CS 2ME3
Assignment 4 Report

Mohid Makhdoomi - makhdoom

April 6, 2017

Constants Module

Module

Constants

Uses

N/A

Syntax

Exported Constants

MAX_GRID = 10 *//max length in the x-direction and y-direction of the grid*
MIN_SIZE = 2 *//minimum ship size*
MAX_SIZE = 5 *//maximum ship size*

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

xcrd():

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

ycrd():

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

dist(p):

- output: $out := \sqrt{(self.xc - p.xcrd())^2 + (self.yc - p.ycrd())^2}$
- 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
startPoint		PointT	
endPoint		PointT	
shipSize		integer	

Semantics

State Variables

start: PointT
end: PointT
length: integer

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($one, two, length$):

- transition: $start, end, length := one, two, length$

- output: $out := self$
- exception: $exc := (((length < \text{Constants.MIN\_SIZE}) \vee (length > \text{Constants.MAX\_SIZE}) \vee (one.\text{dist}(two) \neq length) \vee ((one.\text{xcrd}() \neq two.\text{xcrd}()) \wedge (one.\text{ycrd}() \neq two.\text{ycrd}()))) \Rightarrow \text{InvalidShipException})$

startPoint():

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

endPoint():

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

shipSize():

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

Game State ADT Module

Template Module

GameStateT

Uses

Constants, PointT, ShipT

Syntax

Exported Types

GameStateT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
GameStateT	sequence of ShipT	GameStateT	InvalidShipListException
is_hit	PointT	boolean	
is_all_sunk		boolean	

Semantics

State Variables

ships: sequence of ShipT
hits: sequence of 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: $ships, hits := shipList, < 0, 0, 0, 0, 0 >$
- output: $out := self$

- exception:

  $exc := ((|shipList| \neq 5) \lor (shipList[0].\text{shipSize}() \neq 2) \lor (shipList[1].\text{shipSize}() \neq 3) \lor$
  $(shipList[2].\text{shipSize}() \neq 3) \lor (shipList[3].\text{shipSize}() \neq 4) \lor (shipList[4].\text{shipSize}() \neq 5) \lor$
  $(\exists(i : \mathbb{I} | 0 \leq i < |shipList| : \exists(j : \mathbb{I} | (0 \leq j < |shipList|) \land (i \neq j) :$
  $\text{collision}(shipList[i], shipList[j])))) \Rightarrow \text{InvalidShipListException}$

is_hit(p):

- transition-output: $hits, out :=$
  $(\text{hitCheck}(ships, p) \Rightarrow hits[0..i-1] || <pre(hits)[i] + 1> || hits[i+1..|hits|-1]$
  where $\text{pointInLine}(p, ships[i].\text{startPoint}(), ships[i].\text{endPoint}()) = true$
  $| \neg \text{hitCheck}(ships, p) \Rightarrow hits),$
  $\text{hitCheck}(ships, p)$
- exception: None

is_all_sunk():

- output: $out := \forall(i : \mathbb{I} | 0 \leq i < |ships| : ships[i].\text{shipSize}() = hits[i])$
- exception: None

Local Functions

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

$\text{hitCheck}(shipList, p) \equiv \exists(s : ShipT | s \in shipList : \text{pointInLine}(p, s.\text{startPoint}(), s.\text{endPoint}()))$

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

$\text{collision}(one, two) \equiv \exists(i : \text{PointT} | \text{pointInLine}(i, one.\text{startPoint}(), one.\text{endPoint}()) :$
$\text{pointInLine}(i, two.\text{startPoint}(), two.\text{endPoint}()))$

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

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

Battleship Module

Module

Battleship

Uses

Constants, PointT, ShipT, GameStateT

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
init	sequence of ShipT, sequence of ShipT		
add_shot	PointT	boolean	InvalidMoveException
get_shots		sequence of PointT	
has_won		boolean	

Semantics

State Variables

player1: GameStateT
player2: GameStateT
shotList: sequence of PointT
player1turn: 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(*player1ships*, *player2ships*):

- transition: *player1*, *player2*, *shotList*, *player1turn* := new GameStateT(*player1ships*), new GameStateT(*player2ships*), <>, *true*
- exception: None

add_shot(p):

- transition-output: $player1turn, out := \neg player1turn$ and $shotList$ such that $shotList = pre(shotList)[0..|pre(shotList)|-1]||<p>, (player1turn \Rightarrow player2.\text{is\_hit}(p)|\neg player1turn \Rightarrow player1.\text{is\_hit}(p))$
- exception: $exc := (player1turn \Rightarrow \exists(i : \mathbb{I}|(i\%2 = 0) \wedge (0 \leq i < |shotList|) : samePoint(p, shotList[i]))|\neg player1turn \Rightarrow \exists(i : \mathbb{I}|(i\%2 = 1) \wedge (0 \leq i < |shotList|) : samePoint(p, shotList[i]))) \Rightarrow$ InvalidMoveException

get_shots():

- output: $out := shotList[0..|shotList| - 1]$
- exception: None

has_won():

- output: $out := (player1turn \Rightarrow player2.\text{is\_all\_sunk}()|\neg player1turn \Rightarrow player1.\text{is\_all\_sunk}())$
- exception: None

Local Functions

samePoint : PointT $\times$ PointT $\rightarrow$ boolean

samePoint$(p_1, p_2) \equiv (p_1.\text{xcrd}() = p_2.\text{xcrd}()) \wedge (p_1.\text{ycrd}() = p_2.\text{ycrd}())$