



Formal Modeling of a Tetris Game

Mestrado Integrado em Engenharia Informática e
Computação

Métodos Formais em Engenharia de Software

Grupo 1 Turma 4MIEIC02

Ângela Cardoso - up200204375

Tiago Galvão - up201500034

Nuno Valente - up200204376

January 6, 2017

Contents

1	Informal system description and list of requirements	3
1.1	Informal system description	3
1.2	List of requirements	3
2	Visual UML model	4
2.1	Use case model	4
2.2	Class model	6
3	Formal VDM++ model	6
3.1	Class Game	6
3.2	Class Board	8
3.3	Class Tetromino	8
3.3.1	Class TetrominoI	11
3.3.2	Class TetrominoJ	11
3.3.3	Class TetrominoL	12
3.3.4	Class TetrominoO	13
3.3.5	Class TetrominoS	15
3.3.6	Class TetrominoT	16
3.3.7	Class TetrominoZ	17
4	Model validation	18
4.1	Class TestCaseExtra	22
4.2	Class TestTetris	22
5	Model verification	26
5.1	Example of domain verification	26
5.2	Example of invariant verification	27
6	Conclusions	27
7	References	27
A	Source Code	28
B	Indispensable formal rules	28
C	The 7 tetrominoes	29
D	The position of each mino inside respective tetromino	29
E	All possible orientations of each tetromino	30

1 Informal system description and list of requirements

1.1 Informal system description

Tetris game it's a puzzle game and one of the most recognizable and influential video game brands in the world. It's no wonder why there are hundreds of millions of Tetris products being played, worn, and enjoyed by fans in their everyday lives. The game was born in 1984 and it's living proof of a game that have truly transcended the barriers of culture and language.

A meritorious reference to Alexey Pajitnov because he his the person who developed this popular game. He is a russian video game designer and computer engineer and in his spare time, he drew inspiration from his favorite puzzle board game, pentominoes, and decided to create a computer game for himself. Pajitnov envisioned an electronic game that let players arrange puzzle pieces in real time as they fell from the top of the playing field. The resulting design was a game that used seven distinctive geometric playing pieces (appendixC), each made up of four squares. Pajitnov called this game "Tetris," a combination of "tetra" (the Greek word meaning "four") and "tennis" (his favorite sport).

The rules to play the game are very simple. Tetris game requires players to strategically rotate and drop a chaining of tetrominoes that fall into the rectangular board at increasing speeds. Players attempt to clear as many lines as possible by completing horizontal rows of blocks without empty space, but if the tetrominoes surpass the skyline(top of the board) the game is over! Speed and consequent level advance can make the game ally to strategy more enthusiastic. Formal details about other rules are presented in appendixB.

put a game image here

1.2 List of requirements

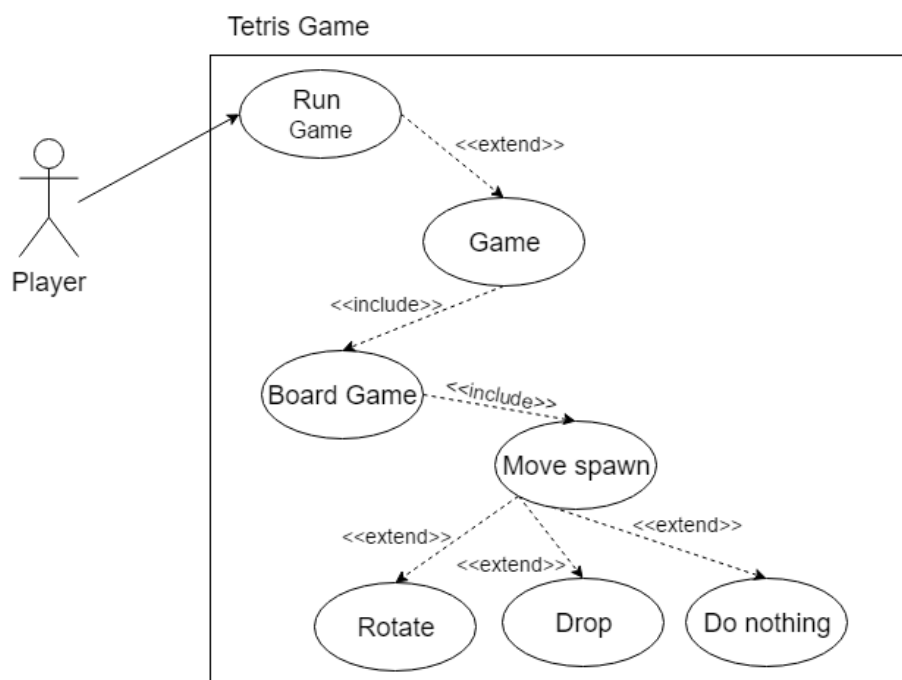
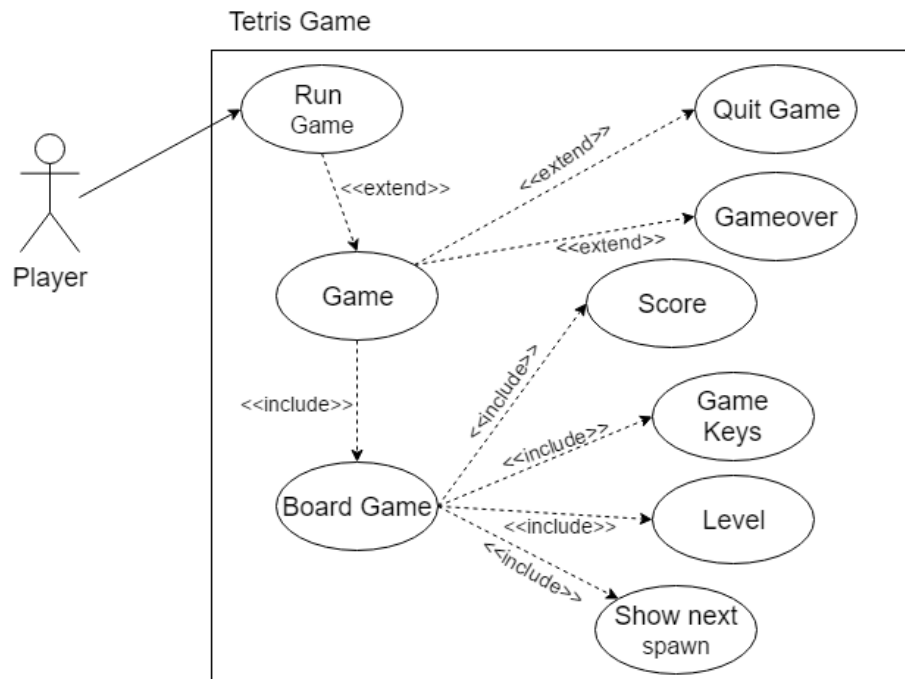
Id	Priority	Description
R1	Mandatory	The player can view his score and level
R2	Mandatory	The game allows two movements - rotation and drop
R3	Opcional	The player should be able to leave the game when he wants
R4	Mandatory	The player has access to a footnote where are the rules to play the game
R5	Mandatory	The spawns appears in a random order to be played
R6	Mandatory	When a row(s) is(are) full of blocks it(they) must be cleared
R7	Mandatory	If tetraminoes surpass the skyline the game is over
R8	Opcional	Time to time the level advance one degree and the game became more difficult
R9	Mandatory	Each tetromino is formed by four squares named minos by us
R10	Mandatory	The player has access to his score and actual level where he is

These requirements are directly translated onto use cases as shown next.

2 Visual UML model

2.1 Use case model

In all use cases we've made an assumption that all keyboard game keys are functioning properly.



The main use cases are described below:

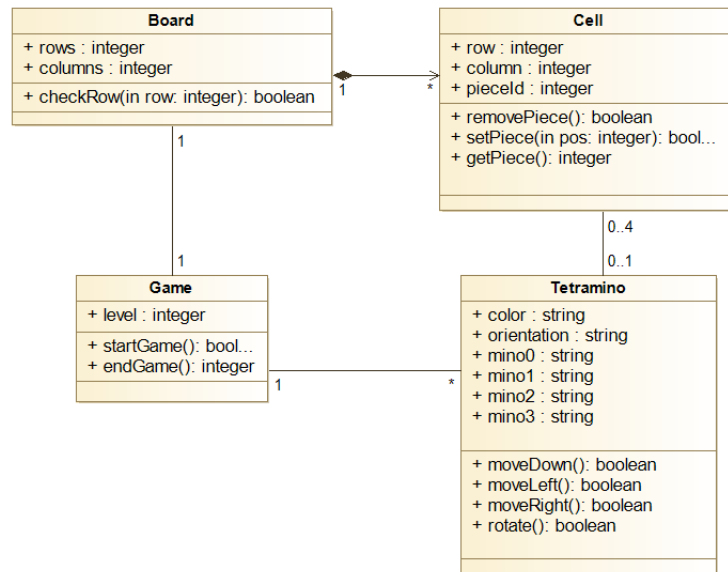
Scenario	View Board Game
Description	The player can see his actual difficulty level, score, keys to play the game and the next spawn
Pre-conditions	The game is running
Post-conditions	The player can view the information updated according to his performance during the game
Steps	(none)
Exceptions	(none)

Scenario	Gameover
Description	The player will finish the game
Pre-conditions	The game is running
Post-conditions	Game will reach the end and close
Steps	<ol style="list-style-type: none"> 1. Play the game pressing the right keys 2. Update score and level 3. The game will eventually reach the end - gameover
Exceptions	(none)

Scenario	Rotation Move
Description	Rotate a tetromino when is falling
Pre-conditions	Tetromino must not be frozen in place
Post-conditions	Tetromino position has changed
Steps	The player must click on a key that allows rotation
Exceptions	When, simultaneously, try to rotate and the spawn reached one final position or the skyline pf the board

Scenario	Drop Move
Description	Accelerates the tetromino falling and position it as it was before reach the final position
Pre-conditions	Tetromino must not be frozen in place
Post-conditions	Tetromino position has changed
Steps	The player must click on a key that allows falling spawn
Exceptions	(none)

2.2 Class model



Class	Description
Game	Core model; defines the state variables and operations available to the players
Board	Defines a game environment for playing and where each piece of tetraminoes can stay
Tetromino	Defines one general piece to play with
TetrominoI/J/L/O/S/T/Z	Defines one specific piece and is a subclass of Tetromino
TestCaseExtra	Superclass for test classes; defines <code>assertEquals</code> and <code>assertTrue</code>
TestTetris	Defines the test/usage scenarios and test cases for the tetris game

Table 1: Description of each class

3 Formal VDM++ model

3.1 Class Game

```
1 class Game
```

```

types

5   public String = seq of char;

instance variables

10  private board : Board;
    private tetramino : Tetramino;
    private gameOver : bool := false;
    private score : nat := 0;
    private lines : nat := 0;
15  private level : nat1 := 1;
    private lineScores : seq of nat := [100, 300, 400,
        800];

operations

20  public Game : () ==> Game
    Game() ==
        board := new Board(); -- startGame

25  public getBoard: () ==> Board
    getBoard() ==
        return board;

    public setGameOver : () ==> ()
30  setGameOver() ==
        gameOver := true;

    public getGameOver : () ==> bool
    getGameOver() ==
35  return gameOver;

    public newTetramino : nat1 ==> ()
    newTetramino(id) == (
        cases id:
40      1 -> tetramino := new TetraminoI(self),
        2 -> tetramino := new TetraminoJ(self),
        3 -> tetramino := new TetraminoL(self),
        4 -> tetramino := new TetraminoO(self),
        5 -> tetramino := new TetraminoS(self),
45      6 -> tetramino := new TetraminoT(self),
        7 -> tetramino := new TetraminoZ(self)
    end;
    );

50  public newRandomTetramino: () ==> ()
    newRandomTetramino() ==
        newTetramino(MATH.rand(7) + 1);

    private incScore : nat ==> ()
55  incScore(inc) ==
        score := score + inc;

    public down : () ==> bool
    down() ==
60  return tetramino.moveDown(board);

```

```

    public left : () ==> bool
    left() ==
        return tetramino.moveLeft(board);
65
    public right : () ==> bool
    right() ==
        return tetramino.moveRight(board);

70
    public rotate : () ==> bool
    rotate() ==
        return tetramino.rotate(board);

    public drop : () ==> nat
75
    drop() == (
        dcl dropDistance: nat := tetramino.drop(board);
        score := score + dropDistance * level;
        return dropDistance;
80
    );

    public checkLines : () ==> nat
    checkLines() == (
        dcl newLines: nat := board.checkRows();
        lines := lines + newLines;
85
        if newLines > 0 then score := score + lineScores(
            newLines) * level;
        if lines > 0 and lines rem 10 = 0 then level :=
            level + 1;
        return newLines
    );

90
    public getScore : () ==> nat
    getScore() == return score;

    public getLines : () ==> nat
    getLines() == return lines;
95

    public getLevel : () ==> nat1
    getLevel() == return level;

    public printBoard : bool * bool * bool ==> String
100
    printBoard(printNow, blackConsole, testPrint) ==
        return board.getBoardPrint(printNow, blackConsole,
            testPrint);

```

end Game

3.2 Class Board

3.3 Class Tetromino

```

1 class Tetramino

    types

5
    public Color = <Cyan> | <Blue> | <Orange>
        | <Yellow> | <Green> | <Purple> | <Red>;
    public Minoes = seq of Board.Position
    inv minoes == len minoes = 4 --and card elems minoes
        = 4

```



```

10      instance variables

      private color : Color := <Cyan>;
      private id   : nat  := 0;
15      private orientation : nat := 0;
      private minoes : Minoes := [[1, 1], [1, 2], [1, 3],
                                   [1, 4]]

      operations

20      public setColor : Color ==> ()
      setColor(c) == color := c;

      public setId : nat ==> ()
25      setId(i) == id := i;

      public getOrientation : () ==> nat
      getOrientation() == return orientation;

30      public setMinoes : Board * Board'Position ==> bool
      setMinoes(board, position) == (
        dcl tempMinoes : Minoes := minoes;
        dcl tempPosition : Board'Position := position;
        removeTetramino(board);
35      for i = 1 to 4 do (
        if (validPosition(board, tempPosition))
        then tempMinoes(i) := tempPosition
        else (
40          addTetramino(board);
          return false
        );
        tempPosition := getNextMino(tempPosition, i);
      );
      minoes := tempMinoes;
45      addTetramino(board);
      return true
    );

      public initialSetMinoes : Game * Board'Position ==>
        ()
50      initialSetMinoes(game, position) == (
        dcl tempPosition : Board'Position := position;
        for i = 1 to 4 do (
          if (validPosition(game.getBoard(), tempPosition))
          then (
55            minoes(i) := tempPosition;
            tempPosition := getNextMino(tempPosition, i)
          )
          else game.setGameOver()
        );
60      if not game.getGameOver() then addTetramino(game.
        getBoard());
    );

      public getNextMino : Board'Position * nat ==> Board'
        Position
      getNextMino(position, index) ==
65      is subclass responsibility;

```

```

    public getRotatedMino : Board'Position ==> Board'
      Position
    getRotatedMino(position) ==
    is subclass responsibility;
70
    public validPosition : Board * Board'Position ==>
      bool
    validPosition(board, position) == (
      if position(1) < 1
      then return false
75      else if position(1) > board.getMaxRow()
      then return false
      else if position(2) < 1
      then return false
      else if position(2) > board.getMaxColumn()
80      then return false
      else if board.getMatrixPosition(position) <> 0
      then return false
      else return true
    );
85
    public removeTetramino : Board ==> ()
    removeTetramino(board) ==
      for mino in minoes do
        board.setMatrixPosition(mino, 0);
90
    public addTetramino : Board ==> ()
    addTetramino(board) ==
      for mino in minoes do
        board.setMatrixPosition(mino, id);
95
    public moveDown : Board ==> bool
    moveDown(board) ==
      return setMinoes(board, [minoes(1)(1) + 1, minoes
        (1)(2)]);
100
    public moveLeft : Board ==> bool
    moveLeft(board) ==
      return setMinoes(board, [minoes(1)(1), minoes(1)(2)
        - 1]);
    public moveRight : Board ==> bool
105
    moveRight(board) ==
      return setMinoes(board, [minoes(1)(1), minoes(1)(2)
        + 1]);
    public rotate : Board ==> bool
    rotate(board) == (
110
      dcl position : Board'Position := getRotatedMino(
        minoes(1));
      orientation := (orientation + 1) mod 4;
      return setMinoes(board, position)
    );
115
    public drop : Board ==> nat
    drop(board) == (
      dcl result : nat := 0;
      while moveDown(board) do
        result := result + 1;
120
      return result

```

```
);
end Tetramino
```

3.3.1 Class TetrominoI

```
1 class TetraminoI is subclass of Tetramino
  operations
    public TetraminoI : Game ==> TetraminoI
5    TetraminoI(game) == (
      Tetramino'setColor(<Cyan>);
      Tetramino'setId(1);
      Tetramino'initialSetMinoes(game, [2, 4]);
      return self
10  );

    public getNextMino: Board'Position * nat ==> Board'
      Position
    getNextMino(position, index) == (
      decl result : Board'Position := position;
15    cases Tetramino'getOrientation():
      0 -> result(2) := position(2) + 1,
      1 -> result(1) := position(1) + 1,
      2 -> result(2) := position(2) - 1,
      3 -> result(1) := position(1) - 1
20    end;
      return result
    );

    public getRotatedMino: Board'Position ==> Board'
      Position
25    getRotatedMino(position) == (
      decl result : Board'Position := position;
      cases Tetramino'getOrientation():
        0 -> (
          result(1) := position(1) - 1;
30          result(2) := position(2) + 2;
        ),
        1 -> (
          result(1) := position(1) + 2;
          result(2) := position(2) + 1;
35          ),
        2 -> (
          result(1) := position(1) + 1;
          result(2) := position(2) - 2;
40          ),
        3 -> (
          result(1) := position(1) - 2;
          result(2) := position(2) - 1;
45          )
      end;
      return result
    );

  end TetraminoI
```

3.3.2 Class TetrominoJ

```
1 class TetraminoJ is subclass of Tetramino
```

```

operations
public TetraminoJ : (Game) ==> TetraminoJ
5 TetraminoJ(game) == (
  Tetramino'setColor(<Blue>);
  Tetramino'setId(2);
  Tetramino'initialSetMinoes(game, [1, 4]);
  return self
10 );

public getNextMino: Board'Position * nat ==> Board'
  Position
getNextMino(position, index) == (
  dcl result : Board'Position := position;
15 cases Tetramino'getOrientation():
  0 -> (
    cases index:
      1 -> result(1) := position(1) + 1,
      others -> result(2) := position(2) + 1
20 end
  ),
  1 -> (
    cases index:
      1 -> result(2) := position(2) - 1,
25 others -> result(1) := position(1) + 1
    end
  ),
  2 -> (
    cases index:
30 1 -> result(1) := position(1) - 1,
    others -> result(2) := position(2) - 1
    end
  ),
  3 -> (
35 cases index:
    1 -> result(2) := position(2) + 1,
    others -> result(1) := position(1) - 1
    end
  )
40 end;
  return result
);

public getRotatedMino: Board'Position ==> Board'
  Position
45 getRotatedMino(position) == (
  dcl result : Board'Position := position;
  cases Tetramino'getOrientation():
    0 -> result(2) := position(2) + 2,
    1 -> result(1) := position(1) + 2,
50 2 -> result(2) := position(2) - 2,
    3 -> result(1) := position(1) - 2
  end;
  return result
);
55 end TetraminoJ

```

3.3.3 Class TetrominoL

```

1 class TetraminoL is subclass of Tetramino

```

```

operations
5  public TetraminoL : (Game) ==> TetraminoL
   TetraminoL(game) == (
     Tetramino'setColor(<Orange>);
     Tetramino'setId(3);
     Tetramino'initialSetMinoes(game, [1, 6]);
10  return self
   );

   public getNextMino: Board'Position * nat ==> Board'
     Position
   getNextMino(position, index) == (
     dcl result : Board'Position := position;
15   cases Tetramino'getOrientation():
     0 -> (
       cases index:
         1 -> result(1) := position(1) + 1,
         others -> result(2) := position(2) - 1
20     end
       ),
     1 -> (
       cases index:
         1 -> result(2) := position(2) - 1,
25     others -> result(1) := position(1) - 1
       end
       ),
     2 -> (
       cases index:
30     1 -> result(1) := position(1) - 1,
       others -> result(2) := position(2) + 1
       end
       ),
     3 -> (
35     cases index:
       1 -> result(2) := position(2) + 1,
       others -> result(1) := position(1) + 1
       end
     )
40   end;
   return result
   );

   public getRotatedMino: Board'Position ==> Board'
     Position
45   getRotatedMino(position) == (
     dcl result : Board'Position := position;
     cases Tetramino'getOrientation():
       0 -> result(1) := position(1) + 2,
       1 -> result(2) := position(2) - 2,
50     2 -> result(1) := position(1) - 2,
       3 -> result(2) := position(2) + 2
     end;
     return result
55   );
end TetraminoL

```

3.3.4 Class TetrominoO

```

1  class TetraminoO is subclass of Tetramino

```

```

operations
5  public Tetramino0 : (Game) ==> Tetramino0
   Tetramino0(game) == (
     Tetramino'setColor(<Yellow>);
     Tetramino'setId(4);
     Tetramino'initialSetMinoes(game, [1, 5]);
10  return self
   );

   public getNextMino: Board'Position * nat ==> Board'
     Position
   getNextMino(position, index) == (
     dcl result : Board'Position := position;
15   cases Tetramino'getOrientation():
     0 -> (
       cases index:
         1 -> result(2) := position(2) + 1,
         2 -> result(1) := position(1) + 1,
20     others -> result(2) := position(2) - 1
       end
     ),
     1 -> (
       cases index:
25     1 -> result(1) := position(1) + 1,
         2 -> result(2) := position(2) - 1,
         others -> result(1) := position(1) - 1
       end
     ),
30     2 -> (
       cases index:
         1 -> result(2) := position(2) - 1,
         2 -> result(1) := position(1) - 1,
         others -> result(2) := position(2) + 1
35     end
     ),
     3 -> (
       cases index:
40     1 -> result(1) := position(1) - 1,
         2 -> result(2) := position(2) + 1,
         others -> result(1) := position(1) + 1
       end
     )
   end;
45  return result
   );

   public getRotatedMino: Board'Position ==> Board'
     Position
   getRotatedMino(position) == (
50   dcl result : Board'Position := position;
     cases Tetramino'getOrientation():
       0 -> result(2) := position(2) + 1,
       1 -> result(1) := position(1) + 1,
       2 -> result(2) := position(2) - 1,
55     3 -> result(1) := position(1) - 1
     end;
     return result
   );

60 end Tetramino0

```

3.3.5 Class TetraminoS

```
1  class TetraminoS is subclass of Tetramino
    operations
    public TetraminoS : (Game) ==> TetraminoS
5   TetraminoS(game) == (
        Tetramino'setColor(<Green>);
        Tetramino'setId(5);
        Tetramino'initialSetMinoes(game, [1, 6]);
        return self
10  );

    public getNextMino: Board'Position * nat ==> Board'
        Position
    getNextMino(position, index) == (
        dcl result : Board'Position := position;
15     cases Tetramino'getOrientation():
        0 -> (
            cases index:
                2 -> result(1) := position(1) + 1,
                others -> result(2) := position(2) - 1
20         end
        ),
        1 -> (
            cases index:
                2 -> result(2) := position(2) - 1,
25         others -> result(1) := position(1) - 1
            end
        ),
        2 -> (
            cases index:
30         2 -> result(1) := position(1) - 1,
            others -> result(2) := position(2) + 1
            end
        ),
        3 -> (
35         cases index:
            2 -> result(2) := position(2) + 1,
            others -> result(1) := position(1) + 1
            end
        )
40     end;
    return result
    );

    public getRotatedMino: Board'Position ==> Board'
        Position
45     getRotatedMino(position) == (
        dcl result : Board'Position := position;
        cases Tetramino'getOrientation():
            0 -> result(1) := position(1) + 2,
            1 -> result(2) := position(2) - 2,
50         2 -> result(1) := position(1) - 2,
            3 -> result(2) := position(2) + 2
        end;
        return result
    );
55 end TetraminoS
```

3.3.6 Class TetrominoT

```
1  class TetraminoT is subclass of Tetramino
    operations
    public TetraminoT : (Game) ==> TetraminoT
5   TetraminoT(game) == (
        Tetramino'setColor(<Purple>);
        Tetramino'setId(6);
        Tetramino'initialSetMinoes(game, [1, 5]);
        return self
10  );

    public getNextMino: Board'Position * nat ==> Board'
        Position
    getNextMino(position, index) == (
        decl result : Board'Position := position;
15   cases Tetramino'getOrientation():
        0 -> (
            cases index:
            1 -> (
                result(1) := position(1) + 1;
20             result(2) := position(2) - 1;
            ),
            others -> result(2) := position(2) + 1
            end
        ),
25     1 -> (
        cases index:
        1 -> (
            result(1) := position(1) - 1;
            result(2) := position(2) - 1;
30         ),
            others -> result(1) := position(1) + 1
            end
        ),
35     2 -> (
        cases index:
        1 -> (
            result(1) := position(1) - 1;
            result(2) := position(2) + 1;
40         ),
            others -> result(2) := position(2) - 1
            end
        ),
45     3 -> (
        cases index:
        1 -> (
            result(1) := position(1) + 1;
            result(2) := position(2) + 1;
50         ),
            others -> result(1) := position(1) - 1
            end
        )
    end;
    return result
55  );

    public getRotatedMino: Board'Position ==> Board'
        Position
```



```

getRotatedMino(position) == (
  dcl result : Board'Position := position;
  cases Tetramino'getOrientation():
60    0 -> (
        result(1) := position(1) + 1;
        result(2) := position(2) + 1;
      ),
    1 -> (
65      result(1) := position(1) + 1;
        result(2) := position(2) - 1;
      ),
    2 -> (
70      result(1) := position(1) - 1;
        result(2) := position(2) - 1;
      ),
    3 -> (
        result(1) := position(1) - 1;
        result(2) := position(2) + 1;
75      )
    end;
    return result
  );
80 end TetraminoT

```

3.3.7 Class TetrominoZ

```

1  class TetraminoZ is subclass of Tetramino
    operations
    public TetraminoZ : (Game) ==> TetraminoZ
5    TetraminoZ(game) == (
        Tetramino'setColor(<Red>);
        Tetramino'setId(7);
        Tetramino'initialSetMinoes(game, [1, 4]);
        return self
10    );

    public getNextMino: Board'Position * nat ==> Board'
      Position
    getNextMino(position, index) == (
      dcl result : Board'Position := position;
15    cases Tetramino'getOrientation():
        0 -> (
            cases index:
              2 -> result(1) := position(1) + 1,
              others -> result(2) := position(2) + 1
20          end
        ),
        1 -> (
            cases index:
              2 -> result(2) := position(2) - 1,
              others -> result(1) := position(1) + 1
25          end
        ),
        2 -> (
            cases index:
30          2 -> result(1) := position(1) - 1,
              others -> result(2) := position(2) - 1
            end
        )
    )

```

```

    ),
    3 -> (
35      cases index:
        2 -> result(2) := position(2) + 1,
        others -> result(1) := position(1) - 1
      end
    )
40  end;
    return result
  );

  public getRotatedMino: Board'Position ==> Board'
    Position
45  getRotatedMino(position) == (
    dcl result : Board'Position := position;
    cases Tetramino'getOrientation():
      0 -> result(2) := position(2) + 2,
      1 -> result(1) := position(1) + 2,
50    2 -> result(2) := position(2) - 2,
      3 -> result(1) := position(1) - 2
    end;
    return result
  );
55 end TetraminoZ

```

4 Model validation

```

1  class TestTetris is subclass of TestCaseExtra

    instance variables
      public printBoard : Game'String := "";
5
    operations
      -----
      initialMatrix_test
      private initialMatrix_test: (Game) ==> ()
      initialMatrix_test(game) == (
10    --print board
      printBoard := game.printBoard(true, true, true);
      );

      -----
      addTetramino_test
15  private addTetramino_test: Game * nat ==> ()
      addTetramino_test(game, id) == (
        game.newTetramino(id);
        --print board
        printBoard := game.printBoard(true, true, true);
20    );

      -----
      dropTetramino_test
      private dropTetramino_test: Game ==> ()
      dropTetramino_test(game) == (
25    dcl value : nat := game.drop();
      if value > 0 then (
        --print board
        printBoard := game.printBoard(true, true, true);
        -- IO'println(value);
      );
    );
  end

```

```

30     IO'println(game.getScore());
      IO'println(game.getLines());
      IO'println(game.getLevel());
    )
  );
35  -----
      downTetramino_test
private downTetramino_test: Game ==> ()
downTetramino_test(game) == (
  dcl result : bool := game.down();
40  --print board
  printBoard := game.printBoard(true, true, true);
  --IO'println(result);
);

45  -----
      rightTetramino_test
private rightTetramino_test: Game ==> ()
rightTetramino_test(game) == (
  dcl result : bool := game.right();
  --print board
50  printBoard := game.printBoard(true, true, true);
  --IO'println(result);
);

  -----
      leftTetramino_test
55 private leftTetramino_test: Game ==> ()
leftTetramino_test(game) == (
  dcl result : bool := game.left();
  --print board
  printBoard := game.printBoard(true, true, true);
60  --IO'println(result);
);

  -----
      rotateTetramino_test
private rotateTetramino_test: Game ==> ()
65 rotateTetramino_test(game) == (
  dcl result : bool := game.rotate();
  --print board
  printBoard := game.printBoard(true, true, true);
  --IO'println(result);
70 );

  -----
      checkLines_test
private checkLines_test: Game ==> ()
checkLines_test(game) == (
75  dcl value : nat := game.checkLines();
  if value > 0 then (
    --print board
    printBoard := game.printBoard(true, true, true);
    --IO'println(value);
80  IO'println(game.getScore());
    IO'println(game.getLines());
    IO'println(game.getLevel());
  )
);

```

```

85 -----
    addRandomTetramino_test
private addRandomTetramino_test: Game ==> ()
addRandomTetramino_test(game) == (
    game.newRandomTetramino();
90    --print board
    printBoard := game.printBoard(true, true, true);
);

-----
    MAIN TESTS
95 public static main: () ==> ()
    main() == (

        dcl game: Game := new Game();
        dcl testInt : nat := 0;
100    IO'print("\n#### TESTS ####\n");

        new TestTetris().initMatrix_test(game);
        new TestTetris().addTetramino_test(game, 1);
105    new TestTetris().downTetramino_test(game);
        new TestTetris().downTetramino_test(game);
        new TestTetris().rotateTetramino_test(game);
        new TestTetris().rotateTetramino_test(game);
        new TestTetris().rotateTetramino_test(game);
110    new TestTetris().rotateTetramino_test(game);
        new TestTetris().dropTetramino_test(game);
        -- new TestTetris().checkLines_test(game);
        new TestTetris().addTetramino_test(game, 2);
        new TestTetris().downTetramino_test(game);
115    new TestTetris().downTetramino_test(game);
        new TestTetris().rotateTetramino_test(game);
        new TestTetris().rotateTetramino_test(game);
        new TestTetris().rotateTetramino_test(game);
        new TestTetris().rotateTetramino_test(game);
120    new TestTetris().leftTetramino_test(game);
        new TestTetris().leftTetramino_test(game);
        new TestTetris().leftTetramino_test(game);
        new TestTetris().dropTetramino_test(game);
        -- new TestTetris().checkLines_test(game);
125    new TestTetris().addTetramino_test(game, 3);
        new TestTetris().downTetramino_test(game);
        new TestTetris().downTetramino_test(game);
        new TestTetris().rotateTetramino_test(game);
        new TestTetris().rotateTetramino_test(game);
130    new TestTetris().rotateTetramino_test(game);
        new TestTetris().rotateTetramino_test(game);
        new TestTetris().rightTetramino_test(game);
        new TestTetris().rightTetramino_test(game);
        new TestTetris().rightTetramino_test(game);
135    new TestTetris().rightTetramino_test(game);
        new TestTetris().dropTetramino_test(game);
        -- new TestTetris().checkLines_test(game);
        new TestTetris().addTetramino_test(game, 4);
        new TestTetris().downTetramino_test(game);
140    new TestTetris().downTetramino_test(game);
        new TestTetris().rotateTetramino_test(game);

```

```

    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
145    new TestTetris().dropTetramino_test(game);
    -- new TestTetris().checkLines_test(game);
    new TestTetris().addTetramino_test(game, 5);
    new TestTetris().downTetramino_test(game);
    new TestTetris().downTetramino_test(game);
150    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
155    new TestTetris().rightTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
    new TestTetris().dropTetramino_test(game);
    -- new TestTetris().checkLines_test(game);
160    new TestTetris().addTetramino_test(game, 6);
    new TestTetris().downTetramino_test(game);
    new TestTetris().downTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
165    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
170    new TestTetris().dropTetramino_test(game);
    -- new TestTetris().checkLines_test(game);
    new TestTetris().addTetramino_test(game, 7);
    new TestTetris().downTetramino_test(game);
    new TestTetris().downTetramino_test(game);
175    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().leftTetramino_test(game);
180    new TestTetris().leftTetramino_test(game);
    new TestTetris().leftTetramino_test(game);
    new TestTetris().dropTetramino_test(game);
    -- new TestTetris().checkLines_test(game);
    new TestTetris().addTetramino_test(game, 5);
185    new TestTetris().downTetramino_test(game);
    new TestTetris().downTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().leftTetramino_test(game);
    new TestTetris().leftTetramino_test(game);
190    new TestTetris().dropTetramino_test(game);
    -- new TestTetris().checkLines_test(game);
    /* while not game.getGameOver() do (
    new TestTetris().addRandomTetramino_test(game);
    if testInt = 0 then (
195    new TestTetris().leftTetramino_test(game);
    new TestTetris().leftTetramino_test(game);
    new TestTetris().leftTetramino_test(game)
    )

```

```

200     else if testInt = 2 then (
        new TestTetris().rightTetramino_test(game);
        new TestTetris().rightTetramino_test(game);
        new TestTetris().rightTetramino_test(game);
        new TestTetris().rightTetramino_test(game);
    );
205     new TestTetris().dropTetramino_test(game);
        new TestTetris().checkLines_test(game);
        testInt := (testInt + 1) mod 3;
    )*/
    );
210 end TestTetris

```

4.1 Class TestCaseExtra

```

1  class TestCaseExtra

    operations
    -- Simulates assertion checking by reducing it to pre
    --condition checking.
    5  -- If 'arg' does not hold, a pre-condition violation
        will be signaled.
        protected assertTrue: bool ==> ()
        assertTrue(arg) ==
            return
        pre arg;
10
    -- Simulates assertion checking by reducing it to
    --post-condition checking.
    -- If values are not equal, prints a message in the
    -- console and generates
    -- a post-conditions violation.
    protected assertEquals: ? * ? ==> ()
15  assertEquals(expected, actual) ==
        if expected <> actual then (
            IO'print("Actual value (");
            IO'print(actual);
            IO'print(") different from expected (");
20            IO'print(expected);
            IO'println(")\n")
        )
        post expected = actual
25 end TestCaseExtra

```

4.2 Class TestTetris

```

1  class TestTetris is subclass of TestCaseExtra

    instance variables
    public printBoard : Game'String := "";
    5
    operations
    -----
        initialMatrix_test
    private initialMatrix_test: (Game) ==> ()
    initialMatrix_test(game) ==(
10    --print board
        printBoard := game.printBoard(true, true, true);
    );

```

```

-----
      addTetramino_test
15  private addTetramino_test: Game * nat ==> ()
      addTetramino_test(game, id) == (
        game.newTetramino(id);
        --print board
        printBoard := game.printBoard(true, true, true);
20  );

-----
      dropTetramino_test
      private dropTetramino_test: Game ==> ()
      dropTetramino_test(game) == (
25    dcl value : nat := game.drop();
        if value > 0 then (
          --print board
          printBoard := game.printBoard(true, true, true);
          -- IO'println(value);
30    IO'println(game.getScore());
          IO'println(game.getLines());
          IO'println(game.getLevel());
        )
      );
35

-----
      downTetramino_test
      private downTetramino_test: Game ==> ()
      downTetramino_test(game) == (
        dcl result : bool := game.down();
40    --print board
        printBoard := game.printBoard(true, true, true);
        --IO'println(result);
      );
45

-----
      rightTetramino_test
      private rightTetramino_test: Game ==> ()
      rightTetramino_test(game) == (
        dcl result : bool := game.right();
        --print board
50    printBoard := game.printBoard(true, true, true);
        --IO'println(result);
      );

-----
      leftTetramino_test
55  private leftTetramino_test: Game ==> ()
      leftTetramino_test(game) == (
        dcl result : bool := game.left();
        --print board
        printBoard := game.printBoard(true, true, true);
60    --IO'println(result);
      );

-----
      rotateTetramino_test
      private rotateTetramino_test: Game ==> ()
65  rotateTetramino_test(game) == (
        dcl result : bool := game.rotate();

```

```

--print board
printBoard := game.printBoard(true, true, true);
--IO'println(result);
70 );

-----
      checkLines_test
private checkLines_test: Game ==> ()
checkLines_test(game) == (
75   dcl value : nat := game.checkLines();
      if value > 0 then (
        --print board
        printBoard := game.printBoard(true, true, true);
        --IO'println(value);
80       IO'println(game.getScore());
        IO'println(game.getLines());
        IO'println(game.getLevel());
      )
    );
85 -----

      addRandomTetramino_test
private addRandomTetramino_test: Game ==> ()
addRandomTetramino_test(game) == (
90   game.newRandomTetramino();
      --print board
      printBoard := game.printBoard(true, true, true);
    );

-----
      MAIN TESTS
95 public static main: () ==> ()
main() == (

  dcl game: Game := new Game();
  dcl testInt : nat := 0;
100
  IO'print("\n#### TESTS ####\n");

  new TestTetris().initMatrix_test(game);
  new TestTetris().addTetramino_test(game, 1);
105 new TestTetris().downTetramino_test(game);
  new TestTetris().downTetramino_test(game);
  new TestTetris().rotateTetramino_test(game);
  new TestTetris().rotateTetramino_test(game);
  new TestTetris().rotateTetramino_test(game);
110 new TestTetris().rotateTetramino_test(game);
  new TestTetris().dropTetramino_test(game);
  -- new TestTetris().checkLines_test(game);
  new TestTetris().addTetramino_test(game, 2);
  new TestTetris().downTetramino_test(game);
115 new TestTetris().downTetramino_test(game);
  new TestTetris().rotateTetramino_test(game);
  new TestTetris().rotateTetramino_test(game);
  new TestTetris().rotateTetramino_test(game);
  new TestTetris().rotateTetramino_test(game);
120 new TestTetris().leftTetramino_test(game);
  new TestTetris().leftTetramino_test(game);
  new TestTetris().leftTetramino_test(game);

```



```

    new TestTetris().dropTetramino_test(game);
--    new TestTetris().checkLines_test(game);
125    new TestTetris().addTetramino_test(game, 3);
    new TestTetris().downTetramino_test(game);
    new TestTetris().downTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
130    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
135    new TestTetris().rightTetramino_test(game);
    new TestTetris().dropTetramino_test(game);
--    new TestTetris().checkLines_test(game);
    new TestTetris().addTetramino_test(game, 4);
    new TestTetris().downTetramino_test(game);
140    new TestTetris().downTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
145    new TestTetris().dropTetramino_test(game);
--    new TestTetris().checkLines_test(game);
    new TestTetris().addTetramino_test(game, 5);
    new TestTetris().downTetramino_test(game);
    new TestTetris().downTetramino_test(game);
150    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
155    new TestTetris().rightTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
    new TestTetris().dropTetramino_test(game);
--    new TestTetris().checkLines_test(game);
160    new TestTetris().addTetramino_test(game, 6);
    new TestTetris().downTetramino_test(game);
    new TestTetris().downTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
165    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
    new TestTetris().rightTetramino_test(game);
170    new TestTetris().dropTetramino_test(game);
--    new TestTetris().checkLines_test(game);
    new TestTetris().addTetramino_test(game, 7);
    new TestTetris().downTetramino_test(game);
    new TestTetris().downTetramino_test(game);
175    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().leftTetramino_test(game);

```

```

180   new TestTetris().leftTetramino_test(game);
    new TestTetris().leftTetramino_test(game);
    new TestTetris().dropTetramino_test(game);
--   new TestTetris().checkLines_test(game);
    new TestTetris().addTetramino_test(game, 5);
185   new TestTetris().downTetramino_test(game);
    new TestTetris().downTetramino_test(game);
    new TestTetris().rotateTetramino_test(game);
    new TestTetris().leftTetramino_test(game);
    new TestTetris().leftTetramino_test(game);
190   new TestTetris().dropTetramino_test(game);
--   new TestTetris().checkLines_test(game);
    /* while not game.getGameOver() do (
        new TestTetris().addRandomTetramino_test(game);
        if testInt = 0 then (
195           new TestTetris().leftTetramino_test(game);
            new TestTetris().leftTetramino_test(game);
            new TestTetris().leftTetramino_test(game)
        )
        else if testInt = 2 then (
200           new TestTetris().rightTetramino_test(game);
            new TestTetris().rightTetramino_test(game);
            new TestTetris().rightTetramino_test(game);
            new TestTetris().rightTetramino_test(game);
        );
205     new TestTetris().dropTetramino_test(game);
        new TestTetris().checkLines_test(game);
        testInt := (testInt + 1) mod 3;
    )*/
    );
210 end TestTetris

```

5 Model verification

5.1 Example of domain verification

One of the proof obligations generated by Overture is:

No.	PO Name	Type
7	Board'checkRow	legal map application

The code under analysis, with the relevant map application, are in line 7:

```

1   public checkRow : int ==> bool
    checkRow(row) == (
        for column = 1 to maxColumn do
            if (matrix([row, column]) = 0) then return false;
5   for i = row - 1 to 1 by -1 do
        for j = 1 to maxColumn do
            matrix([i + 1, j]) := matrix([i, j]);
        return true
    );

```

The proof obligation of Overture also generated a view with the result

forall row:int & ([i, j] in set (dom matrix))

where we can see that each values of i and j are in the domain of the matrix, and only with that values we can access the matrix.

5.2 Example of invariant verification

Another proof obligation generated by Overture is:

No.	PO Name	Type
1	BoardPosition	type invariant satisfiable

The code under analysis, with the relevant map application, are in line 1:

```
1  public Position = seq of nat
   inv position == len position = 2;
   public Matrix = map Position to nat;
   ...

   exists position:Position & ((len position) = 2)
```

6 Conclusions

The model that was developed by us covers all the requirements included implicitly on the theme project and the list of requirements described in section 1.2. In the final and after model verifications, we all see the game developed in VDM++ like one of the projects more consistent and safer that we have ever developed during the course. In addition it is noticed that all the elements of the group have already had contact in the past with the game and continue feeling enthuse with this version developed by us. Maybe in the future we can all add more features to this game and make it appears near the original version. **This project took approximately 16 hours to develop. example in the vending machine**

7 References

1. <https://en.wikipedia.org/wiki/Tetris>
2. <http://tetris.com/>
3. https://tetris.wiki/Tetris_Guideline
4. <http://overturetool.org/>
5. <http://tetris.com/play-tetris/>

A Source Code

maybe not necessary because of section3

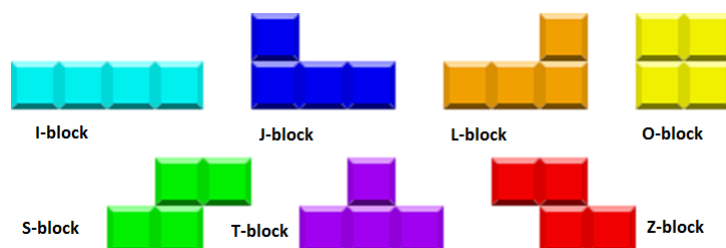
B Indispensable formal rules

In this section we present the formal rules that Tetris game must follow. We already present other rules, named informal and in a player view way in section 1.1.

- Playfield is 10 cells wide and at least 22 cells tall, where rows above 20 are hidden or obstructed by the field frame. We follow in our case 10 cells wide and 22 cells tall where the player can see the first 20 rows and the 2 last cells are invisible to the player.
- The tetromino colors are:
 - Cyan I;
 - Yellow O;
 - Purple T;
 - Green S;
 - Red Z;
 - Blue J;
 - Orange L.
- Each tetromino appear on these exactly locations:
 - The I and O spawn in the middle columns;
 - The rest spawn in the left-middle columns;
 - The tetrominoes spawn horizontally and with their flat side pointed down.
- Super Rotation System (SRS) specifies tetromino rotation.
- Standard mappings for console and handheld gamepads: Up, Down, Left, Right on joystick perform locking hard drop, non-locking soft drop (except first frame locking in some games), left shift, and right shift respectively. Left fire button rotates 90 degrees counterclockwise, and right fire button rotates 90 degrees clockwise.
- Standard mappings different from console/handheld gamepads for computer keyboards
- So-called Random Generator (also called "random bag" or "7 system")
- "Hold piece": The player can press a button to send the falling tetromino to the hold box, and any tetromino that had been in the hold box moves to the top of the screen and begins falling. Hold cannot be used again until after the piece locks down. Games on platforms with fewer than eight usable buttons (such as the version on iPod) may skip this feature. The combination of hold piece and Random Generator would appear to allow the player to play forever.
- Game must have ghost piece function.
- Terms used in the user manual: "Tetriminos" not "tetrominoes" or "tetrads" or "pieces", letter names not "square" or "stick", etc.

- Designated soft drop speed. Details vary between guideline versions.
- Player may only level up by clearing lines or performing T-Spin. Required lines depends in the game.
- The game must use a variant of Roger Dean's Tetris logo, although this was true from around 2000 - before the guidelines emerged.
- Game must include a song called Korobeiniki. (Guideline 2005)
- The player tops out when a piece is spawned overlapping at least one block, or a piece locks completely above the visible portion of the playfield.

C The 7 tetrominoes



D The position of each mino inside respective tetromino



E All possible orientations of each tetromino

