battleship

December 7, 2012

Contents

1 Board

```
class Board
 types
  public Houses = set of House;
 values
  public static BOARD_SIZE: int = 10;

  --Values returned by hit()
  public static MISS : int = 0; -- hit water
  public static HIT : int = 1; -- hit a ship
  public static SHIP_SUNK : int = 2; -- hit and sink a ship

 instance variables

  -- Boards can be of two types:
  --  playable => registers the Players shots on the enemy's board.
  --  not playable (own board) => registers the enemy's shots at the player's board.
  public playable: bool;
  -- Set of Houses that represent the board.
  public houses: Houses := {};

  inv card(houses) >= 1;

 operations

  public Board: bool ==> Board
   Board(type) ==
   (
```

```
    playable := type;

    -- Generates Houses for the Board
    for all x in set {1, ..., BOARD_SIZE}  do (
     for all y in set {1, ..., BOARD_SIZE} do (
      houses := houses union {new House(x,y)};
     )
    );

    -- Sets the board reference in each house to self
    for all h1 in set houses do h1.setBoard(self);
   )
  post card(houses) = BOARD_SIZE * BOARD_SIZE;


  -- returns:
  --  MISS
  --  HIT
  --  SHIP_SUNK
  public hit : seq of int ==> int
   hit(coords) ==
    (
     -- for the house h in the set houses (the board's houses) such that x and y coordinates
     --  of coords is equal to x and y coordinates of h
     let h in set houses be st (h.x = coords(House`X) and h.y = coords(House`Y)) in (

      -- Didnt hit a ship or already hit this house
      if not h.hasShip or h.hit then return MISS;

      -- registers the hit
      h.ship.inc();
      h.hit := true;

      if h.ship.isDown() then return SHIP_SUNK
      else return HIT;
     );
    )
   pre House`checkCoords(coords)
   post RESULT in set {MISS, HIT, SHIP_SUNK};

  -- Replicates the result res of the shot the player made
  public mark : seq of int * int ==> ()
   mark(coords, res) ==
    (
     let h in set houses be st h.x = coords(House`X) and h.y = coords(House`Y) in (
      h.hit := true;
      if res = HIT or res = SHIP_SUNK then
       h.hasShip := true;
     );
    )
   pre House`checkCoords(coords);
end Board
```

Function or operation	Coverage	Calls
Board	100.0%	17
hit	100.0%	119
mark	100.0%	119
Board.vdmpp	100.0%	255

2 CLI

```
class CLI
 types
 values
 instance variables
  public static letters : map nat1 to char := {
   1|->'A', 2|->'B', 3|->'C', 4|->'D', 5|->'E',
   6|->'F', 7|->'G', 8|->'H', 9|->'I',10|->'J'
  };

  public static WATER   : int := 0;
  public static SHIP    : int := 1;
  public static HIT_SHIP  : int := 2;
  public static HIT_WATER : int := 3;

 public static markers : map int to char := {
  WATER   |-> ' ',
  SHIP    |-> 'O',
  HIT_SHIP |-> 'X',
  HIT_WATER |-> '+'
 };

 operations

  public static printInit : Player * Player ==> ()
   printInit (p1, p2) == (

    IO'println("\n**********************************************");
    IO'println(  "*           Battleship War Game           *");
    IO'println(  "**********************************************");

    IO'println(IO'freadval[VDMUtil'String]("res/header.txt").#2);

    IO'print(  "\n************ ");
    IO'print(p1.name);
    IO'print(" Vs ");
    IO'print(p2.name);
    IO'println(" ************\n");

   );

  public static printEnd : Player ==> ()
   printEnd (winner) == (
    IO'print("Player ");
    IO'print(winner.name);
    IO'println(" is Victorious! You fought with bravery young lad! ");
    IO'println("I shall award you with cookies and milk. Enjoy this feast to your content!\n");
    IO'println(IO'freadval[VDMUtil'String]("res/cookies_and_milk.txt").#2);
   );

  public static printHeaders : Player ==> ()
   printHeaders(player) == (
    IO'print("*** ");
    IO'print(player.name);
    IO'println("'s boards ***\n");
   );

  public static printBoard : Board ==> ()
   printBoard(board) == (
    dcl marker : int := WATER;

    IO'print("   ");
```

```
   for x = 1 to Board'BOARD_SIZE do (
    IO'print(letters(x));
    IO'print(" ");
   );
   IO'println(" ");

   for y = 1 to Board'BOARD_SIZE do (
    if y <> 10 then IO'print(" ");
    IO'print(y);
    IO'print("|");
    for x = 1 to Board'BOARD_SIZE do (
     let house in set board.houses be st house.x = x and house.y = y in (

      if house.hasShip then (
       if house.hit then
        marker := HIT_SHIP
       else
        marker := SHIP;
      )
      else (
       if house.hit then (
        marker := HIT_WATER;
       )
       else (
        marker := WATER;
       )
      );

      IO'print(markers(marker));
      IO'print(" ");
     )
    );
    IO'print("|");
    IO'println(y);
   );

   IO'print("    ");
   for x = 1 to Board'BOARD_SIZE do (
    IO'print(letters(x));
    IO'print(" ");
   );
   IO'println(" ");
   IO'println(" ");
  );

 public static printBoardTabbed : Board ==> ()
  printBoardTabbed(board) == (
   dcl marker : int := WATER;

   IO'print("    ");
   IO'print("            ");
   for x = 1 to Board'BOARD_SIZE do (
    IO'print(letters(x));
    IO'print(" ");
   );
   IO'println(" ");

   for y = 1 to Board'BOARD_SIZE do (
    IO'print("            ");
    if y <> 10 then IO'print(" ");
    IO'print(y);
    IO'print("|");
    for x = 1 to Board'BOARD_SIZE do (
     let house in set board.houses be st house.x = x and house.y = y in (
```

4

```
        if house.hasShip then (
         if house.hit then
          marker := HIT_SHIP
         else
          marker := SHIP;
        )
        elseif house.hit then marker := HIT_WATER;

        IO`print(markers(marker));
        IO`print(" ");
        marker := WATER;
       )
      );
      IO`print("|");
      IO`println(y);
     );

     IO`print("   ");
     IO`print("          ");
     for x = 1 to Board`BOARD_SIZE do (
      IO`print(letters(x));
      IO`print(" ");
     );
     IO`println(" ");
     IO`println(" ");
   );

  public static printBoards : Player ==> ()
   printBoards(player) == (
    printHeaders(player);
    printBoard(player.boardplay);
    printBoardTabbed(player.boardown);
   );

end CLI
```

Function or operation	Coverage	Calls
printBoard	97.5%	2
printBoardTabbed	97.7%	2
printBoards	100.0%	2
printEnd	100.0%	1
printHeaders	100.0%	2
printInit	100.0%	1
CLI.vdmpp	98.4%	10

3 GameEngine

```
class GameEngine
 types
 values
  public static PLAYER1 : int = 1;
  public static PLAYER2 : int = 2;

 instance variables
```

```
  public player1: Player;
  public player2: Player;
  public board1own : Board; -- Board of Player 1
  public board1play : Board; -- Board of Player 1 that represents the Player 2's Board
  public board2own : Board; -- Board of Player 2
  public board2play : Board; -- Board of Player 2 that represents the Player 1's Board
  public activePlayer: int := PLAYER1; -- Current Player taking a shot
  public isGameOver : bool := false;

  inv activePlayer in set {PLAYER1, PLAYER2};

operations
  public GameEngine : VDMUtil`String * bool * VDMUtil`String * bool ==> GameEngine
   GameEngine (player1name, isbot1, player2name, isbot2) ==
   (
    player1 := new Player(player1name, 1, isbot1);
    player2 := new Player(player2name, 2, isbot2);

    board1own := player1.boardown;
    board2own := player2.boardown;
    board1play := player1.boardplay;
    board2play := player2.boardplay;
   );

  public turn : () ==> int
   turn() == (

     if activePlayer = PLAYER1 then (
      let coords = player1.play() in
       (
        board1play.mark(coords, board2own.hit(coords));
        isGameOver := checkVictory(player2.ships);

        if isGameOver then return activePlayer;
        activePlayer := PLAYER2;
       )
     )
     elseif activePlayer = PLAYER2 then (
      let coords2 = player2.play() in
       (
        board2play.mark(coords2, board1own.hit(coords2));
        isGameOver := checkVictory(player1.ships);
        if isGameOver then return activePlayer;
        activePlayer := PLAYER1;
       )
     );

     return 0;
   );

  public checkVictory: seq of Ship ==> bool
   checkVictory (ships) ==
   (
    return forall s in set elems ships & s.isDown();
   );

  public getPlayer : int ==> Player
   getPlayer(nr) ==
    let player in set {player1, player2} be st player.playerNumber = nr in (
      return player;
    );

  public getAllBoards : () ==> set of Board
   getAllBoards() ==
    return { board1own, board2own, board1play, board2play};
```

6

```
  -- start the game
  public let_the_slaughter_begin : bool ==> int
   let_the_slaughter_begin (debug) ==
   (
    dcl victorious : int := 0, loser : int := 0;

    if debug then CLI`printInit(player1, player2);

    while not isGameOver do
    (
     victorious := turn();
    );

    if victorious = PLAYER1 then loser := PLAYER2
    else loser := PLAYER1;

    if debug then (
     CLI`printBoards(getPlayer(victorious));
     CLI`printBoards(getPlayer(loser));

     CLI`printEnd(getPlayer(victorious));
    );

    return victorious;
   );

end GameEngine
```

Function or operation	Coverage	Calls
GameEngine	100.0%	4
checkVictory	100.0%	119
getAllBoards	100.0%	1
getPlayer	100.0%	3
let_the_slaughter_begin	100.0%	2
turn	100.0%	119
GameEngine.vdmpp	100.0%	248

4 House

```
class House
 types
  public coords = seq of int;
 values
  public static X : int = 1;
  public static Y : int = 2;
 instance variables

 public x: int;
 public y: int;
 public hit: bool := false;
 public hasShip: bool := false;
 public ship: Ship;
 public board: Board;
```

```
 inv checkCoords([x] ^ [y]);

 operations
  -- Constructor
 public House : int * int ==> House
  House (x1, y1) ==
   (
    x := x1;
    y := y1;
   )
 pre checkCoords([x1] ^ [y1]);

 public setBoard : Board ==> ()
  setBoard(b) ==
   (
    board := b;
   )
 pre is_Board(b);

 functions

  -- Coordinates restriction: x and y must be between 1 and board size.
 public static checkCoords : coords -> bool
  checkCoords(coords) ==
   coords(X) >= 1 and coords(X) <= Board'BOARD_SIZE and
   coords(Y) >= 1 and coords(Y) <= Board'BOARD_SIZE;

end House
```

Function or operation	Coverage	Calls
House	100.0%	1700
checkCoords	100.0%	30840
setBoard	100.0%	1700
House.vdmpp	100.0%	34240

5 Player

```
class Player
 types

 values

 instance variables
  -- Each element represents the size of a ship. (helper to generate the ships)
  -- 4 ships with size 2
  -- 3 ships with size 3
  -- 2 ships with size 4
  -- 1 ship  with size 5
 public static sizes : seq of int := [2, 2, 2, 2, 3, 3, 3, 4, 4, 5];

 public it : int := 0;
 public name : VDMUtil'String;
 public ships : seq of Ship;
 public boarddown : Board;
 public boardplay : Board;
 public playerNumber : int;
```

```
public coords2play : seq of House`coords;
public isBot : bool; -- Player can be a bot (takes sequencial shots in all houses)

inv (len ships) <= len sizes;

operations

public Player : VDMUtil`String * int * bool ==> Player
Player(n, number, isbot) ==
(
 -- helpers to generate the ships
 dcl c : seq of House`coords, orientation: seq of int,
  ship : Ship;

 name := n;
 playerNumber := number;
 isBot := isbot;
 boardown := new Board(false);
 boardplay := new Board(true);

 c := getShipsCoords();
 orientation := getShipsOrientations();

 if not isBot then coords2play := getCoords2Play();

 ships := [];

 for i = 1 to (len sizes) do (
  ship := new Ship(c(i),orientation(i),sizes(i),boardown);
  ships := ships ^ [ship];
 );
);

-- Reads the ships initial coordinates from the file "res/<player_name>.shipscoords"
public getShipsCoords: ()  ==> seq of House`coords
 getShipsCoords() ==
 (
  return IO`freadval[seq of House`coords]("res/" ^ name ^ ".shipscoords").#2;
 )
 post forall coord in set elems RESULT &
   House`checkCoords(coord);

-- Reads the ships orientations from the file "res/<player_name>.orientations"
public getShipsOrientations: () ==> seq of int
 getShipsOrientations() ==
 (
  return IO`freadval[seq of int]("res/" ^ name ^ ".orientations").#2;
 )
 post forall orientation in set elems RESULT &
  Ship`checkOrientation(orientation);

-- Reads the shots the player will make from the file "res/<player_name>.coords2play"
public getCoords2Play: ()  ==> seq of House`coords
 getCoords2Play() ==
 (
  return IO`freadval[seq of House`coords]("res/" ^ name ^ ".coords2play").#2;
 )
 post forall coord in set elems RESULT &
   House`checkCoords(coord);

-- Takes a shot
public play : () ==> House`coords
 play () ==
  (
   dcl coords : House`coords;
```

9

```
    if isBot then coords := bot_genCoords()
    else (
     coords := coords2play(1); -- gets the shot to make
     coords2play := tl coords2play -- removes shot from the available shots to make
    );

    return coords;
   );

  -- Shot generator for the bot.
 public bot_genCoords : () ==> House'coords
  bot_genCoords() ==
  (
   dcl x : int := it - (it div 10) * 10 +1,
    y : int := (it div 10) +1;

   it := it + 1;

   return [x] ^ [y];
  )
  pre it <= 100
  post House'checkCoords(RESULT);

end Player
```

Function or operation	Coverage	Calls
Player	100.0%	8
bot_genCoords	100.0%	59
getCoords2Play	100.0%	2
getShipsCoords	100.0%	8
getShipsOrientations	100.0%	8
play	100.0%	119
Player.vdmpp	100.0%	204

6 Ship

```
class Ship
 types

 values
  -- Orientations of the ships
 public static ORIENTATION_UP : int = 0;
 public static ORIENTATION_RIGHT : int = 1;
 public static ORIENTATION_DOWN : int = 2;
 public static ORIENTATION_LEFT : int = 3;

  -- Mapping of the orientations to the increments to apply to get all the
  -- coordinates of a ship.
 static orientations : map int to (seq of int) = {
  ORIENTATION_UP    |-> [ 0,-1],
  ORIENTATION_RIGHT |-> [+1, 0],
  ORIENTATION_DOWN  |-> [ 0,+1],
  ORIENTATION_LEFT  |-> [-1, 0]
 };
```

```
instance variables

 public static id : int := 0;
 public coord_init : House'coords; -- the first house of the ship
 public coords : set of House'coords := {};
 public orientation : int := 1;
 public hits : int := 0; -- number of hits that the ship as received (in different houses)
 public size: int := 1; -- number of houses that the ship fills
 public board: Board; -- board to which the ship belongs to
 public my_id: int;

 inv checkOrientation(orientation);
 inv len coord_init = 2;
 inv id >= 0;
 inv card(coords) >= 0 and card(coords) <= size;

operations

public Ship: House'coords * int * int * Board ==> Ship
Ship(c,o,s,b) ==
(
 coord_init := c;
 size := s;
 coords := {c};
 orientation := o;
 board := b;
 my_id := id +1;
 id := my_id;

 -- fills the coordinates of the ship given the initial coordinate and the orientation
 for i = 1 to size-1 do (
  coords := coords union {
     [c(House'X)+orientations(o)(House'X)*i,
       c(House'Y)+orientations(o)(House'Y)*i]
      };
 );

)
pre forall x in set {c(House'X),c(House'X)+orientations(o)(House'X)*(s-1)},
    y in set {c(House'Y),c(House'Y)+orientations(o)(House'Y)*(s-1)} &
  House'checkCoords([x] ^ [y])
post fill_houses();

public fill_houses: ()==> bool
 fill_houses() ==
 (
  for all c in set coords do
  (
   let h in set board.houses be st h.x = c(House'X) and
            h.y = c(House'Y) in
   (
    h.hasShip := true;
    h.ship := self;
   )
  );
  return true;
 )
pre forall c in set coords &
 let h in set board.houses be st h.x = c(House'X) and
          h.y = c(House'Y) in
 (
  not h.hasShip
 );
```

```
 public inc: () ==> ()
  inc() == hits := hits +1
 pre hits < size
 post hits <= size;

 public isDown : () ==> bool
  isDown() == return size = hits;

 functions
  public static checkOrientation : int -> bool
   checkOrientation(orientation) ==
    orientation in set {ORIENTATION_UP, ORIENTATION_RIGHT, ORIENTATION_DOWN, ORIENTATION_LEFT};
end Ship
```

Function or operation	Coverage	Calls
Ship	100.0%	80
checkOrientation	100.0%	168
fill_houses	100.0%	80
inc	100.0%	72
isDown	100.0%	383
Ship.vdmpp	100.0%	783

7 Test

```
class Test
 operations

  --Utils
    public static assertTrue : bool ==> ()
        assertTrue(expectedTrue) == return
    pre expectedTrue;

    public static runAllTests : () ==> ()
       runAllTests () == (
        testFullGame1();
        testFullGame2();

        testBoardHousesNumber();
        testCheckCoords();
        testNoOverlapedShips();
       );

       -- #### Full Tests (100% coverage)
       public static testFullGame1 : () ==> ()
        testFullGame1() == (
   dcl ge : GameEngine := new GameEngine("ze", true, "pedro", false),
    victorious : int := ge.let_the_slaughter_begin(false);

   assertTrue(ge.isGameOver);
   assertTrue(victorious = ge.PLAYER2);
  );

       public static testFullGame2 : () ==> ()
        testFullGame2() == (
   dcl ge : GameEngine := new GameEngine("ze", false, "pedro", true),
```

```
      victorious : int := ge.let_the_slaughter_begin(true);

    assertTrue(ge.isGameOver);
    assertTrue(victorious = ge.PLAYER1);
   );


  -- #### Unit Testing

  -- Number of Houses of a Board is exactly BOARD_SIZE * BOARD_SIZE
  public static testBoardHousesNumber : () ==> ()
   testBoardHousesNumber() == (
    dcl board : Board := new Board(false);

    assertTrue(card(board.houses) = Board`BOARD_SIZE * Board`BOARD_SIZE);
   );

  -- Every coords instanciated in the game are between 1 and BOARD_SIZE
  public static testCheckCoords : () ==> ()
   testCheckCoords() == (
    dcl ge : GameEngine := new GameEngine("ze", true, "pedro", true),
     boards : set of Board := ge.getAllBoards();

    assertTrue(
     forall board in set boards &
      forall house in set board.houses &
       House`checkCoords([house.x] ^ [house.y])
      and
      forall ship in set elems (ge.player1.ships ^ ge.player2.ships) &
       forall coord in set ship.coords &
        House`checkCoords([coord(House`X)] ^ [coord(House`Y)])
    );
   );

  public static testNoOverlapedShips : () ==> ()
   testNoOverlapedShips() == (
    dcl ge : GameEngine := new GameEngine("ze", true, "pedro", true),
     coords1 : seq of House`coords := [], coords2 : seq of House`coords := [];

     for all ship in set elems ge.player1.ships do
      coords1 := coords1 ^ VDMUtil`set2seq[House`coords](ship.coords);

     for all ship in set elems ge.player2.ships do
      coords2 := coords2 ^ VDMUtil`set2seq[House`coords](ship.coords);

    assertTrue(
     len coords1 = card elems coords1 and
     len coords2 = card elems coords2
    );
   );

end Test
```

Function or operation	Coverage	Calls
assertTrue	100.0%	7
runAllTests	100.0%	1
testBoardHousesNumber	100.0%	1
testCheckCoords	100.0%	1
testFullGame1	100.0%	1

testFullGame2	100.0%	1
testNoOverlapedShips	100.0%	1
Test.vdmpp	100.0%	13