battleship

December 7, 2012

Contents

1 Board

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

  public static MISS : int = 0;
  public static HIT : int = 1;
  public static SHIP_SUNK : int = 2;

 instance variables

  public playable: bool;
  public houses: Houses := {};

  inv card(houses) >= 1;

 operations

  public Board: bool ==> Board
   Board(t) ==
   (
    playable := t;

    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)};
```

```
     )
    );

    for all h1 in set houses do h1.setBoard(self);
   )
  post card(houses) = BOARD_SIZE * BOARD_SIZE;

  -- returns:
  --0 => miss or already hit
  --1 => hit
  --2 => ship sunk
  public hit : seq of int ==> int
   hit(coords) ==
   (
    let h in set houses be st (h.x = coords(House`X) and h.y = coords(House`Y)) in (
     if not h.hasShip or h.hit then return MISS;

     h.ship.inc();
     h.hit := true;
     if not h.ship.isDown() then return HIT
     else return SHIP_SUNK;
    );
   )
   pre House`checkCoords(coords)
   post RESULT in set {MISS, HIT, SHIP_SUNK};

  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%	8
hit	100.0%	119
mark	100.0%	119
Board.vdmpp	100.0%	246

2 CLI

```
class CLI
 types
 values
 instance variables

 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[VDMUtils`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[VDMUtils`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(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(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(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 (

      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(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	97.4%	4
printBoardTabbed	97.6%	4
printBoards	100.0%	4
printEnd	100.0%	2
printHeaders	100.0%	4
printInit	100.0%	2
CLI.vdmpp	98.2%	20

3 GameEngine

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

 instance variables
  public p1: Player;
  public p2: Player;
  public board1own : Board;
  public board1play : Board;
  public board2own : Board;
  public board2play : Board;
  public activePlayer: int := PLAYER1;
  public win : bool := false;

  inv activePlayer in set {PLAYER1, PLAYER2};

 operations
  public GameEngine : VDMUtils`String * bool * VDMUtils`String * bool ==> GameEngine
   GameEngine (player1name, isbot1, player2name, isbot2) ==
   (
    p1 := new Player(player1name, 1, isbot1);
    p2 := new Player(player2name, 2, isbot2);
```

```
   board1own := p1.boardown;
   board2own := p2.boardown;
   board1play := p1.boardplay;
   board2play := p2.boardplay;
  );

 public let_the_slaughter_begin : bool ==> int
  let_the_slaughter_begin (debug) ==
  (
   dcl victorious : int := 0, loser : int := 0;

   if debug then CLI`printInit(p1, p2);

   while not win 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;
  );

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

   if activePlayer = PLAYER1 then (
    let coords = p1.play() in
    (
     board1play.mark(coords, board2own.hit(coords));
     win := checkVictory(p2.ships);

     if win then return activePlayer;
     activePlayer := PLAYER2;
    )
   )
   elseif activePlayer = PLAYER2 then (
    let coords2 = p2.play() in
    (
     board2play.mark(coords2, board1own.hit(coords2));
     win := checkVictory(p1.ships);
     if win 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 {p1, p2} be st player.playerNumber = nr in (
```

```
      return player;
    );


end GameEngine
```

Function or operation	Coverage	Calls
GameEngine	100.0%	2
checkVictory	100.0%	119
getPlayer	100.0%	6
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;

 -- Coordinates restriction
 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
  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%	800
checkCoords	100.0%	3380
setBoard	100.0%	800
House.vdmpp	100.0%	4980

5 Player

```
class Player
 types

 values

 instance variables
  public static sizes : seq of int := [2, 2, 2, 2, 3, 3, 3, 4, 4, 5];

  public it : int := 0;
  public name : VDMUtils`String;
  public ships : seq of Ship;
  public boardown : Board;
  public boardplay : Board;
  public playerNumber : int;
  public coords2play : seq of House`coords;
  public isBot : bool;

  inv (len ships) <= len sizes;

 operations

  public Player : VDMUtils`String * int * bool ==> Player
  Player(n, number, isbot) ==
  (
   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];
   );
  );
```

```
  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);

  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);

  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);

  public play : () ==> House`coords
   play () ==
   (
    dcl coords : House`coords;

    if isBot then coords := bot_genCoords()
    else (
     coords := coords2play(1);
     coords2play := tl coords2play
    );

    return coords;
   );

  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%	4
bot_genCoords	100.0%	59
getCoords2Play	100.0%	2
getShipsCoords	100.0%	4

getShipsOrientations	100.0%	4
play	100.0%	119
Player.vdmpp	100.0%	192

6 Ship

```
class Ship
 types

 values
  public static ORIENTATION_UP : int = 0;
  public static ORIENTATION_RIGHT : int = 1;
  public static ORIENTATION_DOWN : int = 2;
  public static ORIENTATION_LEFT : int = 3;

  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;
  public coords : set of House`coords := {};
  public orientation : int := 1;
  public hits : int := 0;
  public size: int := 1;
  public board: Board;
  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;

  for i = 1 to size-1 do (
   coords := coords union {[c(1)+orientations(o)(1)*i,c(2)+orientations(o)(2)*i]};
  );

 )
 pre forall x in set {c(1),c(1)+orientations(o)(1)*(s-1)}, y in set {c(2),c(2)+orientations(o)(2)*(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(1) and h.y = c(2) 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(1) and h.y = c(2) 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%	40
checkOrientation	100.0%	120
fill_houses	100.0%	40
inc	100.0%	72
isDown	100.0%	383
Ship.vdmpp	100.0%	655

7 Test

```
class Test
 operations

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

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

```
        );

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

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

end Test
```

Function or operation	Coverage	Calls
assertTrue	100.0%	4
runAllTests	100.0%	1
testFullGame1	100.0%	1
testFullGame2	100.0%	1
Test.vdmpp	100.0%	7