Stratego

December 3, 2013

Contents

1 Code

```
class Code

 instance variables
  public str: seq of char := "  ";

 operations
  public Code : Rank*Color ==> Code
  Code(r,c) ==
  (
   dcl rank : char := '.';
   dcl color : char := 'R';

   cases r.name :
     "two" -> rank := '2',
     "three" -> rank := '3',
     "four" -> rank := '4',
     "five" -> rank := '5',
     "six" -> rank := '6',
     "seven" -> rank := '7',
     "eight" -> rank := '8',
     "nine" -> rank := '9',
     "ten" -> rank := 'M',
     "spy" -> rank := 'S',
     "bomb" -> rank := 'B',
     "flag" -> rank := 'F',
     "water" -> rank := 'W',
     "null" -> rank := '-'
    end;

   cases c.name :
     "red" -> color := 'R',
     "blue" -> color := 'B',
     "null" -> color := '-'
    end;

   str := str ++ {1 |-> rank, 2 |-> color};
  );

end Code
```

Function or operation	Coverage	Calls
Code	80.0%	2765
Code.vdmpp	80.3%	2765

2 Color

```
class Color
 instance variables
  public name : seq of char;
 operations
  public Color : seq of char ==> Color
   Color(n) ==
   (
    name := n;
   )
   post name = n;
end Color
```

Function or operation	Coverage	Calls
Color	100.0%	2765
Color.vdmpp	100.0%	2765

3 Game

```
class Game
 types
  public Position:: column : nat row : nat
 instance variables
  --public player1 : Player;
  --public player2 : Player;
  public turn : seq of char;
  public finish : bool;
  public board : map Position to Piece := {|->};

  --       0    1    2
  --     +----+----+---
  --   0 |    |    |
  --     +----+----+---
  --   1 |    |    |

 --Number pieces in the board
 inv card dom board <= 100;

 --Position columns and rows
 inv forall p in set dom board & (p.column <=9 and p.row <=9);

 --Turn must be either "red" or "blue"
 inv turn = "red" or turn = "blue";

 operations
  public Game : () ==> Game
   Game() ==
  (
   --Players
   --player1 := new Player(new Color("red"));
   --player2 := new Player(new Color("blue"));
   turn := "red";
   finish := false;

   --Fill the empty and water positions
   initialize();

   return self
  );

 --Fill the empty and water positions
 public initialize : () ==> ()
  initialize() ==
  (
   dcl c : nat := 0;
   dcl r : nat := 0;

   --Add empty pieces
   while(r < 10) do
   (
    while(c < 10) do
    (
     addNewPiece(mk_Position(c,r), new Piece(new Rank("null"), new Color("null")));
     c := c+1;
    );

    c := 0;
    r := r+1;
   );

   --Add water
   addNewPiece(mk_Position(2,4), new Piece(new Rank("water"), new Color("null")));
   addNewPiece(mk_Position(2,5), new Piece(new Rank("water"), new Color("null")));
   addNewPiece(mk_Position(3,4), new Piece(new Rank("water"), new Color("null")));
   addNewPiece(mk_Position(3,5), new Piece(new Rank("water"), new Color("null")));

   addNewPiece(mk_Position(6,4), new Piece(new Rank("water"), new Color("null")));
   addNewPiece(mk_Position(6,5), new Piece(new Rank("water"), new Color("null")));
   addNewPiece(mk_Position(7,4), new Piece(new Rank("water"), new Color("null")));
   addNewPiece(mk_Position(7,5), new Piece(new Rank("water"), new Color("null")));
  );

 public getBoardSize : () ==> nat
  getBoardSize() ==
  (
   return card dom board;
  );

 --Check if the number of pieces of a certain type hasn't been surpassed
 public checkAvaiability : Piece ==> bool
  checkAvaiability(piece) ==
  (
   dcl setP : set of Piece := {p | p in set rng board &
        (p.color.name = piece.color.name and p.rank.name = piece.rank.name)};

   return card setP < piece.rank.avaiability;
  );

 --Add new Piece
 public addNewPiece : Position*Piece ==> ()
  addNewPiece(position, piece) ==
  (
   if checkAvaiability(piece)
    then board := board ++ {position |-> piece}
  );
  --pre checkAvaiability(piece);
  --post board(position) = piece;

 --Add Piece
 public addPiece : Position*Piece ==> ()
  addPiece(position, piece) ==
  (
   board := board ++ {position |-> piece}
  );

 --Get piece in given position
 public getPiece : Position ==> Piece
  getPiece(pos) ==
  (
   return board(pos);
  );

 --Get piece in given position
 public getClonePiece : Position ==> Piece
  getClonePiece(pos) ==
  (
   return new Piece(new Rank(board(pos).rank.name), new Color(board(pos).color.name));
  );

 --Empty Piece
  public emptyPiece : () ==> Piece
   emptyPiece() ==
   (
    return new Piece(new Rank("null"), new Color("null"));
   );

 --Check if the piece in the position is the same color as the turn
 public checkTurn : Position ==> bool
  checkTurn(p) ==
   (
    if board(p).color.name = turn
     then return true
    else return false
   );

 --Change turn
 public changeTurn : () ==> ()
  changeTurn() ==
   (
    if turn = "red"
     then turn := "blue"
    else turn := "red"
   );

 --Returns the color of the opponent
 public getOpponentColor : Position ==> seq of char
  getOpponentColor(p) ==
   (
    if board(p).color.name = "red"
     then return "blue"
    else return "red"
   );

 --Check if the "to" position color is valid
 public checkToPositionColor : Position*Position ==> bool
  checkToPositionColor(p1,p2) ==
   (
    if board(p2).color.name = getOpponentColor(p1) or
      board(p2).color.name = "null"
     then return true
    else return false
   )
   pre checkTurn(p1);

--Check if the middle pieces are empty
public checkMiddlePieces : Position*Position ==> bool
 checkMiddlePieces(p_from, p_to) ==
 (
  dcl pSet : set of Position;

  if p_to.column > p_from.column and
    p_to.row = p_from.row -- right
    then pSet := {p | p in set dom board &
   (p.column < p_to.column and p.column > p_from.column and p.row = p_from.row)}
  elseif p_to.column < p_from.column and
      p_to.row = p_from.row -- left
    then pSet := {p | p in set dom board &
   (p.column > p_to.column and p.column < p_from.column and p.row = p_from.row)}
  elseif p_to.column = p_from.column and
      p_to.row < p_from.row -- up
    then pSet := {p | p in set dom board &
   (p.row > p_to.row and p.row < p_from.row and p.column = p_from.column)}
  elseif p_to.column = p_from.column and
      p_to.row > p_from.row -- down
    then pSet := {p | p in set dom board &
   (p.row < p_to.row and p.row > p_from.row and p.column = p_from.column)}
  else return false;

  return forall p in set pSet & board(p).rank.name = "null";
 )
 pre checkTurn(p_from);

--Check if the piece can move the number of cells
public checkMovement : Position*Position ==> bool
 checkMovement(p_from, p_to) ==
 (
  dcl pSet : set of Position;
  dcl length : nat;

  if p_to.column > p_from.column and
    p_to.row = p_from.row -- right
    then pSet := {p | p in set dom board &
   (p.column < p_to.column and p.column > p_from.column and p.row = p_from.row)}
  elseif p_to.column < p_from.column and
      p_to.row = p_from.row -- left
    then pSet := {p | p in set dom board &
   (p.column > p_to.column and p.column < p_from.column and p.row = p_from.row)}
  elseif p_to.column = p_from.column and
      p_to.row < p_from.row -- up
    then pSet := {p | p in set dom board &
   (p.row > p_to.row and p.row < p_from.row and p.column = p_from.column)}
  elseif p_to.column = p_from.column and
      p_to.row > p_from.row -- down
    then pSet := {p | p in set dom board &
   (p.row < p_to.row and p.row > p_from.row and p.column = p_from.column)}
  else return false;

  length := card pSet;
  length := length + 1;

  return length <= board(p_from).rank.movement;
 )
```

```
  pre checkTurn(p_from);

 --Validate move
 public validMove : nat*nat*nat*nat ==> bool
  validMove(fc,fr,tc,tr) ==
  (
   dcl p1 : Position := mk_Position(fc,fr);
   dcl p2 : Position := mk_Position(tc,tr);
   dcl positions : set of Position := {p1,p2};
   return positions subset dom board and
    p1 <> p2 and
    checkToPositionColor(p1,p2) and
    checkMiddlePieces(p1,p2) and
    checkMovement(p1,p2);
  )
  pre fc >= 0 and fc <= 9 and fr >= 0 and fr <= 9 and
    tc >= 0 and tc <= 9 and tr >= 0 and tr <= 9;

  --Set of pieces that can move
  public canMove : () ==> set of Piece
   canMove() ==
   (
    return {p | p in set rng board & p.rank.movement > 0}
   );

  --Move Piece
  public move : nat*nat*nat*nat ==> ()
   move(fc,fr,tc,tr) ==
   (
    dcl p1 : Position := mk_Position(fc,fr);
    dcl p2 : Position := mk_Position(tc,tr);

    if finish = false
     then
     (if getPiece(p2).rank.name = "null" or
        (getPiece(p1).rank.name = "spy" and
       getPiece(p2).rank.name = "ten") or
       (getPiece(p1).rank.name = "three" and
       getPiece(p2).rank.name = "bomb")
      then (addPiece(p2, getClonePiece(p1));
         addPiece(p1, emptyPiece());
         changeTurn();)
     elseif getPiece(p2).rank.name = "bomb"
      then (addPiece(p2, emptyPiece());
         addPiece(p1, emptyPiece());
         changeTurn();)
     elseif getPiece(p2).rank.name = "flag"
      then finish := true
     elseif getPiece(p2).rank.number > getPiece(p1).rank.number
      then (addPiece(p1, emptyPiece());
        changeTurn();)
     elseif getPiece(p2).rank.number < getPiece(p1).rank.number
      then (addPiece(p2, getClonePiece(p1));
         addPiece(p1, emptyPiece());
         changeTurn();)
     );
    --else return false;


    --return true;
   )
   pre validMove(fc,fr,tc,tr);

  --Return true if the game ended
  public gameEnded : () ==> bool
```

```
   gameEnded() ==
   (
    return finish;
   );

  --Return true if the game ended
  public getWinner : () ==> seq of char
   getWinner() ==
   (
    if finish = true
     then return turn;

    return "game not finished...";
   );

  --Print board in the console
  public printBoard : () ==> ()
   printBoard() ==
   (
    dcl r : nat := 0;
    dcl c : nat := 0;

    IO`println("  0  1  2  3  4  5  6  7  8  9");
    IO`println(" +----+----+----+----+----+----+----+----+----+----+");

    while(r < 10) do
    (
     IO`print(" "); IO`print(r);
     while(c < 10) do
     (
      IO`print(" | ");
      IO`print(getPiece(mk_Position(c,r)).code.str);
      c := c+1;
     );
     IO`println(" |");
     IO`println(" +----+----+----+----+----+----+----+----+----+----+");

     c := 0;
     r := r+1;
    );
   );

end Game
```

Function or operation	Coverage	Calls
Game	100.0%	25
addNewPiece	100.0%	2749
addPiece	100.0%	13
canMove	0.0%	0
changeTurn	100.0%	8
checkAvaiability	100.0%	2751
checkMiddlePieces	47.6%	12
checkMovement	49.3%	11
checkToPositionColor	90.4%	10
checkTurn	100.0%	34
emptyPiece	100.0%	8
gameEnded	100.0%	1

getBoardSize	100.0%	1
getClonePiece	100.0%	6
getOpponentColor	81.8%	11
getPiece	100.0%	58
getWinner	0.0%	0
initialize	100.0%	25
move	100.0%	8
printBoard	0.0%	0
validMove	100.0%	10
Game.vdmpp	73.3%	5741

4 Piece

```
class Piece

 instance variables
  public rank : Rank;
  public color : Color;
  public code : Code;

 --Colors
 inv color.name = "red" or
   color.name = "blue" or
   color.name = "null";

 --Ranks
 inv rank.name = "two" or
   rank.name = "three" or
   rank.name = "four" or
   rank.name = "five" or
   rank.name = "six" or
   rank.name = "seven" or
   rank.name = "eight" or
   rank.name = "nine" or
   rank.name = "ten" or
   rank.name = "spy" or
   rank.name = "bomb" or
   rank.name = "flag" or
   rank.name = "water" or
   rank.name = "null";

 --Movement
 inv cases rank.name :
  "water", "bomb", "flag", "null" -> rank.movement = 0,
  "spy" -> rank.movement = 9,
  others -> rank.movement = 1
  end;

 --Avaiability
 inv if color.name = "null"
  then
   cases rank.name :
    "water" -> rank.avaiability = 8,
    "null" -> rank.avaiability = 100
   end
  else
   cases rank.name :
    "ten", "nine", "spy", "flag" -> rank.avaiability = 1,
    "eight" -> rank.avaiability = 2,
    "seven" -> rank.avaiability = 3,
    "six", "five", "four" -> rank.avaiability = 4,
    "three" -> rank.avaiability = 5,
    "bomb" -> rank.avaiability = 6,
    "two" -> rank.avaiability = 8,
    "null" -> rank.avaiability = 20
   end;

 --Number
 inv cases rank.name :
    "two" -> rank.number = 2,
    "three" -> rank.number = 3,
    "four" -> rank.number = 4,
    "five" -> rank.number = 5,
    "six" -> rank.number = 6,
    "seven" -> rank.number = 7,
    "eight" -> rank.number = 8,
    "nine" -> rank.number = 9,
    "ten" -> rank.number = 10,
    "spy" -> rank.number = 1,
    "bomb" -> rank.number = 0,
    "flag" -> rank.number = 0,
    "water" -> rank.number = 0,
    "null" -> rank.number = 0
   end;

 operations
  public Piece : Rank*Color ==> Piece
   Piece(r,c) ==
   (
    rank := r;
    color := c;
    code := new Code(r,c);
   )
   post rank = r and color = c;

end Piece
```

Function or operation	Coverage	Calls
Piece	100.0%	2765
Piece.vdmpp	89.2%	2765

5 Play

```
class Play

 instance variables
  game : Game := new Game();
 operations
  public initializeBoard : () ==> ()
   initializeBoard () ==
   (
    --Red pieces
    game.addPiece(mk_Game`Position(0,0), new Piece(new Rank("two"), new Color("red")));
    game.addPiece(mk_Game`Position(1,0), new Piece(new Rank("three"), new Color("red")));
    game.addPiece(mk_Game`Position(2,0), new Piece(new Rank("eight"), new Color("red")));
    game.addPiece(mk_Game`Position(3,0), new Piece(new Rank("two"), new Color("red")));
    game.addPiece(mk_Game`Position(4,0), new Piece(new Rank("three"), new Color("red")));
    game.addPiece(mk_Game`Position(5,0), new Piece(new Rank("two"), new Color("red")));
    game.addPiece(mk_Game`Position(6,0), new Piece(new Rank("four"), new Color("red")));
    game.addPiece(mk_Game`Position(7,0), new Piece(new Rank("two"), new Color("red")));
    game.addPiece(mk_Game`Position(8,0), new Piece(new Rank("ten"), new Color("red")));
    game.addPiece(mk_Game`Position(9,0), new Piece(new Rank("flag"), new Color("red")));

    game.addPiece(mk_Game`Position(0,1), new Piece(new Rank("three"), new Color("red")));
    game.addPiece(mk_Game`Position(1,1), new Piece(new Rank("two"), new Color("red")));
    game.addPiece(mk_Game`Position(2,1), new Piece(new Rank("six"), new Color("red")));
    game.addPiece(mk_Game`Position(3,1), new Piece(new Rank("three"), new Color("red")));
    game.addPiece(mk_Game`Position(4,1), new Piece(new Rank("four"), new Color("red")));
    game.addPiece(mk_Game`Position(5,1), new Piece(new Rank("five"), new Color("red")));
    game.addPiece(mk_Game`Position(6,1), new Piece(new Rank("three"), new Color("red")));
    game.addPiece(mk_Game`Position(7,1), new Piece(new Rank("six"), new Color("red")));
    game.addPiece(mk_Game`Position(8,1), new Piece(new Rank("four"), new Color("red")));
    game.addPiece(mk_Game`Position(9,1), new Piece(new Rank("four"), new Color("red")));

    game.addPiece(mk_Game`Position(0,2), new Piece(new Rank("bomb"), new Color("red")));
    game.addPiece(mk_Game`Position(1,2), new Piece(new Rank("bomb"), new Color("red")));
    game.addPiece(mk_Game`Position(2,2), new Piece(new Rank("spy"), new Color("red")));
    game.addPiece(mk_Game`Position(3,2), new Piece(new Rank("seven"), new Color("red")));
    game.addPiece(mk_Game`Position(4,2), new Piece(new Rank("bomb"), new Color("red")));
    game.addPiece(mk_Game`Position(5,2), new Piece(new Rank("bomb"), new Color("red")));
    game.addPiece(mk_Game`Position(6,2), new Piece(new Rank("seven"), new Color("red")));
    game.addPiece(mk_Game`Position(7,2), new Piece(new Rank("five"), new Color("red")));
    game.addPiece(mk_Game`Position(8,2), new Piece(new Rank("bomb"), new Color("red")));
    game.addPiece(mk_Game`Position(9,2), new Piece(new Rank("bomb"), new Color("red")));

    game.addPiece(mk_Game`Position(0,3), new Piece(new Rank("six"), new Color("red")));
    game.addPiece(mk_Game`Position(1,3), new Piece(new Rank("two"), new Color("red")));
    game.addPiece(mk_Game`Position(2,3), new Piece(new Rank("five"), new Color("red")));
    game.addPiece(mk_Game`Position(3,3), new Piece(new Rank("eight"), new Color("red")));
    game.addPiece(mk_Game`Position(4,3), new Piece(new Rank("two"), new Color("red")));
    game.addPiece(mk_Game`Position(5,3), new Piece(new Rank("two"), new Color("red")));
    game.addPiece(mk_Game`Position(6,3), new Piece(new Rank("nine"), new Color("red")));
    game.addPiece(mk_Game`Position(7,3), new Piece(new Rank("six"), new Color("red")));
    game.addPiece(mk_Game`Position(8,3), new Piece(new Rank("five"), new Color("red")));
    game.addPiece(mk_Game`Position(9,3), new Piece(new Rank("seven"), new Color("red")));


    --Blue Pieces

    game.addPiece(mk_Game`Position(9,9), new Piece(new Rank("two"), new Color("blue")));
    game.addPiece(mk_Game`Position(8,9), new Piece(new Rank("three"), new Color("blue")));
    game.addPiece(mk_Game`Position(7,9), new Piece(new Rank("eight"), new Color("blue")));
    game.addPiece(mk_Game`Position(6,9), new Piece(new Rank("two"), new Color("blue")));
    game.addPiece(mk_Game`Position(5,9), new Piece(new Rank("three"), new Color("blue")));
    game.addPiece(mk_Game`Position(4,9), new Piece(new Rank("two"), new Color("blue")));
    game.addPiece(mk_Game`Position(3,9), new Piece(new Rank("four"), new Color("blue")));
    game.addPiece(mk_Game`Position(2,9), new Piece(new Rank("two"), new Color("blue")));
    game.addPiece(mk_Game`Position(1,9), new Piece(new Rank("ten"), new Color("blue")));
    game.addPiece(mk_Game`Position(0,9), new Piece(new Rank("flag"), new Color("blue")));

    game.addPiece(mk_Game`Position(9,8), new Piece(new Rank("three"), new Color("blue")));
    game.addPiece(mk_Game`Position(8,8), new Piece(new Rank("two"), new Color("blue")));
    game.addPiece(mk_Game`Position(7,8), new Piece(new Rank("six"), new Color("blue")));
    game.addPiece(mk_Game`Position(6,8), new Piece(new Rank("three"), new Color("blue")));
    game.addPiece(mk_Game`Position(5,8), new Piece(new Rank("four"), new Color("blue")));
    game.addPiece(mk_Game`Position(4,8), new Piece(new Rank("five"), new Color("blue")));
    game.addPiece(mk_Game`Position(3,8), new Piece(new Rank("three"), new Color("blue")));
    game.addPiece(mk_Game`Position(2,8), new Piece(new Rank("six"), new Color("blue")));
    game.addPiece(mk_Game`Position(1,8), new Piece(new Rank("four"), new Color("blue")));
    game.addPiece(mk_Game`Position(0,8), new Piece(new Rank("four"), new Color("blue")));
```

```
    game.addPiece(mk_Game`Position(9,7), new Piece(new Rank("bomb"), new Color("blue")));
    game.addPiece(mk_Game`Position(8,7), new Piece(new Rank("bomb"), new Color("blue")));
    game.addPiece(mk_Game`Position(7,7), new Piece(new Rank("spy"), new Color("blue")));
    game.addPiece(mk_Game`Position(6,7), new Piece(new Rank("seven"), new Color("blue")));
    game.addPiece(mk_Game`Position(5,7), new Piece(new Rank("bomb"), new Color("blue")));
    game.addPiece(mk_Game`Position(4,7), new Piece(new Rank("bomb"), new Color("blue")));
    game.addPiece(mk_Game`Position(3,7), new Piece(new Rank("seven"), new Color("blue")));
    game.addPiece(mk_Game`Position(2,7), new Piece(new Rank("five"), new Color("blue")));
    game.addPiece(mk_Game`Position(1,7), new Piece(new Rank("bomb"), new Color("blue")));
    game.addPiece(mk_Game`Position(0,7), new Piece(new Rank("bomb"), new Color("blue")));

    game.addPiece(mk_Game`Position(9,6), new Piece(new Rank("six"), new Color("blue")));
    game.addPiece(mk_Game`Position(8,6), new Piece(new Rank("two"), new Color("blue")));
    game.addPiece(mk_Game`Position(7,6), new Piece(new Rank("five"), new Color("blue")));
    game.addPiece(mk_Game`Position(6,6), new Piece(new Rank("eight"), new Color("blue")));
    game.addPiece(mk_Game`Position(5,6), new Piece(new Rank("two"), new Color("blue")));
    game.addPiece(mk_Game`Position(4,6), new Piece(new Rank("two"), new Color("blue")));
    game.addPiece(mk_Game`Position(3,6), new Piece(new Rank("nine"), new Color("blue")));
    game.addPiece(mk_Game`Position(2,6), new Piece(new Rank("six"), new Color("blue")));
    game.addPiece(mk_Game`Position(1,6), new Piece(new Rank("five"), new Color("blue")));
    game.addPiece(mk_Game`Position(0,6), new Piece(new Rank("seven"), new Color("blue")));
   );

  public play : nat*nat*nat*nat ==> ()
   play(fc,fr,tc,tr) ==
   (
    game.move(fc,fr,tc,tr);
   );

  public printBoard : () ==> ()
   printBoard() ==
   (
    game.printBoard();
   );

end Play
```

Function or operation	Coverage	Calls
initializeBoard	0.0%	0
play	0.0%	0
printBoard	0.0%	0
Play.vdmpp	0.0%	0

6 Player

```
class Player

 instance variables
  public color: Color;

 --Color must be either red or blue
 inv color.name = "red" or
   color.name = "blue";

 operations
```

```
  public Player : Color ==> Player
  Player(c) ==
  (
   color := c;
  )
  post color = c;

end Player
```

Function or operation	Coverage	Calls
Player	0.0%	0
Player.vdmpp	0.0%	0

7 Rank

```
class Rank

 instance variables
  public name : seq of char;
  public avaiability : nat1;
  public movement : nat;
  public number : nat;

 operations
  public Rank : seq of char ==> Rank
   Rank(n) ==
   (
    name := n;

    --Avaiability
    cases name :
     "water" -> avaiability := 8,
     "null" -> avaiability := 100,
     "ten", "nine", "spy", "flag" -> avaiability := 1,
     "eight" -> avaiability := 2,
     "seven" -> avaiability := 3,
     "six", "five", "four" -> avaiability := 4,
     "three" -> avaiability := 5,
     "bomb" -> avaiability := 6,
     "two" -> avaiability := 8
    end;

    --Movement
    cases name :
     "water", "bomb", "flag", "null" -> movement := 0,
     "spy" -> movement := 9,
     others -> movement := 1
    end;

    --Number
    cases name :
     "two" -> number := 2,
     "three" -> number := 3,
     "four" -> number := 4,
     "five" -> number := 5,
     "six" -> number := 6,
     "seven" -> number := 7,
     "eight" -> number := 8,
     "nine" -> number := 9,
     "ten" -> number := 10,
     "spy" -> number := 1,
     "bomb" -> number := 0,
     "flag" -> number := 0,
     "water" -> number := 0,
     "null" -> number := 0
    end;

   )
   post name = n;

end Rank
```

Function or operation	Coverage	Calls
Rank	77.7%	2765
Rank.vdmpp	77.7%	2765

8 Tests

```
class Tests is subclass of TestCase
 operations
  static public assertTrue : bool ==> ()
               assertTrue(op) == return
        pre op;

  public testgetBoardSize : () ==> ()
   testgetBoardSize() ==
   (
    dcl game : Game := new Game();

    assertTrue(game.getBoardSize() = 100);
    IO`println("getBoardSize : passed");
   );

  --Test if a new position for flag is available
  public testcheckAvaiability1 : () ==> ()
   testcheckAvaiability1() ==
   (
    dcl game : Game := new Game();

    game.addNewPiece(mk_Game`Position(0,0), new Piece(new Rank("eight"), new Color("red")));
    game.addNewPiece(mk_Game`Position(0,1), new Piece(new Rank("eight"), new Color("red")));
    assertTrue(game.checkAvaiability(new Piece(new Rank("eight"), new Color("red"))) = false);
    IO`println("checkAvaiability1 : passed");
   );

  --Test if a new position for 'eight' is available
  public testcheckAvaiability2 : () ==> ()
   testcheckAvaiability2() ==
   (
    dcl game : Game := new Game();

    game.addNewPiece(mk_Game`Position(0,0), new Piece(new Rank("eight"), new Color("red")));
    assertTrue(game.checkAvaiability(new Piece(new Rank("eight"), new Color("red"))));
    IO`println("checkAvaiability2 : passed");
   );

  --Test if the new piece was added
  public testaddNewPiece : () ==> ()
   testaddNewPiece() ==
   (
    dcl game : Game := new Game();
    dcl pos : Game`Position := mk_Game`Position(0,0);

    game.addNewPiece(pos, new Piece(new Rank("eight"), new Color("red")));
    assertTrue(game.board(pos).rank.name = "eight" and game.board(pos).color.name = "red");
    IO`println("addNewPiece : passed");
   );

  --Test if the returned piece is the correct one
  public testgetPiece : () ==> ()
   testgetPiece() ==
   (
    dcl game : Game := new Game();
    dcl pos : Game`Position := mk_Game`Position(0,0);

    game.addNewPiece(pos, new Piece(new Rank("eight"), new Color("red")));
    assertTrue(game.getPiece(pos).rank.name = "eight" and game.getPiece(pos).color.name = "red");
    IO`println("getPiece : passed");
   );

  --Test if the clone of the piece is the correct one
  public testgetClonePiece : () ==> ()
   testgetClonePiece() ==
   (
    dcl game : Game := new Game();
    dcl pos : Game`Position := mk_Game`Position(0,0);
    dcl clone : Piece;

    game.addNewPiece(pos, new Piece(new Rank("eight"), new Color("red")));
    clone := game.getClonePiece(pos);

    assertTrue(clone.rank.name = "eight" and clone.color.name = "red");
    IO`println("getClonePiece : passed");
   );

  --Test if the empty piece is in fact empy
  public testemptyPiece : () ==> ()
   testemptyPiece() ==
   (
    dcl game : Game := new Game();
    dcl pos : Game`Position := mk_Game`Position(0,0);

    game.addNewPiece(pos, game.emptyPiece());

    assertTrue(game.getPiece(pos).rank.name = "null" and game.getPiece(pos).color.name = "null");
    IO`println("emptyPiece :  passed");
   );

  --Test if the piece to be moved belongs to the turn player
  public testcheckTurn : () ==> ()
   testcheckTurn() ==
   (
    dcl game : Game := new Game();
    dcl pos : Game`Position := mk_Game`Position(0,0);

    game.addNewPiece(pos, new Piece(new Rank("eight"), new Color("red")));
    game.turn := "blue";

    assertTrue(game.checkTurn(pos) = false);
    IO`println("checkTurn : passed");
   );

  --Test if the turn is changed
  public testchangeTurn : () ==> ()
   testchangeTurn() ==
   (
    dcl game : Game := new Game();

    game.turn := "blue";
    game.changeTurn();

    assertTrue(game.turn = "red");
    IO`println("changeTurn : passed");
   );

  --Test if the piece to be moved to, belongs to the opponent player
  public testgetOpponentColor : () ==> ()
   testgetOpponentColor() ==
   (
    dcl game : Game := new Game();
    dcl pos : Game`Position := mk_Game`Position(0,0);

    game.addNewPiece(pos, new Piece(new Rank("eight"), new Color("red")));

    assertTrue(game.getOpponentColor(pos) = "blue");
    IO`println("getOpponentColor : passed");
   );

  --Test if the piece to be moved to is no other than the opponent's or empty
  public testcheckToPositionColor : () ==> ()
   testcheckToPositionColor() ==
   (
    dcl game : Game := new Game();
    dcl pos1 : Game`Position := mk_Game`Position(0,0);
    dcl pos2 : Game`Position := mk_Game`Position(0,1);

    game.addNewPiece(pos1, new Piece(new Rank("eight"), new Color("red")));
    game.addNewPiece(pos2, new Piece(new Rank("eight"), new Color("blue")));

    assertTrue(game.checkToPositionColor(pos1,pos2));
    IO`println("checkToPositionColor : passed");
   );

  --Test if the pieces in the middle ar empty (all empty)
  public testcheckMiddlePieces1 : () ==> ()
   testcheckMiddlePieces1() ==
   (
    dcl game : Game := new Game();
    dcl pos1 : Game`Position := mk_Game`Position(0,0);
    dcl pos2 : Game`Position := mk_Game`Position(0,1);
    dcl pos3 : Game`Position := mk_Game`Position(0,2);
    dcl pos4 : Game`Position := mk_Game`Position(0,3);

    game.addNewPiece(pos1, new Piece(new Rank("eight"), new Color("red")));
    game.addNewPiece(pos2, new Piece(new Rank("null"), new Color("null")));
    game.addNewPiece(pos3, new Piece(new Rank("null"), new Color("null")));
    game.addNewPiece(pos4, new Piece(new Rank("eight"), new Color("blue")));

    assertTrue(game.checkMiddlePieces(pos1,pos4));
    IO`println("checkMiddlePieces1 : passed");
   );

  --Test if the pieces in the middle ar empty (one water)
  public testcheckMiddlePieces2 : () ==> ()
```

```
 testcheckMiddlePieces2() ==
 (
  dcl game : Game := new Game();
  dcl pos1 : Game`Position := mk_Game`Position(0,0);
  dcl pos2 : Game`Position := mk_Game`Position(0,1);
  dcl pos3 : Game`Position := mk_Game`Position(0,2);
  dcl pos4 : Game`Position := mk_Game`Position(0,3);

  game.addNewPiece(pos1, new Piece(new Rank("eight"), new Color("red")));
  game.addNewPiece(pos2, new Piece(new Rank("water"), new Color("null")));
  game.addNewPiece(pos3, new Piece(new Rank("null"), new Color("null")));
  game.addNewPiece(pos4, new Piece(new Rank("eight"), new Color("blue")));

  assertTrue(game.checkMiddlePieces(pos1,pos4));
  IO`println("checkMiddlePieces2 : passed");
 );

--Test if the pieces in the middle ar empty (one piece)
public testcheckMiddlePieces3 : () ==> ()
 testcheckMiddlePieces3() ==
 (
  dcl game : Game := new Game();
  dcl pos1 : Game`Position := mk_Game`Position(0,0);
  dcl pos2 : Game`Position := mk_Game`Position(0,1);
  dcl pos3 : Game`Position := mk_Game`Position(0,2);
  dcl pos4 : Game`Position := mk_Game`Position(0,3);

  game.addNewPiece(pos1, new Piece(new Rank("eight"), new Color("red")));
  game.addNewPiece(pos2, new Piece(new Rank("null"), new Color("null")));
  game.addNewPiece(pos3, new Piece(new Rank("eight"), new Color("blue")));
  game.addNewPiece(pos4, new Piece(new Rank("null"), new Color("null")));

  assertTrue(game.checkMiddlePieces(pos1,pos4) = false);
  IO`println("checkMiddlePieces3 : passed");
 );

--Test if the number of steps in a move is less or equal to the piece steps (2 steps in 1 step piece)
public testcheckMovement1 : () ==> ()
 testcheckMovement1() ==
 (
  dcl game : Game := new Game();
  dcl pos1 : Game`Position := mk_Game`Position(0,0);
  dcl pos2 : Game`Position := mk_Game`Position(0,1);
  dcl pos3 : Game`Position := mk_Game`Position(0,2);

  game.addNewPiece(pos1, new Piece(new Rank("eight"), new Color("red")));
  game.addNewPiece(pos2, new Piece(new Rank("null"), new Color("null")));
  game.addNewPiece(pos3, new Piece(new Rank("null"), new Color("null")));

  assertTrue(game.checkMovement(pos1,pos3) = false);
  IO`println("checkMovement1 : passed");
 );


--Test if the number of steps in a move is less or equal to the piece steps (2 steps in 8 step piece)
public testcheckMovement2 : () ==> ()
 testcheckMovement2() ==
 (
  dcl game : Game := new Game();
  dcl pos1 : Game`Position := mk_Game`Position(0,0);
  dcl pos2 : Game`Position := mk_Game`Position(0,1);
  dcl pos3 : Game`Position := mk_Game`Position(0,2);

  game.addNewPiece(pos1, new Piece(new Rank("spy"), new Color("red")));
  game.addNewPiece(pos2, new Piece(new Rank("null"), new Color("null")));
```

```
    game.addNewPiece(pos3, new Piece(new Rank("null"), new Color("null")));

    assertTrue(game.checkMovement(pos1,pos3));
    IO`println("checkMovement2 : passed");
  );

--Test if the movement to be made is valid (same piece)
public testvalidMove1 : () ==> ()
 testvalidMove1() ==
  (
    dcl game : Game := new Game();
    dcl pos : Game`Position := mk_Game`Position(0,0);

    game.addNewPiece(pos, new Piece(new Rank("eight"), new Color("red")));

    assertTrue(game.validMove(0,0,0,0) = false);
    IO`println("validMove1 : passed");
  );

--Test if the movement to be made is valid (empty cell)
public testvalidMove2 : () ==> ()
 testvalidMove2() ==
  (
    dcl game : Game := new Game();
    dcl pos1 : Game`Position := mk_Game`Position(0,0);
    dcl pos2 : Game`Position := mk_Game`Position(0,1);

    game.addNewPiece(pos1, new Piece(new Rank("eight"), new Color("red")));
    game.addNewPiece(pos2, new Piece(new Rank("null"), new Color("null")));

    assertTrue(game.validMove(0,0,0,1));
    IO`println("validMove2 : passed");
  );

--Test if the movement made was valid (spy, 2 steps to an empty cell)
public testMove1 : () ==> ()
 testMove1() ==
  (
    dcl game : Game := new Game();
    dcl pos1 : Game`Position := mk_Game`Position(0,0);
    dcl pos2 : Game`Position := mk_Game`Position(0,1);
    dcl pos3 : Game`Position := mk_Game`Position(0,2);

    game.addNewPiece(pos1, new Piece(new Rank("spy"), new Color("red")));
    game.addNewPiece(pos2, new Piece(new Rank("null"), new Color("null")));
    game.addNewPiece(pos3, new Piece(new Rank("null"), new Color("null")));
    game.move(0,0,0,2);

    assertTrue(game.getPiece(pos3).rank.name = "spy" and game.getPiece(pos1).rank.name = "null");
    IO`println("Move1 : passed");
  );

--Test if the movement made was valid (spy, 2 steps to a major cell)
public testMove2 : () ==> ()
 testMove2() ==
  (
    dcl game : Game := new Game();
    dcl pos1 : Game`Position := mk_Game`Position(0,0);
    dcl pos2 : Game`Position := mk_Game`Position(0,1);
    dcl pos3 : Game`Position := mk_Game`Position(0,2);

    game.addNewPiece(pos1, new Piece(new Rank("spy"), new Color("red")));
    game.addNewPiece(pos2, new Piece(new Rank("null"), new Color("null")));
    game.addNewPiece(pos3, new Piece(new Rank("ten"), new Color("blue")));
    game.move(0,0,0,2);
```

```
   assertTrue(game.getPiece(pos3).rank.name = "spy" and game.getPiece(pos1).rank.name = "null");
   IO`println("Move2 : passed");
  );

 --Test if the movement made was valid (three, 1 step to a bomb)
 public testMove3 : () ==> ()
  testMove3() ==
  (
   dcl game : Game := new Game();
   dcl pos1 : Game`Position := mk_Game`Position(0,0);
   dcl pos2 : Game`Position := mk_Game`Position(0,1);

   game.addNewPiece(pos1, new Piece(new Rank("three"), new Color("red")));
   game.addNewPiece(pos2, new Piece(new Rank("bomb"), new Color("blue")));
   game.move(0,0,0,1);

   assertTrue(game.getPiece(pos2).rank.name = "three" and game.getPiece(pos1).rank.name = "null");
   IO`println("Move3 : passed");
  );

 --Test if the movement made was valid (two, 1 step to a bomb)
 public testMove4 : () ==> ()
  testMove4() ==
  (
   dcl game : Game := new Game();
   dcl pos1 : Game`Position := mk_Game`Position(0,0);
   dcl pos2 : Game`Position := mk_Game`Position(0,1);

   game.addNewPiece(pos1, new Piece(new Rank("two"), new Color("red")));
   game.addNewPiece(pos2, new Piece(new Rank("bomb"), new Color("blue")));
   game.move(0,0,0,1);

   assertTrue(game.getPiece(pos2).rank.name = "null" and game.getPiece(pos1).rank.name = "null");
   IO`println("Move4 : passed");
  );

 --Test if the movement made was valid (two, 1 steps to higher opponent)
 public testMove5 : () ==> ()
  testMove5() ==
  (
   dcl game : Game := new Game();
   dcl pos1 : Game`Position := mk_Game`Position(0,0);
   dcl pos2 : Game`Position := mk_Game`Position(0,1);

   game.addNewPiece(pos1, new Piece(new Rank("two"), new Color("red")));
   game.addNewPiece(pos2, new Piece(new Rank("three"), new Color("blue")));
   game.move(0,0,0,1);

   assertTrue(game.getPiece(pos2).rank.name = "three" and game.getPiece(pos1).rank.name = "null");
   IO`println("Move5 : passed");
  );

 --Test if the movement made was valid (three, 1 steps to lower opponent)
 public testMove6 : () ==> ()
  testMove6() ==
  (
   dcl game : Game := new Game();
   dcl pos1 : Game`Position := mk_Game`Position(0,0);
   dcl pos2 : Game`Position := mk_Game`Position(0,1);

   game.addNewPiece(pos1, new Piece(new Rank("three"), new Color("red")));
   game.addNewPiece(pos2, new Piece(new Rank("two"), new Color("blue")));
   game.move(0,0,0,1);
  assertTrue(game.getPiece(pos2).rank.name = "three" and game.getPiece(pos1).rank.name = "null");
  IO`println("Move6 : passed");
 );

--Test if after a 'move' the turn changes
public testturnChanged : () ==> ()
 testturnChanged() ==
 (
  dcl game : Game := new Game();
  dcl pos1 : Game`Position := mk_Game`Position(0,0);
  dcl pos2 : Game`Position := mk_Game`Position(0,1);

  game.turn := "red";
  game.addNewPiece(pos1, new Piece(new Rank("three"), new Color("red")));
  game.addNewPiece(pos2, new Piece(new Rank("two"), new Color("blue")));
  game.move(0,0,0,1);

  assertTrue(game.turn = "blue");
  IO`println("testturnChanged : passed");
 );

--Test if after a flag capture the game ends
public testgameEnded : () ==> ()
 testgameEnded() ==
 (
  dcl game : Game := new Game();
  dcl pos1 : Game`Position := mk_Game`Position(0,0);
  dcl pos2 : Game`Position := mk_Game`Position(0,1);

  game.addNewPiece(pos1, new Piece(new Rank("two"), new Color("red")));
  game.addNewPiece(pos2, new Piece(new Rank("flag"), new Color("blue")));
  game.move(0,0,0,1);

  assertTrue(game.gameEnded());
  IO`println("gameEnded : passed");
 );

--Test if the winner is correct
public testgetWinner : () ==> ()
 testgetWinner() ==
 (
  dcl game : Game := new Game();
  dcl pos1 : Game`Position := mk_Game`Position(0,0);
  dcl pos2 : Game`Position := mk_Game`Position(0,1);

  game.addNewPiece(pos1, new Piece(new Rank("two"), new Color("red")));
  game.addNewPiece(pos2, new Piece(new Rank("flag"), new Color("blue")));
  game.move(0,0,0,1);

  assertTrue(game.getWinner() = "red");
  IO`println("getWinner :  passed");
 );

public testAll : () ==> ()
 testAll() ==
 (
  testgetBoardSize();
  testcheckAvaiability1();
  testcheckAvaiability2();
  testaddNewPiece();
  testgetPiece();
  testgetClonePiece();
  testcheckTurn();
  testchangeTurn();
  testgetOpponentColor();
```

```
    testcheckToPositionColor();
    testcheckMiddlePieces1();
    testcheckMiddlePieces2();
    testcheckMiddlePieces3();
    testcheckMovement1();
    testcheckMovement2();
    testvalidMove1();
    testvalidMove2();
    testMove1();
    testMove2();
    testMove3();
    testMove4();
    testMove5();
    testMove6();
    testgameEnded();
    testturnChanged();
   );
end Tests
```

Function or operation	Coverage	Calls
assertTrue	100.0%	25
testAll	100.0%	1
testMove1	100.0%	1
testMove2	100.0%	1
testMove3	100.0%	1
testMove4	100.0%	1
testMove5	100.0%	1
testMove6	100.0%	1
testaddNewPiece	100.0%	1
testchangeTurn	100.0%	1
testcheckAvaiability1	100.0%	1
testcheckAvaiability2	100.0%	1
testcheckMiddlePieces1	100.0%	1
testcheckMiddlePieces2	100.0%	1
testcheckMiddlePieces3	100.0%	1
testcheckMovement1	100.0%	1
testcheckMovement2	100.0%	1
testcheckToPositionColor	100.0%	1
testcheckTurn	100.0%	1
testemptyPiece	0.0%	0
testgameEnded	100.0%	1
testgetBoardSize	100.0%	1
testgetClonePiece	100.0%	1
testgetOpponentColor	100.0%	1
testgetPiece	100.0%	1
testgetWinner	0.0%	0
testturnChanged	100.0%	1
testvalidMove1	100.0%	1
testvalidMove2	100.0%	1
Tests.vdmpp	94.0%	51