereira Cardoso - up200204375
Nuno Miguel Rainho Valente - up200204376

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

11 de Outubro de 2015

1 O Jogo Dominup

Dominup é uma variação do jogo Dominó para 2 a 4 jogadores, em que, tal como o nome sugere, é possível colocar peças em cima de outras.

No típico Dominó existem 28 peças duplas numeradas de 0 a 6, à semelhança das faces de um dado. Já no Dominup há 36 peças duplas numeradas de 0 a 7, usando códigos binários: o ponto no centro representa 1, o circulo pequeno representa 2 e o circulo grande representa 4, como se pode ver na figura 1. Este desenho das peças, juntamente com as regras do Dominup e de dois outros jogos, foram criadas por Néstor Romeral Andrés em 2014, sendo o conjunto publicado por nestorgames[1].

Figura 1: Exemplo da peça $3 \cdot 6$.

Existem dois tipos de colocação de peças no Dominup:

- subir - a peça é colocada em cima de duas peças adjacentes que estejam ao mesmo nível, de forma a que os números da peça colocada sejam iguais aos que ficam por baixo (um em cada peça de suporte), tal como mostra a figura 2.
- expandir - a peça é colocada na superfície de jogo, de forma a que fique adjacente e ortogonal a pelo menos uma peça já colocada, como, por exemplo, as duas peças já colocadas na figura 2.

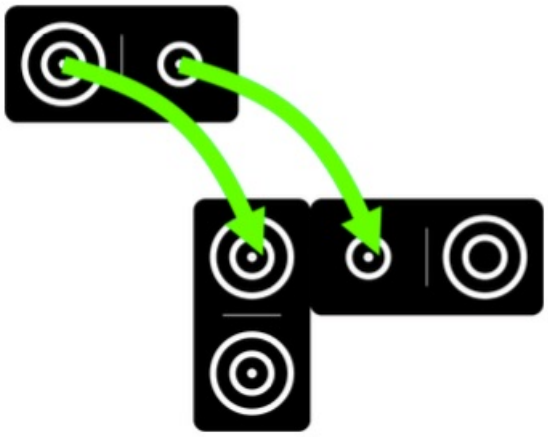

Figura 2: Exemplo de um posicionamento a subir válido.

Tal como no Dominó, as regras são relativamente simples. Começa-se por distribuir as peças aleatoriamente e de forma equilibrada pelos jogadores, mantendo a face voltada para baixo.

O jogador com o duplo 7 inicia o jogo, colocando essa peça no centro da superfície de jogo e determinando a ordem dos restantes jogadores, que é dada pelo sentido contrário ao ponteiro dos relógios.

Começando no segundo, cada jogador, na sua vez, realiza ambos os passos seguintes:

[1] http://www.nestorgames.com

1. Enquanto for possível, coloca peças a subir, podendo escolher a ordem em que o faz;

2. Se ainda tiver alguma peça, coloca-a a expandir.

Se, no final da sua vez, o jogador ficar sem peças, é declarado vencedor e o jogo termina. Alternativamente, os restantes jogadores podem continuar, de forma a determinar o segundo, terceiro e quarto lugares.

Na figura 3 pode ser observado um possível jogo de Dominup a decorrer.

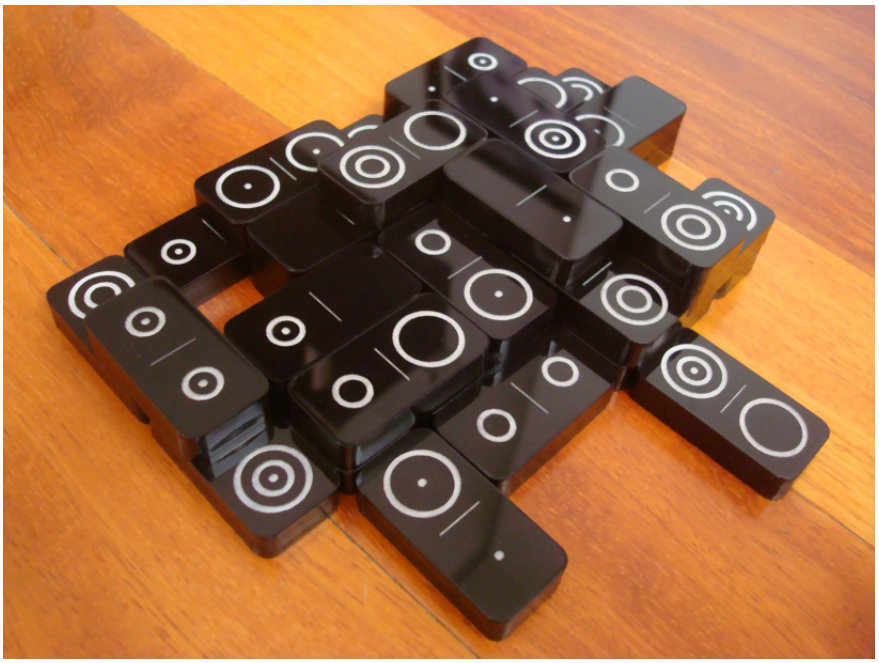

Figura 3: Exemplo de um jogo de Dominup.

2 Representação do Estado do Jogo

O tabuleiro de jogo no Dominup é um quadriculado, cujo tamanho não deve limitar o posicionamento de peças a expandir. Uma vez que expansões sucessivas têm que ser feitas ortogonalmente, uma linha expansiva numa só direção ocupa $2 + 1 + 2 + 1 + \cdots$ quadrículas, assumindo que a primeira peça está colocada horizontalmente. Ao todo temos 36 peças, por isso seriam necessárias $2*18+18 = 54$ quadrículas para acomodar uma tal linha de expansão. Dado que a primeira peça é colocada no centro do tabuleiro, se a expansão fosse feita sempre no mesmo sentido, poderíamos ter que considerar um tabuleiro com 108 quadrículas de lado.

Uma análise mais cuidada das regras do jogo revela que as peças são preferencialmente colocadas a subir. De facto, em cada vez, um jogador coloca tantas peças a subir quanto possível e no máximo uma peça a expandir. Além disso, em geral não será boa estratégia para nenhum dos jogadores expandir sempre no mesmo sentido. Tendo em consideração as limitações de um computador, quer em termos de capacidade de processamento, quer em termos de tamanho do ecrã, decidimos considerar um tabuleiro quadriculado de lado 24. Desta forma, é possível colocar pelo menos 8 peças em cada um dos 4 sentidos a partir do centro do tabuleiro. Eventualmente, com a experiência, podemos decidir reduzir ou aumentar este tamanho. O ideal é que o tabuleiro nunca limite as jogadas, mantendo-se tão pequeno quanto possível para facilitar o processamento.

Sendo assim, o tabuleiro em Prolog é representado por uma lista `board` de 24 linhas do tabuleiro. Por sua vez, cada linha é uma lista de 24 elementos, um para cada quadrícula. Cada elemento de uma linha é um termo complexo `halfPiece(Number, Level, Cardinal)` representando a meia peça de dominó lá colocada, com o seguinte significado:

- `Number` é o número da meia peça;
- `Level` é o nível do tabuleiro em que a peça está colocada, 1 se for colocada em cima do tabuleiro, 2 se for colocada em cima dessa, etc;
- `Cardinal` é o ponto cardeal que indica a posição da outra metade da peça.

Assim, numa quadrícula vazia temos `halfPiece(_, _, _)`. Quando é colocado o dominó duplo 7, teremos `halfPiece(7, 1, e)` e `halfPiece(7, 1, o)` em duas quadrículas adjacentes. Numa fase mais avançada pode ser colocado o dominó $2 \cdot 6$ no nível 3 na vertical com `halfPiece(2, 3, n)` e `halfPiece(6, 3, s)`.

Além do tabuleiro, o estado de jogo contém as peças de cada jogador. Dado que apenas consideramos dois jogadores nesta implementação, teremos as listas `set1` e `set2` com as peças dos jogadores 1 e 2, respetivamente. À medida que vão sendo colocadas no tabuleiro, as peças são retiradas destas listas. Cada peça é representada pelo termo complexo `piece(Number1, Number2)`, por exemplo, `piece(2, 6)`, convencionando-se que o primeiro número é sempre menor ou igual ao segundo.

Finalmente, o estado de jogo terá indicação de qual é o próximo jogador a jogar na variável `playerTurn`, que pode ser 1 ou 2. Assim, o estado de jogo é representado pelo termo `state(set1, set2, playerTurn, board)`.

Um possível estado inicial, já com as peças distribuídas aleatoriamente pelos jogadores, é dado por:

- ```
  set1([piece(0, 1), piece(0, 3), piece(0, 4), piece(0, 6), piece(1, 1),
        piece(1, 2), piece(1, 4), piece(1, 5), piece(1, 7), piece(2, 2),
        piece(2, 3), piece(2, 6), piece(3, 3), piece(3, 7), piece(4, 5),
        piece(5, 6), piece(6, 7), piece(7, 7)]);
  ```
- ```
  set2([piece(0, 0), piece(0, 2), piece(0, 5), piece(0, 7), piece(1, 3),
        piece(1, 6), piece(2, 4), piece(2, 5), piece(2, 7), piece(3, 4),
        piece(3, 5), piece(3, 6), piece(4, 4), piece(4, 6), piece(4, 7),
        piece(5, 5), piece(5, 7), piece(6, 6)]);
  ```
- ```
  playerTurn(1);
  ```
- ```
  board([[halfPiece(_, _, _), halfPiece(_, _, _), ..., halfPiece(_, _, _)],
         [halfPiece(_, _, _), halfPiece(_, _, _), ..., halfPiece(_, _, _)],
         ...,
         [halfPiece(_, _, _), halfPiece(_, _, _), ..., halfPiece(_, _, _)]]).
  ```

Após algumas jogadas, podemos chegar ao estado ilustrado pela Figura 4. Em cima estão as peças do jogador 1, dado que esse é o próximo a jogar. Em baixo está o tabuleiro com as peças que contém até então. Cada meia peça colocada no tabuleiro tem do lado esquerdo o seu número e do lado direito o nível em que se encontra. Este estado intermédio é dado pelos seguintes termos:

- ```
  set1([piece(0, 1), piece(0, 3), piece(0, 4), piece(0, 6), piece(1, 2),
        piece(1, 4), piece(1, 5), piece(1, 7), piece(2, 2), piece(2, 6),
        piece(3, 7), piece(4, 5), piece(5, 6), piece(6, 7)]);
  ```
- ```
  set2([piece(0, 0), piece(0, 2), piece(0, 7), piece(1, 3), piece(1, 6),
        piece(2, 7), piece(3, 4), piece(3, 6), piece(4, 4), piece(4, 6),
        piece(5, 5), piece(5, 7)]);
  ```
- ```
  playerTurn(1);
  ```
  ```

- `board([[halfPiece(_, _, _), halfPiece(_, _, _), ..., halfPiece(_, _, _)],`
  `...,`
  `[halfPiece(_, _, _), halfPiece(_, _, _), ..., halfPiece(_, _, _)],`
  `[..., halfPiece(1, 1, s), halfPiece(7, 1, e), halfPiece(7, 2, s), ...]`
  `[..., halfPiece(1, 1, n), halfPiece(_, _, _), halfPiece(4, 2, n), ...]`
  `[..., halfPiece(2, 3, e), halfPiece(5, 3, o), halfPiece(0, 1, o),`
  `halfPiece(6, 1, s), ...]`
  `[..., halfPiece(3, 2, n), halfPiece(3, 2, n), halfPiece(_, _, _),`
  `halfPiece(6, 1, n), ...]`
  `[halfPiece(_, _, _), halfPiece(_, _, _), ..., halfPiece(_, _, _)],`
  `...,`
  `[halfPiece(_, _, _), halfPiece(_, _, _), ..., halfPiece(_, _, _)]]).`

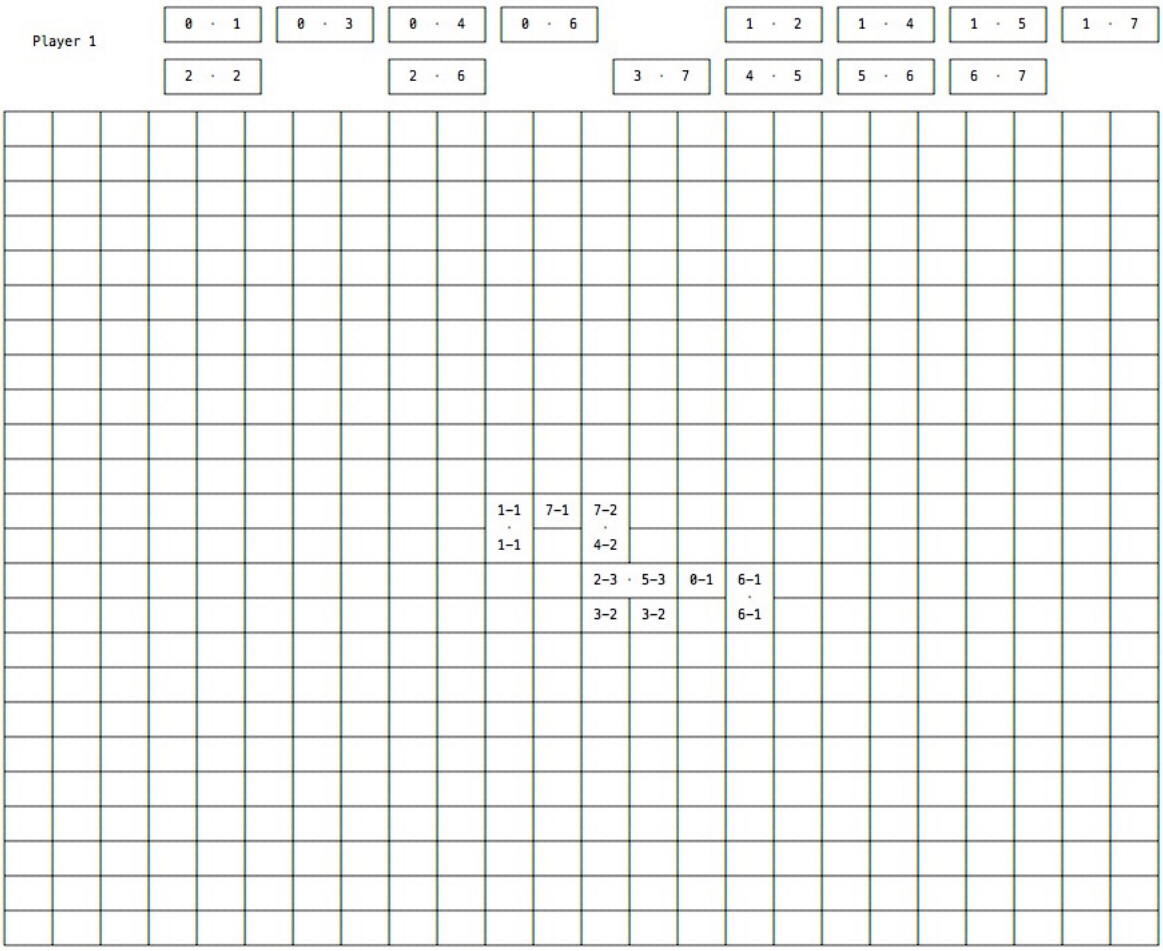

Figura 4: Estado intermédio do jogo.

Um possível estado final, correspondente ao jogo da Figura 3, deverá ter a ilustração da Figura 5 e, à exceção do tabuleiro, cuja representação é demasiado complexa para traduzir neste relatório, tem os seguintes termos:

- `set1([]);`

- `set2([]);`
- `playerTurn(2);`

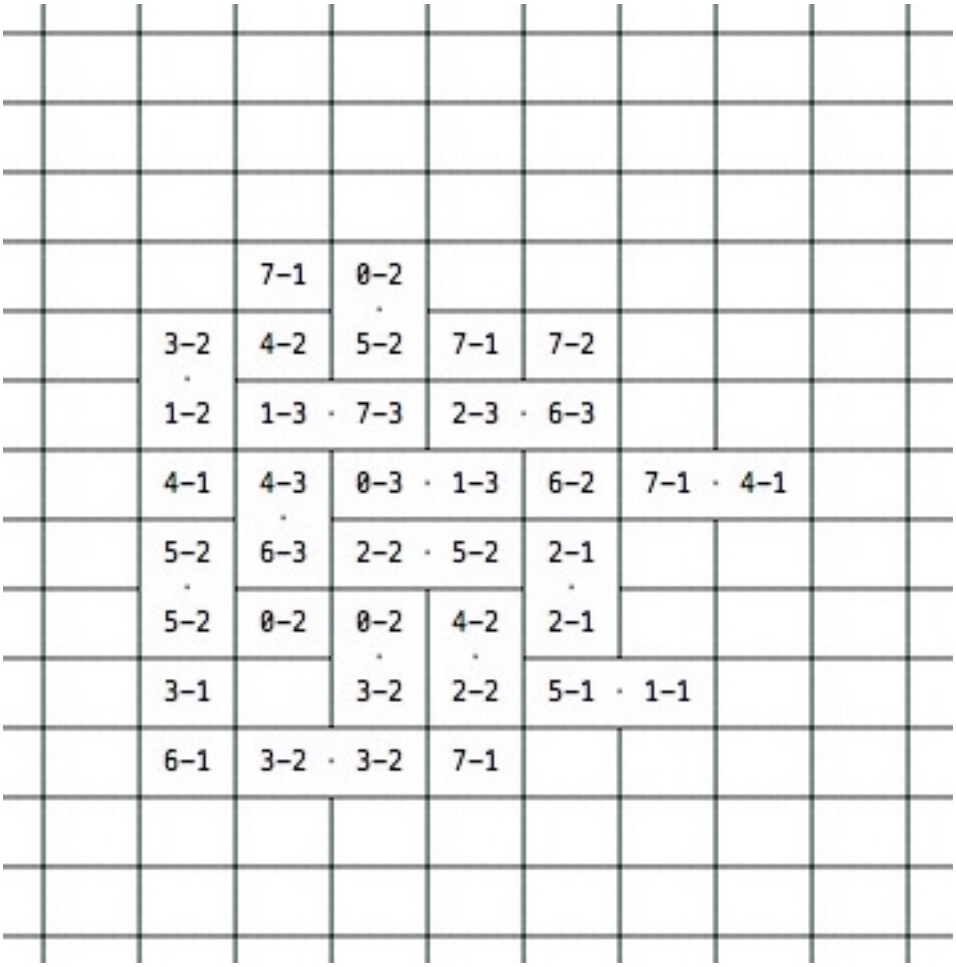

Figura 5: Estado final do jogo.

3 Visualização do Tabuleiro

Descrever a forma de visualização do tabuleiro em modo de texto e o(s) predicado(s) Prolog construídos para o efeito. Deve ser incluída pelo menos uma imagem correspondente ao output produzido pelo predicado de visualização.

4 Movimentos

De acordo com as regras, na sua vez, um jogador pode realizar vários movimentos, isto é, colocar várias peças no tabuleiro. Mais especificamente, devem ser colocadas todas as peças possíveis a subir (pela ordem que o jogador escolher) e, caso ainda exista, uma peça a expandir. Para realizar cada um destes posicionamentos de peças vamos utilizar os seguintes predicados:

- `climb(playerTurn, piece, position, orientation)` para colocar uma peça a subir. Por exemplo, `climb(playerTurn(1), piece(0, 6), position(14, 15), orientation(e))` é usado para colocar o dominó $0 \cdot 6$ do jogador 1, com a metade do 0 na linha 14 e coluna 15 e a metade do 6 a este desta, ou seja, na linha 14 e coluna 16. Na ilustração da Figura 4, este movimento é legal e coloca o dominó no nível 2 em altura, em cima das peças $0 \cdot 5$ e $6 \cdot 6$, como mostra a Figura 6.
- `expand(playerTurn, piece, position, orientation)` para colocar uma peça a expandir. Por exemplo, `expand(playerTurn(1), piece(2, 6), position(13, 12), orientation(s))` é usado para colocar o dominó $2 \cdot 6$ do jogador 1, com a metade do 2 na linha 13 e coluna 12 e a metade do 6 a sul desta, ou seja, na linha 13 e coluna 12. Na ilustração da Figura 4, este movimento é legal e coloca o dominó no nível 1 em altura, ortogonal à peça $7 \cdot 7$ que foi colocada logo no início, como mostra a Figura 7.

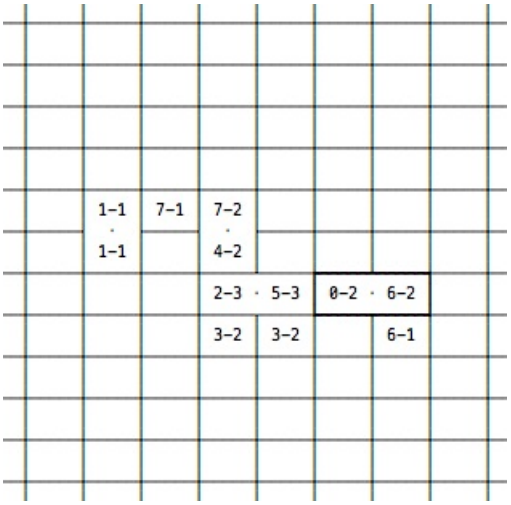

Figura 6: Movimento a subir.

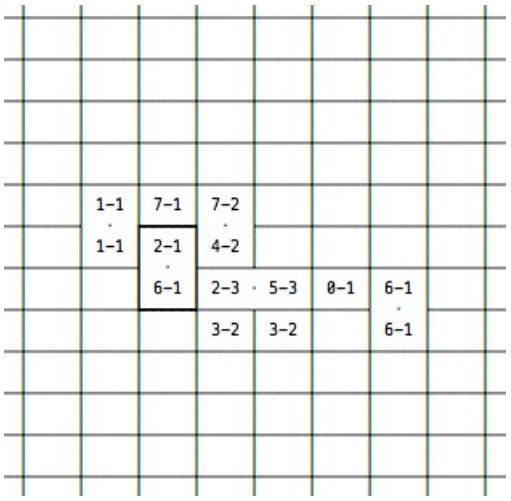

Figura 7: Movimento a expandir.