Assignment 5 Specification

Tasnim Bari Noshin (noshint)

April 5, 2017

The purpose of this software design exercise is to design, specify, implement and test a module for storing the state of an Battelship game. The game board is represented as a two dimensional sequence, with the first dimension the row and the second dimension the column. The indexes are relative to the upper left hand corner of the board; that is, row 0 and column 0 are at the top left.

Constants Module

Module

Constants

Uses

N/A

Syntax

Exported Constants

MAX_BOARD_ROWS = 9
MAX_BOARD_COLUMN = 11
SHIP_NUMBER = 5

Semantics

State Variables

none

Point ADT Module

Template Module

pointADT

Uses

N/A

Syntax

Exported Types

PointT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new PointT	integer, integer	PointT	
xcrd		integer	
ycrd		integer	

Semantics

State Variables

xc: integer
yc: integer

State Invariant

None

Assumptions

None

Access Routine Semantics

new PointT (x, y):

- transition: $xc, yc := x, y$
- output: $out := self$
- exception: none

xcrd:

- output: $out := xc$
- exception: none

ycrd:

- output: $out := yc$
- exception: none

Board ADT Module

Template Module

BoardADT

Uses

Constants, PointADT

Syntax

Exported Types

BoardT = ?
PointT = { FREE, SHIP, MISS, HIT }

Exported Access Programs

Routine name	In	Out	Exceptions
new BoardT		BoardT	
placeShip	integer, PointT, PointT	sequence of PointT	OutOfBoundsException, InvalidMoveException, InvalidShipException
shot	PointT, BoardT		OutOfBoundsException, InvalidMoveException
getTraceShot		sequence of PointT	
getBoard		sequence of sequence of PointT	
getShipList		sequence of sequence of PointT	
addToShipList	sequence of PointT		
hasSunk	sequence of PointT		
isLosing		boolean	InvalidShipException
turn		boolean	
chances		real	

Semantics

State Variables

traceShot : sequence of PointT
board : sequence of sequence of PointT
shipList : sequence of sequence of PointT
shipSunk : integer
switchTurn : boolean
possible_ships = { 2, 3, 3, 4, 5 }

State Invariant

| board | = MAX_BOARD_ROWS
| board [0 ... |board| -1] | = MAX_BOARD_COLUMNS
| shipList | ≤ SHIP_NUMBER

Assumptions

The constructor BoardT is called for each abstract object before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

boardT():

- transition:

$$
board := < \begin{array}{c}
< \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\
< \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\
< \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\
< \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\
< \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\
< \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\
< \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\
< \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\
< \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} >
\end{array} >
$$

- output: *out* := *self*

- exception: *exc* := *none*

placeShip($size$, $start$, end):

- transition: $board$:= (is_valid_ship($size, start, end,$ board) = 1 $\Rightarrow$ ($\forall(i : \mathbb{N}|(i \leq$ end.ycrd() $\Rightarrow$ (board[beg.xcrd()][(beg.ycrd() + i)] = SHIP))))|($\forall(i : \mathbb{N}|(i \leq$ end.xcrd() $\Rightarrow$ board[beg.xcrd() + i][beg.ycrd()] = SHIP))))
- output: $tempShip$:= (is_valid_ship($size, start, end,$ board) = 1 $\Rightarrow$ ($\forall(i : \mathbb{N}|(i \leq$ end.ycrd() $\Rightarrow$ tempShip||(board[beg.xcrd(), (beg.ycrd() + i)]))))|($\forall(i : \mathbb{N}|(i \leq$ end.xcrd() $\Rightarrow$ tempShip||(board[beg.xcrd()][beg.ycrd() + i])))))
- exception exc := (is_valid_ship($size, start, end, board$) $\Rightarrow$ InvalidMove, OutOfBoundsException, Inval

shotTaken($move$, $opBoard$):

- transition: $opBoard$:= (is_valid_move($move, board$)$\wedge$(opBoard[$move$.xcrd()][$move$.ycrd()] = SHIP $\Rightarrow$ (opBoard[$move$.xcrd()][$move$.ycrd()] = HIT, $opBoard$.getShipList()[0..$move$.xcrd()$-$1][0..$move$.ycrd()$-$1]||$opBoard$.getShipList()[$move$.xcrd()+1..|$shipList$|$-$1][$move$.ycrd()+1..|$move$.xcrd()| $-$ 1]|opBoard[$move$.xcrd()][$move$.ycrd()] = MISS))),
  $switch$:= $\neg(switchTurn)$
  $traceShot$:= (is_valid_move($move, board$) $\Rightarrow$ ($traceShot$||$move$))
- exception exc := (is_valid_move($move, board$) $\Rightarrow$ InvalidMove, OutOfBoundsException)

getTraceShot():

- output out := $traceShot$
- exception exc := $none$

getBoard():

- output out := $board$
- exception exc := $none$

addToShipList($ship$):

- transition $shipList$:= $shipList$||$ship$
- exception exc := $none$

getShipList():

- output out := $shipList$

- exception $exc := none$

hasSunk($ship$):

- transition $shipSunk := (ship \in \text{shipList} \Rightarrow |ship| = 0 \Rightarrow shipSunk + 1)$
- exception $exc := none$

turn():

- output $out := switchTurn$
- exception $exc := none$

isLosing():

- output $out := (shipSunk = \text{SHIP\_NUMBER} \Rightarrow True)$
- exception $exc := (shipSunk > \text{SHIP\_NUMBER} \Rightarrow InvalidShip))$

chances():

- output $out := ((shipSunk/\text{SHIP\_NUMBER}) * 100)$
- exception $exc := none$

Local Types

board = sequence [MAX_BOARD_ROWS, MAX_BOARD_COLUMNS] of PointT
tempShip = sequence of PointT

Local Functions

is_valid_ship : integer × PointT × Point × boardT → integer

is_valid_ship($size, start, end, board$) ≡ (($size \notin$ possible_ships ⇒ InvalidShip) ∨ ((end.ycrd() > MAX_BOARD_COLUMNS∨$start$.ycrd() > MAX_BOARD_COLUMNS) ⇒ OutOfBoundsException) ∨ ((end.xcrd() > MAX_BOARD_ROWS ∨ $start$.xcrd() > MAX_BOARD_ROWS) ⇒ OutOfBoundsException) ∨ (($end \in$ shipList ∨ $start \in$ shipList) ⇒ InvalidMove)∨((end.xcrd() ≠ $start$.xcrd())∨(end.ycrd() ≠ $start$.ycrd()) ⇒

InvalidShip) $\vee$ (end.xcrd() = $start$.xcrd() $\Rightarrow$ 1|2))

is_valid_move : Point $\times$ boardT $\rightarrow$ boolean

is_valid_move($move, board$) $\equiv$ (((end.ycrd() > MAX_BOARD_COLUMNS$\vee$$start$.ycrd() > MAX_BOARD_COLUMNS) $\Rightarrow$ OutOfBoundsException)$\vee$((end.xcrd() > MAX_BOARD_ROWS$\vee$ $start$.xcrd() > MAX_BOARD_ROWS) $\Rightarrow$ OutOfBoundsException)$\vee$(($end \in$ board$\vee$ $start \in$ board) $\Rightarrow$ InvalidMove) $\vee$ $True$)