Assignment 2

COMP SCI 2ME3 and SFWR ENG 2AA4

Rober Boshra

April 10, 2017

Specifications of battleship are split across three modules: ShipADT, BoardADT, and GameModule. GameModule provides basic access to starting a game, making a move, and checking whether the game is over. BoardADT contains the BoardT type, specifying a single board pertaining to one player and including both ship placements and hits/misses. BoardT was designed to have a sequence of ShipTs and a sequence of already hit gridcells; this is opposed to a complete two-dimensional grid. Each ship in the game is represented as an instance of ShipT with its predefined size and position on the board. A ShipT also tracks how many of its parts have been hit; however, there is no explicit tracking of which part is hit, as that is already covered by BoardT's list of hit cells.

ShipADT Module

Template Module

ShipADT

Uses

N/A

Syntax

Exported Constants

MISS_ID $= -1$
EMPTY_ID $= 0$

HIT_ID = 1
SHIPDESTROYED_ID = 2

Exported Types

Gridcell = tuple of $(xpos : integer, ypos : integer)$
ShipT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new ShipT	integer, integer, integer, boolean	ShipT	SHIPERROR
hit		integer	
span		sequence of Gridcell	
posx		integer	
posy		integer	
size		integer	
hor		boolean	
partsdestroyed		integer	

Semantics

State Variables

x: integer //X-position of leftmost part of the ship
y: integer //Y-position of the topmost part of the ship
s: integer //Size of the ship
h: boolean //True if the ship is placed horizontally, false otherwise
d: integer //Number of ship parts hit

State Invariant

None

Assumptions

None

Access Routine Semantics

new ShipT (xin, yin, s, hin):

- transition: $x, y, s, h, d := xin, yin, s, i, hin, 0$
- output: $out := self$
- exception: $exc := (s < 1 \vee posx < 0 \vee posy < 0 \Rightarrow SHIPERROR)$

span ()

- output:

$$out := ||(\forall xb, yb : \mathbb{N}|(yb = y \wedge xb \in [x..x+s] \wedge h)$$
$$\vee(xb = x \wedge yb \in [y..y+s] \wedge \neg h) :< \langle xb, yb \rangle >)$$

hit ()

- transition: $d := d + 1$
- output: $out := (d = s \Rightarrow \text{SHIPDESTROYED\_ID} | d < s \Rightarrow \text{HIT\_ID})$

posx ():

- output: $out := x$

posy ():

- output: $out := y$

hor ():

- output: $out := h$

size ():

- output: $out := s$

partsdestroyed ():

- output: $out := d$

BoardADT Module

Template Module

BoardADT

Uses

ShipADT

Syntax

Exported Constants

SIZE_X = 11
SIZE_Y = 9
SHIP_SIZES = $< 2, 3, 3, 4, 5 >$

Exported Types

BoardT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new BoardT	sequence of ShipT	BoardT	SETUPERROR
hit	integer, integer	integer	HITERROR
checkCell	integer, integer	integer	LOCATIONERROR
lose		boolean	

Semantics

State Variables

ss: sequence of ShipT //All ships that are part of the board
$hits$: set of Gridcell //Tracks all actions taken against this board

State Invariant

None

Assumptions

Ships are passed to the constructor in the same order, in terms of size/type, as in SHIP_SIZES.

Access Routine Semantics

new BoardT ($ships$):

- transition: $ss, hits := ships, \{\}$
- output: $out := self$
- exception: $exec := (\neg correctsetup(ships) \Rightarrow \text{SETUPERROR})$

hit (x, y)

- transition: $hits := hits || \{\langle x, y\rangle\}$
- output: $out := (\exists s : ShipT | s \in ss \wedge \langle x, y\rangle \in s.span() \Rightarrow s.hit() | \nexists s : ShipT | s \in ss \wedge \langle x, y\rangle \in s.span() \Rightarrow -1)$
- exception: $exec := \langle x, y\rangle \in hits \vee x \notin [1..\text{SIZE\_X}] \vee y \notin [1..\text{SIZE\_Y}] \Rightarrow \text{HITERROR})$

checkCell (x, y)

- output:

$$\begin{aligned}&out := (\exists s : ShipT | s \in ss \wedge \langle x, y\rangle \in s.span()\\&\wedge \langle x, y\rangle \in hits \Rightarrow \text{ShipADT.HIT\_ID} | \langle x, y\rangle \notin hits \Rightarrow \text{ShipADT.EMPTY\_ID}\\&| \nexists s : ShipT | s \in ss \wedge \langle x, y\rangle \in s.span() \wedge \langle x, y\rangle \in hits \Rightarrow \text{ShipADT.MISS\_ID})\end{aligned}$$

- exception: $exec := x \notin [1..\text{SIZE\_X}] \vee y \notin [1..\text{SIZE\_Y}] \Rightarrow \text{LOCATIONERROR})$

lose ()

- output: $out := (\forall s : ShipT | s \in ships \wedge s.size() = s.partsdestroyed())$

Local Functions

correctsetup: sequence of ShipT $\rightarrow$ boolean
correctsetup($ships$) $\equiv$

$$\begin{aligned}&(\forall i : \mathbb{N} | ships[i].size() = SHIP\_SIZES[i] \wedge i < |SHIP\_SIZES|)\\&\wedge(\forall xp, yp, s : \mathbb{N}, \mathbb{N}, ShipT | s \in ships \wedge \langle xp, yp\rangle \in s.span() \wedge xp \in [1..\text{SIZE\_X}]\\&\vee yp \in [1..\text{SIZE\_Y}]) \wedge (\nexists cell, s1, s2 : Gridcell, ShipT, ShipT | s1 \in ships \wedge s2 \in ships\\&\wedge cell \in s1.span() \wedge cell \in s2.span \wedge s1 \neq s2)\end{aligned}$$

Battleship Game Module

Module

GameModule

Uses

BoardADT

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
Game_init	BoardT, BoardT		
Game_p2turn		boolean	
Game_hit	integer, integer	integer	
Game_won		integer	
Game_view	integer, integer	integer	

Semantics

State Variables

$p1$: BoardT //First player's board (2nd player attacks it)
$p2$: BoardT //Second player's board
$turn$: boolean //Set to true if it is the second player's turn, the first player's if false

Assumptions

Game_init is called before any other access program.
Game_won is called after every Game_hit. Game ends if the output is not 0 (a player won)

Access Routine Semantics

Game_init(x, y):

- transition: $p1, p2, turn := x, y, false$

Game_hit(x, y):

- transition: $turn := \neg turn$
- output: $out := currenttargetboard().hit(x, y)$

Game_p2turn():

- output: $out := turn$

Game_won():

- output: $out := p1.lose() \Rightarrow 2 | p2.lose() \Rightarrow 1 | \neg(p1.lose \vee p2.lose) \Rightarrow 0$

Game_view(x,y):

- output: $out := currenttargetboard().checkCell(x, y)$

Local Functions

currenttargetboard: () $\rightarrow$ BoardT
currenttargetboard() $\equiv (turn \Rightarrow p1) | (\neg turn \Rightarrow p2)$