

Mestrado Integrado em Engenharia Informática e Computação

**Métodos Formais em Engenharia de Software**

**2013/2014**

**Relatório Final**

**T10 Cascade Game em VDM++**

Turma: 4MIEIC04

Grupo: T4G10

Miao Sun – 200803743 – [ei08162@fe.up.pt](mailto:ei08162@fe.up.pt)

Victor Filipe Carneiro Cerqueira – 201009027 – ei10055@fe.up.pt

08 de Dezembro de 2013

Índice

[1. Descrição 3](#_Toc374303042)

[2. Lista de Requisitos 4](#_Toc374303043)

[3. Principais Restrições ao Funcionamento Correto 5](#_Toc374303044)

[4. Especificação em VDM++ 6](#_Toc374303045)

[5. Diagrama de classes UML 7](#_Toc374303046)

[6. Classes e Scripts de Teste 8](#_Toc374303047)

[Classe TestCell 8](#_Toc374303048)

[Classe TestBoard 9](#_Toc374303049)

[Classe TestCascade 13](#_Toc374303050)

[7. Matriz de Rastreabilidade 15](#_Toc374303051)

[8. Definição completa das classes 16](#_Toc374303052)

[9. Cobertura de Testes 17](#_Toc374303053)

[Classe Cell 17](#_Toc374303054)

[Classe TestCell 17](#_Toc374303055)

[Classe Board 17](#_Toc374303056)

[Classe TestBoard 18](#_Toc374303057)

[Classe Cascade 18](#_Toc374303058)

[Classe TestCascade 18](#_Toc374303059)

[10. Consistência do Modelo 20](#_Toc374303060)

# 1. Descrição

Este trabalho foi realizado no âmbito da unidade curricular de Métodos Formais e Engenharia de Software. O objetivo é elaborar e aplicar os conhecimentos desenvolvidos para implementar um sistema de software em VDM++. O tema escolhido pelo grupo foi Cascade Game, que é um jogo de tabuleiro que contém 3 cores diferentes de blocos, o jogador tem que selecionar grupos que contém igual ou mais que 3 blocos da mesma cor para os eliminar, mais blocos eliminado duma vez mais pontuação recebe. Tem um bónus de uma bomba se conseguir eliminar igual ou mais que 15 blocos numa vez, se selecionar, a bomba elimina os blocos perto de si. O sistema insere uma nova linha de blocos com um intervalo constante. O jogador perde o jogo quando deixar os blocos exceder o tamanho do tabuleiro.

Foi usado o IDE Overture e editor de texto Microsoft Word para a realização da especificação formal do jogo, VDM++ Toolbox para validação de sintaxes e consistência da especificação, testar o funcionamento das funções e converter a especificação em código fonte Java, finalmente usado IDE Eclipse para a implementação da interface gráfica e umas funções que não podem ser realizado em VDM++.

# 

# 2. Lista de Requisitos

|  |  |
| --- | --- |
| **Código Identificador** | **Requisitos** |
| REQ01 | Os blocos só podem conter cores “red”, “yellow”, “green”, ou ser vazio ou conter uma bomba. |
| REQ02 | Atualizar o tabuleiro após de cada jogada, de forma não deixar existir espaço entre blocos verticalmente e a primeira linha do tabuleiro horizontalmente. |
| REQ03 | O sistema insere automaticamente uma linha de blocos num determinado intervalo de tempo. |
| REQ04 | Atualizar o tabuleiro após de cada inserção de blocos feita automaticamente pelo sistema. |
| REQ05 | Atribuir um bloco especial (bomba) se eliminar igual ou mais que 15 blocos numa só jogada. |
| REQ06 | Calcula a pontuação de cada jogada, e atualizar a pontuação global do jogo após cada jogada. |
| REQ07 | Terminar jogo se os blocos exceder o tamanho do tabuleiro verticalmente. |
| REQ08 | O jogo tem vários níveis, entre automaticamente no próximo nível se o jogador conseguir manter jogar durante um determinado tempo. |
| REQ09 | O jogador ganha o jogo se conseguir terminar o último nível do jogo. |

# 3. Principais Restrições ao Funcionamento Correto

|  |  |
| --- | --- |
| **Código** | **Restrição** |
| RES01 | O tabuleiro tem um limite de tamanho determinado, os blocos não pode exceder o limite do tabuleiro. |
| RES02 | O jogo começa com 6 linhas de blocos pré-gerados. |
| RES03 | Na geração de blocos não pode gerar um bloco vazio nem gerar uma bomba, tem que conter uma cor ou “red” ou “yellow” ou “green”. |
| RES04 | O sistema só insere automaticamente linhas de blocos quando a próxima iteração não termina o jogo. |
| RES05 | Não pode tiver nenhum bloco exceder o limite vertical do tabuleiro, se tiver termina o jogo. |
| RES06 | Não pode selecionar blocos vazios para obter vizinhos vazios ou eliminar grupo de blocos vazios. |
| RES07 | Só pode eliminar grupos de blocos construído por 3 ou mais blocos de mesmo cor, exceto o bloco especial (bomba). |
| RES08 | Só a bomba pode destruir blocos a volta, seja qualquer cor for. |
| RES09 | Não pode existir espaço entre blocos verticalmente. |
| RES10 | Não pode existir espaço entre blocos horizontalmente na linha base do tabuleiro. |
| RES11 | O jogo começa no nível 1, e tem um limite de níveis determinados. Não pode exceder o limite de níveis. |
| RES12 | O jogo só atribui pontuação positivos às jogadas que eliminam igual ou mais que 3 blocos. |
| RES13 | O jogo só entre no próximo nível se o jogador conseguir manter jogar durante determinado tempo. |

4. Especificação em VDM++

5. Diagrama de classes UML

6. Classes e Scripts de Teste

## Classe TestCell

class TestCell

operations

public AssertTrue : bool ==> ()

AssertTrue(a) == return

pre a;

public runAllTestsCell : () ==> ()

runAllTestsCell() ==

(

TestCreateCell();

TestClearCell();

TestMakeBomb();

);

public TestCreateCell : () ==> ()

TestCreateCell() ==

(

dcl cell : Cell := new Cell(<red>);

dcl cell2: Cell := new Cell();

AssertTrue(cell.color = <red>);

AssertTrue(cell2.color = <empty>);

);

public TestClearCell : () ==> ()

TestClearCell() ==

(

dcl cell : Cell := new Cell(<red>);

cell.clear();

AssertTrue(cell.color = <empty>);

);

public TestMakeBomb : () ==> ()

TestMakeBomb() ==

(

dcl cell : Cell := new Cell();

cell.makeBomb();

AssertTrue(cell.color = <bomb>);

);

end TestCell

## Classe TestBoard

class TestBoard

instance variables

bd\_h : Board := new Board();

bd\_v : Board := new Board();

operations

public AssertTrue : bool ==> ()

AssertTrue(a) == return

pre a;

public AuxBoard\_H : () ==> ()

AuxBoard\_H() ==

(

bd\_h.pieces := bd\_h.pieces ++ {mk\_Board`Position(6,6) |-> new Cell(<red>), mk\_Board`Position(6,7) |-> new Cell(<red>), mk\_Board`Position(6,8) |-> new Cell(<red>), mk\_Board`Position(5,6) |-> new Cell(<green>), mk\_Board`Position(5,7) |-> new Cell(<yellow>), mk\_Board`Position(5,8) |-> new Cell(<green>), mk\_Board`Position(7,6) |-> new Cell(<green>), mk\_Board`Position(7,7) |-> new Cell(<bomb>), mk\_Board`Position(7,8) |-> new Cell(<green>), mk\_Board`Position(6,5) |-> new Cell(<green>), mk\_Board`Position(6,9) |-> new Cell(<empty>)};

);

public AuxBoard\_V : () ==> ()

AuxBoard\_V() ==

(

bd\_v.pieces := bd\_v.pieces ++ {mk\_Board`Position(5, y) |-> new Cell() | y in set {0,...,5}};

bd\_v.pieces := bd\_v.pieces ++ {mk\_Board`Position(6, y) |-> new Cell() | y in set {0,...,5}};

);

public runAllTestsBoard : () ==> ()

runAllTestsBoard () ==

(

TestCreateBoard();

TestPrint();

TestPrintCell();

TestGenerateColor();

TestInsertNewLine();

TestWillFinish();

TestSelectPiece();

TestSelectPieceIter();

TestGetNeighbours();

TestDeletePieces();

TestExplode();

TestFixBoard();

TestFixBoardVertical();

TestDistLeft();

TestDistRight();

TestMoveLeft();

TestMoveRight();

);

public TestCreateBoard : () ==> ()

TestCreateBoard() ==

(

dcl board : Board := new Board();

AssertTrue(board.pieces <> {|->});

);

public TestPrint : () ==> ()

TestPrint() ==

(

dcl board : Board := new Board();

AssertTrue(board.print() = true);

);

public TestPrintCell : () ==> ()

TestPrintCell() ==

(

dcl c1 : Cell := new Cell();

dcl c2 : Cell := new Cell(<red>);

dcl c3 : Cell := new Cell(<green>);

dcl c4 : Cell := new Cell(<yellow>);

dcl c5 : Cell := new Cell(<bomb>);

dcl io : IO := new IO();

AssertTrue(Board`printCell(c1) = true);

AssertTrue(Board`printCell(c2) = io.echo("R "));

AssertTrue(Board`printCell(c3) = io.echo("G "));

AssertTrue(Board`printCell(c4) = io.echo("Y "));

AssertTrue(Board`printCell(c5) = io.echo("# "));

);

public TestGenerateColor : () ==> ()

TestGenerateColor() ==

(

dcl board : Board := new Board();

dcl c : Board`Color := board.generateColor();

AssertTrue(c in set {<yellow>, <green>, <red>});

);

public TestInsertNewLine : () ==> ()

TestInsertNewLine() ==

(

dcl board : Board := new Board();

dcl oldPieces : map Board`Position to Cell := board.pieces;

AssertTrue(oldPieces = board.pieces);

board.insertNewLine();

AssertTrue(oldPieces <> board.pieces);

);

public TestWillFinish : () ==> ()

TestWillFinish() ==

(

dcl board : Board := new Board();

AssertTrue(board.willFinish() = false);

board.insertNewLine();

board.insertNewLine();

board.insertNewLine();

board.insertNewLine();

board.insertNewLine();

board.insertNewLine();

AssertTrue(board.willFinish() = true);

);

public TestSelectPiece : () ==> ()

TestSelectPiece() ==

(

AuxBoard\_H();

AssertTrue(bd\_h.selectPiece(mk\_Board`Position(6,6)) = bd\_h.selectPieceIter(mk\_Board`Position(6,6), {}));

AssertTrue(bd\_h.selectPiece(mk\_Board`Position(6,6)) = {mk\_Board`Position(6,6), mk\_Board`Position(6,7), mk\_Board`Position(6,8)});

);

public TestSelectPieceIter : () ==> ()

TestSelectPieceIter() ==

(

AuxBoard\_H();

AssertTrue(bd\_h.selectPieceIter(mk\_Board`Position(6,6), {}) = {mk\_Board`Position(6,6), mk\_Board`Position(6,7), mk\_Board`Position(6,8)});

AssertTrue(mk\_Board`Position(6,6) in set bd\_h.selectPieceIter(mk\_Board`Position(6,6), {}));

);

public TestGetNeighbours : () ==> ()

TestGetNeighbours() ==

(

AuxBoard\_H();

AssertTrue(bd\_h.getNeighbours(mk\_Board`Position(6,6)) = {mk\_Board`Position(6,6)} union {mk\_Board`Position(6,7)});

AssertTrue(mk\_Board`Position(6,6) in set bd\_h.getNeighbours(mk\_Board`Position(6,6)));

AssertTrue(card bd\_h.getNeighbours(mk\_Board`Position(6,6)) = 2);

);

public TestDeletePieces : () ==> ()

TestDeletePieces() ==

(

AuxBoard\_H();

AssertTrue(bd\_h.deletePieces(mk\_Board`Position(6,6))=3);

AssertTrue(bd\_h.pieces(mk\_Board`Position(6,6)).color = <empty>);

AuxBoard\_H();

bd\_h.pieces(mk\_Board`Position(6,8)).clear();

AssertTrue(bd\_h.deletePieces(mk\_Board`Position(6,9)) = 0);

AssertTrue(bd\_h.deletePieces(mk\_Board`Position(6,6)) = 2);

bd\_h.pieces := bd\_h.pieces ++ {mk\_Board`Position(x,5) |-> new Cell(<red>) | x in set {0,...,15}};

bd\_h.pieces := bd\_h.pieces ++ {mk\_Board`Position(x,4) |-> new Cell(<green>) | x in set {0,...,15}};

AssertTrue(bd\_h.deletePieces(mk\_Board`Position(3,5)) = 18);

AssertTrue(bd\_h.pieces(mk\_Board`Position(3,5)).color = <bomb>);

AuxBoard\_H();

AssertTrue(bd\_h.deletePieces(mk\_Board`Position(7,7)) = 0);

);

public TestExplode : () ==> ()

TestExplode() ==

(

AuxBoard\_H();

bd\_h.explode(mk\_Board`Position(7,7));

AssertTrue(bd\_h.pieces(mk\_Board`Position(7,7)).color = <empty>);

AssertTrue(bd\_h.pieces(mk\_Board`Position(7,8)).color = <empty>);

);

public TestFixBoard : () ==> ()

TestFixBoard() ==

(

AuxBoard\_H();

bd\_h.pieces(mk\_Board`Position(6,7)).clear();

bd\_h.fixBoard();

AssertTrue(bd\_h.pieces(mk\_Board`Position(6,7)).color = <red>);

AssertTrue(bd\_h.pieces(mk\_Board`Position(6,8)).color = <empty>);

);

public TestFixBoardVertical : () ==> ()

TestFixBoardVertical() ==

(

AuxBoard\_V();

bd\_v.fixBoardVertical();

AssertTrue(bd\_v.pieces(mk\_Board`Position(5,0)).color <> <empty>); AssertTrue(bd\_v.pieces(mk\_Board`Position(6,0)).color <> <empty>);

);

public TestDistLeft : () ==> ()

TestDistLeft() ==

(

AuxBoard\_V();

AssertTrue(bd\_v.distLeft(6) = 4); AssertTrue(bd\_v.distLeft(8) = 8);

);

public TestDistRight : () ==> ()

TestDistRight() ==

(

AuxBoard\_V();

AssertTrue(bd\_v.distRight(5) = 7);

AssertTrue(bd\_v.distRight(8) = 8);

);

public TestMoveLeft : () ==> ()

TestMoveLeft() ==

(

AuxBoard\_V();

AssertTrue(bd\_v.moveLeft(10) = 72);

);

public TestMoveRight : () ==> ()

TestMoveRight() ==

(

AuxBoard\_V();

AssertTrue(bd\_v.moveRight(5) = 72);

);

end TestBoard

## Classe TestCascade

class TestCascade

operations

public AssertTrue : bool ==> ()

AssertTrue(a) == return

pre a;

public runAllTestsCascade : () ==> ()

runAllTestsCascade() ==

(

TestCalculateScore();

TestPlay();

TestNextLevel();

TestEndGame();

TestIsVictory();

);

public TestCalculateScore : () ==> ()

TestCalculateScore() ==

(

dcl cascade : Cascade := new Cascade();

dcl n1 : nat := 3;

dcl n2 : nat := 5;

AssertTrue(cascade.CalculateScore(n1) = 50);

AssertTrue(cascade.CalculateScore(n2) = 90);

);

public TestPlay : () ==> ()

TestPlay() ==

(

dcl cascade : Cascade := new Cascade();

AssertTrue(cascade.play(mk\_Board`Position(6,6)) = 0);

AssertTrue(cascade.play(mk\_Board`Position(4,1)) = 50);

);

public TestNextLevel : () ==> ()

TestNextLevel() ==

(

dcl cascade : Cascade := new Cascade();

dcl b : bool := false;

AssertTrue(cascade.nextLevel() = true);

for all x in set {1,...,13} do

b := cascade.nextLevel();

AssertTrue(cascade.nextLevel() = false);

);

public TestEndGame : () ==> ()

TestEndGame() ==

(

dcl cascade : Cascade := new Cascade();

cascade.endGame();

AssertTrue(cascade.isPlayable = false);

);

public TestIsVictory : () ==> ()

TestIsVictory() ==

(

dcl cascade : Cascade := new Cascade();

AssertTrue(cascade.isVictory() = false);

cascade.level := 16;

AssertTrue(cascade.isVictory() = true);

)

end TestCascade

7. Matriz de Rastreabilidade

TestCell

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RES  01 | RES  02 | RES  03 | RES  04 | RES  05 | RES  06 | RES  07 | RES  08 | RES  09 | RES  10 | RES  11 | RES  12 | RES  13 |
| TestCreateCell() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestClearCell() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestMakeBomb() |  |  |  |  |  |  |  |  |  |  |  |  |  |

TestBoard

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RES  01 | RES  02 | RES  03 | RES  04 | RES  05 | RES  06 | RES  07 | RES  08 | RES  09 | RES  10 | RES  11 | RES  12 | RES  13 |
| TestCreateBoard () |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestPrint () |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestPrintCell () |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestGenerateColor() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestInsertNewLine() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestWillFinish() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestSelectPiece() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestSelectPieceIter() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestGetNeighbours() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestDeletePieces() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestExplode() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestFixBoard() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestFixBoardVertical() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestDistLeft() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestDistRight() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestMoveLeft() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestMoveRight() |  |  |  |  |  |  |  |  |  |  |  |  |  |

TestCascade

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RES  01 | RES  02 | RES  03 | RES  04 | RES  05 | RES  06 | RES  07 | RES  08 | RES  09 | RES  10 | RES  11 | RES  12 | RES  13 |
| TestCalculateScore() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestPlay () |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestNextLevel () |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestEndGame() |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestIsVictory() |  |  |  |  |  |  |  |  |  |  |  |  |  |

8. Definição completa das classes

9. Cobertura de Testes

### Classe Cell

|  |  |  |
| --- | --- | --- |
| ***name*** | ***#calls*** | ***coverage*** |
| Cell`clear | 1 | 100% |
| Cell`Cell | 2 | 100% |
| Cell`makeBomb | 1 | 100% |
| Cell`Cell | 2 | 100% |
| ***total*** |  | ***100%*** |

### Classe TestCell

|  |  |  |
| --- | --- | --- |
| ***name*** | ***#calls*** | ***coverage*** |
| TestCell`AssertTrue | 4 | 100% |
| TestCell`TestMakeBomb | 1 | 100% |
| TestCell`TestClearCell | 1 | 100% |
| TestCell`TestCreateCell | 1 | 100% |
| TestCell`runAllTestsCell | 1 | 100% |
| ***total*** |  | ***100%*** |

### Classe Board

|  |  |  |
| --- | --- | --- |
| ***name*** | ***#calls*** | ***coverage*** |
| Board`Board | 7 | 100% |
| Board`print | 1 | 100% |
| Board`explode | 2 | 100% |
| Board`distLeft | 3 | 100% |
| Board`fixBoard | 10 | 100% |
| Board`moveLeft | 2 | 100% |
| Board`distRight | 3 | 100% |
| Board`moveRight | 2 | 100% |
| Board`printCell | 213 | 100% |
| Board`willFinish | 9 | 100% |
| Board`selectPiece | 5 | 100% |
| Board`deletePieces | 5 | 100% |
| Board`generateColor | 897 | 100% |
| Board`getNeighbours | 41 | 100% |
| Board`insertNewLine | 7 | 100% |
| Board`selectPieceIter | 38 | 100% |
| Board`fixBoardVertical | 2 | 100% |
| ***total*** |  | ***100%*** |

### Classe TestBoard

|  |  |  |
| --- | --- | --- |
| ***name*** | ***#calls*** | ***coverage*** |
| TestBoard`TestPrint | 1 | 100% |
| TestBoard`AssertTrue | 38 | 100% |
| TestBoard`AuxBoard\_H | 8 | 100% |
| TestBoard`AuxBoard\_V | 5 | 100% |
| TestBoard`TestExplode | 1 | 100% |
| TestBoard`TestDistLeft | 1 | 100% |
| TestBoard`TestFixBoard | 1 | 100% |
| TestBoard`TestMoveLeft | 1 | 100% |
| TestBoard`TestDistRight | 1 | 100% |
| TestBoard`TestMoveRight | 1 | 100% |
| TestBoard`TestPrintCell | 1 | 100% |
| TestBoard`TestWillFinish | 1 | 100% |
| TestBoard`TestCreateBoard | 1 | 100% |
| TestBoard`TestSelectPiece | 1 | 100% |
| TestBoard`TestDeletePieces | 1 | 100% |
| TestBoard`runAllTestsBoard | 1 | 100% |
| TestBoard`TestGenerateColor | 1 | 100% |
| TestBoard`TestGetNeighbours | 1 | 100% |
| TestBoard`TestInsertNewLine | 1 | 100% |
| TestBoard`TestSelectPieceIter | 1 | 100% |
| TestBoard`TestFixBoardVertical | 1 | 100% |
| ***total*** |  | ***100%*** |

### Classe Cascade

|  |  |  |
| --- | --- | --- |
| ***name*** | ***#calls*** | ***coverage*** |
| Cascade`play | 2 | 100% |
| Cascade`Cascade | 5 | 100% |
| Cascade`endGame | 1 | 100% |
| Cascade`isVictory | 2 | 100% |
| Cascade`nextLevel | 15 | 100% |
| Cascade`CalculateScore | 4 | 100% |
| ***total*** |  | ***100%*** |

### Classe TestCascade

|  |  |  |
| --- | --- | --- |
| ***name*** | ***#calls*** | ***coverage*** |
| TestCascade`TestPlay | 1 | 100% |
| TestCascade`AssertTrue | 9 | 100% |
| TestCascade`TestEndGame | 1 | 100% |
| TestCascade`TestIsVictory | 1 | 100% |
| TestCascade`TestNextLevel | 1 | 100% |
| TestCascade`TestCalculateScore | 1 | 100% |
| TestCascade`runAllTestsCascade | 1 | 100% |
| ***total*** |  | ***100%*** |

10. Consistência do Modelo

Segundo as regras observadas do jogo exemplo Cascade, podemos concluir que todas as regras do jogo estão devidamente implementadas. A lógica do jogo e as respetivas especificações do jogo foram implementadas em VDM++. Foram implementadas varias classes, nas quais 3 são principais, nomeadamente a classe Cell, a classe Board e a classe Cascade.

A classe Cell modela os blocos do tabuleiro, só tem um atributo que é *colo,* tem funções para criação do bloco, esvaziar bloco ou tornar-se um bomba.

A classe Board tem atributo *pieces* e next\_*pieces,* ambas são um mapa de posições para um bloco, representa o tabuleiro do jogo e a linha que se insere automaticamente no jogo em cada determinado intervalo. Tem também muitas funções que são as operações necessários para garantir o correto funcionamento do jogo. Como *generateColor()* e *insertNewLine()* que gera e insere uma linha de blocos com cores sortidas, *selectPiece()* e *deletePieces()* que seleciona um grupo de blocos que é construído por 3 ou mais cores iguais e elimina-los, *fixBoard()* e *fixBoardVertical()* para garantir não existir espaços entre blocos verticalmente, e horizontalmente na linha mais abaixo do tabuleiro.

A classe Cascade tem atributos *board*, *level*, *score*, *timeToNextLine* e *isPlayable* que representa os componentes do jogo, tabuleiro, nível e pontuação do jogo, controlo de nível e a terminação do jogo. Tem funções play que faça a jogada (elimina os blocos selecionados) e atualiza a pontuação global do jogo, *nextLevel()*, *endGame()* e *isVictory()* para controlar o nível, terminação do jogo.

NewlineThread e NextLevelThread são 2 classes que fazem controlo da inserção de blocos e nível do jogo, precisa manipulação de tempo real do computador, como não conseguimos arranjar nenhuma maneira de realizar o tal em VDM++, foi desenvolvido a parte as funções em Java.

IO e MATH são 2 bibliotecas abordados para permitir a impressão no ecrã e fazer random para a geração de blocos.   
Todas as classes mencionadas tem as regras implementadas para o correto funcionamento do jogo, tem invariantes, pré e pós condições para maior parte de atributos e funções.