# MFES

January 5, 2017

## Contents



# 1 Board

```
class Board
 types

  -- type representing each area segment of the board
  public Slot = <FREE> | <OCCUPIED> | <VISITED> | <WALL> | <NOWALL>;

 instance variables

  -- the game instance
  private game : Game;
  -- invariant for the board class
  -- the board must always have 289 positions and there must always be 0, 1, 2, 3 or 4 occupied
      places on the board
  -- odd rows and odd columns are for wall positioning whilst the others are for player
      positioning
  inv card(dom board) = 289 and (card dom(board :> {<OCCUPIED>}) <= 4)
    and forall [row, col] in set dom board &
      (if row rem 2 = 0 then board([row, col]) = <NOWALL> or board([row, col]) = <WALL>
        else (if col rem 2 = 0 then board([row, col]) = <NOWALL> or board([row, col]) = <WALL>
          else (board([row, col]) = <FREE> or board([row, col]) = <OCCUPIED>)));
```

```
-- the representation of the board
public static board : map seq of int to Slot :=
  {
   [1,1] |-> <FREE>, [1,2] |-> <NOWALL>, [1,3] |-> <FREE>, [1,4] |-> <NOWALL>, [1,5] |-> <FREE
       >, [1,6] |-> <NOWALL>, [1,7] |-> <FREE>, [1,8] |-> <NOWALL>, [1,9] |-> <FREE>, [1,10]
       |-> <NOWALL>,
   [1,11] |-> <FREE>, [1,12] |-> <NOWALL>, [1,13] |-> <FREE>, [1,14] |-> <NOWALL>, [1,15] |-> <
       FREE>, [1,16] |-> <NOWALL>, [1,17] |-> <FREE>,
   [2,1] |-> <NOWALL>, [2,2] |-> <NOWALL>, [2,3] |-> <NOWALL>, [2,4] |-> <NOWALL>, [2,5] |-> <
       NOWALL>, [2,6] |-> <NOWALL>, [2,7] |-> <NOWALL>, [2,8] |-> <NOWALL>, [2,9] |-> <NOWALL>,
        [2,10] |-> <NOWALL>,
   [2,11] |-> <NOWALL>, [2,12] |-> <NOWALL>, [2,13] |-> <NOWALL>, [2,14] |-> <NOWALL>, [2,15]
       |-> <NOWALL>, [2,16] |-> <NOWALL>, [2,17] |-> <NOWALL>,

   [3,1] |-> <FREE>, [3,2] |-> <NOWALL>, [3,3] |-> <FREE>, [3,4] |-> <NOWALL>, [3,5] |-> <FREE
       >, [3,6] |-> <NOWALL>, [3,7] |-> <FREE>, [3,8] |-> <NOWALL>, [3,9] |-> <FREE>, [3,10]
       |-> <NOWALL>,
   [3,11] |-> <FREE>, [3,12] |-> <NOWALL>, [3,13] |-> <FREE>, [3,14] |-> <NOWALL>, [3,15] |-> <
       FREE>, [3,16] |-> <NOWALL>, [3,17] |-> <FREE>,
   [4,1] |-> <NOWALL>, [4,2] |-> <NOWALL>, [4,3] |-> <NOWALL>, [4,4] |-> <NOWALL>, [4,5] |-> <
       NOWALL>, [4,6] |-> <NOWALL>, [4,7] |-> <NOWALL>, [4,8] |-> <NOWALL>, [4,9] |-> <NOWALL>,
        [4,10] |-> <NOWALL>,
   [4,11] |-> <NOWALL>, [4,12] |-> <NOWALL>, [4,13] |-> <NOWALL>, [4,14] |-> <NOWALL>, [4,15]
       |-> <NOWALL>, [4,16] |-> <NOWALL>, [4,17] |-> <NOWALL>,

   [5,1] |-> <FREE>, [5,2] |-> <NOWALL>, [5,3] |-> <FREE>, [5,4] |-> <NOWALL>, [5,5] |-> <FREE
       >, [5,6] |-> <NOWALL>, [5,7] |-> <FREE>, [5,8] |-> <NOWALL>, [5,9] |-> <FREE>, [5,10]
       |-> <NOWALL>,
   [5,11] |-> <FREE>, [5,12] |-> <NOWALL>, [5,13] |-> <FREE>, [5,14] |-> <NOWALL>, [5,15] |-> <
       FREE>, [5,16] |-> <NOWALL>, [5,17] |-> <FREE>,
   [6,1] |-> <NOWALL>, [6,2] |-> <NOWALL>, [6,3] |-> <NOWALL>, [6,4] |-> <NOWALL>, [6,5] |-> <
       NOWALL>, [6,6] |-> <NOWALL>, [6,7] |-> <NOWALL>, [6,8] |-> <NOWALL>, [6,9] |-> <NOWALL>,
        [6,10] |-> <NOWALL>,
   [6,11] |-> <NOWALL>, [6,12] |-> <NOWALL>, [6,13] |-> <NOWALL>, [6,14] |-> <NOWALL>, [6,15]
       |-> <NOWALL>, [6,16] |-> <NOWALL>, [6,17] |-> <NOWALL>,

   [7,1] |-> <FREE>, [7,2] |-> <NOWALL>, [7,3] |-> <FREE>, [7,4] |-> <NOWALL>, [7,5] |-> <FREE
       >, [7,6] |-> <NOWALL>, [7,7] |-> <FREE>, [7,8] |-> <NOWALL>, [7,9] |-> <FREE>, [7,10]
       |-> <NOWALL>,
   [7,11] |-> <FREE>, [7,12] |-> <NOWALL>, [7,13] |-> <FREE>, [7,14] |-> <NOWALL>, [7,15] |-> <
       FREE>, [7,16] |-> <NOWALL>, [7,17] |-> <FREE>,
   [8,1] |-> <NOWALL>, [8,2] |-> <NOWALL>, [8,3] |-> <NOWALL>, [8,4] |-> <NOWALL>, [8,5] |-> <
       NOWALL>, [8,6] |-> <NOWALL>, [8,7] |-> <NOWALL>, [8,8] |-> <NOWALL>, [8,9] |-> <NOWALL>,
        [8,10] |-> <NOWALL>,
   [8,11] |-> <NOWALL>, [8,12] |-> <NOWALL>, [8,13] |-> <NOWALL>, [8,14] |-> <NOWALL>, [8,15]
       |-> <NOWALL>, [8,16] |-> <NOWALL>, [8,17] |-> <NOWALL>,

   [9,1] |-> <FREE>, [9,2] |-> <NOWALL>, [9,3] |-> <FREE>, [9,4] |-> <NOWALL>, [9,5] |-> <FREE
       >, [9,6] |-> <NOWALL>, [9,7] |-> <FREE>, [9,8] |-> <NOWALL>, [9,9] |-> <FREE>, [9,10]
       |-> <NOWALL>,
   [9,11] |-> <FREE>, [9,12] |-> <NOWALL>, [9,13] |-> <FREE>, [9,14] |-> <NOWALL>, [9,15] |-> <
       FREE>, [9,16] |-> <NOWALL>, [9,17] |-> <FREE>,
   [10,1] |-> <NOWALL>, [10,2] |-> <NOWALL>, [10,3] |-> <NOWALL>, [10,4] |-> <NOWALL>, [10,5]
       |-> <NOWALL>, [10,6] |-> <NOWALL>, [10,7] |-> <NOWALL>, [10,8] |-> <NOWALL>, [10,9] |->
       <NOWALL>, [10,10] |-> <NOWALL>,
   [10,11] |-> <NOWALL>, [10,12] |-> <NOWALL>, [10,13] |-> <NOWALL>, [10,14] |-> <NOWALL>,
       [10,15] |-> <NOWALL>, [10,16] |-> <NOWALL>, [10,17] |-> <NOWALL>,

   [11,1] |-> <FREE>, [11,2] |-> <NOWALL>, [11,3] |-> <FREE>, [11,4] |-> <NOWALL>, [11,5] |-> <
       FREE>, [11,6] |-> <NOWALL>, [11,7] |-> <FREE>, [11,8] |-> <NOWALL>, [11,9] |-> <FREE>,
       [11,10] |-> <NOWALL>,
   [11,11] |-> <FREE>, [11,12] |-> <NOWALL>, [11,13] |-> <FREE>, [11,14] |-> <NOWALL>, [11,15]
       |-> <FREE>, [11,16] |-> <NOWALL>, [11,17] |-> <FREE>,
```

```
    [12,1] |-> <NOWALL>, [12,2] |-> <NOWALL>, [12,3] |-> <NOWALL>, [12,4] |-> <NOWALL>, [12,5]
        |-> <NOWALL>, [12,6] |-> <NOWALL>, [12,7] |-> <NOWALL>, [12,8] |-> <NOWALL>, [12,9] |->
        <NOWALL>, [12,10] |-> <NOWALL>,
    [12,11] |-> <NOWALL>, [12,12] |-> <NOWALL>, [12,13] |-> <NOWALL>, [12,14] |-> <NOWALL>,
        [12,15] |-> <NOWALL>, [12,16] |-> <NOWALL>, [12,17] |-> <NOWALL>,

    [13,1] |-> <FREE>, [13,2] |-> <NOWALL>, [13,3] |-> <FREE>, [13,4] |-> <NOWALL>, [13,5] |-> <
        FREE>, [13,6] |-> <NOWALL>, [13,7] |-> <FREE>, [13,8] |-> <NOWALL>, [13,9] |-> <FREE>,
        [13,10] |-> <NOWALL>,
    [13,11] |-> <FREE>, [13,12] |-> <NOWALL>, [13,13] |-> <FREE>, [13,14] |-> <NOWALL>, [13,15]
        |-> <FREE>, [13,16] |-> <NOWALL>, [13,17] |-> <FREE>,
    [14,1] |-> <NOWALL>, [14,2] |-> <NOWALL>, [14,3] |-> <NOWALL>, [14,4] |-> <NOWALL>, [14,5]
        |-> <NOWALL>, [14,6] |-> <NOWALL>, [14,7] |-> <NOWALL>, [14,8] |-> <NOWALL>, [14,9] |->
        <NOWALL>, [14,10] |-> <NOWALL>,
    [14,11] |-> <NOWALL>, [14,12] |-> <NOWALL>, [14,13] |-> <NOWALL>, [14,14] |-> <NOWALL>,
        [14,15] |-> <NOWALL>, [14,16] |-> <NOWALL>, [14,17] |-> <NOWALL>,

    [15,1] |-> <FREE>, [15,2] |-> <NOWALL>, [15,3] |-> <FREE>, [15,4] |-> <NOWALL>, [15,5] |-> <
        FREE>, [15,6] |-> <NOWALL>, [15,7] |-> <FREE>, [15,8] |-> <NOWALL>, [15,9] |-> <FREE>,
        [15,10] |-> <NOWALL>,
    [15,11] |-> <FREE>, [15,12] |-> <NOWALL>, [15,13] |-> <FREE>, [15,14] |-> <NOWALL>, [15,15]
        |-> <FREE>, [15,16] |-> <NOWALL>, [15,17] |-> <FREE>,
    [16,1] |-> <NOWALL>, [16,2] |-> <NOWALL>, [16,3] |-> <NOWALL>, [16,4] |-> <NOWALL>, [16,5]
        |-> <NOWALL>, [16,6] |-> <NOWALL>, [16,7] |-> <NOWALL>, [16,8] |-> <NOWALL>, [16,9] |->
        <NOWALL>, [16,10] |-> <NOWALL>,
    [16,11] |-> <NOWALL>, [16,12] |-> <NOWALL>, [16,13] |-> <NOWALL>, [16,14] |-> <NOWALL>,
        [16,15] |-> <NOWALL>, [16,16] |-> <NOWALL>, [16,17] |-> <NOWALL>,

    [17,1] |-> <FREE>, [17,2] |-> <NOWALL>, [17,3] |-> <FREE>, [17,4] |-> <NOWALL>, [17,5] |-> <
        FREE>, [17,6] |-> <NOWALL>, [17,7] |-> <FREE>, [17,8] |-> <NOWALL>, [17,9] |-> <FREE>,
        [17,10] |-> <NOWALL>,
    [17,11] |-> <FREE>, [17,12] |-> <NOWALL>, [17,13] |-> <FREE>, [17,14] |-> <NOWALL>, [17,15]
        |-> <FREE>, [17,16] |-> <NOWALL>, [17,17] |-> <FREE>
  };

 -- an auxiliary board to validate logical plays (droping walls)
 public conectivity : map seq of nat1 to Slot;

operations

 -- constructor

 public Board : Game ==> Board
 Board(gameObj) ==
 (
  resetBoard();
  conectivity := board;
  game := gameObj;
 )
 post conectivity = board;


 public resetBoard : () ==> ()
 resetBoard() ==
 (
  board := {
    [1,1] |-> <FREE>, [1,2] |-> <NOWALL>, [1,3] |-> <FREE>, [1,4] |-> <NOWALL>, [1,5] |-> <FREE
        >, [1,6] |-> <NOWALL>, [1,7] |-> <FREE>, [1,8] |-> <NOWALL>, [1,9] |-> <FREE>, [1,10]
        |-> <NOWALL>,
    [1,11] |-> <FREE>, [1,12] |-> <NOWALL>, [1,13] |-> <FREE>, [1,14] |-> <NOWALL>, [1,15] |-> <
        FREE>, [1,16] |-> <NOWALL>, [1,17] |-> <FREE>,
    [2,1] |-> <NOWALL>, [2,2] |-> <NOWALL>, [2,3] |-> <NOWALL>, [2,4] |-> <NOWALL>, [2,5] |-> <
        NOWALL>, [2,6] |-> <NOWALL>, [2,7] |-> <NOWALL>, [2,8] |-> <NOWALL>, [2,9] |-> <NOWALL>,
         [2,10] |-> <NOWALL>,
```

```
[2,11] |-> <NOWALL>, [2,12] |-> <NOWALL>, [2,13] |-> <NOWALL>, [2,14] |-> <NOWALL>, [2,15]
    |-> <NOWALL>, [2,16] |-> <NOWALL>, [2,17] |-> <NOWALL>,

[3,1] |-> <FREE>, [3,2] |-> <NOWALL>, [3,3] |-> <FREE>, [3,4] |-> <NOWALL>, [3,5] |-> <FREE
    >, [3,6] |-> <NOWALL>, [3,7] |-> <FREE>, [3,8] |-> <NOWALL>, [3,9] |-> <FREE>, [3,10]
    |-> <NOWALL>,
[3,11] |-> <FREE>, [3,12] |-> <NOWALL>, [3,13] |-> <FREE>, [3,14] |-> <NOWALL>, [3,15] |-> <
    FREE>, [3,16] |-> <NOWALL>, [3,17] |-> <FREE>,
[4,1] |-> <NOWALL>, [4,2] |-> <NOWALL>, [4,3] |-> <NOWALL>, [4,4] |-> <NOWALL>, [4,5] |-> <
    NOWALL>, [4,6] |-> <NOWALL>, [4,7] |-> <NOWALL>, [4,8] |-> <NOWALL>, [4,9] |-> <NOWALL>,
     [4,10] |-> <NOWALL>,
[4,11] |-> <NOWALL>, [4,12] |-> <NOWALL>, [4,13] |-> <NOWALL>, [4,14] |-> <NOWALL>, [4,15]
    |-> <NOWALL>, [4,16] |-> <NOWALL>, [4,17] |-> <NOWALL>,

[5,1] |-> <FREE>, [5,2] |-> <NOWALL>, [5,3] |-> <FREE>, [5,4] |-> <NOWALL>, [5,5] |-> <FREE
    >, [5,6] |-> <NOWALL>, [5,7] |-> <FREE>, [5,8] |-> <NOWALL>, [5,9] |-> <FREE>, [5,10]
    |-> <NOWALL>,
[5,11] |-> <FREE>, [5,12] |-> <NOWALL>, [5,13] |-> <FREE>, [5,14] |-> <NOWALL>, [5,15] |-> <
    FREE>, [5,16] |-> <NOWALL>, [5,17] |-> <FREE>,
[6,1] |-> <NOWALL>, [6,2] |-> <NOWALL>, [6,3] |-> <NOWALL>, [6,4] |-> <NOWALL>, [6,5] |-> <
    NOWALL>, [6,6] |-> <NOWALL>, [6,7] |-> <NOWALL>, [6,8] |-> <NOWALL>, [6,9] |-> <NOWALL>,
     [6,10] |-> <NOWALL>,
[6,11] |-> <NOWALL>, [6,12] |-> <NOWALL>, [6,13] |-> <NOWALL>, [6,14] |-> <NOWALL>, [6,15]
    |-> <NOWALL>, [6,16] |-> <NOWALL>, [6,17] |-> <NOWALL>,

[7,1] |-> <FREE>, [7,2] |-> <NOWALL>, [7,3] |-> <FREE>, [7,4] |-> <NOWALL>, [7,5] |-> <FREE
    >, [7,6] |-> <NOWALL>, [7,7] |-> <FREE>, [7,8] |-> <NOWALL>, [7,9] |-> <FREE>, [7,10]
    |-> <NOWALL>,
[7,11] |-> <FREE>, [7,12] |-> <NOWALL>, [7,13] |-> <FREE>, [7,14] |-> <NOWALL>, [7,15] |-> <
    FREE>, [7,16] |-> <NOWALL>, [7,17] |-> <FREE>,
[8,1] |-> <NOWALL>, [8,2] |-> <NOWALL>, [8,3] |-> <NOWALL>, [8,4] |-> <NOWALL>, [8,5] |-> <
    NOWALL>, [8,6] |-> <NOWALL>, [8,7] |-> <NOWALL>, [8,8] |-> <NOWALL>, [8,9] |-> <NOWALL>,
     [8,10] |-> <NOWALL>,
[8,11] |-> <NOWALL>, [8,12] |-> <NOWALL>, [8,13] |-> <NOWALL>, [8,14] |-> <NOWALL>, [8,15]
    |-> <NOWALL>, [8,16] |-> <NOWALL>, [8,17] |-> <NOWALL>,

[9,1] |-> <FREE>, [9,2] |-> <NOWALL>, [9,3] |-> <FREE>, [9,4] |-> <NOWALL>, [9,5] |-> <FREE
    >, [9,6] |-> <NOWALL>, [9,7] |-> <FREE>, [9,8] |-> <NOWALL>, [9,9] |-> <FREE>, [9,10]
    |-> <NOWALL>,
[9,11] |-> <FREE>, [9,12] |-> <NOWALL>, [9,13] |-> <FREE>, [9,14] |-> <NOWALL>, [9,15] |-> <
    FREE>, [9,16] |-> <NOWALL>, [9,17] |-> <FREE>,
[10,1] |-> <NOWALL>, [10,2] |-> <NOWALL>, [10,3] |-> <NOWALL>, [10,4] |-> <NOWALL>, [10,5]
    |-> <NOWALL>, [10,6] |-> <NOWALL>, [10,7] |-> <NOWALL>, [10,8] |-> <NOWALL>, [10,9] |->
    <NOWALL>, [10,10] |-> <NOWALL>,
[10,11] |-> <NOWALL>, [10,12] |-> <NOWALL>, [10,13] |-> <NOWALL>, [10,14] |-> <NOWALL>,
    [10,15] |-> <NOWALL>, [10,16] |-> <NOWALL>, [10,17] |-> <NOWALL>,

[11,1] |-> <FREE>, [11,2] |-> <NOWALL>, [11,3] |-> <FREE>, [11,4] |-> <NOWALL>, [11,5] |-> <
    FREE>, [11,6] |-> <NOWALL>, [11,7] |-> <FREE>, [11,8] |-> <NOWALL>, [11,9] |-> <FREE>,
    [11,10] |-> <NOWALL>,
[11,11] |-> <FREE>, [11,12] |-> <NOWALL>, [11,13] |-> <FREE>, [11,14] |-> <NOWALL>, [11,15]
    |-> <FREE>, [11,16] |-> <NOWALL>, [11,17] |-> <FREE>,
[12,1] |-> <NOWALL>, [12,2] |-> <NOWALL>, [12,3] |-> <NOWALL>, [12,4] |-> <NOWALL>, [12,5]
    |-> <NOWALL>, [12,6] |-> <NOWALL>, [12,7] |-> <NOWALL>, [12,8] |-> <NOWALL>, [12,9] |->
    <NOWALL>, [12,10] |-> <NOWALL>,
[12,11] |-> <NOWALL>, [12,12] |-> <NOWALL>, [12,13] |-> <NOWALL>, [12,14] |-> <NOWALL>,
    [12,15] |-> <NOWALL>, [12,16] |-> <NOWALL>, [12,17] |-> <NOWALL>,

[13,1] |-> <FREE>, [13,2] |-> <NOWALL>, [13,3] |-> <FREE>, [13,4] |-> <NOWALL>, [13,5] |-> <
    FREE>, [13,6] |-> <NOWALL>, [13,7] |-> <FREE>, [13,8] |-> <NOWALL>, [13,9] |-> <FREE>,
    [13,10] |-> <NOWALL>,
[13,11] |-> <FREE>, [13,12] |-> <NOWALL>, [13,13] |-> <FREE>, [13,14] |-> <NOWALL>, [13,15]
    |-> <FREE>, [13,16] |-> <NOWALL>, [13,17] |-> <FREE>,
[14,1] |-> <NOWALL>, [14,2] |-> <NOWALL>, [14,3] |-> <NOWALL>, [14,4] |-> <NOWALL>, [14,5]
    |-> <NOWALL>, [14,6] |-> <NOWALL>, [14,7] |-> <NOWALL>, [14,8] |-> <NOWALL>, [14,9] |->
```

```
        <NOWALL>, [14,10] |-> <NOWALL>,
    [14,11] |-> <NOWALL>, [14,12] |-> <NOWALL>, [14,13] |-> <NOWALL>, [14,14] |-> <NOWALL>,
        [14,15] |-> <NOWALL>, [14,16] |-> <NOWALL>, [14,17] |-> <NOWALL>,

    [15,1] |-> <FREE>, [15,2] |-> <NOWALL>, [15,3] |-> <FREE>, [15,4] |-> <NOWALL>, [15,5] |-> <
        FREE>, [15,6] |-> <NOWALL>, [15,7] |-> <FREE>, [15,8] |-> <NOWALL>, [15,9] |-> <FREE>,
        [15,10] |-> <NOWALL>,
    [15,11] |-> <FREE>, [15,12] |-> <NOWALL>, [15,13] |-> <FREE>, [15,14] |-> <NOWALL>, [15,15]
        |-> <FREE>, [15,16] |-> <NOWALL>, [15,17] |-> <FREE>,
    [16,1] |-> <NOWALL>, [16,2] |-> <NOWALL>, [16,3] |-> <NOWALL>, [16,4] |-> <NOWALL>, [16,5]
        |-> <NOWALL>, [16,6] |-> <NOWALL>, [16,7] |-> <NOWALL>, [16,8] |-> <NOWALL>, [16,9] |->
        <NOWALL>, [16,10] |-> <NOWALL>,
    [16,11] |-> <NOWALL>, [16,12] |-> <NOWALL>, [16,13] |-> <NOWALL>, [16,14] |-> <NOWALL>,
        [16,15] |-> <NOWALL>, [16,16] |-> <NOWALL>, [16,17] |-> <NOWALL>,

    [17,1] |-> <FREE>, [17,2] |-> <NOWALL>, [17,3] |-> <FREE>, [17,4] |-> <NOWALL>, [17,5] |-> <
        FREE>, [17,6] |-> <NOWALL>, [17,7] |-> <FREE>, [17,8] |-> <NOWALL>, [17,9] |-> <FREE>,
        [17,10] |-> <NOWALL>,
    [17,11] |-> <FREE>, [17,12] |-> <NOWALL>, [17,13] |-> <FREE>, [17,14] |-> <NOWALL>, [17,15]
        |-> <FREE>, [17,16] |-> <NOWALL>, [17,17] |-> <FREE>
 };
);

-- resets the auxiliar board with the current board status to future path processing

public resetConectivity : () ==> bool
resetConectivity() ==
(
 conectivity := board;
 return true;
)
post conectivity = board;

-- adds a new wall to the board, if possible

public addWall : nat1 * nat1 ==> [bool]
addWall(row, col) ==
(
 dcl oneRow : int := row + 1;
 dcl twoRow : int := row + 2;
 dcl oneCol : int := col + 1;
 dcl twoCol : int := col + 2;
 dcl players : seq of Player := game.getPlayers();
 if row mod 2 = 1
 then
 (
  if col < 18 and row < 16 and board([row,col]) = <NOWALL> and board([oneRow,col]) = <NOWALL>
      and board([twoRow,col]) = <NOWALL>
  then
  (
   board := board ++ {[row,col] |-> <WALL>};
   board := board ++ {[oneRow,col] |-> <WALL>};
   board := board ++ {[twoRow,col] |-> <WALL>};

   if exists p in seq players & (resetConectivity() and not pathToDestination(p, p.getPosition
       ().getX() , p.getPosition().getY()))
   then
   (
    board := board ++ {[row,col] |-> <NOWALL>};
    board := board ++ {[oneRow,col] |-> <NOWALL>};
    board := board ++ {[twoRow,col] |-> <NOWALL>};

    return false;
   )
   else return true;
```

```
   )
   else return false;
  )
  else
  (
   if col < 16 and row < 18 and board([row,col]) = <NOWALL> and board([row,oneCol]) = <NOWALL>
       and board([row,twoCol]) = <NOWALL>
   then
   (
    board := board ++ {[row,col] |-> <WALL>, [row,oneCol] |-> <WALL>, [row,twoCol] |-> <WALL>};

    if exists p in seq players & (resetConectivity() and not pathToDestination(p, p.getPosition
        ().getX() , p.getPosition().getY()))
    then
    (
     board := board ++ {[row,col] |-> <NOWALL>, [row,oneCol] |-> <NOWALL>, [row,twoCol] |-> <
         NOWALL>};

     return false;
    )
    else return true;
   )
   else return false;
  )
 )
 pre row > 0 and row < 18 and col > 0 and col < 18
 post numberOfPlacedWalls(board~) = numberOfPlacedWalls(board) or numberOfPlacedWalls(board~) +
     1 = numberOfPlacedWalls(board);


 -- checks whether the player can get to the target row or not

 public pathToDestination : Player * nat1 * nat1 ==> bool
 pathToDestination(p, row, col) ==
 (
  if p.getTargetRow() <> 0 and p.getTargetRow() = row
   then return true
  else if p.getTargetCol() <> 0 and p.getTargetCol() = col
   then return true;

  if conectivity([row,col]) = <VISITED>
   then return false;

  conectivity := conectivity ++ {[row,col] |-> <VISITED>};

  if row = 1
  then
  (
   if col = 1
    then return checkRight(p, row, col + 1) or checkDown(p, row + 1, col)
   else if col = 17
    then return checkLeft(p, row, col - 1) or checkDown(p, row + 1, col)
   else return checkLeft(p, row, col - 1) or checkRight(p, row, col + 1) or checkDown(p, row +
       1, col)
  );

  if row = 17
  then
  (
   if col = 1
    then return checkRight(p, row, col + 1) or checkUp(p, row - 1, col)
   else if col = 17
    then return checkLeft(p, row, col - 1) or checkUp(p, row - 1, col)
   else return checkLeft(p, row, col - 1) or checkRight(p, row, col + 1) or checkUp(p, row - 1,
       col)
```

```
 );

 if col = 1
 then return checkRight(p, row, col + 1) or checkUp(p, row - 1, col) or checkDown(p, row + 1,
     col)
 else if col = 17
 then return checkLeft(p, row, col - 1) or checkUp(p, row - 1, col) or checkDown(p, row + 1,
     col)
 else return checkLeft(p, row, col - 1) or checkRight(p, row, col + 1) or checkUp(p, row - 1,
     col) or checkDown(p, row + 1, col);
)
pre (p.getTargetRow() <> 0 or p.getTargetCol() <> 0) and row mod 2 = 1 and col mod 2 = 1 and
    row >= 1 and col >= 1 and row < 18 and col < 18;

-- checks path availability on the direction down

private checkDown : Player * nat1 * nat1 ==> bool
checkDown(p, row, col) ==
(
 if conectivity([row,col]) <> <WALL>
  then return pathToDestination(p, row + 1, col)
 else return false;
);

-- checks path availability on the direction up

private checkUp : Player * nat1 * nat1 ==> bool
checkUp(p, row, col) ==
(
 if conectivity([row,col]) <> <WALL>
  then return pathToDestination(p, row - 1, col)
 else return false;
);

-- checks path availability on the direction right

private checkRight : Player * nat1 * nat1 ==> bool
checkRight(p, row, col) ==
(
 if conectivity([row,col]) <> <WALL>
  then return pathToDestination(p, row, col + 1)
 else return false;
);

-- checks path availability on the direction left

private checkLeft : Player * nat1 * nat1 ==> bool
checkLeft(p, row, col) ==
(
 if conectivity([row,col]) <> <WALL>
  then return pathToDestination(p, row, col - 1)
 else return false;
);
-- Sets the board position as occupied

public setBoardPosition: Point ==> ()
setBoardPosition(p) ==
(
 board := board ++ {[p.getX(),p.getY()] |-> <OCCUPIED>};
);

-- Sets the occupied board cell back to free

public unsetBoardPosition: Point ==> ()
unsetBoardPosition(p) ==
```

```
  (
   board := board ++ {[p.getX(),p.getY()]  |-> <FREE>};
  );


functions
  -- determines if a wall can be placed in a certain position

  public dropableWall: map seq of nat1 to Slot * nat1 * nat1 +> [seq of nat1]
  dropableWall(board, row, col) ==
  (
   if (col mod 2) = 0 and (row mod 2 = 1)
   then (
    if (board([row,col]) = <NOWALL> and board([row+1,col]) = <NOWALL> and board([row+2,col]) = <
        NOWALL> and (board([row+1,col-1]) = <NOWALL> or board([row+1,col+1]) = <NOWALL>))
    then [row+2,col]
    else nil
   )
   else
   if (row mod 2) = 0 and (col mod 2) = 1
   then (
    if (board([row,col]) = <NOWALL> and board([row,col+1]) = <NOWALL> and board([row,col+2]) = <
        NOWALL> and (board([row-1,col+1]) = <NOWALL> or board([row+1,col+1]) = <NOWALL>))
    then [row,col+2]
    else nil
   )
   else nil
  )
  pre row > 0 and row < 18 and col > 0 and col < 18;

  -- counts the number of walls in the board

  public numberOfPlacedWalls: map seq of nat1 to Slot +> nat
  numberOfPlacedWalls(board) ==
  card(dom(board :> {<WALL>})) / 3

traces
-- TODO Define Combinatorial Test Traces here
end Board
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| Board | 73 | 100.0% | 240 |
| addWall | 141 | 100.0% | 325 |
| checkDown | 238 | 100.0% | 4479 |
| checkLeft | 265 | 100.0% | 24009 |
| checkRight | 256 | 100.0% | 14721 |
| checkUp | 247 | 100.0% | 4815 |
| dropableWall | 289 | 100.0% | 74 |
| numberOfPlacedWalls | 310 | 100.0% | 628 |
| pathToDestination | 196 | 100.0% | 46560 |
| resetBoard | 82 | 100.0% | 240 |
| resetConectivity | 132 | 100.0% | 579 |
| setBoardPosition | 273 | 100.0% | 662 |
| unsetBoardPosition | 280 | 100.0% | 198 |
| Board.vdmpp | | 100.0% | 97530 |

# 2 Game

```
class Game

instance variables
 -- the players on the game
 private players : seq of Player := [];
 -- the player that is currently making a move
 private currentPlayerID : nat1 := 1;
 -- the board of the game
 private board : [Board];
 -- invariant for game class
 -- the current player's id can never be higher than the number of players in game
 inv (len players) >= currentPlayerID;
 -- it is not possible for two different Players to have the same ID
 inv not exists m, n in seq players & m.getPlayerID() = n.getPlayerID() and m <> n;

operations

  -- game constructor

  public Game : nat1 ==> Game
  Game(number) ==
  (
   if number = 2
   then players := [new Player(1,10), new Player(2,10)]
   else if number = 4
   then players := [new Player(1,5), new Player(2,5), new Player(3,5), new Player(4,5)];
   currentPlayerID := 1;
   board := new Board(self);
   updateBoard();
  );

  -- sets a new position in the board for a given player

  public move: nat1 * nat1 * [Player] ==> ()
  move(new_x, new_y, player) ==
  (
   eraseOldPosition(player.getPosition());
   player.setPosition(new_x, new_y);
   updateBoard();
  )
  pre player <> nil and new_x rem 2 <> 0 and new_y rem 2 <> 0
  post player.getPosition().getX() = new_x and player.getPosition().getY() = new_y;

  -- switches the current player to the next one

  public switchPlayer : () ==> ()
  switchPlayer() ==
  if len players = 2
  then
  (
   if currentPlayerID = 1 then currentPlayerID := 2
   else currentPlayerID := 1;
  )
  else if len players = 4
  then
  (
   if currentPlayerID = 1 then currentPlayerID := 2
   else if currentPlayerID = 2 then currentPlayerID := 3
   else if currentPlayerID = 3 then currentPlayerID := 4
   else currentPlayerID := 1;
  );
```

```
-- returns the player with the specified id

public getPlayer :nat1 ==>Player
getPlayer(id) ==
return [player | player in seq players  &  player.getPlayerID() = id](1);

-- returns the current player's id

public getCurrentPlayer : () ==> nat1
getCurrentPlayer() ==
return currentPlayerID;

-- verify wether the player won

public currentPlayerWin : () ==> bool
currentPlayerWin() ==
(
 if (getPlayer(currentPlayerID).getTargetRow() <> 0 and getPlayer(currentPlayerID).getPosition
     ().getX() = getPlayer(currentPlayerID).getTargetRow())
   or
  (getPlayer(currentPlayerID).getTargetCol() <> 0 and getPlayer(currentPlayerID).getPosition().
      getY() = getPlayer(currentPlayerID).getTargetCol())
   then return true
 else return false;
);

-- adds a player to the players list

public addPlayer: Player ==> ()
 addPlayer(p) ==
 (
  players := players^[p];
 )
 post len players <> 0;

-- retrieve board from game instance

public getBoard : () ==> Board
getBoard() == return self.board;

-- retrieve players from game instance

public getPlayers : () ==> seq of Player
getPlayers() == return self.players;

-- adds a wall to the board on the specified coordinates

public addWall : nat1 * nat1 ==> bool
addWall(row, col) ==
(
 dcl player : Player := getPlayer(getCurrentPlayer());
 if(player.getWalls() > 0 and board.addWall(row,col))
  then return player.decWalls()
  else return false;
)
pre ((row mod 2 = 1) and (col mod 2 = 0)) or ((row mod 2 = 0) and (col mod 2 = 1));

-- returns the possible moves for the current player

public getPossibleMoves: () ==> seq of Point
getPossibleMoves() ==
(
 dcl moves: seq of Point := [],
 special_moves: seq of Point := [],
```

```
p: Player := getPlayer(currentPlayerID),
p_x: nat1 := p.getPosition().getX(),
p_y: nat1 := p.getPosition().getY(),
x: nat1, y: nat1;

-- upper position exists
if(p_x <> 1)
then
(
 if board.board([p_x-1,p_y]) = <NOWALL>
 then
 (
  if board.board([p_x-2,p_y]) = <FREE>
  then
  (
   x := (p_x-2);
   y := p_y;
   moves := moves^[new Point(x, y)];
  )
  else
  (
   -- jump over another player
   special_moves := special_moves^verifyPlayerJump(p_x-2, p_y, "up");
   moves := moves^special_moves;
  );
 );
);
-- left position exists
if(p_y <> 1)
then
(
 if board.board([p_x,p_y-1]) = <NOWALL>
 then
 (
  if board.board([p_x,p_y-2]) = <FREE>
  then
  (
   x := p_x;
   y := (p_y-2);
   moves := moves^[new Point(x, y)];
  )
  else
  (
   -- jump over another player
   special_moves := special_moves^verifyPlayerJump(p_x, p_y-2, "left");
   moves := moves^special_moves;
  );
 );
);
-- right position exists
if(p_y <> 17)
then
(
 if board.board([p_x,p_y+1]) = <NOWALL>
 then
 (
  if board.board([p_x,p_y+2]) = <FREE>
  then
  (
   x := p_x;
   y := (p_y+2);
   moves := moves^[new Point(x, y)];
  )
  else
  (
```

```
      -- jump over another player
      special_moves := special_moves^verifyPlayerJump(p_x, p_y+2, "right");
      moves := moves^special_moves;
     );
    );
   );
   -- bottom position exists
   if(p_x <> 17)
   then
   (
    if board.board([p_x+1,p_y]) = <NOWALL>
    then
    (
     if board.board([p_x+2,p_y]) = <FREE>
     then
     (
      x := (p_x+2);
      y := p_y;
      moves := moves^[new Point(x, y)];
     )
     else
     (
      -- jump over another player
      special_moves := special_moves^verifyPlayerJump(p_x+2, p_y, "down");
      moves := moves^special_moves;
     );
    );
   );

   return moves;
  );

  -- verifies whether the player can jump over another player

  public verifyPlayerJump: nat1 * nat1 * seq of char ==> seq of Point
  verifyPlayerJump(x, y, direction) ==
  (
   if board.board([x,y]) = <OCCUPIED>
   then
   (
    if direction = "up"
    then
    (
     return checkUpMove(x, y);
    )
    else
    (
     if direction = "left"
     then
     (
      return checkLeftMove(x, y);
     )
     else
     (
      if direction = "right"
      then
      (
       return checkRightMove(x, y);
      )
      else
      (
       if direction = "down"
       then
       (
        return checkDownMove(x, y);
```

```
      )
      else
      (
       return [];
      );
     );
    );
   );
  )
  else
  (
   return [];
  );
);

-- check possible up movement

private checkUpMove : nat1 * nat1 ==> seq of Point
checkUpMove(x, y) ==
(

 dcl special: seq of Point := [];

 if(x <> 1)
 then
 (
  if board.board([x-1,y]) = <NOWALL>
  then
  (
   if board.board([x-2,y]) = <FREE>
   then
   (
    special := special^[new Point(x-2, y)];
   )
  )
  -- check special case with oponent + wall followed by each other
  else
  (
   special := special^checkDiagonalHorizontal(x, y);
  );
 );

 return special;

);

-- check possible left movement

private checkLeftMove : nat1 * nat1 ==> seq of Point
checkLeftMove(x, y) ==
(

 dcl special: seq of Point := [];

 if(y <> 1)
 then
 (
  if board.board([x,y-1]) = <NOWALL>
  then
  (
   if board.board([x,y-2]) = <FREE>
   then
   (
    special := special^[new Point(x, y-2)];
   )
```

```
   )
   -- check special case with oponent + wall followed by each other
   else
   (
    special := special^checkDiagonalVertical(x, y);
   );
  );

  return special;

);

-- check possible right movement

private checkRightMove : nat1 * nat1 ==> seq of Point
checkRightMove(x, y) ==
(

 dcl special: seq of Point := [];

 if(y <> 17)
 then
 (
  if board.board([x,y+1]) = <NOWALL>
  then
  (
   if board.board([x,y+2]) = <FREE>
   then
   (
    special := special^[new Point(x, y+2)];
   )
  )
  -- check special case with oponent + wall followed by each other
  else
  (
   special := special^checkDiagonalVertical(x, y);
  );
 );

 return special;

);

-- check possible down movement

private checkDownMove : nat1 * nat1 ==> seq of Point
checkDownMove(x, y) ==
(

 dcl special: seq of Point := [];

 if(x <> 17)
 then
 (
  if board.board([x+1,y]) = <NOWALL>
  then
  (
   if board.board([x+2,y]) = <FREE>
   then
   (
    special := special^[new Point(x+2, y)];
   )
  )
  -- check special case with oponent + wall followed by each other
  else
```

```
   (
    special := special^checkDiagonalHorizontal(x, y);
   );
  );

  return special;

 );

 -- check diagonal movement direction horizontal in special cases

 private checkDiagonalHorizontal : nat1 * nat1 ==> seq of Point
 checkDiagonalHorizontal(x, y) ==
 (

  dcl special: seq of Point := [];

  -- check if the movement to the left is possible
  if y <> 1
  then
  (
   -- check for walls on the left
   if board.board([x,y-1]) = <NOWALL>
   then
   (
    if board.board([x,y-2]) = <FREE>
    then
    (
     special := special^[new Point(x, y-2)];
    )
   );
  );
  -- check if the movement to the right is possible
  if y <> 17
  then
  (
   -- check for walls on the right
   if board.board([x,y+1]) = <NOWALL>
   then
   (
    if board.board([x,y+2]) = <FREE>
    then
    (
     special := special^[new Point(x, y+2)];
    )
   );
  );

  return special;

 );

 -- check diagonal movement direction vertical in special cases

 private checkDiagonalVertical : nat1 * nat1 ==> seq of Point
 checkDiagonalVertical(x, y) ==
 (

  dcl special: seq of Point := [];

  -- check if the movement up is possible
  if x <> 1
  then
  (
   -- check for walls up
```

```
   if board.board([x-1,y]) = <NOWALL>
   then
   (
    if board.board([x-2,y]) = <FREE>
    then
    (
     special := special^[new Point(x-2, y)];
    )
   );
  );
  -- check if the movement down is possible
  if x <> 17
  then
  (
   -- check for walls down
   if board.board([x+1,y]) = <NOWALL>
   then
   (
    if board.board([x+2,y]) = <FREE>
    then
    (
     special := special^[new Point(x+2, y)];
    )
   );
  );

  return special;

 );

 -- update board according to player's positions

 public updateBoard : () ==> ()
 updateBoard() ==
 (
  if(len players = 2)
  then (
   dcl player1: Player := getPlayer(1),
   player2: Player := getPlayer(2),
   p1_position: Point := player1.getPosition(),
   p2_position: Point := player2.getPosition();

   board.setBoardPosition(p1_position);
   board.setBoardPosition(p2_position);
  )
  else if(len players = 4)
  then (
   dcl player1: Player := getPlayer(1),
   player2: Player := getPlayer(2),
   player3: Player := getPlayer(3),
   player4: Player := getPlayer(4),
   p1_position: Point := player1.getPosition(),
   p2_position: Point := player2.getPosition(),
   p3_position: Point := player3.getPosition(),
   p4_position: Point := player4.getPosition();

   board.setBoardPosition(p1_position);
   board.setBoardPosition(p2_position);
   board.setBoardPosition(p3_position);
   board.setBoardPosition(p4_position);
  );
 );

 -- update board by erasing the old player's position
```

```
  public eraseOldPosition : Point ==> ()
  eraseOldPosition(old_position) ==
  (
   board.unsetBoardPosition(old_position);
  );

end Game
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| Game | 19 | 100.0% | 131 |
| addPlayer | 82 | 100.0% | 12 |
| addWall | 98 | 100.0% | 60 |
| checkDiagonalHorizontal | 379 | 100.0% | 16 |
| checkDiagonalVertical | 421 | 100.0% | 16 |
| checkDownMove | 349 | 100.0% | 24 |
| checkLeftMove | 289 | 100.0% | 16 |
| checkRightMove | 319 | 100.0% | 16 |
| checkUpMove | 259 | 100.0% | 24 |
| currentPlayerWin | 71 | 100.0% | 36 |
| eraseOldPosition | 495 | 100.0% | 134 |
| getBoard | 90 | 100.0% | 36 |
| getCurrentPlayer | 66 | 100.0% | 144 |
| getPlayer | 61 | 100.0% | 1204 |
| getPlayers | 94 | 100.0% | 316 |
| getPossibleMoves | 109 | 100.0% | 80 |
| move | 32 | 100.0% | 186 |
| switchPlayer | 43 | 100.0% | 108 |
| updateBoard | 463 | 100.0% | 198 |
| verifyPlayerJump | 212 | 100.0% | 84 |
| Game.vdmpp | | 100.0% | 2841 |

# 3 Player

```
class Player
instance variables

   -- identifier of the player
   private numberID : [nat1];
   -- coordinates of the player's current location
   private position: [Point];
   -- remaining walls that the player can still placed in the map
   private walls : [nat];
   -- row that the player aims to reach in order to complete the objective
   private targetRow : nat := 0;
   -- column that the player aims to reach in order to complete the objective (for 4 players)
   private targetCol : nat := 0;
   -- invariant for the class
   -- target rows and columns can never be bigger than the board
   inv targetRow <= 17 and targetCol <= 17;
   -- the player position coordinates must always be odd in row and col
   inv (position.getX() mod 2 = 1) and (position.getY() mod 2 = 1);
```

```
operations

   -- constructor

   public Player: nat * nat ==> Player
   Player(id, numWalls) ==
   (
    if id = 1
    then (
     position := new Point(1,9);
     targetRow := 17;
     numberID := id;
     walls := numWalls;
    )
    else if id = 2
    then (
     position := new Point(17,9);
     targetRow := 1;
     numberID := id;
     walls := numWalls;
    )
    else if id = 3
    then (
     position := new Point(9,1);
     targetCol := 17;
     numberID := id;
     walls := numWalls;
    )
    else if id = 4
    then (
     position := new Point(9,17);
     targetCol := 1;
     numberID := id;
     walls := numWalls;
    );
   )
   pre id > 0 and id < 5
   post position <> nil and (targetRow <> 0 or targetCol <> 0) and numberID <> nil and (walls =
       10 or walls = 5);

   -- getting the current position of the player

   public pure getPosition : () ==> Point
   getPosition() == return position;

   -- editing the position of the player

   public setPosition: nat1 * nat1 ==> ()
   setPosition(new_x, new_y) ==
    position := new Point(new_x, new_y)
   post position <> nil;

   -- get the player id

   public pure getPlayerID : () ==> nat1
   getPlayerID() ==
    return numberID
   pre numberID <> nil;

   -- set the player targetRow

   public setTargetRow: nat1 ==> ()
   setTargetRow(target) ==
   (
```

```
    targetRow := target;
   )
   post targetRow <> 0;

   -- set the player targetCol

   public setTargetCol: nat1 ==> ()
   setTargetCol(target) ==
   (
    targetCol := target;
   )
   post targetCol <> 0;

   -- retrieves the index of the row the player has to reach

   public pure getTargetRow : () ==> nat
   getTargetRow() == return self.targetRow;

   -- retrieves the index of the column the player has to reach

   public pure getTargetCol : () ==> nat
   getTargetCol() == return self.targetCol;

   -- get the number of walls left for the player

   public getWalls : () ==> nat
   getWalls() == return self.walls;

   -- decrement the value of walls for the player

   public decWalls : () ==> bool
   decWalls() ==
   (
    if(self.walls > 0)
    then
    (
     walls := self.walls - 1;
     return true;
    )
    else return false;
   );

end Player
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| Player | 23 | 100.0% | 348 |
| decWalls | 103 | 100.0% | 235 |
| getPlayerID | 69 | 100.0% | 8471 |
| getPosition | 59 | 100.0% | 2642 |
| getTargetCol | 95 | 100.0% | 46389 |
| getTargetRow | 91 | 100.0% | 140419 |
| getWalls | 99 | 100.0% | 94 |
| setPosition | 63 | 100.0% | 203 |
| setTargetCol | 83 | 100.0% | 28 |
| setTargetRow | 75 | 100.0% | 33 |
| Player.vdmpp | | 100.0% | 198862 |

# 4 Point

```
class Point
instance variables

   -- Coordinates of the point
   private x : [nat1];
   private y : [nat1];

operations

   -- constructor

   public Point: nat1 * nat1 ==> Point
       Point(x_input, y_input) == (x := x_input; y := y_input)
      post x <> nil and y <> nil;

      -- setting new x and y parameters of the class

      public setPoint : nat1 * nat1 ==> ()
      setPoint(x_set, y_set) ==
     (
      x := x_set; y := y_set;
     )
     post x <> nil and y <> nil;

     -- getters of the parameters of the class

      public pure getX : () ==> nat1
      getX() ==
       return x;


      public pure getY : () ==> nat1
      getY() ==
       return y;

end Point
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| Point | 11 | 100.0% | 1165 |
| getX | 24 | 100.0% | 2583 |
| getY | 28 | 100.0% | 2570 |
| setPoint | 16 | 100.0% | 18 |
| Point.vdmpp | | 100.0% | 6336 |

# 5 MyTest

```
class MyTest

operations


 protected assertTrue : bool ==> ()
```

```
  assertTrue(a) == return
  pre a;


 protected assertFalse : bool ==> ()
  assertFalse(a) == return
  pre not a;


 protected assertEqual : ? * ? ==> ()
 assertEqual(e,a) == return
 pre e = a;


 protected assertNotEqual : ? * ? ==> ()
 assertNotEqual(e,a) == return
 pre e <> a;

end MyTest
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| assertEqual | 13 | 100.0% | 235 |
| assertFalse | 9 | 100.0% | 111 |
| assertNotEqual | 17 | 100.0% | 17 |
| assertTrue | 5 | 100.0% | 1127 |
| MyTest.vdmpp | | 100.0% | 1490 |

# 6 TestBoard

```
class TestBoard is subclass of MyTest

instance variables

 private board : [Board] := nil;

operations


 public test : () ==> ()
 test() ==
 (
  testConstructor();
  testResetConectivity();
  testPathToDestination();
  testAddWall();
  testDropWall();
 );


 private testConstructor : () ==> ()
 testConstructor() ==
 (
  board := new Board(new Game(2));
  assertTrue(card(dom board.board) = 289);
  assertEqual(board.board, board.conectivity);
 );
```

```
private testResetConectivity : () ==> ()
testResetConectivity() ==
(
 board := new Board(new Game(2));
 board.conectivity := {|->};
 assertNotEqual(board.board, board.conectivity);
 assertTrue(board.resetConectivity());
 assertEqual(board.board, board.conectivity);
);


private testPathToDestination: () ==> ()
testPathToDestination() ==
(
 dcl
 p: Player := new Player(3, 10);
 board := new Board(new Game(2));
 board.board := board.board ++ {[2,1] |-> <WALL>, [2,3] |-> <WALL>, [2,5] |-> <WALL>, [2,7] |->
     <WALL>, [2,9] |-> <WALL>, [2,11] |-> <WALL>, [2,13] |-> <WALL>, [2,15] |-> <WALL>, [3,16]
     |-> <WALL>, [5,16] |-> <WALL>, [6,17] |-> <NOWALL>};
 assertTrue(board.resetConectivity());
 p.setTargetCol(9);
 assertTrue(board.pathToDestination(p, 17, 9));
 assertTrue(board.resetConectivity());
 p.setTargetRow(1);
 assertTrue(board.pathToDestination(p, 1, 9));
);


private testAddWall : () ==> ()
testAddWall() ==
(
 board := new Board(new Game(2));
 assertTrue(board.addWall(1,6));
 assertTrue(board.board([1,6]) = <WALL> and board.board([2,6]) = <WALL> and board.board([3,6]) =
      <WALL>);
 assertFalse(board.addWall(3,6));
 assertTrue(board.board([3,6]) = <WALL> and board.board([4,6]) = <NOWALL> and board.board([5,6])
      = <NOWALL>);
 assertTrue(board.addWall(2,7));
 assertTrue(board.board([2,7]) = <WALL> and board.board([2,8]) = <WALL> and board.board([2,9]) =
      <WALL>);
 assertFalse(board.addWall(1,10));
 assertTrue(board.board([1,10]) = <NOWALL> and board.board([2,10]) = <NOWALL> and board.board
     ([3,10]) = <NOWALL>);
 assertFalse(board.addWall(2,9));
 assertTrue(board.board([2,9]) = <WALL> and board.board([2,10]) = <NOWALL> and board.board
     ([2,11]) = <NOWALL>);
 assertTrue(board.addWall(15,8));
 assertTrue(board.board([15,8]) = <WALL> and board.board([16,8]) = <WALL> and board.board
     ([17,8]) = <WALL>);
 assertTrue(board.addWall(16,9));
 assertTrue(board.board([16,9]) = <WALL> and board.board([16,10]) = <WALL> and board.board
     ([16,11]) = <WALL>);
 assertTrue(board.addWall(16,12));
 assertTrue(board.board([16,12]) = <WALL> and board.board([16,13]) = <WALL> and board.board
     ([16,14]) = <WALL>);
 assertFalse(board.addWall(16,15));
 assertTrue(board.board([16,15]) = <NOWALL> and board.board([16,16]) = <NOWALL> and board.board
     ([16,17]) = <NOWALL>);

 -- second attempt
 board := new Board(new Game(2));
```

```
 assertTrue(board.addWall(2,1));
 assertEqual(board.numberOfPlacedWalls(board.board),1);
 assertTrue(board.board([2,1]) = <WALL> and board.board([2,3]) = <WALL> and board.board([2,5]) =
      <NOWALL> and board.board([2,6]) = <NOWALL> and board.board([2,7]) = <NOWALL>);
 assertTrue(board.addWall(2,5));
 assertEqual(board.numberOfPlacedWalls(board.board),2);
 assertTrue(board.addWall(2,9));
 assertTrue(board.addWall(2,13));
 assertEqual(board.numberOfPlacedWalls(board.board),4);
 assertTrue(board.addWall(3,16));
 assertFalse(board.addWall(6,15));
 assertTrue(
  board.board([2,1]) = <WALL> and board.board([2,3]) = <WALL> and board.board([2,5]) = <WALL>
      and
  board.board([2,7]) = <WALL> and board.board([2,9]) = <WALL> and board.board([2,11]) = <WALL>
      and
  board.board([2,13]) = <WALL> and board.board([2,15]) = <WALL> and board.board([3,16]) = <WALL>
       and
  board.board([5,16]) = <WALL> and board.board([6,15]) = <NOWALL> and board.board([6,17]) = <
      NOWALL>
 );

);


private testDropWall : () ==> ()
testDropWall() ==
(
 -- verify whether the wall can be put on that position or not
 dcl
 board : Board := new Board(new Game(2)),
 position: [seq of nat1] := board.dropableWall(board.board, 3, 2),
 position2: [seq of nat1] := board.dropableWall(board.board, 2, 3);

 assertEqual(hd position, 5);
 assertEqual(hd tl position, 2);

 assertEqual(hd position2, 2);
 assertEqual(hd tl position2, 5);

 assertTrue(board.addWall(3, 16));

 assertEqual(board.dropableWall(board.board, 2, 15), [2,17]);
 assertEqual(board.dropableWall(board.board, 6, 15), [6,17]);
 assertEqual(board.dropableWall(board.board, 4, 15), nil);

 assertTrue(board.addWall(4, 3));
 assertEqual(board.dropableWall(board.board, 3, 2), [5,2]);
 assertEqual(board.dropableWall(board.board, 3, 6), [5,6]);
 assertEqual(board.dropableWall(board.board, 3, 4), nil);

 assertEqual(board.dropableWall(board.board, 1, 7), nil);
 assertEqual(board.dropableWall(board.board, 2, 2), nil);

 assertEqual(board.dropableWall(board.board, 1, 1), nil);
);

end TestBoard
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|

| | | | |
|---|---|---|---|
| test | 9 | 100.0% | 18 |
| testAddWall | 52 | 100.0% | 17 |
| testConstructor | 19 | 100.0% | 18 |
| testDropWall | 96 | 100.0% | 10 |
| testPathToDestination | 37 | 100.0% | 17 |
| testResetConectivity | 27 | 100.0% | 17 |
| TestBoard.vdmpp | | 100.0% | 97 |

# 7 TestGame

```
class TestGame is subclass of MyTest

instance variables

 private game : Game;

operations


 public test : () ==> ()
 test() ==
 (
  testConstructor();
  testMove();
  testFourPlayers();
 );


 private testConstructor : () ==> ()
 testConstructor() ==
 (

  dcl
  game: Game := new Game(2),
  p2: Player := new Player(3, 10);

  game.switchPlayer();
  assertTrue(game.getCurrentPlayer() = 2);

  game.move(1, 17, game.getPlayer(2));
  assertTrue(game.currentPlayerWin());

  game.switchPlayer();
  assertTrue(game.getCurrentPlayer() = 1);
  assertTrue(game.currentPlayerWin() = false);

  game.addPlayer(p2);
  assertTrue(len game.getPlayers() = 3);

 );


 private testMove : () ==> ()
 testMove() ==
 (
  dcl
  game: Game := new Game(2),
  p1: Player := game.getPlayer(1),
  p2: Player := game.getPlayer(2),
```

```
moves: seq of Point;

game.eraseOldPosition(new Point(1, 17));
assertTrue(game.getBoard().board([1,17]) = <FREE>);

-- normal move case
game.move(1, 1, game.getPlayer(1));
assertTrue(game.getBoard().board([1,1]) = <OCCUPIED> and p1.getPosition().getX() = 1 and p1.
    getPosition().getY() = 1);
game.move(3, 3, game.getPlayer(2));
assertTrue(game.getBoard().board([3,3]) = <OCCUPIED> and p2.getPosition().getX() = 3 and p2.
    getPosition().getY() = 3);

-- special move cases

-- player up + wall up
game.move(5, 7, game.getPlayer(1));
game.move(3, 7, game.getPlayer(2));
moves := game.getPossibleMoves();
assertTrue(len moves = 4);

assertTrue(game.addWall(2, 7));
moves := game.getPossibleMoves();
assertTrue(len moves = 5);

-- player left + wall left
game.move(5, 9, game.getPlayer(1));
game.move(5, 7, game.getPlayer(2));
moves := game.getPossibleMoves();
assertTrue(len moves = 4);

assertTrue(game.addWall(5, 6));
assertFalse(game.addWall(5, 6));
moves := game.getPossibleMoves();
assertTrue(len moves = 5);

-- player right + wall right
game.move(5, 7, game.getPlayer(1));
game.move(5, 9, game.getPlayer(2));
moves := game.getPossibleMoves();
assertTrue(len moves = 3);

assertTrue(game.addWall(5, 10));
moves := game.getPossibleMoves();
assertTrue(len moves = 4);

-- player down + wall down
game.move(3, 9, game.getPlayer(1));
game.move(5, 9, game.getPlayer(2));
moves := game.getPossibleMoves();
assertTrue(len moves = 3);

-- test horizontal diagonal
game.move(5, 15, game.getPlayer(1));
game.move(7, 15, game.getPlayer(2));
moves := game.getPossibleMoves();
assertTrue(len moves = 4);

assertTrue(game.addWall(8, 15));
moves := game.getPossibleMoves();
assertTrue(len moves = 5);

-- test checkUpMove
game.move(7, 17, game.getPlayer(1));
game.move(5, 17, game.getPlayer(2));
```

```
 moves := game.getPossibleMoves();
 assertTrue(len moves = 2);

);


private testFourPlayers : () ==> ()
testFourPlayers() ==
(

 dcl
 game: Game := new Game(4);

 game.switchPlayer();
 assertTrue(game.getCurrentPlayer() = 2);
 game.switchPlayer();
 assertTrue(game.getCurrentPlayer() = 3);
 game.switchPlayer();
 assertTrue(game.getCurrentPlayer() = 4);
 game.switchPlayer();
 assertTrue(game.getCurrentPlayer() = 1);
 game.switchPlayer();
 game.switchPlayer();
 game.switchPlayer();
 assertTrue(game.getCurrentPlayer() = 4);

 game.getPlayer(4).setTargetCol(1);
 assertTrue(game.currentPlayerWin() = false);

 game.move(7, 3, game.getPlayer(1));
 game.move(11, 3, game.getPlayer(3));
 assertEqual(game.verifyPlayerJump(9, 3, "up"), []);
 game.move(9, 3, game.getPlayer(2));
 assertEqual(game.verifyPlayerJump(9, 3, "teste"), []);

);

end TestGame
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| test | 9 | 100.0% | 13 |
| testConstructor | 17 | 100.0% | 13 |
| testFourPlayers | 115 | 100.0% | 13 |
| testMove | 40 | 100.0% | 13 |
| TestGame.vdmpp | | 100.0% | 52 |

# 8 TestMain

```
class TestMain

operations


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

```
  new TestPoint().test();
  new TestPlayer().test();
  new TestBoard().test();
  new TestGame().test();
 );

end TestMain
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| main | 5 | 100.0% | 18 |
| TestMain.vdmpp | | 100.0% | 18 |

# 9 TestPlayer

```
class TestPlayer is subclass of MyTest

operations


 public test : () ==> ()
 test() ==
 (

  dcl p1 : Player := new Player(1,10);
  dcl p2 : Player := new Player(2,10);

  dcl point : Point := new Point(1,9);
  assertTrue(p1.getPosition().getX() = point.getX() and p1.getPosition().getY() = point.getY());
  point := new Point(17,9);
  assertTrue(p2.getPosition().getX() = point.getX() and p2.getPosition().getY() = point.getY());
  p1.setPosition(5,7);
  assertTrue(p1.getPosition().getX() = 5 and p1.getPosition().getY() = 7);
  p1.setTargetRow(17);
  assertTrue(p1.getTargetRow() = 17);
  assertEqual(p1.getPlayerID(), 1);

  assertEqual(p1.getWalls(), 10);
  assertTrue(p1.decWalls());
  assertEqual(p1.getWalls(), 9);
  assertTrue(p1.decWalls());
  assertTrue(p1.decWalls());
  assertTrue(p1.decWalls());
  assertTrue(p1.decWalls());
  assertTrue(p1.decWalls());
  assertTrue(p1.decWalls());
  assertTrue(p1.decWalls());
  assertTrue(p1.decWalls());
  assertTrue(p1.decWalls());
  assertFalse(p1.decWalls());

 );

end TestPlayer
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| test | 5 | 100.0% | 18 |
| TestPlayer.vdmpp | | 100.0% | 18 |

# 10 TestPoint

```
class TestPoint is subclass of MyTest
operations


 public test: () ==> ()
 test() ==
 (
  dcl point : Point := new Point(1,2);
  assertTrue(point.getX() = 1);
  assertTrue(point.getY() = 2);
  point.setPoint(2,1);
  assertTrue(point.getX() = 2 and point.getY() = 1);
 );


end TestPoint
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| test | 4 | 100.0% | 18 |
| TestPoint.vdmpp | | 100.0% | 18 |