

# battleship-VDM

January 2, 2017

## Contents

<b>1</b>	<b>Board</b>	<b>1</b>
<b>2</b>	<b>Game</b>	<b>5</b>
<b>3</b>	<b>Player</b>	<b>7</b>
<b>4</b>	<b>TestBase</b>	<b>10</b>
<b>5</b>	<b>TestBoard</b>	<b>10</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 + i)(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 : () ==> int
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 l 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	19	100.0%	4
cellToString	123	0.0%	0
countCellType	59	100.0%	1
emptyValidCells	35	0.0%	0
getComponent	36	0.0%	0
getShips	89	0.0%	0
getShipsCount	92	0.0%	0
placeShip	68	0.0%	0
printBoard	129	0.0%	0
printParallelBoards	140	0.0%	0
printRemainShips	159	0.0%	0
registerMove	95	0.0%	0
setComponent	25	100.0%	2

setComponentCol	31	100.0%	1
shipToString	108	0.0%	0
Board.vdmpp		22.5%	8

## 2 Game

```

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

instance variables
  private inGame: bool := false;
  private playerA: Player;
  private playerB: Player;
  private currPlayer: Player;
  private players: set of Player := {};
  private 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

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 and card players = 0
  post card players = 2;

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;

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;

public switchTurns: () ==> ()

```

```

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

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

public startGame: () ==> Player`String
startGame() == (
  inGame := true;
  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>;

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>;

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

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	0.0%	0
changePlayers	39	0.0%	0
createPlayer	31	0.0%	0
getOtherPlayer	56	0.0%	0
guessShipPosition	102	0.0%	0
shipPlacement	75	0.0%	0
startGame	62	0.0%	0
startRounds	93	0.0%	0
switchTurns	49	0.0%	0
Game.vdmpp		0.0%	0

### 3 Player

```

class Player
types
    public String = seq of char;

instance variables
    private name: String;
    private wins: nat;
    private losses: nat;
    private ownBoard: [Board] := nil;
    private enemyBoard: [Board] := nil;
    private myShips: set of Board`CellContent := {}; -- ships placed and my ships during game (
        decrease)
    private enemiesShips: set of Board`CellContent := {}; -- enemies ships destroyed (increase)

    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;

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

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;

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

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

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

public registerAttack: char * nat1 ==> Board`CellContent
registerAttack(colCh, line) == (
    dcl shipHit : Board`CellContent := ownBoard.registerMove(colCh, line);
    if ownBoard.countCellType(shipHit) = 0 then (
        myShips := myShips \ {shipHit};
        if card myShips = 0 then losses := losses + 1;
        return shipHit;
    )
    else if shipHit <> <Empty> then return <Hit>
    else return <Miss>;
);

public registerResult: Board`CellContent * char * nat1 ==> bool
registerResult(code, colCh, line) == (
    if code = <Miss> then enemyBoard.setComponentCol(<Miss>, line, colCh)
    else(
        enemyBoard.setComponentCol(<Hit>, line, colCh);
        if code <> <Hit> then enemiesShips := enemiesShips union {code};

```





registerResult	72	0.0%	0
shipPlacement	36	0.0%	0
startRounds	46	0.0%	0
Player.vdmpp		0.0%	0

## 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();
    tb.run();

  );
end TestBase

```

Function or operation	Line	Coverage	Calls
AssertTrue	4	0.0%	0
assertEquals	9	100.0%	5
assertTrue	5	0.0%	0
runAllTests	9	100.0%	1
TestBase.vdmpp		80.0%	6

## 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`assertEquals(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')));
);

public run: () ==> ()
run() == (
  testCreateBoard();
  testColMap();
);
end TestBoard

```

Function or operation	Line	Coverage	Calls
run	9	100.0%	1
testColMap	11	100.0%	1
testCreateBoard	4	100.0%	1
TestBoard.vdmpp		100.0%	3