Assignment 4 - 2AA4

Mohamed Bengezi bengezim

April 5, 2017

The contents of this report include the specifications of the game state for a game a Battleship.

Battleship Module

Module

Battleship

Uses

PointT, ShipT, GameStateT

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
init	sequence[5] of ShipT sequence[5] of ShipT		
all_shots		sequence of PointT	
place_shot	PointT	boolean	InvalidShotException
is_winner		boolean	

Semantics

State Variables

shots_taken: sequence of PointT
firstPlayerTurn: boolean
first_player: GameStateT for the first player
second_player: GameStateT for the second player

State Invariant

None

Assumptions

The init method is called for the abstract object before any other access routine is called for that object. The init method can be used to return the state of the game to the state of a new game.

Access Routine Semantics

init(*firstPlayerShipList*, *secondPlayerShipList*):

- transition: $shots\_taken, firstPlayerTurn, first\_player, second\_player := <>, true$, new GameStateT(firstPlayerShipList), new GameStateT(secondPlayerShipList)
- exception: None

all_shots():

- output: $out := shots\_taken$
- exception: None

place_shot(p):

- transition-output: $firstPlayerTurn, out := \neg firstPlayerTurn$ and $shots\_taken$ such that $shots\_taken = pre(shots\_taken)[0..|pre(shots\_taken)|-1]||<p>, (firstPlayerTurn \Rightarrow second\_player.has\_been\_shot(p) | \neg firstPlayerTurn \Rightarrow first\_player.has\_been\_shot(p))$
- exception: $exc := (firstPlayerTurn \Rightarrow \exists(i : \mathbb{I} | (i\%2 = 0) \wedge (0 \leq i < |shots\_taken|) : repeatedShot(p, shots\_taken[i])) | \neg firstPlayerTurn \Rightarrow \exists(i : \mathbb{I} | (i\%2 = 1) \wedge (1 \leq i < |shots\_taken|) : repeatedShot(p, shots\_taken[i]))) \Rightarrow$ InvalidShotException

is_winner():

- output: $out := (firstPlayerTurn \Rightarrow second_player.all_ships_lost() | \neg firstPlayerTurn \Rightarrow first_player.all_ships_lost())$
- exception: None

Local Functions

repeatedShot : PointT $\times$ PointT $\rightarrow$ boolean

repeatedShot$(p, q) \equiv (p.xcrd() = q.xcrd()) \wedge (p.ycrd() = q.ycrd())$

Game State Module

Template Module

GameStateT

Uses

ShipT, PointT

Syntax

Exported Types

GameStateT = ?

Exported Constants

MAX_ROWS = 10
MAX_COLUMNS = 10

Exported Access Programs

Routine name	In	Out	Exceptions
GameStateT	sequence of ShipT	GameStateT	InvalidConfigurationException
has_been_shot	PointT	boolean	
all_ships_lost		boolean	

Semantics

State Variables

hitList: sequence <> of integer *//integer array representing the list of shots hit*
shipList: sequence <> of ShipT *//ShipT array representing the shipList*
hitIndex: Integer

State Invariant

None

Assumptions

The GameStateT() constructor is called for each abstract object before any other access routine is called for that object. The constructor can only be called once.

Access Routine Semantics

GameStateT($shipList$):

- transition: $shipList, hitList := shipList, <>$
- output: $out := self$
- exception:

$$exc := (\neg(|shipList| = 5) \vee \neg(shipList[0].get\_length() = 2) \vee \neg(shipList[1].get\_length() = 3) \vee \neg(shipList[2].get\_length() = 3) \vee \neg(shipList[3].get\_length() = 4) \vee \neg(shipList[4].get\_length() = 5) \vee (\exists(i : \mathbb{I} | 0 \leq i < |shipList| : \exists(j : \mathbb{I} | (0 \leq j < |shipList|) \wedge (i \neq j) : \text{conflict}(shipList[i], shipList[j])))))) \Rightarrow \text{InvalidConfigurationException}$$

has_been_shot(p):

- transition-output: $hitList, out := (checkShot(shipList, p) \Rightarrow hitList[0..hitIndex - 1]|| < pre(hitList)[hitIndex] + 1 > ||hitList[hitIndex + 1..|hitList| - 1] where\ pointInLine(p, shipList[i].get\_head(), shipList[i].get\_tail()) = true | \neg checkShot(shipList, p) \Rightarrow hitList) checkShot(shipList, p)$
- exception: None

all_ships_lost():

- output: $out := \forall(i : \mathbb{I} | 0 \leq i < |shipList| : shipList[i].\text{get\_length}() = hitList[i])$
- exception: None

Local Functions

pointInLine : PointT × PointT × PointT → boolean

pointInLine$(p, head, tail) \equiv (head.dist(p) + tail.dist(p) = head.dist(tail))$

conflict : ShipT × ShipT → boolean

conflict$(first, second) \equiv \exists(i : PointT | pointInLine(i, first.get_head(), firstget_tail()) : pointInLine(i, second.get_head(), second.get_tail()) \rightarrow hitIndex = index_o f_i)$

checkShot : sequence of ShipT $\times$ PointT $\rightarrow$ boolean

checkShot$(shipList, p) \equiv \exists(s : ShipT | s \in shipList : pointInLine(p, s.get_head(), s.get_tail()))$

Ship Module

Template Module

ShipT

Uses

PointT

Syntax

Exported Types

ShipT = ?

Exported Constants

MAX_SIZE = 5 *//Maximum size of a ship*

MIN_SIZE = 2 *//Minimum size of a ship*

Exported Access Programs

Routine name	In	Out	Exceptions
ShipT	integer, PointT, PointT	ShipT	InvalidShipException
get_length		integer	
get_head		PointT	
get_tail		PointT	

Semantics

State Variables

head: PointT *//Start point of the boat*
tail: PointT *//End point of the boat*
length: integer *//Length of the boat*

State Invariant

Assumptions

The ShipT constructor is called for each abstract object before any other access routine is called for that object. The constructor can only be called once.

Access Routine Semantics

ShipT(l, a, b):

- transition: $length, head, tail := l, a, b$
- output: $out := self$
- exception: $exc := (\neg(MIN\_SIZE \leq length \leq \text{MAX\_SIZE}) \vee ((head.xcrd() \neq tail.xrd()) \wedge (head.ycrd() \neq tail.ycrd())) \Rightarrow \text{InvalidShipException})$

get_length():

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

get_head():

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

get_tail():

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

Point ADT Module

Template Module

PointT

Uses

N/A

Syntax

Exported Types

PointT = ?

Exported Constants

MAX_X = 10 MAX_Y = 10

Exported Access Programs

Routine name	In	Out	Exceptions
PointT	real, real	PointT	InvalidPointException
xcrd		real	
ycrd		real	
dist	PointT	real	

Semantics

State Variables

xc: real
yc: real

State Invariant

none

Assumptions

The constructor PointT 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

PointT(x, y):

- transition: $xc, yc := x, y$
- output: $out := self$
- exception $exc := ((\neg(0 \leq x \leq \text{MAX\_X}) \vee \neg(0 \leq y \leq \text{MAX\_Y})) \Rightarrow \text{InvalidPointException})$

xcrd():

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

ycrd():

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

dist(p):

- output: $out := \sqrt{(self.xc - p.xc)^2 + (self.yc - p.yc)^2}$
- exception: none