

# battleship-VDM

January 3, 2017

## Contents

<b>1</b>	<b>Board</b>	<b>1</b>
<b>2</b>	<b>Game</b>	<b>1</b>
<b>3</b>	<b>Player</b>	<b>4</b>
<b>4</b>	<b>TestBase</b>	<b>6</b>
<b>5</b>	<b>TestBoard</b>	<b>7</b>
<b>6</b>	<b>TestGame</b>	<b>9</b>
<b>7</b>	<b>TestPlayer</b>	<b>11</b>

## 1 Board

```
class Board
types
  public CellContent = <Empty> | <Carrier> | <Battleship> | <Cruiser> | <Submarine> | <Destroyer>
    | <Hit> | <Miss>;
  public Direction = <Up> | <Down> | <Left> | <Right>

values
  protected static shipSize: map CellContent to nat1 = {
    <Carrier> |-> 5, <Battleship> |-> 4, <Cruiser> |-> 3, <Submarine> |-> 3, <Destroyer> |-> 2
  };
  protected static colMap: map char to nat1 = {
    'A' |-> 1, 'B' |-> 2, 'C' |-> 3, 'D' |-> 4, 'E' |-> 5, 'F' |-> 6, 'G' |-> 7, 'H' |-> 8, 'I' |->
      9, 'J' |-> 10
  }; -- ndice de cada coluna

instance variables
  protected cells: seq of seq of CellContent := []; --matriz de clulas

  inv len cells = 10; -- cada tabuleiro tem 10 linhas
  inv card {i| i in set inds cells & len cells(i) = 10} = 10; -- cada linha tem 10 colunas

operations
--cria tabuleiro com todas clulas vazias

  public Board: () ==> Board
  Board() == (
    cells := [[<Empty>| x in set {1,...,10}]] x in set {1,...,10}];
```

```

    return self;
  );

--preenche a clula (line,col) com content

public setComponent: CellContent * nat1 * nat1 ==> ()
  setComponent(content, line, col) == (
    cells(line) := cells(line) ++ {col |-> content}
  )
  pre col <= 10 and line <= 10 and content <> <Empty>;

--mapeia coluna para ndice e chama fun o para preencher clula

public setComponentCol: CellContent * nat1 * char ==> ()
  setComponentCol(content, line, colCh) == setComponent(content, line, colMap(colCh))
  pre colCh in set dom colMap;

-- tendo a clula A=(line,col) e size-1 culas numa determinada direc o (dir)
-- todas estas clulas esto dentro dos limites do tabuleiro e esto vazias

pure public emptyValidCells: nat1 * nat1 * Direction * nat1 ==> bool
  emptyValidCells(line, col, dir, size) == (
    if dir = <Up> then
      for i = 0 to size - 1 do (
        if line - i <= 0 then return false;
        if cells(line - i)(col) <> <Empty> then return false;
      )
    else if dir = <Down> then
      for i = 0 to size - 1 do (
        if line + i > 10 then return false;
        if cells(line + 1)(col) <> <Empty> then return false;
      )
    else if dir = <Left> then
      for i = 0 to size - 1 do (
        if col - i <= 0 then return false;
        if cells(line)(col - i) <> <Empty> then return false;
      )
    else for i = 0 to size - 1 do (
      if col + i > 10 then return false;
      if cells(line)(col + i) <> <Empty> then return false;
    );
    return true;
  );

-- verifica o nmero de clulas do tabuleiro preenchidas com determinada componente

pure public countCellType: CellContent ==> nat
  countCellType(type) == (
    dcl sum : nat := 0;
    for i = 1 to len cells do
      for j = 1 to len cells(i) do
        if cells(i)(j) = type then sum := sum + 1;
      return sum;
  );

-- coloca um navio no tabuleiro dada clula A=(line, colCh) numa determinada direc o

public placeShip: CellContent * char * nat1 * Direction ==> ()
  placeShip(ship, colCh, line, dir) == (
    dcl col: nat1 := colMap(colCh);
    if dir = <Up> then
      for i = 0 to shipSize(ship) - 1 do
        setComponent(ship, line - i, col)
    else if dir = <Down> then
      for i = 0 to shipSize(ship) - 1 do

```

```

        setComponent(ship, line + i, col)
    else if dir = <Left> then
        for i = 0 to shipSize(ship) - 1 do
            setComponent(ship, line, col - i)
        else for i = 0 to shipSize(ship) - 1 do
            setComponent(ship, line, col + i)
    )
    pre ship in set dom shipSize
    and colCh in set dom colMap
    and countCellType(ship) = 0
    and emptyValidCells(line, colMap(colCh), dir, shipSize(ship)) --check if cells that a ship
        takes are empty and if ship fits in the board
    post countCellType(ship) = shipSize(ship);

-- retorna a lista de navios numa frota

public getShips : () ==> set of CellContent
getShips() == return dom shipSize;

-- retorna o nmero de navios numa frota

public getShipsCount : () ==> nat1
getShipsCount() == return card dom shipSize;

-- verifica se numa clula existe um navio e regista com <Hit> ou <Miss>

public registerMove: char * nat1 ==> CellContent
registerMove(colCh, line) == (
    dcl col: nat1 := colMap(colCh);
    dcl oldValue: CellContent := cells(line)(col);
    if oldValue = <Empty> then cells(line)(col) := <Miss>
    else cells(line)(col) := <Hit>;
    return oldValue;
)
pre colCh in set dom colMap and cells(line)(colMap(colCh)) not in set {<Hit>, <Miss>}
post cells(line)(colMap(colCh)) in set {<Hit>, <Miss>};

--- print to console

-- retorna o navio em string

public shipToString : CellContent ==> Player'string
shipToString(type) == (
    cases type:
    <Empty> -> return "Empty",
    <Carrier> -> return "Carrier",
    <Battleship> -> return "Battleship",
    <Cruiser> -> return "Cruiser",
    <Submarine> -> return "Submarine",
    <Destroyer> -> return "Destroyer",
    <Hit> -> return "Hit",
    <Miss> -> return "Miss"
    end;
    return "Unknow"
);

-- retorna a representao de um navio no tabuleiro

public cellToString : CellContent ==> Player'string
cellToString(type) == (
    dcl fullStr: Player'string := shipToString(type);
    return [fullStr(i) | i in set inds fullStr & i < 4];
);

-- imprime um tabuleiro e peas 1 inseridas

```

```

public printBoard : () ==> Player'string
printBoard() == (
  dcl board: Player'string := [];
  for i = 1 to len cells do (
    if i < 10 then board := board ^ "0";
    board := board ^ VDMUtil.val2seq_of_char[nat](i) ^ " ";
    for j = 1 to len cells(i) do board:= board ^ cellToString(cells(i)(j)) ^ " ";
    board := board ^ "\n";
  );
  return "      A      B      C      D      E      F      G      H      I      J      \n\n" ^
    board);

-- imprime dois tabuleiros lado a lado

public printParallelBoards : Board ==> Player'string
printParallelBoards(enemyBoard) == (
  dcl pBoard: Player'string := [];
  for i = 1 to len cells do (
    if i < 10 then pBoard := pBoard ^ "0";
    pBoard := pBoard ^ VDMUtil.val2seq_of_char[nat](i) ^ " ";
    for j = 1 to len cells(i) do pBoard:= pBoard ^ cellToString(cells(i)(j)) ^ " ";

    pBoard := pBoard ^ "\t\t\t";
    if i < 10 then pBoard := pBoard ^ "0";
    pBoard := pBoard ^ VDMUtil.val2seq_of_char[nat](i) ^ " ";
    for j = 1 to len enemyBoard.cells(i) do pBoard:= pBoard ^ cellToString(enemyBoard.cells(i)(j)
      ) ^ " ";

    pBoard := pBoard ^ "\n";
  );
  return "      A      B      C      D      E      F      G      H      I      J      " ^
    "\t\t\t" ^ "      A      B      C      D      E      F      G      H      I      J      \n\n" ^
    pBoard);

-- imprime os navios de uma lista em formato string

public printRemainShips: set of CellContent ==> Player'string
printRemainShips(ships) == (
  dcl res: Player'string := [];
  for all ship in set ships do res := res ^ shipToString(ship) ^ ": " ^ VDMUtil.val2seq_of_char[
    nat](shipSize(ship)) ^ " ";
  return res;
)

end Board

```

Function or operation	Line	Coverage	Calls
Board	22	100.0%	46
cellToString	137	100.0%	12802
countCellType	67	100.0%	232
emptyValidCells	42	100.0%	88
getShips	99	100.0%	26
getShipsCount	103	100.0%	48
placeShip	77	100.0%	70
printBoard	144	100.0%	48
printParallelBoards	156	100.0%	40
printRemainShips	175	100.0%	48

registerMove	107	100.0%	84
setComponent	29	100.0%	298
setComponentCol	36	100.0%	58
shipToString	121	93.1%	13214
Board.vdmpp		99.6%	27102

## 2 Game

```

class Game
types
  protected State = <Off> | <Start> | <Placed> | <Round>

instance variables
  protected playerA: Player := new Player("Default_1");
  protected playerB: Player := new Player("Default_2");
  protected currPlayer: Player := playerA;
  protected players: set of Player := {playerA, playerB};
  protected currState: State := <Off>;

  inv currPlayer in set {playerA, playerB};
  inv playerA in set players;
  inv playerB in set players;
  inv forall p1, p2 in set players & p1 <> p2 => p1.getName() <> p2.getName();

operations

  -- cria jogo

  public Game: Player`string * Player`string ==> Game
    Game(name1, name2) == (
      playerA := new Player(name1);
      playerB := new Player(name2);
      currPlayer := playerA;
      players := {playerA, playerB};
      return self
    )
    pre currState= <Off> and name1 <> name2
    post card players = 2;

    -- cria novo jogador com determinado nome se nome ainda no estiver registrado

  public createPlayer: (Player`string) ==> Player
    createPlayer(name) == (
      dcl player: Player := new Player(name);
      players := players union {player};
      return player
    )
    pre not exists p in set players & p.getName() = name;

    --muda os jogadores que v o realizar jogo

  public changePlayers: Player`string * Player`string ==> ()
    changePlayers(name1, name2) == (
      dcl tmpPlayer: Player := iota p in set players & p.getName() = name1;
      atomic(playerA := tmpPlayer;
        playerB := iota p in set players & p.getName() = name2;
        currPlayer := tmpPlayer;
      );
    )

```

```

pre currState = <Off> and exists p1, p2 in set players & p1.getName() = name1 and p2.getName()
    = name2;

    -- muda o turno para jogar em cada ronda

public switchTurns: () ==> ()
switchTurns() == (
    if currPlayer = playerA then currPlayer := playerB
    else currPlayer := playerA
)
pre currState <> <Off>;

    -- retorna o jogador que aguarda o ataque do adversario

public getOtherPlayer: () ==> Player
getOtherPlayer() == (
    if currPlayer = playerA then return playerB
    else return playerA;
);

    -- inicia jogo

public startGame: () ==> Player'string
startGame() == (
    playerA.addBoards();
    playerB.addBoards();
    currState := <Start>;
    return "Game started with following players:\n"
    ^ playerA.printInfo()
    ^ playerB.printInfo() ^ "\n\n\n\n"
    ^ currPlayer.printPlacementStatus();
)
pre currState = <Off>;

    -- jogador ativo colca os navios no tabuleiro

public shipPlacement: Board'CellContent * char * nat1 * Board'Direction ==> Player'string
shipPlacement(ship, colCh, line, dir) == (
    dcl ret: Player'string;
    currPlayer.shipPlacement(ship, colCh, line, dir);
    ret := currPlayer.printPlacementStatus();
    if currPlayer.allShipsPlaced() then (
        switchTurns();
        ret := ret ^ "\n\n\n\n\n";
        if currPlayer.allShipsPlaced() then (
            currState := <Placed>;
            ret := ret ^ "All ships placed\n";
        )
    else ret := ret ^ currPlayer.printPlacementStatus();
    );
    return ret;
)
pre currState = <Start>;

    -- inicia rondas no jogo

public startRounds: () ==> Player'string
startRounds() == (
    playerA.startRounds();
    playerB.startRounds();
    currState := <Round>;
    return currPlayer.printGameStatus();
)
pre currState = <Placed>;

```

```

-- jogador activo tenta adivinhar posi o de um navio adversario

public guessShipPosition: char * nat1 ==> Player`string
guessShipPosition(colCh, line) == (
  dcl othPlayer: Player := getOtherPlayer();
  dcl code: Board`CellContent := othPlayer.registerAttack(colCh, line);
  dcl ret: Player`string := [];
  dcl final: bool := currPlayer.registerResult(code, colCh, line);
  if code = <Miss> then (
    ret := ret ^ "\n\nSplash!! You missed!\n";
    switchTurns();
  )
  else if code = <Hit> then ret := ret ^ "\n\nGreat strike\n"
  else ret := ret ^ currPlayer.printTakeDown(code);
  if final then(
    ret := ret ^ currPlayer.printVictory();
    currState := <Off>;
    playerA.clearData();
    playerB.clearData();
    return ret;
  );
  ret := ret ^ "\n\n\n\n" ^ currPlayer.printGameStatus();
  return ret;
)
pre currState = <Round>;

end Game

```

Function or operation	Line	Coverage	Calls
Game	20	100.0%	6
changePlayers	41	100.0%	2
createPlayer	32	100.0%	2
getOtherPlayer	60	100.0%	40
guessShipPosition	109	100.0%	40
shipPlacement	80	100.0%	40
startGame	67	100.0%	4
startRounds	99	100.0%	2
switchTurns	52	100.0%	14
Game.vdmpp		100.0%	150

### 3 Player

```

class Player
types
  public string = seq of char;

instance variables
  protected name: string := [];
  protected wins: nat;
  protected losses: nat;
  protected ownBoard: [Board] := nil; --tabuleiro aonde coloca-se os proprios navios e regista-se
    as tentativas do inimigo
  protected enemyBoard: [Board] := nil; --tabuleiro das nossas tentativas de destoir a frota
    inimiga

```

```

protected myShips: set of Board'CellContent := {}; -- navios colocados antes da ronda e meus
    navios durante as rondas (diminui)
protected enemiesShips: set of Board'CellContent := {}; -- navios inimigos (aumenta)

inv len name < 256;

operations

public Player: string ==> Player
Player(nameArg) == (
    name := nameArg;
    wins := 0;
    losses := 0;
    return self
);

pure public getName : () ==> string
getName() == return name;

-- reinicia jogo

public addBoards : () ==> ()
addBoards() == (
    ownBoard := new Board();
    enemyBoard := new Board();
    myShips := ownBoard.getShips();
)
pre ownBoard = nil and enemyBoard = nil;

-- coloca navio no tabuleiro se este ainda n o tiver sido colocado

public shipPlacement: Board'CellContent * char * nat1 * Board'Direction ==> ()
shipPlacement(ship, colCh, line, dir) == (
    ownBoard.placeShip(ship, colCh, line, dir);
    myShips := myShips \ {ship}
)
pre ship in set myShips;

-- retorna verdadeiro se todos os navios esto colocados no tabuleiro

public allShipsPlaced: () ==> bool
allShipsPlaced() == return card myShips = 0;

-- inicia parte do jogo com o objetivo de destruir o navio inimigo

public startRounds: () ==> ()
startRounds () == (
    myShips := ownBoard.getShips();
    enemiesShips := {};
);

-- limpa tabuleiro e registro dos navios

public clearData: () ==> ()
clearData() == (
    ownBoard := nil;
    enemyBoard := nil;
    myShips := {};
    enemiesShips := {};
);

-- registra tentativa de afundano do adversario

public registerAttack: char * nat1 ==> Board'CellContent

```





```

    printVictory() == return "\nEnemy fleet destroyed. Victory!\n";
end Player

```

Function or operation	Line	Coverage	Calls
Player	17	100.0%	38
addBoards	29	100.0%	16
allShipsPlaced	46	100.0%	52
clearData	57	100.0%	6
getName	25	100.0%	490
printGameStatus	108	100.0%	40
printInfo	96	100.0%	8
printPlacementStatus	102	100.0%	48
printTakeDown	120	100.0%	10
printVictory	124	100.0%	2
registerAttack	66	100.0%	78
registerResult	79	100.0%	54
shipPlacement	38	100.0%	60
startRounds	50	100.0%	8
Player.vdmpp		100.0%	910

## 4 TestBase

```

class TestBase

operations

    static public assertTrue : bool ==> ()
        assertTrue(cond) == return
    pre cond;

    static public assertEquals : ? * ? ==> ()
        assertEquals(result, expected) == return
    post result = expected;

    static public runAllTests: () ==> ()
        runAllTests() == (
            dcl tb: TestBoard := new TestBoard();
            dcl tp: TestPlayer := new TestPlayer();
            dcl tg: TestGame := new TestGame();
            tb.run();
            tp.run();
            tg.run();
        );
end TestBase

```

Function or operation	Line	Coverage	Calls
assertEqual	9	100.0%	92
assertTrue	5	100.0%	36
runAllTests	13	100.0%	2
TestBase.vdmpp		100.0%	130

## 5 TestBoard

```

class TestBoard is subclass of Board

operations
-- todas as clulas esto vazias ao criar o tabuleiro

private testCreateBoard : () ==> ()
testCreateBoard() == (
  dcl b: Board := new Board();
  TestBase`assertEqual(b.countCellType(<Empty>),100);
);

-- testa mapeamento entre colunas char e seu ndice

private testColMap : () ==> ()
testColMap() == (
  dcl b: Board := new Board();
  dcl c: Board := new Board();
  TestBase`assertEqual(2,colMap('B'));
  TestBase`assertEqual(2-1,colMap('A'));
  TestBase`assertEqual(2+1,colMap('C'));

  b.setComponent(<Cruiser>,1,2);
  c.setComponentCol(<Cruiser>,1,'B');
  TestBase`assertEqual(b.cells(1)(2),c.cells(1)(colMap('B')));
);

-- testa limites do tabuleiro e disponibilidade de uma clula

private testEmptyBeforePlacement: () ==> ()
testEmptyBeforePlacement() == (
  dcl b: Board := new Board();
  b.setComponentCol(<Cruiser>,6,'F');
  TestBase`assertTrue(b.emptyValidCells(2,colMap('C'),<Right>,3));
  TestBase`assertTrue(not b.emptyValidCells(7,colMap('F'),<Up>,3));
  TestBase`assertTrue(not b.emptyValidCells(1,colMap('F'),<Up>,3));
  TestBase`assertTrue(not b.emptyValidCells(1,colMap('A'),<Left>,2));
  TestBase`assertTrue(not b.emptyValidCells(9,colMap('H'),<Down>,3));
  TestBase`assertTrue(not b.emptyValidCells(5,colMap('F'),<Down>,3));
  TestBase`assertTrue(not b.emptyValidCells(6,colMap('C'),<Right>,5));
  TestBase`assertTrue(not b.emptyValidCells(6,colMap('H'),<Left>,4));
  TestBase`assertTrue(not b.emptyValidCells(2,colMap('H'),<Right>,5));

  TestBase`assertEqual(b.countCellType(<Submarine>),0);
  TestBase`assertEqual(shipSize(<Submarine>),3);
  b.placeShip(<Submarine>,'C',2,<Right>);
  TestBase`assertEqual(b.countCellType(<Submarine>),3);

  b.placeShip(<Carrier>,'J',1,<Down>);
  TestBase`assertEqual(b.countCellType(<Carrier>),5);

  b.placeShip(<Destroyer>,'J',9,<Left>);

```

```

TestBase`assertEqual(b.countCellType(<Destroyer>),2);

b.placeShip(<Battleship>,'A',7,<Up>);
TestBase`assertEqual(b.countCellType(<Battleship>),4);
);

-- teste simples get da frota

private testGetShips: () ==> ()
testGetShips() == (
  dcl b: Board := new Board();
  TestBase`assertEqual(b.getShipsCount(),5);
  TestBase`assertEqual(b.getShips(),{<Carrier>, <Battleship>, <Cruiser>, <Submarine>, <Destroyer>});
);

-- testa uma tentativa de afundar um navio

private testRegisterMove: () ==> ()
testRegisterMove() == (
  dcl b: Board := new Board();
  dcl ret: CellContent := <Empty>;
  b.placeShip(<Submarine>,'C',2,<Right>);
  ret := b.registerMove('A',3);
  TestBase`assertEqual(ret,<Empty>);
  TestBase`assertEqual(b.cells(3)(colMap('A')),<Miss>);

  ret := b.registerMove('D',2);
  TestBase`assertEqual(ret,<Submarine>);
  TestBase`assertEqual(b.cells(2)(colMap('D')),<Hit>);

  ret := b.registerMove('C',2);
  TestBase`assertEqual(ret,<Submarine>);
  TestBase`assertEqual(b.cells(2)(colMap('C')),<Hit>);
);

private testPrints: () ==> ()
testPrints() == (
  TestBase`assertEqual(shipToString(<Carrier>),"Carrier");
  TestBase`assertEqual(cellToString(<Carrier>),"Car");
);

public run: () ==> ()
run() == (
  testCreateBoard();
  testColMap();
  testEmptyBeforePlacement();
  testGetShips();
  testRegisterMove();
  testPrints();
);
end TestBoard

```

Function or operation	Line	Coverage	Calls
run	88	100.0%	2
testColMap	12	100.0%	2
testCreateBoard	5	100.0%	2
testEmptyBeforePlacement	26	100.0%	2

testGetShips	56	100.0%	2
testPrints	82	100.0%	2
testRegisterMove	64	100.0%	2
TestBoard.vdmpp		100.0%	14

## 6 TestGame

```

class TestGame is subclass of Game

operations

--testa a cria o de um jogo

private testCreateGame: () ==> ()
testCreateGame() == (
  dcl g: Game := new Game("John","Diana");
  dcl p: Player:= g.playerA;
  TestBase`assertEqual(card g.players,2);

  p:= g.createPlayer("Paul");
  TestBase`assertEqual(card g.players,3);
  TestBase`assertEqual(g.playerA.getName(),"John");
  TestBase`assertEqual(g.playerB.getName(),"Diana");

  g.changePlayers("Paul","John");
  TestBase`assertEqual(g.playerA.getName(),"Paul");
  TestBase`assertEqual(g.playerB.getName(),"John");
);

-- teste a colocao dos navios

public testEmplacement: () ==> ()
testEmplacement() == (

  dcl g: Game := new Game("Ana","Paula");
  dcl str: Player`string := [];
  str := g.startGame();
  str := g.shipPlacement(<Submarine>,'C',9,<Right>);
  TestBase`assertEqual(g.currState,<Start>);
  TestBase`assertEqual(g.currPlayer,g.playerA);

  str := g.shipPlacement(<Carrier>,'H',9,<Up>);
  str := g.shipPlacement(<Cruiser>,'G',2,<Right>);
  str := g.shipPlacement(<Battleship>,'A',2,<Down>);
  str := g.shipPlacement(<Destroyer>,'C',4,<Down>);

  TestBase`assertEqual(g.currState,<Start>);
  TestBase`assertEqual(g.currPlayer,g.playerB);

  str := g.shipPlacement(<Submarine>,'C',9,<Right>);
  str := g.shipPlacement(<Carrier>,'H',9,<Up>);
  str := g.shipPlacement(<Cruiser>,'G',2,<Right>);
  str := g.shipPlacement(<Battleship>,'A',2,<Down>);
  str := g.shipPlacement(<Destroyer>,'C',4,<Down>);

  TestBase`assertEqual(g.currState,<Placed>);
);

--testa as consecutivas rondas de um jogo

```

```

private testShipGuess: () ==> ()
testShipGuess() == (
  dcl g: Game := new Game("Ana","Paula");
  dcl str: Player`string := [];
  str := g.startGame();
  str := g.shipPlacement(<Submarine>,'C',9,<Right>);
  str := g.shipPlacement(<Carrier>, 'H', 9, <Up>);
  str := g.shipPlacement(<Cruiser>, 'G',2,<Right>);
  str := g.shipPlacement(<Battleship>,'A',2,<Down>);
  str := g.shipPlacement(<Destroyer>,'C',4,<Down>);

  str := g.shipPlacement(<Submarine>,'C',9,<Right>);
  str := g.shipPlacement(<Carrier>, 'H', 9, <Up>);
  str := g.shipPlacement(<Cruiser>, 'G',2,<Right>);
  str := g.shipPlacement(<Battleship>,'A',2,<Down>);
  str := g.shipPlacement(<Destroyer>,'C',4,<Down>);
  str := g.startRounds();

  TestBase`assertEqual(g.currPlayer,g.playerA);
  str := g.guessShipPosition('H',1);

  TestBase`assertEqual(g.currPlayer,g.playerB);
  str := g.guessShipPosition('C',4);
  TestBase`assertEqual(g.currPlayer,g.playerB);
  str := g.guessShipPosition('C',5);
  str := g.guessShipPosition('C',6);

  str := g.guessShipPosition('F',9);

  str := g.guessShipPosition('C',9);
  str := g.guessShipPosition('D',9);
  str := g.guessShipPosition('E',9);

  str := g.guessShipPosition('A',2);
  str := g.guessShipPosition('A',3);
  str := g.guessShipPosition('A',4);
  str := g.guessShipPosition('A',5);

  str := g.guessShipPosition('G',2);
  str := g.guessShipPosition('H',2);
  str := g.guessShipPosition('I',2);

  str := g.guessShipPosition('H',9);
  str := g.guessShipPosition('H',8);
  str := g.guessShipPosition('H',7);
  str := g.guessShipPosition('H',6);
  str := g.guessShipPosition('H',5);

  TestBase`assertEqual(g.currState,<Off>);
);

public run: () ==> ()
run() == (
  testCreateGame();
  testEmplacement();
  testShipGuess();
);
end TestGame

```

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

run	102	100.0%	2
testCreateGame	6	100.0%	2
testEmplacement	23	100.0%	2
testShipGuess	51	100.0%	2
TestGame.vdmpp		100.0%	8

## 7 TestPlayer

```

class TestPlayer is subclass of Player

operations

private testCreatePlayer: () ==> ()
testCreatePlayer() == (
  dcl pl: Player := new Player("John");
  TestBase`assertEqual(pl.losses+pl.wins,0);
  TestBase`assertTrue(pl.ownBoard = nil);
  pl.addBoards();
  TestBase`assertTrue(pl.ownBoard <> nil);
  TestBase`assertEqual(pl.getName(),"John");
);

private testShipPlacement: () ==> ()
testShipPlacement() == (
  dcl pl: Player := new Player("John");
  pl.addBoards();
  pl.shipPlacement(<Cruiser>,'B',2, <Right>);
  TestBase`assertTrue(pl.ownBoard.countCellType(<Cruiser>) <> 0);
  pl.shipPlacement(<Battleship>,'A',1, <Down>);
  TestBase`assertTrue(not pl.allShipsPlaced());
  pl.shipPlacement(<Carrier>,'J',4, <Left>);
  pl.shipPlacement(<Submarine>,'J',10, <Up>);
  pl.shipPlacement(<Destroyer>,'G',8, <Left>);
  TestBase`assertTrue(pl.allShipsPlaced());
);

private testAttackResponse: () ==> ()
testAttackResponse() == (
  dcl pl: Player := new Player("John");
  dcl ret: Board`CellContent := <Empty>;
  dcl nbShips: nat1 := 1;
  dcl nbLosses: nat := pl.losses;
  pl.addBoards();
  pl.shipPlacement(<Cruiser>,'B',2, <Right>);
  pl.shipPlacement(<Battleship>,'A',1, <Down>);
  pl.shipPlacement(<Carrier>,'J',4, <Left>);
  pl.shipPlacement(<Submarine>,'J',10, <Up>);
  pl.shipPlacement(<Destroyer>,'G',8, <Left>);
  pl.startRounds();
  nbShips := card pl.myShips;
  TestBase`assertTrue(nbShips > 0);

  ret := pl.registerAttack('A',10);
  TestBase`assertEqual(ret,<Miss>);

  ret := pl.registerAttack('G',8);
  TestBase`assertEqual(ret,<Hit>);

```

```

ret := pl.registerAttack('F',8);
TestBase`assertEqual(ret,<Destroyer>);
TestBase`assertEqual(card pl.myShips,nbShips-1);

ret := pl.registerAttack('J',8);
ret := pl.registerAttack('J',9);
ret := pl.registerAttack('J',10);

ret := pl.registerAttack('J',4);
ret := pl.registerAttack('I',4);
ret := pl.registerAttack('H',4);
ret := pl.registerAttack('G',4);
ret := pl.registerAttack('F',4);
ret := pl.registerAttack('E',4);

ret := pl.registerAttack('A',1);
ret := pl.registerAttack('A',2);
ret := pl.registerAttack('A',3);
ret := pl.registerAttack('A',4);

ret := pl.registerAttack('B',2);
ret := pl.registerAttack('C',2);
ret := pl.registerAttack('D',2);

TestBase`assertEqual(pl.losses,nbLosses+1);

pl.clearData();
TestBase`assertTrue(pl.ownBoard = nil);
);

private testAttackResult : () ==> ()
testAttackResult() == (
  dcl pl: Player := new Player("John");
  dcl ret: bool := false;
  dcl nbShips: nat := 0;
  dcl nbWins: nat := pl.wins;
  pl.addBoards();
  pl.startRounds();

  nbWins := pl.wins;
  nbShips := card pl.enemiesShips;
  ret := pl.registerResult(<Miss>,'A',5);
  TestBase`assertTrue(not ret);
  ret := pl.registerResult(<Hit>,'D',10);
  TestBase`assertEqual(card pl.enemiesShips, nbShips);

  ret := pl.registerResult(<Submarine>,'A',6);
  TestBase`assertEqual(card pl.enemiesShips, nbShips+1);

  ret := pl.registerResult(<Cruiser>,'J',2);
  ret := pl.registerResult(<Carrier>,'D',10);
  ret := pl.registerResult(<Destroyer>,'I',6);
  ret := pl.registerResult(< Battleship>,'C',4);
  TestBase`assertTrue(ret);
  TestBase`assertEqual(pl.wins,nbWins+1);
);

public run: () ==> ()
run() == (
  testCreatePlayer();
  testShipPlacement();
  testAttackResponse();

```



```

    testAttackResult();
);
end TestPlayer

```

Function or operation	Line	Coverage	Calls
run	107	100.0%	2
testAttackResponse	28	100.0%	2
testAttackResult	80	100.0%	2
testCreatePlayer	4	100.0%	2
testShipPlacement	14	100.0%	2
TestPlayer.vdmpp		100.0%	10