Assignment 4

Muyideen Jimoh, MacID: jimohma

April 5, 2017

This game of battleship is playable by two players only. This battleship game has three modules in total; The Board Module, The Ships Module and The Gameplay Module. The ship module is designed as an abstract data type, therefore it is template module where both players have access to the ships available in this module. There are three ships in total present in this module namely; Titanic, Hokage and Kazekage. These ships each have a method in the ship module that returns where a particular ship is located on the board. The location of any particular ship is in the form of a list of tuples which has a size of 4 meaning that all three types of ship owned by a player has an exact size of 4. The board module is one both players have access to in order to mark the coordinates where they hit or miss a ship after a fire attack. The size of the board is a 10x10 which means that a player could make a maximum of 100 attacks in an attemp to sink all the ships of its opponent. Finally, the module for the actual game of battleship between the two players is called Gameplay. In this module is where different methods that describes the state of the game are defined. This module has the gameOver method which basically checks to see if all the ships have been sunk and it returns a boolean. In this module is the method which takes care of the actual game-play between player 1 and player 2. Lastly, when a coordinate of a ship is guessed correctly(i.e an attack by a player hit a coordinate where the enemy's ship is located), that coordinate(a tuple) is removed from the list of tuples which represents the ships location.

Board Module

Module

Board

Uses

N/A

Syntax

Exported Constants

MAX_INDEX = 10 // *Dimension in the x and y-direction of the 10 X 10 sized board*
LEAST_INDEX = 1 //*Row or Col index for the first coordinate on the game board*

Exported Access Programs

Routine name	In	Out	Exceptions
init	List		
get_state	List, real, real	String	INVALID_SHIP_POSITION
set_state	List, real, real, String	String	INVALID_SHIP_POSITION

Semantics

State Variables

grid: sequence of List

State Invariant

$|grid|$ = MAX_INDEX

Assumptions

None

Access Routine Semantics

init:

- transition: $grid := <>$
- exception: none

get_state($grid, row, col$):

- output: $out := grid[row][col]$
- exception: $exc := (i \in [0..|grid|] \wedge j \in [0..|grid|] \wedge \text{LEAST\_INDEX} > i > \text{MAX\_INDEX} \vee \text{LEAST\_INDEX} > j > \text{MAX\_INDEX} \Rightarrow \text{INVALID\_SHIP\_POSITION})$

set_state($grid, row, col, player$):

- transition: $grid := (grid[row][col] = player)$
- exception: $exc := (i \in [0..|grid|] \wedge j \in [0..|grid|] \wedge \text{LEAST\_INDEX} > i > \text{MAX\_INDEX} \vee \text{LEAST\_INDEX} > j > \text{MAX\_INDEX} \Rightarrow \text{INVALID\_SHIP\_POSITION})$

Ship Module

Template Module

The_ships

Uses

Constants

Syntax

Exported Types

Ships = ?

Exported Constants

MAX_GRID = 10 *//Maximum coordinate index in the x and y-direction of the board*
MIN_GRID = 1 *//Least coordinate index in the x and y-direction of the board*
SHIP_SIZE = 4 *//Constant size for all the ships*

Exported Access Programs

Routine name	In	Out	Exceptions
new Ships	List, List, List	Ships	
ship_Titanic		List	INVALID_SHIP_POSITION, WRONG_SHIP_SIZE
ship_Hokage		List	INVALID_SHIP_POSITION, WRONG_SHIP_SIZE
ship_Kazekage		List	INVALID_SHIP_POSITION, WRONG_SHIP_SIZE

Semantics

State Variables

$Titanic$: sequence of Tuple
$Hokage$: sequence of Tuple
$Kazekage$: sequence of Tuple

State Invariant

None

Assumptions

None

Access Routine Semantics

new Ships($ship1, ship2, ship3$):

- transition: $Titanic, Hokage, Kazekage := ship1, ship2, ship3$
- output: $out := self$
- exception: none

ship_Titanic:

- output: $out := Titanic$
- exception: $exc :=$ ($|Titanic| \neq$ SHIP_SIZE $\Rightarrow$ WRONG_SHIP_SIZE $\mid i \in [0..|Titanic|-1] \wedge$ MIN_GRID $> Titanic[i][0] >$ MAX_GRID $\vee$ MIN_GRID $> Titanic[i][1] >$ MAX_GRID $\Rightarrow$ INVALID_SHIP_POSITION)

ship_Hokage:

- output: $out := Hokage$
- exception: $exc :=$ ($|Hokage| \neq$ SHIP_SIZE $\Rightarrow$ WRONG_SHIP_SIZE $\mid i \in [0..|Hokage|-1] \wedge$ MIN_GRID $> Hokage[i][0] >$ MAX_GRID $\vee$ MIN_GRID $> Hokage[i][1] >$ MAX_GRID $\Rightarrow$ INVALID_SHIP_POSITION)

ship_Kazekage:

- output: $out := Kazekage$
- exception: $exc :=$ ($|Kazekage| \neq$ SHIP_SIZE $\Rightarrow$ WRONG_SHIP_SIZE $\mid i \in [0..|Kazekage|-1] \wedge$ MIN_GRID $> Kazekage[i][0] >$ MAX_GRID $\vee$ MIN_GRID $> Kazekage[i][1] >$ MAX_GRID $\Rightarrow$ INVALID_SHIP_POSITION)

Game-Play Module

Template Module

Gameplay

Uses

Board, The_Ships

Syntax

Exported Types

GamePlay = ?

Exported Constants

MAX_FIRE = 100 *//Maximum attack to destroy all opponent's ship*
SHIP_SIZE = 4 *//Constant size for all the ships*
MAX_HIT = 12 *// Exact number of hits needed in order to sink all of enemy's ship and win the game*

Exported Access Programs

Routine name	In	Out	Exceptions
new GamePlay	List, List, List, List	GamePlay	
gameOver	List, List, List	boolean	
shipSunk	List	boolean	
winner_mssg	String	String	
play		String	

Semantics

State Variables

p1_ship: sequence of List *// Contains the three ship positions for player 1*
p2_ship: sequence of List *// Contains the three ship positions for player 2*
p1_fire: sequence of Tuple *// Contains the attacks fired by player 1*
p2_fire: sequence of Tuple *// Contains the attacks fired by player 2*

State Invariant

$|p1\_ship|, |p2\_ship| = \text{SHIP\_SIZE}$
$|p1\_fire|, |p2\_fire| = \text{MAX\_FIRE}$

Assumptions

The RegionT 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

new GamePlay($ship\_P1, ship\_P2, fire\_P1, fire\_P2$):

- transition: $p1\_ship, p2\_ship, p1\_fire, p2\_fire := ship\_P1, ship\_P2, fire\_P1, fire\_P2$
- output: $out := self$
- exception: none

gameOver($count$):

- output: $out := (count == \text{MAX\_HIT})$
- exception: none

winner_mssg($winner$):

- output: $out :=$ "$Message$"
- exception: none

play:

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