Q!nto

Relatório Intercalar

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo 2:
Filipa Marilia Monteiro Ramos - up201305378
Inês Alexandra dos Santos Carneiro - up201303501

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

11 de Outubro de 2015

1 O Jogo Q!nto

Figura 1: Caixa e cartas do jog.

1.1 História

Q!nto é um jogo de estratégia abstrata que foi desenvolvido pelo designer Gene Mackles e publicado pela PDG games. O seu lançamento no mercado data do ano 2014. É adequado para todas as idades a partir dos 8 anos podendo ser jogado por um mínimo de 2 e um máximo de 4 jogadores. Existem 3 variações de Q!nto: Q!nto clássico que será o desenvolvido pelo grupo; Q!nto Plus que permite a contagem de pontos numa diagonal de 3 ou mais cartas e Q!nto Light no qual as cartas são divididas igualmente pelos jogadores e o vencedor é o que esvazia a sua mão primeiro.

1.2 Regras

Cada jogador possui 5 cartas sendo que existem 5 formas e 5 cores possíveis. Existem cartas, em menor número, que permitem escolher ou a sua forma ou a sua cor ou ambas.

1. Carta que permite escolher a cor: tem uma forma específica e o utilizador escolhe a cor que esta possuí.
2. Carta que permite escolher a forma: tem uma cor definida e permite a escolha da forma da mesma.
3. Carta universal: forma e cor definível.

Uma jogada é válida se for feita uma coluna ou linha com cartas nas seguintes condições:

1. símbolos iguais;
2. cores iguais;
3. símbolos e cores diferentes.

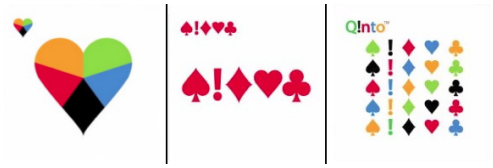

Figura 2: Carta que permite escolher a cor, forma e universal. (1,2 e 3 respetivamente).

A pontuação é calculada com base no número de cartas por linha e coluna englobada na jogada. A cada carta é dada a pontuação de uma unidade. A mesma carta pode ser contabilizada duas vezes visto que os pontos são cotados horizontal e verticalmente. Quando se obtêm linhas com 5 elementos o jogador recebe uma pontuação extra de 5 pontos. Esta jogada é apelidada de Q!nto. Também se obtém 5 pontos extra quando são jogadas todas as cartas na mão e quando é a última jogada. O jogo acaba quando não existem mais cartas a serem jogadas e não existem cartas na pilha. Ganha o jogador com mais pontos.

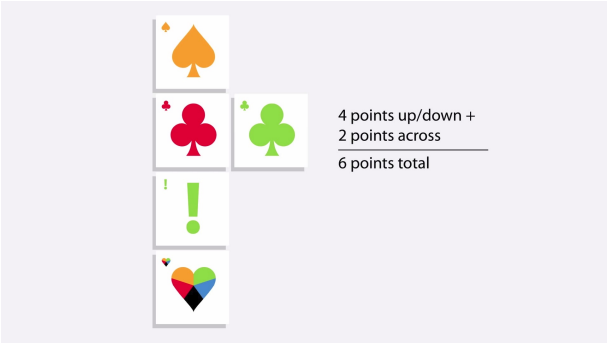

Figura 3: Contagem dos pontos.

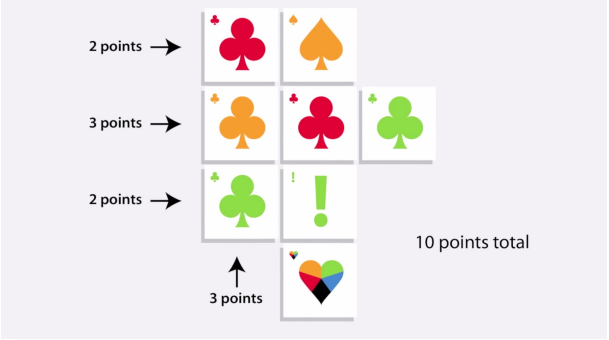

Figura 4: Contagem dos pontos (exemplo 2).

Figura 5: Jogada Q!into.

2 Representação do Estado do Jogo

O menu inicial permitirá a escolha entre as seguintes possibilidades:

1. Jogador vs Jogador;
2. Jogador vs Computador;
3. Computador vs Computador;
4. Sair do Jogo.

O tabuleiro inicial corresponde a uma grelha de 5x5. À medida que o jogo evolui, o tabuleiro é aumentado 5 casas horizontal e verticalmente. Inicialmente, o tabuleiro tem apenas uma carta no centro que é escolhida de forma aleatória da pilha. Por baixo do tabuleiro é representada a mão do jogador (5 cartas). A cada jogada é atualizado o tabuleiro posicionando as cartas jogadas e são dadas cartas a quem as gastou de modo a perfazer 5. O estado de vitória ou derrota corresponde ao final do jogo e é identificada através de uma mensagem no ecrã.

```
    A    B    C
  ---- ---- ----
1 |    |    |    | 1
  |----|----|----|
2 |    | r* |    | 2
  |----|----|----|
3 |    |    |    | 3
  ---- ---- ----
    A    B    C
```

Figura 6: Exemplo de tabuleiro inicial de 3x3 com carta gerada aleatoriamente no centro.

3 Visualização do Tabuleiro

A representação do tabuleiro será feita de forma textual através de barras e *underscores*.

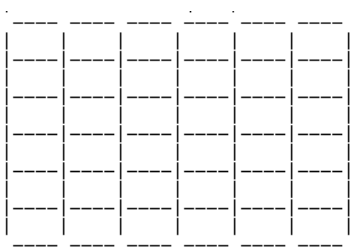

Figura 7: Tabuleiro exemplo visualizado no SICStus de 6x6.

Os predicados implementados foram os seguintes:

```
fguideLine(N, CC) :- CC==N, write('    ').
fguideLine(N, CC) :- CC < N, CC2 is CC+1, write('    '),
L is 65+CC, format('~1c', [L]) , write(' '), fguideLine(N, CC2).
```

- escreve os números das linhas e as letras identificadoras das colunas.

```
fHorizontalLine(0).
fHorizontalLine(N) :- N>0 , write('  '), write('----'), N1 is N-1,
fHorizontalLine(N1).
```

- f significa "first", ou seja, os predicados com f desenham a primeira linha do tabuleiro.

```
sHorizontalLine(0, [], _CC) :- write('|'),write(' '), write(_CC).
sHorizontalLine(N, [L|R], _CC) :- N>0 , write(_CC), write(' '),
write('|'), writetile(L), N1 is N-1, sHorizontalLine(N1, R, _CC2).
```

- s significa "second", ou seja, estes predicados desenham a "segunda"linha do tabuleiro o que corresponde a todas as linhas entre a primeira e a última.

```
tHorizontalLine(0) :-  write('|').
tHorizontalLine(N) :- N>0, write(' '), write('|'),
write('----'), N1 is N-1 , tHorizontalLine(N1).
```

- t refere-se a "third"sendo estes os predicados que desenham a última linha do tabuleiro.

```
displayBoardaux([]).
displayBoardaux([L1]) :- length(L1,N1), sHorizontalLine(N1,L1,0), nl.
displayBoardaux([L1|R]) :- R \= [], length(L1,N1),
sHorizontalLine(N1, L1,0), nl, tHorizontalLine(N1), nl, displayBoardaux(R).
```

- predicados auxiliares que ajudam a fazer o display do tabuleiro.

```
listElement([],0, _X).
listElement([X|Xs], N, X) :- N1 is N - 1,  listElement(Xs, N1, X).
```

- mais predicados que auxiliam na construção do tabuleiro.

```
displayBoard([L1|R]) :-  length(L1,N1), nl, fguideLine(N1, 0), nl,
fHorizontalLine(N1), nl, displayBoardaux([L1|R]),
fHorizontalLine(N1), nl, fguideLine(N1, 0), nl.
```

- predicado que faz o display do tabuleiro.

```
createMatrix(W, H, Matrix) :- listElement(L,W,tile(' ',' ')),
listElement(Matrix,H,L).
```

- cria a matriz que corresponde ao tabuleiro.

```
createBoard(W,H) :- createMatrix(W,H, B), displayBoard(B).
```

- faz output do tabuleiro e cria a matrix correspondente.

```
expand_matrix_up(Matrix, [L|Matrix]) :- empty_tile( E ),
matrix_width(Matrix, W), listElement(L, W, E).
expand_matrix_down(Matrix, NewMatrix) :- empty_tile( E ),
matrix_width(Matrix, W), listElement(L, W, E),
append(Matrix, [L], NewMatrix).

expand_matrix_left([],[]).
expand_matrix_left([L|Matrix], [NL|NewMatrix]):- empty_tile( E ),
append([E], L, NL), expand_matrix_left(Matrix, NewMatrix).
expand_matrix_right([],[]).
expand_matrix_right([L|Matrix], [NL|NewMatrix]):- empty_tile( E ),
append(L, [E], NL), expand_matrix_right(Matrix, NewMatrix).
```

- para expandir a matriz a cada jogada.

4 Movimentos

Os movimentos possíveis são em linha tanto horizontal como vertical. A cada movimento feito tem de ser verificada a validade da jogada. Para isto serão criados predicados de verificação e validação e predicados para permitir os movimentos de cartas no tabuleiro.

```
color(r).
color(b).
color(g).
color(y).
color(c).
```

- definição das cores possíveis.

```
shape('*').
shape('!').
shape('#').
shape('+').
shape('&').
shape(s).
```

- definição das formas possíveis.

```
oneTile(X) :- color(C), shape(F), X = tile(C,F).
tile(' ',' ').
tile(C, F) :- color(C), shape(F).
```

- criação de cartas.

```
writetile(tile(C,F)) :- write(' '), print(C), print(F), write(' ').
```

- predicado que imprime as cartas no tabuleiro.

```
placeTile(X,Y).
```

- predicado que permitirá colocar uma carta no tabuleiro na posição (x,y).

```
verifyMovement(M).
```

- predicado que verificará se a jogada M é válida ou não. Este predicado necessitará de várias funções auxiliares para verificar as 3 possíveis jogadas válidas referidas na secção 1.2 (segunda enumeração).

5 Bibliografia

1. http://www.pdggames.com/images/resources/TriplePlayRulesmore.pdf
2. http://www.pdggames.com/product/qnto
3. https://www.youtube.com/watch?v=t0RolluapXg
4. https://boardgamegeek.com/boardgame/164972/qnto