SWR ENG 2AA4
Assignment 4

AMIT BINU - binua

April 5, 2017

Constants Module

Module

Constants

Uses

N/A

Syntax

Exported Constants

MAX_X= 9 *//max length of the grid in the x-direction*
MAX_Y = 9*//max length of the grid in the y-direction*
MIN_SIZE = 2 *//minimum ship size, like specified in the specifications*
MAX_SIZE = 5 *//minimum ship size, like specified in the specifications*

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

Point ADT Module

Template Module

PointT

Uses

Constants

Syntax

Exported Types

PointT = ?

Exported Access Programs

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

Semantics

State Variables

xc: integer
yc: integer

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: $exc := ((\neg(0 \leq x \leq \text{Contants.MAX\_X}) \vee \neg(0 \leq y \leq \text{Constants.MAX\_Y})) \Rightarrow$ 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

Ship ADT Module

Template Module

ShipT

Uses

Constants, PointT

Syntax

Exported Types

ShipT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
ShipT	PointT, PointT, integer	ShipT	InvalidShipException
FirstPoint		PointT	
SecondPoint		PointT	
Size		integer	
list	integer	list	

Semantics

State Variables

$FirstPoint$: PointT
$SecondPoint$: PointT
$size$: integer
$list$: sequence of PointT

State Invariant

None

Assumptions

The constructor ShipT 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

ShipT($first, second$):

- transition: $FirstPoint, SecondPoint, size, list := first, second, first.dist(second), <>$
- output: $out := self$
- exception: $exc := ((first.dist(second) < \text{Constants.MIN\_SIZE}) \vee (first.dist(second) > \text{Constants.MAX\_SIZE}) \vee ((first.\text{xcrd}() \neq second.\text{xcrd}()) \wedge (first.\text{ycrd}() \neq second.\text{ycrd}())) \Rightarrow \text{InvalidShipException})$

firstPoint():

- output: $out := FirstPoint$
- exception: None

secondPoint():

- output: $out := SecondPoint$
- exception: None

SIZE():

- output: $out := size$
- exception: None

list(count):

- transition: $\forall(i : \mathbb{I} | first.\text{xcrd}() \leq i \leq second.\text{xcrd}() : \forall(j : \mathbb{I} | first.\text{ycrd}() \leq j \leq second.\text{ycrd}() : list[count++] := PointT(i++, j++)))$
- output: list
- exception: None

Game Status ADT Module

Template Module

GameStatusT

Uses

Constants, PointT, ShipT

Syntax

Exported Types

GameStateT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
GameStatusT	sequence of ShipT	GameStatusT	InvalidPositionException
is_hit	PointT	boolean	
is_game_over		boolean	

Semantics

State Variables

$ListOfships$: sequence of ShipT
$guess$: sequence of integer $count$: an integer

State Invariant

None

Assumptions

The GameStatusT() 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($list$):

- transition: $ListOfships, guess := list, <>$
- output: $out := self$

- exception:

$$exc := ((|list| \neq 5) \vee (\exists(i : \mathbb{I} | 0 \leq i < |list| : list[i].SIZE() < 2 \vee list[i].SIZE() > 5)) \\ \vee ShipLength(0) \neq 2 \vee \\ (\exists(i : \mathbb{I} | 0 \leq i < |list| : \exists(j : \mathbb{I} | (0 \leq j < |list|) \wedge (i \neq j) : \\ \text{collision}(list[i], list[j])))) \Rightarrow \text{InvalidPositionException}$$

is_hit(p):

- transition:

$$\exists(i : \mathbb{I} | 0 \leq i < |ships| : \\ \text{pointInLine}(p, ListOfships[i].\text{firstPoint}(), ListOfships[i].\text{secondPoint}()) \\ \Rightarrow guess[i] := guess[i] + 1)$$

- output:

$$out := \exists(i : \mathbb{I} | 0 \leq i < |ships| : \\ \text{pointInLine}(p, ListOfships[i].\text{firstPoint}(), ListOfships[i].\text{secondPoint}()))$$

- exception: None

is_game_over():

- output: $out := \forall(i : \mathbb{I} | 0 \leq i < |ListOfships| : ListOfships[i].\text{SIZE}() \equiv guess[i])$
- exception: None

ships():

- output: $out := ListOfships$
- exception: None

Local Functions

ShipLength : Integer → Integer

ShipLength(count):

- $\exists(i : \mathbb{I} | 0 \leq i < |GameStatustT.ships()| : GameStatustT.ships()[i].SIZE() \equiv 3 \rightarrow count++))$

- output: $out := \text{count}$

pointInLine : PointT $\times$ PointT $\times$ PointT $\rightarrow$ boolean

$\text{pointInLine}(p, start, end) \equiv (start.\text{dist}(p) + end.\text{dist}(p) = start.\text{dist}(end))$

collision : ShipT $\times$ ShipT $\rightarrow$ boolean

$\text{collision(one,two)} \equiv \forall(i : \mathbb{I} | 0 \leq i \leq |one.list(0)| : \forall(j : \mathbb{I} | 0 \leq j \leq |two.list(0)| : one.list(0)[i].\text{xcrd}() \equiv second.list(0)[j].\text{xcrd}() \wedge one.list(0)[i].\text{ycrd}() \equiv second.list(0)[j].\text{ycrd}()))$

Battleship Module

Module

Battleship

Uses

Constants, PointT, ShipT, GameStateT

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
init			
move	integer, integer, cellT		OutOfBoundsException, InvalidMoveException, WrongPlayerException
switch_turn			ValidMoveExistsException
getb	integer, integer	cellT	OutOfBoundsException
get_turn		cellT	
count	cellT	integer	
is_valid_move	integer, integer, cellT	boolean	OutOfBoundsException
is_winning	cellT	boolean	
is_any_valid_move	cellT	boolean	
is_game_over		boolean	

Semantics

State Variables

made_List: sequence of PointT
hit_List: sequence of boolean
player1: boolean *player2*: boolean

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():

- transition: $player1, hit\_List, made\_List := true, <>, <>$
- exception none

move(i):

- transition: $blacksturn := \neg blacksturn$ and b such that GameStatusT.is_hit(i)
- exception $exc :=$ (InvalidPosition$(i, j) \Rightarrow$ OutOfBoundsException$|\neg$is_correctPlayer$([player1, c) \Rightarrow$ WrongPlayerException)

switch_turn():

- transition: $player1 := \neg player1$
- exception $exc :=$ (is_any_valid_move() $\Rightarrow$ ValidMoveExistsException)

get_turn():

- output: $out :=$ ($blacksturn \Rightarrow$ BLACK$|\neg blacksturn \Rightarrow$ WHITE)
- exception: none

is_game_over(): *//Returns true if neither player has a valid move*

- output: out := GameStatustT.is_game_over()
- exception: none

Local Types

boardT = sequence [SIZE, SIZE] of cellT

Local Functions

InvalidPosition: integer $\times$ integer $\rightarrow$ boolean

InvalidPosition$(i, j) \equiv \neg((0 \leq i$.xcrd() $<$ Constants.MAX_X$) \wedge (0 \leq i$.ycrd() $\leq$ Constants.MAX_Y$) \wedge$ $(0 \leq j$.xcrd() $<$ Constants.MAX_X$) \wedge (0 \leq j$.ycrd() $\leq$ Constants.MAX_Y$))$

is_correctPlayer: boolean $\times$ cellT $\rightarrow$ boolean

is_correctPlayer$(bt, c) \equiv (bt \Rightarrow c =$ player1$|\neg bt \Rightarrow c =$ player2$)$