**Ano Lectivo 2012/2013**


Universidade do Porto

Faculdade de Engenharia

FEUP

**Métodos formais em VDM++**

# Scrabble

*José Carlos Portela Pereira* – ei09012@fe.up.pt

*Mário Jorge Pereira* - ei09010@fe.up.pt

Porto, 6 de Dezembro de 2012

# Índice

# Introdução

No âmbito da cadeira de Métodos Formais de Engenharia de Software foi-nos atribuído o desenvolvimento de um trabalho que consiste na elaboração, documentação e teste de um modelo formal executável de um sistema de software em VDM++. Linguagem que permite para além da modelação de problemas, o uso de ferramentas de automatização de testes unitários às classes e suas respectivas funções e operações do modelo, e também de geração de código em diversas linguagens.

O sistema a desenvolver é um jogo de tabuleiro bastante popular, o "Scrabble", cuja descrição é explicada nas secções seguintes, assim como são especificadas as suas regras e restrições.

No decorrer do relatório também está demonstrada toda a documentação relativa à implementação das classes e testes unitários(Test-Driven Development) do sistema de software. É também feita uma análise à sua fiabilidade e tiradas algumas conclusões finais.

# O Scrabble

O *Scrabble* consiste num jogo de pelo menos dois jogadores (obrigatoriamente), em que cada um recebe um conjunto de 7 peças retiradas aleatoriamente de um saco(baralho) que contêm várias peças.

Com essas peças ele vai ter que formar palavras válidas, ou seja, que estejam no dicionário e que possam ser colocadas no tabuleiro enxaixando nas peças que já se encontram lá. Consoante as peças que constituem a palavra o jogador recebe uma determinada pontuação.

O jogador para além de tentar formar palavras com as suas peças, pode passar a vez, ou também, trocar as letras que tenha na sua mão com as do saco, desde que tenha sempre 7 na sua mão. As palavras válidas são colocadas num tabuleiro 15 por 15, que contêm vários tipos de células, umas vazias, outras ocupadas e outras que contêm símbolos que funcionam como multiplicadores da pontuação.

O jogo pode terminar de várias maneiras, por exemplo, quando um jogador passa a vez duas vezes consecutivas, quando algum jogador escolha sair do jogo e quando o baralho de peças ficar vazio.


**Fig. 1** - Tabuleiro de Scrabble

## Requisitos e restrições do sistema

Relativamente a requisitos e restrições do sistema, é aqui apresentada uma lista dos principais requisitos relativos ao jogo, no que diz respeito ao manuseamento de peças no jogo, tanto ao nível de baralho e mão , como de jogadas(colocação de peças no tabuleiro).

## Requisitos:

<u>Manuseamento de peças na mão do jogador e no baralho:</u>

- **Req.1**: baralhar de modo aleatório a sequência de peças que representa o baralho;

- **Req.2**: colocar as peças na mao do jogador, sempre que necessário;

- **Req.3**: trocar as posições das peças na mão do jogador de modo a ajudar numa nova jogada(shuffle);

- **Req.4**: verificar se a mão tem sempre 7 peças , e no caso de não ter acrescentar até atingir as 7 peças novamente;

Jogadas:

- **Req.5**: verificar se uma jogada foi efectuada de modo a passar a vez do jogador;

- **Req.6**: saber quando é necessária a intervenção de multiplicadores e e a sua influência na pontuação do jogador para o cálculo de pontuações;

- **Req.7**: respeitar as regras pré-definidas do Scrabble aquando da colocação de peças no tabuleiro;

- **Req.8**: verificar limites do tabuleiro;

- **Req.9**: saber se foi criada uma palavra existente no dicionário e atribuir a pontuação de acordo com a pontuação verificável em cada peça;

- **Req.10**: O tabuleiro:
  - . (ponto final) simboliza uma casa vazia;
  - * Casa de duplicação de valor da letra;
  - # Casa de triplicação de valor da letra;
  - $ Casa de duplicação de valor da palavra;
  - % Casa de Triplicação de valor da palavra;

## Restrições:

Tabuleiro:

- As dimensões do tabuleiro são de 15 x 15 (225 células);

- As células do tabuleiro apenas podem ser caracteres;
- Os multiplicadores do tabuleiro são apenas o ‘#’, ‘$’, ‘*’ e o ‘%’;

Jogadores:

- Existem apenas 2 jogadores

- Cada jogador tem apenas uma tentativa de jogada, e em cada turno pode:
    - Formar uma palavra no tabuleiro;
    - Trocar peças com o baralho;
    - Fazer *shuffle* à sua mão;
    - Passar a sua vez;

Peças:

- Cada peça tem de ter obrigatoriamente uma pontuação;
- O caracter de uma peça é uma letra do alfabeto mais o ‘_’ que simboliza uma peça vazia;

Jogo:

- O baralho contêm 120 peças;
- A troca de peças permite que um jogador troque qualquer número de peças que tem na mão por peças do baralho;
- No caso de um jogador passar de vez duas vezes seguidas o jogo termina;
- O jogo termina quando todas as peças tiverem sido retiradas do baralho e um jogador tiver usado todas as que tinha disponíveis na mão;
- A decisão de qual o jogador que inicia o jogo é aleatória;
- O vencedor é aquele que no final do jogo tiver maior pontuação;
- O jogador apenas pode colocar palavras na horizontal(da esquerda para a direita) e na vertical(de cima para baixo);
- O jogador apenas pode colocar uma palavra se tiver peças na sua mão que a formem;
- A palavra que vai ser colocada no tabuleiro tem que encaixar com pelo menos uma peça que esteja no tabuleiro;

## Restrições identificadas em VDM++

Nesta secção são especificadas as restrições, em VDM++, que servem para garantir o sucesso das restrições apontadas:

Classe Celula:

- O simbolo de uma célula do tabuleiro tem que ser uma letra do alfabeto, uma peça vazia, um ponto final(casa vazia) ou um multiplicador:

```
inv simbolo  in set
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P'
,'Q','R','S','T','U','V','X','Y','W','Z','#','*','$','%','.','
','_'};
```

Classe Jogador:

- A mão de um jogador durante o jogo nunca pode atingir um tamanho maior do que 7 peças. Como o jogo é apenas para dois jogadores o *id* do jogador apenas pode ser 1 ou 2. A pontuação que um jogador tem ao longo do jogo nunca pode ser menor do que 0:

```
inv len mao <= 7;
inv id = 1 or id = 2;
inv pontos >= 0;
```

Classe Peca:

- A peça que vai ser colocada no tabuleiro tem de ter obrigatoriamente uma pontuação superior ou igual a 0 e fazer parte do alfabeto, ou então ter o caracter '_'(peça vazia):

```
inv letra  in set
{'A','B','C','Ç','D','E','F','G','H','I','J','K','L','M','N','O','P','
Q','R','S','T','U','V','X','Y','W','Z','_'};
inv pontuacao >= 0;
```

Classe Tabuleiro:

- O tabuleiro não pode ter mais do que 225(15*15) células ou peças:

```
inv (card (dom tabuleiro))  <= 226;
inv (card (rng tabuleiro))  <= 226;
inv card posicoesOcupadas <= 226 ;
```

Classe Jogo:

- O baralho durante o jogo nunca pode chegar a ter um tamanho superior a 120 peças. O nº de jogadores do jogo nunca pode maior que dois. A variável da classe jogo *vez,* que determina de quem a vez de jogar só pode ter o valor 1, caso seja a vez do jogador 1 ou o valor 2, caso seja a vez do outro jogador:

```
inv len baralho <= 120;
inv len jogadores <= 2;
inv vez = 1 or vez = 2;
```

## Diagrama conceptual de classes

O modelo de classes abaixo representa a estrutura e comportamentos das classes representativas do problema. Este diagrama foi gerado automaticamente através do *Overture*.

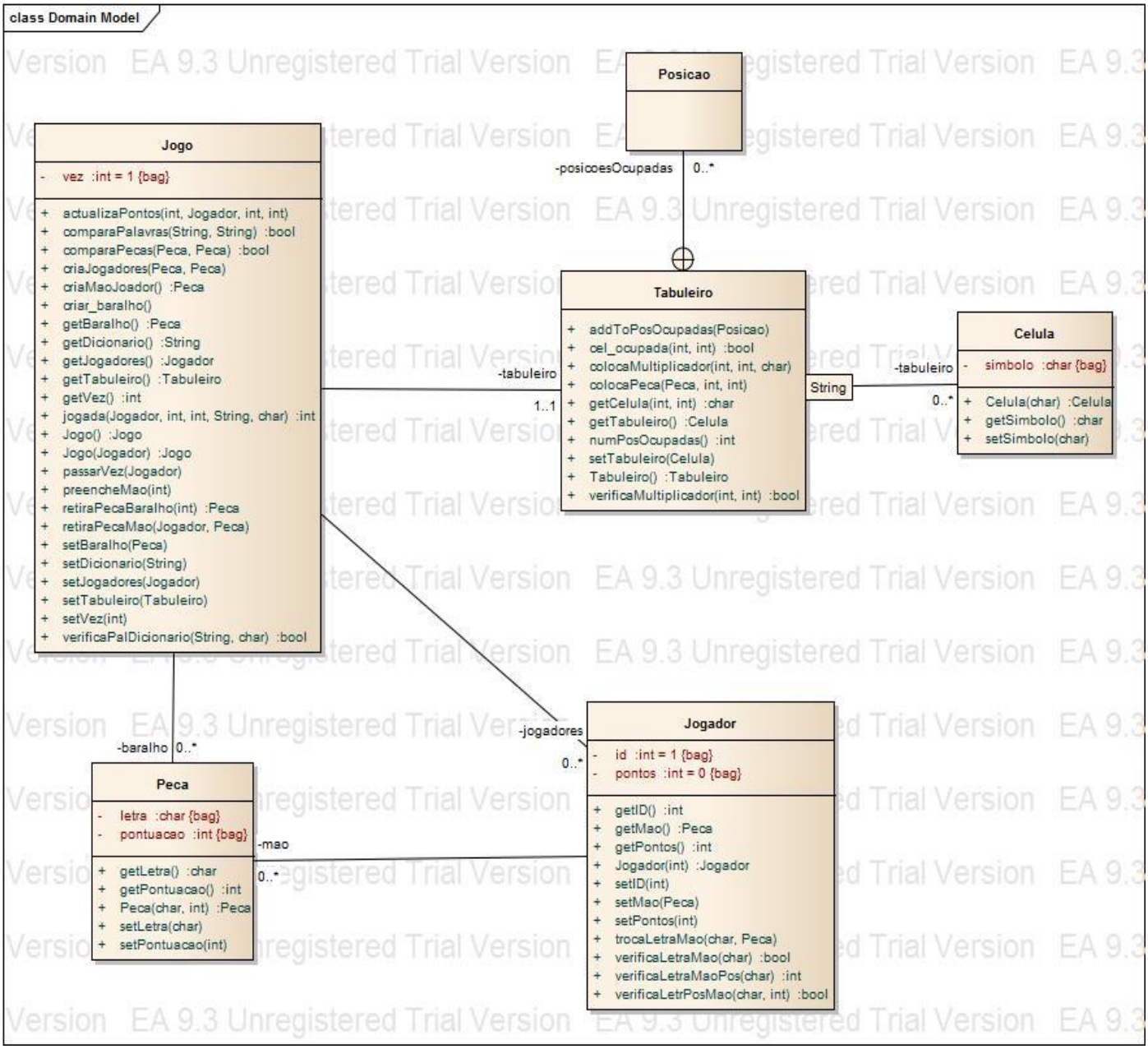


**Fig. 2** – Diagrama UML

## Classes e scripts de testes

Estão aqui as classes e scripts de teste usados para verificar o funcionamento das operações constituintes do jogo implementado.

Para cada classe definida existe uma classe em que se testam as operações criadas. Estas operações de teste tentam verificar o maior número de situações possíveis.

De seguida é feita uma breve descrição de cada uma das classes de teste:

- **assertTrue()** - este teste está presente em todas as classes pois é usado para verificar se a expressão que é passada como parâmetro é verdadeira;

### Classe Jogo - Classe TestJogo

- **testaCriarJogadores: ()** - verifica se foram criados os dois jogadores participantes no jogo:

```
static public testaCriarJogadores: () ==> ()
                  testaCriarJogadores () == (

               dcl tmpJogs : seq of Jogador := [];
            dcl jg: Jogo := new Jogo(tmpJogs);
            dcl peca1 : Peca := new Peca('T',14);
              dcl peca2 : Peca := new Peca('J',10);
              dcl peca3 : Peca := new Peca('E',18);
                dcl tmp2 : seq of Peca := [peca1,peca2];
            dcl tmp1 : seq of Peca := [peca3];

           jg.criaJogadores(tmp1,tmp2);

              AssertTrue(len jg.getJogadores() = 2);
   );
```

- **testaCriaBaralho: ()** - verifica se o baralho está a ser criado correctamente:

```
static public testaCriaBaralho: () ==> ()
                  testaCriaBaralho () == (

                  dcl players: seq of Jogador := [];
                  dcl jgb: Jogo := new Jogo(players);

                   AssertTrue(jgb.getBaralho()(1).getLetra() = 'A');
                   AssertTrue(jgb.getBaralho()(16).getPontuacao() = 3);
                   AssertTrue (jgb.getBaralho()(120).getLetra() = '_');
                   AssertTrue(jgb.getBaralho()(2).getPontuacao() <> 2);

       );
```

- **testaCriaMao: ()** - verifica se é atribuida uma mao a cada jogador:

```
static public testaCriaMao: () ==> ()
                        testaCriaMao () == (

                         dcl j : Jogo := new Jogo();
                         dcl mao_j1: seq of Peca := [];
                         dcl j1 : Jogador := new Jogador(1);

                         mao_j1 := j.criaMaoJoador();
                         j1.setMao(mao_j1);

                          AssertTrue( len j1.getMao() = 7);
                  );
```

- **testaComparaPalavras: ()** - verifica se duas palavras são iguais:

```
static public testaComparaPalavras: () ==> ()
                             testaComparaPalavras() == (

                             dcl jogo : Jogo := new Jogo();
                             dcl pal1 : seq of char :=
['c','a','s','a'];
                             dcl pal2 : seq of char :=
['c','a','s','a'];
                             dcl pal3 : seq of char :=
['c','a','s','a','l'];
                             dcl pal4 : seq of char :=
['c','a','s','t'];



      AssertTrue(jogo.comparaPalavras(pal1,pal2) = true);

      AssertTrue(jogo.comparaPalavras(pal1,pal3) = false);

      AssertTrue(jogo.comparaPalavras(pal1,pal4) = false);

      );
```

- **testaPalDicionario: ()** - verifica se a palavra inserida existe no dicionário:

```
  static public testaPalDicionario: () ==> ()
                              testaPalDicionario() == (

                              dcl jogo : Jogo := new Jogo();
                              dcl pal1 : seq of char :=
['c','a','s','a'];
                              dcl pal2 : seq of char :=
['k','a','s','a'];
                              dcl dicionario : set of seq of char :=
{['c','a','s','a'],['c','a','s','a','l'],['c','a','s','t']};


      AssertTrue(jogo.verificaPalDicionario(pal1,dicionario) = true);

      AssertTrue(jogo.verificaPalDicionario(pal2,dicionario) = false);
       );
```

- **testaPassaVez: ()** - testa o método de passar a vez de um jogador para o outro:

```
   static public testaPassaVez: () ==> ()
                              testaPassaVez() == (

                         dcl jogo : Jogo := new Jogo();
                         dcl jog1 : Jogador := new Jogador(1);
                         dcl jog2 : Jogador := new Jogador(2);

                         jogo.setJogadores([jog1,jog2]);

                         jogo.passarVez(jog1);
                         jogo.passarVez(jog2);

                         AssertTrue( jogo.getVez() = 1);
       );
```

- **testaRetirapecaMao: ()** - testa a operação em que é passada por parâmetro a letra a retirar da mão e verifica se existe, e nesse caso retira-a:

```
static public testaRetiraPecaMao: () ==> ()
                              testaRetiraPecaMao() == (

                         dcl jogo : Jogo := new Jogo();
                         dcl jog1 : Jogador := new Jogador(1);
                         dcl jog2 : Jogador := new Jogador(2);
                         dcl peca1 : Peca := new Peca('B',10);
                         dcl peca2 : Peca := new Peca('D',20);
                         dcl mao_jog1 : seq of Peca := [];

                         jog1.setMao([peca1,peca2]);

                         jogo.setJogadores([jog1,jog2]);

                         jogo.retiraPecaMao(jog1,peca1);
                         -- jogo.retiraPecaMao(jog1,peca2);
                         mao_jog1 := jog1.getMao();
```

```
                        AssertTrue( len mao_jog1 = 1);
                        --AssertTrue( len mao_jog1 = 0);
                    );
```

- **testaRetiraPecaBaralho: ()** - testa a operação em que é passada por parâmetro a letra a retirar do baralho e verifica se existe, e nesse caso retira-a:

```
static public testaRetiraPecaBaralho: () ==> ()
                              testaRetiraPecaBaralho() == (

                            dcl jogo : Jogo := new Jogo();
                            dcl baralho_act : seq of Peca := [];
                          dcl peca_ret : Peca :=
jogo.retiraPecaBaralho(3);

                            baralho_act := jogo.getBaralho();

                               AssertTrue( peca_ret.getLetra() = 'A');
                               AssertTrue( len baralho_act = 119);
      );
```

- **testaJogada: ()** - testa um tipo de jogada:

```
static public testaJogada: () ==> ()
                              testaJogada() == (


                              dcl jogo : Jogo := new Jogo();
                              dcl jog1 : Jogador := new Jogador(1);
                              dcl peca1 : Peca := new Peca('F',10);
                              dcl peca2 : Peca := new Peca('C',20);
                              dcl peca3 : Peca := new Peca('A',18);
                            dcl peca4 : Peca := new Peca('S',14);
                            dcl peca5 : Peca := new Peca('A',10);
                            dcl peca6 : Peca := new Peca('I',18);
                            dcl peca7 : Peca := new Peca('O',40);
                            dcl tam_mao : nat;
                            dcl dicionario : set of seq of char :=
{['C','A','S','A'],['C','A','S','A','l'],['C','A','S','T']};
                              dcl tabuleiro : Tabuleiro;
                              dcl result : nat;


                              jogo.setDicionario(dicionario);

      jog1.setMao([peca1,peca2,peca3,peca4,peca5,peca6,peca7]);

                              result :=
jogo.jogada(jog1,6,0,['C','A','S','A'],'D');

                              tabuleiro := jogo.getTabuleiro();
                              tam_mao := len jog1.getMao();

                              AssertTrue( tabuleiro.getCelula(7,0) =
'C');
                              AssertTrue( tabuleiro.getCelula(8,0) =
'A');
```

```
                                AssertTrue( tabuleiro.getCelula(9,0) =
'S');
                                AssertTrue( tabuleiro.getCelula(10,0) =
'A');
                                --AssertTrue( tam_mao = 3);
                                AssertTrue( result = 1);

                                );
```

- **testaJogada2: ()** - testa outro tipo de jogada:

```
static public testaJogada2: () ==> ()
                                testaJogada2() == (


                           dcl jogo : Jogo := new Jogo();
                           dcl jog1 : Jogador := new Jogador(1);
                           dcl peca1 : Peca := new Peca('F',10);
                           dcl peca2 : Peca := new Peca('I',20);
                           dcl peca3 : Peca := new Peca('T',18);
                             dcl peca4 : Peca := new Peca('E',14);
                             dcl peca5 : Peca := new Peca('L',10);
                             dcl peca6 : Peca := new Peca('H',18);
                             dcl peca7 : Peca := new Peca('A',40);
                             dcl tam_mao : nat;
                             dcl dicionario : set of seq of char :=
{['C','A','S','A'],['C','A','S','A','l'],['T','E','L','H','A']};
                                dcl tabuleiro : Tabuleiro;
                                dcl result : nat;


                                jogo.setDicionario(dicionario);

      jog1.setMao([peca1,peca2,peca3,peca4,peca5,peca6,peca7]);

                                result :=
jogo.jogada(jog1,6,0,['T','E','L','H','A'],'B');

                                tabuleiro := jogo.getTabuleiro();
                                tam_mao := len jog1.getMao();

                                AssertTrue( tabuleiro.getCelula(6,1) =
'T');
                                AssertTrue( tabuleiro.getCelula(6,2) =
'E');
                                AssertTrue( tabuleiro.getCelula(6,3) =
'L');
                                AssertTrue( tabuleiro.getCelula(6,4) =
'H');
                                AssertTrue( tabuleiro.getCelula(6,5) =
'A');
                                -- AssertTrue( tam_mao = 2);
                                AssertTrue( result = 1);

                           );
```

- **testaCompletaMao: ()** - assegura que é possível completar a mão do jogador com as peças em falta:

```
      static public testaCompletaMao: () ==> ()
                  testaCompletaMao () == (

            dcl peca1 : Peca := new Peca('T',14);
            dcl peca2 : Peca := new Peca('J',10);
            dcl peca3 : Peca := new Peca('E',18);
            dcl peca4 : Peca := new Peca('M',14);
            dcl peca5 : Peca := new Peca('R',10);
            dcl peca6 : Peca := new Peca('I',18);
            dcl peca7 : Peca := new Peca('O',40);
            dcl peca8 : Peca := new Peca('A',14);
            dcl peca9 : Peca := new Peca('B',10);
            dcl peca10 : Peca := new Peca('C',18);
            dcl peca11: Peca := new Peca('E',14);
            dcl peca12 : Peca := new Peca('F',10);
            dcl peca13: Peca := new Peca('G',18);
            dcl peca14: Peca := new Peca('H',40);
            dcl sim_baralho : seq of Peca := [peca1, peca2, peca3,
peca4, peca5, peca6, peca7, peca8, peca9, peca10, peca11, peca12,
peca13, peca14];
            dcl jgc: Jogo := new Jogo();
            dcl tmp5 : seq of Peca := [];
            dcl tmp6 : seq of Peca := [];


            jgc.criaJogadores(tmp5,tmp6);

            jgc.setBaralho(sim_baralho);

            jgc.preencheMao(2);

            AssertTrue( len (jgc.getJogadores()(2).getMao()) = 7 );

      );
```

- **testaComparaPecas: ()** - este teste assegura a possibilidade de comparar peças:

```
static public testaComparaPecas: () ==> ()
                  testaComparaPecas () == (

                  dcl jogo : Jogo := new Jogo();
                  dcl peca1 : Peca := new Peca('B',10);
                  dcl peca2 : Peca := new Peca('D',20);
                  dcl peca3 : Peca := new Peca('B',30);

                  AssertTrue( jogo.comparaPecas(peca1,peca2) = false);
                  AssertTrue( jogo.comparaPecas(peca1,peca3) = true);
      );
```

- **testaJogo: ()** – testa parcialmete uma simulação de um jogo:

```
static public testaJogo: () ==> ()
                                    testaJogo() == (

                          dcl jogo : Jogo := new Jogo();
                          dcl jog1 : Jogador := new Jogador(1);
                          dcl jog2 : Jogador := new Jogador(2);

                          dcl peca1 : Peca := new Peca('F',10);
                          dcl peca2 : Peca := new Peca('I',20);
                          dcl peca3 : Peca := new Peca('G',18);
                            dcl peca4 : Peca := new Peca('E',14);
                            dcl peca5 : Peca := new Peca('L',10);
                            dcl peca6 : Peca := new Peca('S',18);
                            dcl peca7 : Peca := new Peca('J',18);
                            dcl peca8 : Peca := new Peca('A',40);
                            dcl peca9 : Peca := new Peca('T',18);
                            dcl peca10 : Peca := new Peca('E',14);
                            dcl peca11 : Peca := new Peca('C',10);
                            dcl peca12 : Peca := new Peca('O',18);
                            dcl peca13 : Peca := new Peca('A',40);
                            dcl tabuleiro : Tabuleiro;

                            dcl result : nat;

                               dcl dicionario : set of seq of char :=
{['C','A','S','A'],['C','A','S','A','l'],['T','E','L','H','A'],['J','O
','G','O'],['T','E','C','O']};
                               jogo.setDicionario(dicionario);


      jog1.setMao([peca1,peca2,peca3,peca4,peca5,peca6,peca7]);

      jog2.setMao([peca7,peca8,peca9,peca10,peca11,peca12,peca13]);

                               jogo.setVez(1);
                               result :=
jogo.jogada(jog1,7,7,['C','A','N','O','A'],'D');

                               --como canoa nao esta no dicionario e
passada a vez
                               --jogo.passarVez(jog1);

                               result :=
jogo.jogada(jog2,8,2,['T','E','C','O'],'D');

                               tabuleiro := jogo.getTabuleiro();

                               AssertTrue( tabuleiro.getCelula(9,2) =
'T');
                               AssertTrue( tabuleiro.getCelula(10,2) =
'E');
                               AssertTrue( tabuleiro.getCelula(11,2) =
'C');
                               AssertTrue( tabuleiro.getCelula(12,2) =
'O');

);
```

- **runAllJogoTests: ()** – testa todos os testes desta classe:

```
static public runAllJogoTests: () ==> ()
                                          runAllJogoTests() == (


                                        testaCriarJogadores();
                                        testaCriaBaralho();
                                        testaCriaMao();
                                        testaCompletaMao();
                                        testaComparaPecas();
                                        testaComparaPalavras();
                                        testaPalDicionario();
                                        testaPassaVez();
                                        testaRetiraPecaMao();
                                        testaRetiraPecaBaralho();
                                        testaJogada();
                                        testaJogada2();
                                        testaJogo();
);
```

## Classe Tabuleiro - Classe TestTabuleiro

- **testaTabuleiro: ()** - verifica a criação do tabuleiro:

```
static public testaTabuleiro: () ==> ()
       testaTabuleiro () == (

            dcl t : Tabuleiro := new Tabuleiro();
            dcl tam : nat := 0;

            tam := (card (dom t.getTabuleiro() ));

            -- 15*15 = 225
            AssertTrue( tam = 225);

       );
```

- **testaAddToPosOcupadas: ()** - testa a ocupação de posições de jogo em campo:

```
static public testaAddToPosOcupadas: () ==> ()
            testaAddToPosOcupadas () == (

            dcl t : Tabuleiro := new Tabuleiro();

            t.addToPosOcupadas(mk_Tabuleiro`Posicao(2,3));
            t.addToPosOcupadas(mk_Tabuleiro`Posicao(6,10));
            AssertTrue( t.numPosOcupadas()  = 2);

            );
```

- **testaColocaPeca: ()** - aqui, é testada a operação de colocação de uma peça no tabuleiro, em que se ocupa uma posição do tabuleiro:

```
static public testaColocaPeca: () ==> ()
                testaColocaPeca () == (

                dcl t : Tabuleiro := new Tabuleiro();
                dcl peca1 : Peca := new Peca('H',10);
                dcl peca2 : Peca := new Peca('A',20);
                dcl peca3 : Peca := new Peca('T',30);

                t.colocaPeca(peca1,1,1);
                t.colocaPeca(peca2,14,1);
                t.colocaPeca(peca3,15,1);

                AssertTrue( t.getCelula(1,1)  = 'H');
                AssertTrue( t.getCelula(14,1)  = 'A');
                AssertTrue( t.getCelula(15,1)  = 'T');
     );
```

- **testaCelOcupada: ()** - verifca se uma célula está ocupada ou não:

```
static public testaCelOcupada: () ==> ()
                testaCelOcupada() == (

                dcl t : Tabuleiro := new Tabuleiro();
                dcl peca1 : Peca := new Peca('H',10);
                dcl peca2 : Peca := new Peca('A',20);

                t.colocaPeca(peca1,3,2);
                t.colocaPeca(peca2,4,5);

                AssertTrue( t.cel_ocupada(3,2) = true);
                AssertTrue( t.cel_ocupada(4,5) = true);
                AssertTrue( t.cel_ocupada(2,3) = false);
     );
```

- **testaColocaMulti: ()** - coloca um multiplicador no tabuleiro:

```
static public testaColocaMulti: () ==> ()
                testaColocaMulti() == (

                dcl t : Tabuleiro := new Tabuleiro();

                t.colocaMultiplicador(2,2,'#');
                t.colocaMultiplicador(5,3,'*');
                t.colocaMultiplicador(8,10,'$');
                -- teste que falha, pois nao respeita a pre-condicao
                -- t.colocaMultiplicador(4,2,'d');

                AssertTrue( t.getCelula(2,2) = '#');
                AssertTrue( t.getCelula(5,3) = '*');
                AssertTrue( t.getCelula(8,10) = '$');
      );
```

- **testaVerificaMultiplicador: ()** - testa a detecção de um multiplicador numa determinada posição do tabuleiro:

```
static public testaVerificaMultiplicador: () ==> ()
                testaVerificaMultiplicador () == (

                   dcl t : Tabuleiro := new Tabuleiro();

                   t.colocaMultiplicador(5,2,'#');
                   t.colocaMultiplicador(4,7,'*');

                   AssertTrue (t.verificaMultiplicador(5,2) = true);
                   AssertTrue (t.verificaMultiplicador(4,7) = true);
                   AssertTrue (t.verificaMultiplicador(10,11) <>
true);
                );
```

- **runAllTabuleiroTests: ()** - testa a execução de todos os testes referentes à classe Tabuleiro:

```
static public runAllTabuleiroTests: () ==> ()
                   runAllTabuleiroTests() == (

                        testaTabuleiro();
                        testaAddToPosOcupadas();
                        testaColocaPeca();
                        testaCelOcupada();
                        testaColocaMulti();
                        testaVerificaMultiplicador();
```

## Classe Jogador - Classe TestJogador

- **testaPontosJogador: ()** - teste da obtenção dos pontos do jogador;

```
static public testaPontosJogador: () ==> ()
            testaPontosJogador () == (

            dcl j : Jogador := new Jogador(1);

            -- os pontos iniciais de um jogador e 0
            AssertTrue( j.getPontos() = 0);
      );
```

- **testaPontos: ()** - teste da pontuacao de peças existentes na mão do jogador;

```
  static public testaPontos: () ==> ()
                  testaPontos () == (

              dcl peca1 : Peca := new Peca('A',14);
              dcl peca2 : Peca := new Peca('B',10);
              dcl peca3 : Peca := new Peca('C',18);
              dcl peca4 : Peca := new Peca('E',14);
              dcl peca5 : Peca := new Peca('F',10);
              dcl peca6 : Peca := new Peca('G',18);
              dcl peca7 : Peca := new Peca('H',40);
                    dcl j: Jogador := new Jogador();

              j.setMao([peca1,peca2,peca3,peca4,peca5,peca6,peca7]);

                  j.setPontos(20);
                  AssertTrue (j.getPontos() = 20);
       );
```

- **testaMaoJogador: ()** - testa a atribuicao de uma mão de peças a um jogador;

```
static public testaMaoJogador: () ==> ()
            testaMaoJogador () == (

                  dcl j : Jogador := new Jogador(1);
                  dcl peca1 : Peca := new Peca('A',14);
                  dcl peca2 : Peca := new Peca('B',10);
                  dcl peca3 : Peca := new Peca('C',18);
                  dcl peca4 : Peca := new Peca('A',14);
                  dcl peca5 : Peca := new Peca('B',10);
                  dcl peca6 : Peca := new Peca('C',18);
                  -- dcl peca7 : Peca := new Peca('a',14);
                  -- dcl peca8 : Peca := new Peca('b',10);

            j.setMao([peca1,peca2,peca3,peca4,peca5,peca6]);
            AssertTrue( (len j.getMao()) = 6);

            -- Teste falha, pois o jogador não pode ter mais de 7 peças

            --
j.setMao([peca1,peca2,peca3,peca4,peca5,peca6,peca7,peca8]);
            -- AssertTrue( (len j.getMao()) = 8);

      );
```

- **testaPontuacao: ()** - teste da pontuação de peças existentes na mão do jogador;

```
static public testaPontuacao: () ==> ()
                        testaPontuacao () == (

              dcl peca1 : Peca := new Peca('A',14);
              dcl peca2 : Peca := new Peca('B',10);
              dcl peca3 : Peca := new Peca('C',18);
```

```
              dcl peca4 : Peca := new Peca('E',14);
              dcl peca5 : Peca := new Peca('F',10);
              dcl peca6 : Peca := new Peca('G',18);
              dcl peca7 : Peca := new Peca('H',40);
                    dcl j: Jogador := new Jogador();

              j.setMao([peca1,peca2,peca3,peca4,peca5,peca6,peca7]);

                  j.setPontos(20);
                  AssertTrue (j.getMao()(1).getPontuacao() = 14);
                  AssertTrue (j.getMao()(2).getPontuacao() = 10);
                  AssertTrue (j.getMao()(3).getPontuacao() = 18);
                  AssertTrue (j.getMao()(4).getPontuacao() = 14);
                  AssertTrue (j.getMao()(5).getPontuacao() = 10);
                  AssertTrue (j.getMao()(6).getPontuacao() = 18);
                  AssertTrue (j.getMao()(7).getPontuacao() = 40);
    );
```

- **testaVerifLetraMao: ()** - teste que verifica se uma letra se encontra na mão do jogador:

```
static public testaVerifLetraMao: () ==> ()
                                  testaVerifLetraMao() == (

                                  dcl j : Jogador := new Jogador(1);
                                  dcl peca1 : Peca := new
Peca('A',14);
                                  dcl peca2 : Peca := new
Peca('B',10);
                                  dcl peca3 : Peca := new
Peca('C',18);
                                  dcl peca4 : Peca := new
Peca('D',14);
                                  dcl peca5 : Peca := new
Peca('A',10);
                                  dcl peca6 : Peca := new
Peca('F',18);


      j.setMao([peca1,peca2,peca3,peca4,peca5,peca6]);

                                  AssertTrue( j.verificaLetraMao('D')
= true);
                                  AssertTrue( j.verificaLetraMao('K')
= false);
                        );
```

- **testaTrocaLetraMao: ()** - teste que verifica se uma letra da mão é trocada correctamente:

```
static public testaTrocaLetraMao : () ==> ()
                                  testaTrocaLetraMao() == (

                                  dcl j : Jogador := new Jogador(1);
```

```
                                    dcl peca1 : Peca := new
Peca('A',14);
                                    dcl peca2 : Peca := new
Peca('B',10);
                                    dcl peca3 : Peca := new
Peca('C',18);
                                    dcl peca4 : Peca := new
Peca('D',14);
                                    dcl mao : seq of Peca := [];

                                    dcl nova_peca : Peca := new
Peca('K',15);



      j.setMao([peca1,peca2,peca3,peca4]);
                                    j.trocaLetraMao('B',nova_peca);

                                    mao := j.getMao();

                                    AssertTrue( (len j.getMao()) = 4);
                                    );
```

- **runAllJogadorTests: ()** - teste que verifica se todos os testes referentes à classe Jogador são executados correctamente:

```
static public runAllJogadorTests: () ==> ()
                    runAllJogadorTests() == (

                         testaPontosJogador();
                         testaPontos();
                         testaPontuacao();
                         testaMaoJogador();
                         testaVerifLetraMao();
                         testaTrocaLetraMao(););
```

## Classe TestAll

- **runAllTests: ()** – teste que executa todos os testes implementados no programa:

```
static public runAllTests: () ==> ()
                                    runAllTests() == (


                                    dcl teste_jogador : TestJogador :=
new TestJogador();
                                    dcl teste_tabuleiro : TestTabuleiro
:= new TestTabuleiro();
                                    dcl teste_jogo : TestJogo := new
TestJogo();


                                    teste_jogador.runAllJogadorTests();
```

```
    teste_tabuleiro.runAllTabuleiroTests();
                                    teste_jogo.runAllJogoTests();
                        );
```

## Matriz de rastreabilidade

A matriz de rastreabilidade é usada para representação da relação entre os requisitos e os testes usados para comprovar o seu cumprimento. Além dos testes apresentados, existem outros que não são discriminados.

| Testes/ Requisitos | Req. 1 | Req. 2 | Req. 3 | Req. 4 | Req. 5 | Req. 6 | Req. 7 | Req. 8 | Req. 9 | Req. 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **testaCriaBaralho()** | X | | | | | | | | | |
| **testaCompletaMao()** | | | X | | | | | | | |
| **testaPontos()** | | | | | | | | | X | |
| **testaPontuacao()** | | | | | | | | | X | |
| **testaMaoJogador()** | | | X | | | | | | | |
| **testaTabuleiro()** | | | | | | | | X | | |
| **testaAddToPosOcupadas()** | | | | | | | | | | X |
| **testaColocaPeca()** | | | | | | | | | | X |
| **testaCelOcupada()** | | | | | | | | | | X |
| **testaColocaMulti()** | | | | | | X | | | | X |
| **testaVerificaMultiplicador()** | | | | | | X | | | | X |
| **testaCriaMao()** | | | | | | | | | | |
| **testaComparaPalavras()** | | | | | | | X | | | |
| **testaPalDicionario()** | | | | X | | | | | | |
| **testaPassaVez()** | | | | X | X | | | | | |
| **testaRetiraPecaMao()** | | | | X | | | | | | |
| **testaRetiraPecaBaralho()** | | | X | | | | | | | |
| **testaJogada()** | | | | | | | X | | | |
| **testaJogada2()** | | | | | | | X | | | |

# Classes em VDM++

## Classe Celula

- Descrição: Nesta classe são armazenadas as características necessárias para manipular uma célula, sendo que esa, corresponde a uma posição do tabuleiro;

- Atributos: simbolo: char;
- Operações:

```
class Celula


      instance variables

            private simbolo : char;

            -- As casas vazias são representadas por '.' e os bonus
podem ser:
            -- * - duplica valor letra
        -- # - triplica valor letra
        -- $ - duplica valor palavra
        -- % - triplica valor palavra
            inv simbolo  in set
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','
R','S','T','U','V','X','Y','W','Z','#','*','$','%','.',' ','_'};

      operations

            -- As celulas são constituidos por casas normais e por
casas com bonus
            public Celula : char ==> Celula
                  Celula(simb) == (
                  simbolo:= simb;
                  return self)
                  post simbolo = simb;


            public getSimbolo : () ==> char
                  getSimbolo() == return simbolo;

            public setSimbolo: char  ==> ()
                  setSimbolo(simb) == (
                        simbolo := simb);


end Celula
```

## Classe Jogador

- Descrição: Aqui é definido e alterado aquilo que o jogador precisa de manipular para poder jogar, como por exemplo a sua mão de peças e a sua pontuação;

- Atributos: pontos: nat;  | id:nat | seq of Peca;

- Operações:

```
class Jogador

instance variables

            private pontos: nat := 0;
            private id: nat1 := 1;
            private mao: seq of Peca := [];


            inv len mao <= 7;
            inv id = 1 or id = 2;
            inv pontos >= 0;


      operations

            public Jogador : nat1  ==> Jogador
                  Jogador(id_n) == (

                  id := id_n;
                  pontos := 0;

                  )
                  pre id_n = 1 or id_n = 2
                  post id = id_n;


            public getID : () ==> nat1
                  getID() == return id;

            public setID : nat1 ==> ()
                  setID(id_novo) == id := id_novo;

            public getPontos : () ==> nat
                  getPontos() == return pontos;

            public setPontos : nat ==> ()
                  setPontos(pts) == pontos := pts;

            public getMao : () ==> seq of Peca
```

```
                getMao() == return mao;


          public setMao: seq of Peca ==> ()
                setMao(nova_mao) == mao := nova_mao;



             -- verifica se a letra se encontra na mão
          public verificaLetraMao: char  ==> bool
                        verificaLetraMao(letra) ==
                        (
                        dcl tam_mao : nat := len mao;
                        dcl result : bool := false;
                        dcl i : nat := 1;

                        while(i <= tam_mao) do(

                              if( mao(i).getLetra() = letra)
                                          then result := true;

                              i := i+1;
                        );

                              return result;
                        )
                        pre letra in set
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','
R','S','T','U','V','X','Y','W','Z','_'};

          -- igual ao metodo anterior só que devolve a posicao da peca
na mao
          public verificaLetraMaoPos: char  ==> nat
                        verificaLetraMaoPos(letra) ==
                        (
                        dcl tam_mao : nat := len mao;
                        dcl result : nat := 0;
                        dcl i : nat := 1;

                        while(i <= tam_mao) do(

                              if( mao(i).getLetra() = letra)
                                          then return i;

                              i := i+1;
                        );

                              return result;
                        )
                        pre letra in set
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','
R','S','T','U','V','X','Y','W','Z','_'};


           public verificaLetrPosMao : char * nat ==> bool
                              verificaLetrPosMao(letra,pos) == (

                              dcl result : bool := false;

                              if( mao(pos).getLetra() = letra)
                                    then result := true;
```

```
                        return result;
                    )
                    pre letra in set
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','
R','S','T','U','V','X','Y','W','Z'} and
                    pos >= 0 and pos <= 7;



        public trocaLetraMao: char * Peca ==> ()
                trocaLetraMao(letra_p_trocar,nova_peca) == (

                    dcl tam_mao : nat := len mao;
                    dcl nova_mao : seq of Peca := [];
                    dcl i : nat := 1;

                    while(i <= tam_mao) do (

                    if(verificaLetrPosMao(letra_p_trocar,i) = true
)
                        then skip
                        else nova_mao := nova_mao^[mao(i)];

                        i := i+1;
                    );

                    nova_mao := nova_mao^[nova_peca];

                    setMao(nova_mao);
                )
                pre letra_p_trocar in set
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','
R','S','T','U','V','X','Y','W','Z'};



end Jogador
```

```
Jogador
```

## Classe Peca

- Descrição: classe que representa uma peça do jogo que vai ser colocada no tabuleiro. Cada peça é constituída por uma letra sendo que cada uma tem um valor de pontuação diferente;

- Atributos: letra:char | pontuacao:nat;
    - *Types:* Posicao:: x:nat y:nat;

- Operações:

```
class Peca


instance variables

      private letra: char;
      private pontuacao: nat;

      inv letra  in set
{'A','B','C','Ç','D','E','F','G','H','I','J','K','L','M','N','O','P','
Q','R','S','T','U','V','X','Y','W','Z','_'};
      inv pontuacao >= 0;


operations
            public Peca : char * nat ==> Peca
            Peca(l,pont) == (letra := l; pontuacao := pont)
                  post letra  = l  and pontuacao = pont ;

            public getLetra: () ==> char
                  getLetra() == return letra;


            public getPontuacao : () ==> nat
                  getPontuacao() == return pontuacao
                  post RESULT = pontuacao;

            public setPontuacao: nat ==> ()
                  setPontuacao(pont) ==
                        pontuacao := pont;


            public setLetra: char ==> ()
                  setLetra(letr) ==
                        letra := letr;
end Peca
```

```
Peca
```

## Classe Tabuleiro

- Descrição: Abstracção de um tabuleiro de Scrabble no qual acontecem todas as movimentações de peças;

- Atributos: tabuleiro : map Posicao to Celula | posicoesOcupadas : set of Posicao;
    - *Types*: Posicao:: x:nat y:nat;

- Operações:

```
class Tabuleiro

      -- NOTAS: seq permite repetidos, set nao
      types
            public Posicao:: x:nat  y:nat



      instance variables

            private tabuleiro : map Posicao to Celula := {|->};
        private posicoesOcupadas : set of Posicao := {};

      -- O tabuleiro e constituido por 225 celulas
      inv (card (dom tabuleiro))  <= 226;
      inv (card (rng tabuleiro))  <= 226;
      inv card posicoesOcupadas <= 226 ;


      operations

            public Tabuleiro:() ==> Tabuleiro
                  Tabuleiro() == (

            dcl x:nat :=0;
            dcl y:nat :=0;

              while(y < 15) do (
                  --for all y in set {0,...,19} do (
                   while(x < 15) do(
                         tabuleiro := tabuleiro ++ {mk_Posicao(x,y)|->
new Celula('.')};
                         x := x+1;
                   );

                  x := 0;
                  y := y+1;
                   );

                   -- * - duplica valor letra
                  colocaMultiplicador(0,3,'*');
                  colocaMultiplicador(3,0,'*');
                  colocaMultiplicador(11,0,'*');
                  colocaMultiplicador(0,11,'*');
                  colocaMultiplicador(14,3,'*');
                  colocaMultiplicador(3,14,'*');
                  colocaMultiplicador(11,14,'*');
                  colocaMultiplicador(14,11,'*');
```

```
            colocaMultiplicador(6,6,'*');
            colocaMultiplicador(8,8,'*');
            colocaMultiplicador(8,6,'*');
            colocaMultiplicador(6,8,'*');
            colocaMultiplicador(6,2,'*');
            colocaMultiplicador(2,6,'*');
            colocaMultiplicador(7,4,'*');
            colocaMultiplicador(4,7,'*');
            colocaMultiplicador(8,2,'*');
            colocaMultiplicador(2,8,'*');
            colocaMultiplicador(12,6,'*');
            colocaMultiplicador(6,12,'*');
            colocaMultiplicador(11,7,'*');
            colocaMultiplicador(7,11,'*');
            colocaMultiplicador(12,8,'*');
            colocaMultiplicador(8,12,'*');

            -- # - triplica valor letra
            colocaMultiplicador(1,5,'#');
            colocaMultiplicador(5,1,'#');
            colocaMultiplicador(9,1,'#');
            colocaMultiplicador(1,9,'#');
            colocaMultiplicador(5,13,'#');
            colocaMultiplicador(13,5,'#');
            colocaMultiplicador(9,13,'#');
            colocaMultiplicador(13,9,'#');
            colocaMultiplicador(5,5,'#');
            colocaMultiplicador(9,9,'#');
            colocaMultiplicador(5,9,'#');
            colocaMultiplicador(9,5,'#');


        -- $ - duplica valor palavra
            colocaMultiplicador(1,1,'$');
            colocaMultiplicador(1,13,'$');
            colocaMultiplicador(2,2,'$');
            colocaMultiplicador(2,12,'$');
            colocaMultiplicador(3,3,'$');
            colocaMultiplicador(3,11,'$');
            colocaMultiplicador(4,4,'$');
            colocaMultiplicador(4,10,'$');
            colocaMultiplicador(7,7,'$');
            colocaMultiplicador(10,4,'$');
            colocaMultiplicador(10,10,'$');
            colocaMultiplicador(11,3,'$');
            colocaMultiplicador(11,11,'$');
            colocaMultiplicador(12,2,'$');
            colocaMultiplicador(12,12,'$');
            colocaMultiplicador(13,1,'$');
            colocaMultiplicador(13,13,'$');

            -- % - triplica valor palavra
            colocaMultiplicador(0,0,'%');
            colocaMultiplicador(0,7,'%');
            colocaMultiplicador(0,14,'%');
            colocaMultiplicador(7,0,'%');
            colocaMultiplicador(7,14,'%');
            colocaMultiplicador(14,0,'%');
            colocaMultiplicador(14,7,'%');
            colocaMultiplicador(14,14,'%');
```

```
                return self;
        );

        public getTabuleiro : () ==> map Posicao to Celula
            getTabuleiro() == return tabuleiro;

        public setTabuleiro : map Posicao to Celula ==> ()
                setTabuleiro(tab) == tabuleiro := tab;


        public addToPosOcupadas : Posicao ==> ()
            addToPosOcupadas(pos) ==
                posicoesOcupadas := posicoesOcupadas union {pos};

        public numPosOcupadas : () ==> nat
            numPosOcupadas() == (
                return (card posicoesOcupadas);
        );

        public colocaPeca: Peca * nat * nat ==> ()
            colocaPeca(peca,pos_horiz,pos_vert) == (

                tabuleiro := tabuleiro ++ {
mk_Posicao(pos_horiz,pos_vert) |-> new Celula(peca.getLetra() ) };

        );


       -- Dado um x e um y devolve o caracter que estiver nessa
posicao
        public getCelula: nat * nat ==> char
                getCelula(pos_x,pos_y) == (

                dcl posicao : Posicao := mk_Posicao(pos_x,pos_y);
                dcl simbolo : char := ' ';
                dcl cel : Celula := new Celula(' ');

                cel := tabuleiro(posicao);
                simbolo := cel.getSimbolo();

                return simbolo;
        )
         pre pos_x >= 0 and pos_x <= 15 and pos_y >= 0 and pos_y <=
15;


        public cel_ocupada: nat * nat ==> bool
            cel_ocupada(pos_x,pos_y) == (

            dcl posicao : Posicao := mk_Posicao(pos_x,pos_y);
            dcl cel : Celula := new Celula(' ');
            dcl simbolo : char := ' ';
            dcl result : bool := false;

            cel := tabuleiro(posicao);
            simbolo := cel.getSimbolo();

            if(simbolo = '.' or simbolo ='#' or simbolo ='*' or simbolo
= '%' or simbolo = '$')
```

```
                then result := false
            else result := true;

            return result;
        )
         pre pos_x >= 0 and pos_x <= 15 and pos_y >= 0 and pos_y <=
15;


        -- * - duplica valor letra
        -- # - triplica valor letra
        -- $ - duplica valor palavra
        -- % - triplica valor palavra
        -- Coloca um multiplicador numa determinada posicao do
tabuleiro
       public colocaMultiplicador: nat * nat * char ==> ()
            colocaMultiplicador(pos_x,pos_y,simbolo) == (


                tabuleiro := tabuleiro ++ { mk_Posicao(pos_x,pos_y)
|-> new Celula(simbolo) };
            )
            pre simbolo in set {'#','*','$','%'} and pos_x >= 0 and
pos_x <= 15 and pos_y >= 0 and pos_y <= 15;


       -- Verifica se uma determina posicao do tabuleiro tem um
multiplicador
       public verificaMultiplicador: nat * nat ==> bool
                verificaMultiplicador(pos_x,pos_y) == (

                dcl posicao : Posicao := mk_Posicao(pos_x,pos_y);
                  dcl cel : Celula := new Celula(' ');
                dcl simbolo : char := ' ';

                cel := tabuleiro(posicao);
                simbolo := cel.getSimbolo();

                if simbolo = '#' then return true;
                if simbolo = '*' then return true;
                if simbolo = '%' then return true;
                if simbolo = '$' then return true
                  else return false
              )
              pre pos_x >= 0 and pos_x <= 15 and pos_y >= 0 and pos_y
<= 15;
end Tabuleiro
```

```
Tabuleiro
```

## Classe Jogo

- Descrição: Nesta classe estão localizadas as principais operações/funções que controlam o jogo;

- Atributos: baralho: seq of Peca | dicionario: set of String | jogadores: seq of Jogador |orientacao: char ;
    - *Types:* String = seq of char; | Posicao:: x:nat  y:nat;

- Operações:

```
class Jogo

      types
            public String = seq of char;

      instance variables

            private baralho : seq of Peca := [];
            private dicionario : set of String := {};
            private jogadores : seq of Jogador := [];
            private tabuleiro : Tabuleiro;
            --1 se e a vez do jogador1 jogar, 2 se e a vez do jogador 2
            private vez : nat1 := 1;

            inv len baralho <= 120;
            inv len jogadores <= 2;
            inv vez = 1 or vez = 2;

      operations

             public Jogo : () ==> Jogo
                        Jogo() == (

                        criar_baralho();
                        tabuleiro := new Tabuleiro();

                        return self;
                  );



            public Jogo :  seq of Jogador ==> Jogo
               Jogo(jogs) == (

               jogadores := jogs ;
               tabuleiro := new Tabuleiro();
               criar_baralho();
```

```
            return self;
        )
        post len jogadores <= 2;

    public getVez: () ==> nat1
            getVez() == return vez;

     public setVez: nat1 ==> ()
            setVez(n_vez) == vez := n_vez;

    public getDicionario: () ==> set of String
                getDicionario() == return dicionario;

        -- o set de Strings que e passado como parametro e gerado
por uma funcao em Java
        public setDicionario: set of String ==> ()
                setDicionario(dic) ==dicionario := dic;

        public getTabuleiro: () ==> Tabuleiro
                    getTabuleiro() == return tabuleiro;


        public setTabuleiro: Tabuleiro ==> ()
                        setTabuleiro(tab) == tabuleiro := tab;


        public getJogadores: () ==> seq of Jogador
                getJogadores() == return jogadores;

        public setJogadores: seq of Jogador ==> ()
                setJogadores(jogs) ==jogadores := jogs;


        public getBaralho: () ==> seq of Peca
           getBaralho() == return baralho;

        public setBaralho: seq of Peca ==> ()
              setBaralho(bar) == baralho := bar;

        public criar_baralho: () ==> ()
                    criar_baralho() == (

        dcl peca1 : Peca := new Peca('A',  1);
        dcl peca2 : Peca := new Peca('A',  1);
        dcl peca3 : Peca := new Peca('A',  1);
        dcl peca4 : Peca := new Peca('A',  1);
        dcl peca5 : Peca := new Peca('A',  1);
        dcl peca6 : Peca := new Peca('A',  1);
        dcl peca7 : Peca := new Peca('A',  1);
        dcl peca8 : Peca := new Peca('A',  1);
        dcl peca9 : Peca := new Peca('A',  1);
        dcl peca10 : Peca := new Peca('A', 1);
        dcl peca11 : Peca := new Peca('A', 1);
        dcl peca12 : Peca := new Peca('A', 1);
        dcl peca13 : Peca := new Peca('A', 1);
        dcl peca14 : Peca := new Peca('A', 1);

        dcl peca15 : Peca := new Peca('B', 3);
        dcl peca16 : Peca := new Peca('B', 3);
        dcl peca17 : Peca := new Peca('B', 3);
```

```
            dcl peca18 : Peca := new Peca('C', 2);
            dcl peca19 : Peca := new Peca('C', 2);
            dcl peca20 : Peca := new Peca('C', 2);
            dcl peca21 : Peca := new Peca('C', 2);

            dcl peca22 : Peca := new Peca('Ç', 3);
            dcl peca23 : Peca := new Peca('Ç', 3);

            dcl peca24 : Peca := new Peca('D', 2);
            dcl peca25 : Peca := new Peca('D', 2);
            dcl peca26 : Peca := new Peca('D', 2);
            dcl peca27 : Peca := new Peca('D', 2);
            dcl peca28 : Peca := new Peca('D', 2);

            dcl peca29 : Peca := new Peca('E', 1);
            dcl peca30 : Peca := new Peca('E', 1);
            dcl peca31 : Peca := new Peca('E', 1);
            dcl peca32 : Peca := new Peca('E', 1);
            dcl peca33 : Peca := new Peca('E', 1);
            dcl peca34 : Peca := new Peca('E', 1);
            dcl peca35 : Peca := new Peca('E', 1);
            dcl peca36 : Peca := new Peca('E', 1);
            dcl peca37 : Peca := new Peca('E', 1);
            dcl peca38 : Peca := new Peca('E', 1);
            dcl peca39 : Peca := new Peca('E', 1);

            dcl peca40 : Peca := new Peca('F', 4);
            dcl peca41 : Peca := new Peca('F', 4);

            dcl peca42 : Peca := new Peca('G', 4);
            dcl peca43 : Peca := new Peca('G', 4);

            dcl peca44 : Peca := new Peca('H', 4);
            dcl peca45 : Peca := new Peca('H', 4);

            dcl peca46 : Peca := new Peca('I', 1);
            dcl peca47 : Peca := new Peca('I', 1);
            dcl peca48 : Peca := new Peca('I', 1);
            dcl peca49 : Peca := new Peca('I', 1);
            dcl peca50 : Peca := new Peca('I', 1);
            dcl peca51 : Peca := new Peca('I', 1);
            dcl peca52 : Peca := new Peca('I', 1);
            dcl peca53 : Peca := new Peca('I', 1);
            dcl peca54 : Peca := new Peca('I', 1);
            dcl peca55 : Peca := new Peca('I', 1);

            dcl peca56 : Peca := new Peca('J', 5);
            dcl peca57 : Peca := new Peca('J', 5);

            dcl peca58 : Peca := new Peca('L', 2);
            dcl peca59 : Peca := new Peca('L', 2);
            dcl peca60 : Peca := new Peca('L', 2);
            dcl peca61 : Peca := new Peca('L', 2);
            dcl peca62 : Peca := new Peca('L', 2);

            dcl peca63 : Peca := new Peca('M', 1);
            dcl peca64 : Peca := new Peca('M', 1);
            dcl peca65 : Peca := new Peca('M', 1);
            dcl peca66 : Peca := new Peca('M', 1);
            dcl peca67 : Peca := new Peca('M', 1);
            dcl peca68 : Peca := new Peca('M', 1);
```

```
dcl peca69 : Peca := new Peca('N', 3);
dcl peca70 : Peca := new Peca('N', 3);
dcl peca71 : Peca := new Peca('N', 3);
dcl peca72 : Peca := new Peca('N', 3);

dcl peca73 : Peca := new Peca('O', 1);
dcl peca74 : Peca := new Peca('O', 1);
dcl peca75 : Peca := new Peca('O', 1);
dcl peca76 : Peca := new Peca('O', 1);
dcl peca77 : Peca := new Peca('O', 1);
dcl peca78 : Peca := new Peca('O', 1);
dcl peca79 : Peca := new Peca('O', 1);
dcl peca80 : Peca := new Peca('O', 1);
dcl peca81 : Peca := new Peca('O', 1);
dcl peca82 : Peca := new Peca('O', 1);

dcl peca83 : Peca := new Peca('P', 2);
dcl peca84 : Peca := new Peca('P', 2);
dcl peca85 : Peca := new Peca('P', 2);
dcl peca86 : Peca := new Peca('P', 2);

dcl peca87 : Peca := new Peca('Q', 6);

dcl peca88 : Peca := new Peca('R', 1);
dcl peca89 : Peca := new Peca('R', 1);
dcl peca90 : Peca := new Peca('R', 1);
dcl peca91 : Peca := new Peca('R', 1);
dcl peca92 : Peca := new Peca('R', 1);
dcl peca93 : Peca := new Peca('R', 1);

dcl peca94 : Peca := new Peca('S', 1);
dcl peca95 : Peca := new Peca('S', 1);
dcl peca96 : Peca := new Peca('S', 1);
dcl peca97 : Peca := new Peca('S', 1);
dcl peca98 : Peca := new Peca('S', 1);
dcl peca99 : Peca := new Peca('S', 1);
dcl peca100 : Peca := new Peca('S', 1);
dcl peca101 : Peca := new Peca('S', 1);

dcl peca102 : Peca := new Peca('T', 1);
dcl peca103 : Peca := new Peca('T', 1);
dcl peca104 : Peca := new Peca('T', 1);
dcl peca105 : Peca := new Peca('T', 1);
dcl peca106 : Peca := new Peca('T', 1);

dcl peca107 : Peca := new Peca('U', 1);
dcl peca108 : Peca := new Peca('U', 1);
dcl peca109 : Peca := new Peca('U', 1);
dcl peca110 : Peca := new Peca('U', 1);
dcl peca111 : Peca := new Peca('U', 1);
dcl peca112 : Peca := new Peca('U', 1);
dcl peca113 : Peca := new Peca('U', 1);

dcl peca114 : Peca := new Peca('V', 4);
dcl peca115 : Peca := new Peca('V', 4);

dcl peca116 : Peca := new Peca('X', 8);

dcl peca117 : Peca := new Peca('Z', 8);
```

```
            dcl peca118 : Peca := new Peca('_',0);
            dcl peca119 : Peca := new Peca('_',0);
            dcl peca120 : Peca := new Peca('_',0);

            baralho := baralho^[peca1];
            baralho := baralho^[peca2];
            baralho := baralho^[peca3];
            baralho := baralho^[peca4];
            baralho := baralho^[peca5];
            baralho := baralho^[peca6];
            baralho := baralho^[peca7];
            baralho := baralho^[peca8];
            baralho := baralho^[peca9];
            baralho := baralho^[peca10];
            baralho := baralho^[peca11];
            baralho := baralho^[peca12];
            baralho := baralho^[peca13];
            baralho := baralho^[peca14];
            baralho := baralho^[peca15];
            baralho := baralho^[peca16];
            baralho := baralho^[peca17];
            baralho := baralho^[peca18];
            baralho := baralho^[peca19];
            baralho := baralho^[peca20];
            baralho := baralho^[peca21];
            baralho := baralho^[peca22];
            baralho := baralho^[peca23];
            baralho := baralho^[peca24];
            baralho := baralho^[peca25];
            baralho := baralho^[peca26];
            baralho := baralho^[peca27];
            baralho := baralho^[peca28];
            baralho := baralho^[peca29];
            baralho := baralho^[peca30];
            baralho := baralho^[peca31];
            baralho := baralho^[peca32];
            baralho := baralho^[peca33];
            baralho := baralho^[peca34];
            baralho := baralho^[peca35];
            baralho := baralho^[peca36];
            baralho := baralho^[peca37];
            baralho := baralho^[peca38];
            baralho := baralho^[peca39];
            baralho := baralho^[peca40];
            baralho := baralho^[peca41];
            baralho := baralho^[peca42];
            baralho := baralho^[peca43];
            baralho := baralho^[peca44];
            baralho := baralho^[peca45];
            baralho := baralho^[peca46];
            baralho := baralho^[peca47];
            baralho := baralho^[peca48];
            baralho := baralho^[peca49];
            baralho := baralho^[peca50];
            baralho := baralho^[peca51];
            baralho := baralho^[peca52];
            baralho := baralho^[peca53];
            baralho := baralho^[peca54];
            baralho := baralho^[peca55];
            baralho := baralho^[peca56];
            baralho := baralho^[peca57];
```

```
            baralho := baralho^[peca58];
            baralho := baralho^[peca59];
            baralho := baralho^[peca60];
            baralho := baralho^[peca61];
            baralho := baralho^[peca62];
            baralho := baralho^[peca63];
            baralho := baralho^[peca64];
            baralho := baralho^[peca65];
            baralho := baralho^[peca66];
            baralho := baralho^[peca67];
            baralho := baralho^[peca68];
            baralho := baralho^[peca69];
            baralho := baralho^[peca70];
            baralho := baralho^[peca71];
            baralho := baralho^[peca72];
            baralho := baralho^[peca73];
            baralho := baralho^[peca74];
            baralho := baralho^[peca75];
            baralho := baralho^[peca76];
            baralho := baralho^[peca77];
            baralho := baralho^[peca78];
            baralho := baralho^[peca79];
            baralho := baralho^[peca80];
            baralho := baralho^[peca81];
            baralho := baralho^[peca82];
            baralho := baralho^[peca83];
            baralho := baralho^[peca84];
            baralho := baralho^[peca85];
            baralho := baralho^[peca86];
            baralho := baralho^[peca87];
            baralho := baralho^[peca88];
            baralho := baralho^[peca89];
            baralho := baralho^[peca90];
            baralho := baralho^[peca91];
            baralho := baralho^[peca92];
            baralho := baralho^[peca93];
            baralho := baralho^[peca94];
            baralho := baralho^[peca95];
            baralho := baralho^[peca96];
            baralho := baralho^[peca97];
            baralho := baralho^[peca98];
            baralho := baralho^[peca99];
            baralho := baralho^[peca100];
            baralho := baralho^[peca101];
            baralho := baralho^[peca102];
            baralho := baralho^[peca103];
            baralho := baralho^[peca104];
            baralho := baralho^[peca105];
            baralho := baralho^[peca106];
            baralho := baralho^[peca107];
            baralho := baralho^[peca108];
            baralho := baralho^[peca109];
            baralho := baralho^[peca110];
            baralho := baralho^[peca111];
            baralho := baralho^[peca112];
            baralho := baralho^[peca113];
            baralho := baralho^[peca114];
            baralho := baralho^[peca115];
            baralho := baralho^[peca116];
            baralho := baralho^[peca117];
            baralho := baralho^[peca118];
```

```
        baralho := baralho^[peca119];
        baralho := baralho^[peca120];


    );

    public criaJogadores: seq of Peca * seq of Peca ==> ()
     criaJogadores(m1,m2) == (

          dcl jog1 : Jogador := new Jogador(1);
          dcl jog2 : Jogador := new Jogador(2);

          jog1.setMao(m1);
          jog2.setMao(m2);

          jogadores := jogadores ^ [jog1,jog2];
    )
    pre len m1 <= 7 and len m2 <= 7;


    public criaMaoJoador: () ==> seq of Peca
                    criaMaoJoador() == (

                          dcl mao : seq of Peca := [];
                          dcl i : nat := 1;

                          while i <= 7 do
                          (

                                mao := mao^[baralho(i)];

                                i:= i +1;
                          );

                          return mao;

                );



    public preencheMao: nat ==> ()
           preencheMao (i) == (

             dcl tmp : seq of Peca := jogadores(i).getMao();

           if(len jogadores(i).getMao() < 7) then (
           while(len tmp < 7) do

                   (tmp := tmp ^ [baralho(len baralho-1)];
                   jogadores(i).setMao(tmp);
           )
          );
     )
     pre i = 1 or i = 2;

     public actualizaPontos: nat * Jogador * nat * nat ==> ()
                actualizaPontos(pts_adicionar, jog, pos1, pos2) == (

                      dcl tab : Tabuleiro := new Tabuleiro();

                      if tab.verificaMultiplicador(pos1, pos2) then (
```

```
                        dcl cel : Celula := new Celula(' ');
                  dcl simbolo : char := ' ';
                        -- cel := tab(mk Tabuleiro`Posicao(pos1,pos2));
                        simbolo := cel.getSimbolo();

                        if simbolo = '#' then
jog.setPontos(jog.getPontos()+ 3 * pts_adicionar);
                        if simbolo = '*' then
jog.setPontos(jog.getPontos()+ 2 * pts_adicionar);
                     --if simbolo = '%' then
jog.setPontos(og.getPontos()+  * pts_adicionar);
                     --if simbolo = '$' then
jog.setPontos(jog.getPontos()+ * pts_adicionar);
              ) else

                  jog.setPontos(jog.getPontos()+pts_adicionar);
            )
            pre pts_adicionar >= 0 and pos1 >= 0 and pos1 <= 15 and
pos2 >= 0 and pos2 <= 15;

            public comparaPecas: Peca * Peca ==> bool
                  comparaPecas(peca1,peca2) == (

                        dcl letra1 : char := peca1.getLetra();
                        dcl letra2 : char := peca2.getLetra();
                        dcl result : bool := false;

                        if(letra1 = letra2)
                              then  result := true
                        else result := false;

                        return result;
                  );


            -- TRUE se palavras iguais, senão FALSE
            public comparaPalavras: String * String ==> bool
                              comparaPalavras(pal1,pal2) == (

                              dcl tam_pal1 : nat := len pal1;
                              dcl tam_pal2 : nat := len pal2;
                              dcl i : nat1 := 1;

                              if(tam_pal1 <> tam_pal2) then return
false;

                              while(i <= tam_pal1) do
                              (
                                    if(pal1(i) <> pal2(i))
                                          then return false;

                                    i := i+1;

                              );

                              return true;
                        );

            -- RETURN true, se a palavra existe no dicionário
```

```
            public verificaPalDicionario: String * set of seq of char
==> bool
                              verificaPalDicionario(palavra,dicion) ==
(

                              dcl result : bool := false;

                              result := palavra in set dicion;

                              return result;
                  );


            -- recebe o jogador que vai passar a vez
            public passarVez: Jogador ==> ()
                        passarVez(jog) == (

                        if(jog.getID() = jogadores(1).getID())
                              then setVez(2);

                        if(jog.getID() = jogadores(2).getID())
                              then setVez(1);

                        );

            public retiraPecaMao: Jogador * Peca ==> ()
                              retiraPecaMao(jogador,pec) == (

                              dcl mao_jog : seq of Peca :=
jogador.getMao();
                              dcl tam_mao : nat := len
jogador.getMao();
                              dcl nova_mao : seq of Peca := [];
                              dcl nao_procura_mais : bool := false;
                              dcl letra_p_retirar : char :=
pec.getLetra();
                              dcl i : nat1 := 1;

                              while(i <= tam_mao) do (


      if(jogador.verificaLetrPosMao(letra_p_retirar,i) = true and
nao_procura_mais = false  )
                              then nao_procura_mais := true

                              else nova_mao := nova_mao^[mao_jog(i)];

                              i := i+1;
                        );

                        jogador.setMao(nova_mao);

                  );


                  public retiraPecaBaralho: nat ==> Peca
                              retiraPecaBaralho(pos_aleat) == (

                              dcl baralho : seq of Peca;
                              dcl tam_bar : nat := len getBaralho();
                              dcl baralho_act : seq of Peca := [];
```

```
                              dcl nao_procura_mais : bool := false;
                              dcl peca_ret : Peca;
                              dcl i : nat1 := 1;

                              baralho := getBaralho();
                              peca_ret := baralho(pos_aleat);

                              while(i <= tam_bar) do (

                              if(   peca_ret.getLetra() =
baralho(i).getLetra() and nao_procura_mais = false)
                                          then nao_procura_mais := true

                                          else baralho_act :=
baralho_act^[baralho(i)];

                                    i := i+1;
                        );



                              setBaralho(baralho_act);

                              return peca_ret;
                        )
                        pre pos_aleat <= 120;


            -- jogada que coloca uma palavra no tabuleiro
            -- retorna um inteiro conforme a situacao
            public jogada: Jogador * nat * nat * String * char ==>  nat
                        jogada(jogador,pos_x,pos_y,palavra,orient) == (

                        -- result = 0 em caso de nao se poder colocar a
palavra, 1 se sucesso
                        dcl result : nat := 0;
                        dcl dicio :  set of String;
                        dcl tabuleiro : Tabuleiro := getTabuleiro();
                        dcl mao_jogador : seq of Peca :=
jogador.getMao();
                        dcl i : nat1 := 1;

                        dicio := getDicionario();

                        -- Se a palavra que o jogador quer inserir não
se encontra no dicionario retorna FALSE
                        if ( verificaPalDicionario(palavra,dicionario)
= false)
                                    then return 3;

                        -- Vai colocar a palavra na vertical
                        -- percorre a palavra
                        while(i <= len palavra) do
                        (
                              if(orient = 'b' or orient = 'B') then
                              (
                                    -- se a celula esta livre em baixo
esta livre
                                    if(
tabuleiro.cel_ocupada(pos_x,pos_y + i) = false) then
                                    (
```

```
                                        if(
jogador.verificaLetraMao(palavra(i)) = true) then
                                        (
                                              dcl pos_na_mao : nat :=
jogador.verificaLetraMaoPos(palavra(i));

                                              tabuleiro.colocaPeca(
new
Peca(palavra(i),mao_jogador(pos_na_mao).getPontuacao()),pos_x,pos_y+i)
;

      retiraPecaMao(jogador,mao_jogador(pos_na_mao) );
                                              result := 1;
                                        );

                                  );

                              )else(
                                        -- se vai colocar a palavra
na horizontal
                                        if(orient = 'd' or orient =
'D') then(

                                              -- se a celula esta
livre em baixo
                                              if(
tabuleiro.cel_ocupada(pos_x+i,pos_y) = false) then
                                              (
                                                    if(
jogador.verificaLetraMao(palavra(i)) = true) then
                                                    (
                                                          dcl
pos_na_mao : nat := jogador.verificaLetraMaoPos(palavra(i));


      tabuleiro.colocaPeca( new
Peca(palavra(i),mao_jogador(pos_na_mao).getPontuacao()),pos_x+i,pos_y)
;

      retiraPecaMao(jogador,mao_jogador(pos_na_mao));
                                                                result :=
1;
                                                    );
                                              );
                                        );
                              );

                              i := i+1;
                        );


                        return result;
                  )
                  pre pos_x >= 0 and pos_x <= 15 and pos_y >= 0 and
pos_y <= 15 and orient = 'd' or orient = 'D' or orient = 'b' or orient
= 'B';

end Jogo
```

```
Jogo
```

## Informação de cobertura dos testes

Classe Celula:

| ***name*** | ***#calls*** | ***coverage*** |
|---|---|---|
| Celula`Celula | 2013 | 100% |
| Celula`getSimbolo | 6 | 100% |
| Celula`setSimbolo | 0 | 0% |
| ***total*** | | ***76%*** |

Classe Peca:

| ***name*** | ***#calls*** | ***coverage*** |
|---|---|---|
| Peca`Peca | 15 | 25% |
| Peca`getLetra | 0 | 0% |
| Peca`setLetra | 0 | 0% |
| Peca`getPontuacao | 0 | 0% |
| Peca`setPontuacao | 0 | 0% |
| ***total*** | | ***13%*** |

Classe Jogador:

| ***name*** | ***#calls*** | ***coverage*** |
|---|---|---|
| Jogador`getID | 0 | 0% |
| Jogador`setID | 0 | 0% |
| Jogador`getMao | 0 | 0% |
| Jogador`setMao | 0 | 0% |
| Jogador`Jogador | 3 | 100% |

| | | |
|---|---|---|
| Jogador`getPontos | 1 | 100% |
| Jogador`setPontos | 0 | 0% |
| Jogador`trocaLetraMao | 0 | 0% |
| Jogador`verificaLetraMao | 0 | 0% |
| Jogador`verificaLetrPosMao | 0 | 0% |
| Jogador`verificaLetraMaoPos | 0 | 0% |
| ***total*** | | ***5%*** |

Classe Tabuleiro:

| ***name*** | ***#calls*** | ***coverage*** |
|---|---|---|
| Tabuleiro`Tabuleiro | 7 | 100% |
| Tabuleiro`getCelula | 3 | 100% |
| Tabuleiro`colocaPeca | 0 | 0% |
| Tabuleiro`cel_ocupada | 0 | 0% |
| Tabuleiro`getTabuleiro | 1 | 100% |
| Tabuleiro`setTabuleiro | 0 | 0% |
| Tabuleiro`numPosOcupadas | 1 | 100% |
| Tabuleiro`addToPosOcupadas | 2 | 100% |
| Tabuleiro`colocaMultiplicador | 432 | 100% |
| Tabuleiro`verificaMultiplicador | 3 | 90% |
| ***total*** | | ***86%*** |

Classe Jogo:

| ***name*** | ***#calls*** | ***coverage*** |
|---|---|---|
| Jogo`getVez | 0 | 0% |
| Jogo`jogada | 0 | 0% |
| Jogo`setVez | 0 | 0% |
| Jogo`Jogo | 7 | 28% |
| Jogo`passarVez | 0 | 0% |
| Jogo`getBaralho | 0 | 0% |
| Jogo`setBaralho | 0 | 0% |
| Jogo`preencheMao | 0 | 0% |
| Jogo`comparaPecas | 0 | 0% |
| Jogo`getJogadores | 0 | 0% |
| Jogo`getTabuleiro | 0 | 0% |
| Jogo`setJogadores | 0 | 0% |
| Jogo`setTabuleiro | 0 | 0% |
| Jogo`criaJogadores | 0 | 0% |
| Jogo`criaMaoJoador | 0 | 0% |
| Jogo`criar_baralho | 9 | 0% |
| Jogo`getDicionario | 0 | 0% |
| Jogo`retiraPecaMao | 0 | 0% |
| Jogo`setDicionario | 0 | 0% |
| Jogo`actualizaPontos | 0 | 0% |
| Jogo`comparaPalavras | 0 | 0% |
| Jogo`retiraPecaBaralho | 0 | 0% |
| Jogo`Jogo | 2 | 77% |

| | | |
|---|---|---|
| Jogo`verificaPalDicionario | 0 | 0% |
| ***total*** | | ***0%*** |

## Análise da consistência do modelo

A construção do modelo apesar de incompleta, pois faltou implementar algumas funcionalidades que o jogo requer, foi relativamente bem efectuada. Para validar, de modo a obter um maior grau de confiança no modelo, existem três abordagens:

- Propriedades de integração;
- Testes sistemáticos;
- Protótipos;

As propriedades de integração foram geradas automaticamente pela ferramenta VDMTools garantindo assim já algum nível de confiança na robustez do sistema.

A segunda opção foi a escolha para a implementação manual de testes, que além de permitir saber se as operações/funções estavam a funcionar de acordo com o pretendido, permite estabelecer um grau de confiança relativamente elevado no modelo apresentado. Apesar de termos testado quase todas as operações implementadas acabamos por não escrever um teste para testar o funcionamento do jogo em geral, desde o inicio da partida até ao fim desta.

Quanto ao último ponto , a ferramenta *Overture* não produz nenhum tipo de *output*, o que faz com que a fiabilidade dos testes tenha que ser maior. Nós optamos então por começar a desenvolver uma interface gráfica em *Java,* mas infelizmente devido à chegada do limite de entrega do trabalho vimo-nos forçados a parar com o desenvolvimento desta, tendo ela ficado incompleta.

# Conclusão

O projecto permitiu aos membros da equipa obter bons conhecimentos no que diz respeito ao desenvolvimento e verificação de sistemas através de VDM++. Este tipo de abordagem por abstrações permitiu obter um maior nível de qualidade e fiabilidade na implementação de um software, muito devido também à utilização de invariantes, pré e pós condições.

A linguagem de VDM++ possui várias formas de testar o mesmo modelo, como foi explicado no ponto anterior, o que a torna muito útil a nível de fiabilidade de software, o que é muito importante em sistemas a serem utilizados em operações de alto risco em que nada pode falhar(p.ex. Lançamento de foguetões, etc).

Por fim, apesar de não termos conseguido acabar a interface básica nem cobrir a funcionalidade total de um jogo desde o seu inicio até à sua conclusão determinando o vencedor da partida, achamos que identificamos as potencialidades deste tipo de implementação e linguagem, podendo esta vir a ser uma futura ferramenta no desenvolvimento de software.