Battleship Module Interface Specification

Michael Balas

January 9, 2019

The purpose of this software design exercise is to design and specify a module for storing the state of a Battleship game. The game board is represented as a two dimensional sequence, with the first dimension as the row and the second dimension as the column. The indexes are relative to the upper left hand corner of the board; that is, row 0 and column 0 are at the upper left.
If anything seems ambiguous, I recommend checking the code in *Battleship.java*, as it follows the specification exactly but is clearer and contains some documentation. The test cases in *BattleshipTests.java* may also provide further clarity.

Battleship Module

Template Module

BattleShip

Uses

N/A

Syntax

Exported Constants

SIZE = 10
NUM_SHIPS = 5
CARRIER_LENGTH = 5
BATTLESHIP_LENGTH = 4
CRUISER_LENGTH = 3
SUBMARINE_LENGTH = 3
DESTROYER_LENGTH = 2
SHIPS = [CARRIER_LENGTH, BATTLESHIP_LENGTH, CRUISER_LENGTH, SUBMARINE_LENGTH, DESTROYER_LENGTH]
HITPOINTS = (CARRIER_LENGTH)+(BATTLESHIP_LENGTH)+(CRUISER_LENGTH)+ (SUBMARINE_LENGTH) + (DESTROYER_LENGTH)

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
boardInit			
addShip	integer, integer, integer, boolean,		InvalidLocationException, ShipAdditionException, InvalidShipTypeException
shoot	integer, integer		InvalidLocationException, PrematureShotException, GameOverException
switchTurn			GameOverException
getTurn		boolean	
percentPlayerShips		real	
percentOpponentShips		real	
numShotsFired		integer	
numShotsTaken		integer	
isWinning		boolean	GameOverException
isWinner		boolean	

Semantics

State Variables

$playerBoard$: sequence [SIZE][SIZE] of integers
$opponentBoard$: sequence [SIZE][SIZE] of integers
$shotsFired$: sequence [SIZE][SIZE] of integers
$shotsTaken$: sequence [SIZE][SIZE] of integers
$playerTurn$: boolean
$readyToFire$: boolean

State Invariant

None

Assumptions

boardInit() is called before any other access routine.

Access Routine Semantics

boardInit():

- transition: $playerBoard, opponentBoard, shotsFired, shotsTaken, playerTurn, readyToFire :=$
  $(i, j : \mathbb{N} | i \in [0..\text{SIZE} - 1] \wedge j \in [0..\text{SIZE} - 1] : playerBoard[i][j] = 0)$,
  $(i, j : \mathbb{N} | i \in [0..\text{SIZE} - 1] \wedge j \in [0..\text{SIZE} - 1] : opponentBoard[i][j] = 0)$,
  $(i, j : \mathbb{N} | i \in [0..\text{SIZE} - 1] \wedge j \in [0..\text{SIZE} - 1] : shotsFired[i][j] = 0)$,
  $(i, j : \mathbb{N} | i \in [0..\text{SIZE} - 1] \wedge j \in [0..\text{SIZE} - 1] : shotsTaken[i][j] = 0)$,
  $True$,
  $False$

- exception: none

addShip($row, column, shipType, orientation$):

- transition: $(i : \mathbb{N} | i \in [0..SHIPS[shipType]] : playerTurn \Rightarrow (isValidPlacement($row, column, shipType, orientation$) \Rightarrow (orientation \Rightarrow playerBoard[row][column{+}i] = 1 | \neg orientation \Rightarrow playerBoard[row + i][column] = 1))$
  $\neg playerTurn \Rightarrow (isValidPlacement($row, column, shipType, orientation$) \Rightarrow (orientation \Rightarrow opponentBoard[row][column{+}i] = 1 | \neg orientation \Rightarrow opponentBoard[row{+}i][column] = 1)))$,
  $(allShipsPlaced() \Rightarrow readyToFire = True))$

- exception: $exc := \neg isValidPlacement(row, column, shipType, orientation) \Rightarrow$ InvalidLocationException$| shipType \notin [0..|SHIPS|{-}1] \Rightarrow InvalidShipTypeException$ $| readyToFire \Rightarrow ShipAdditionException)$

switchTurn():

- transition: $playerTurn := \neg playerTurn$

- exception: $exc := ((readyToFire = True) \wedge (percentPlayerShips = 0 \vee percentOpponentShips = 0) \Rightarrow GameOverException)$

getTurn():

- output: $out := playerTurn$

- exception: none

shoot(row, column):

- transition: $(isValidShot \Rightarrow (playerTurn \Rightarrow (shotsFired[row][column] = 1 \wedge opponentBoard[row][column] = 0) \wedge (percentOpponentShips() > 0) \Rightarrow switchTurn() | \neg playerTurn \Rightarrow (shotsTaken[row][column] = 1 \wedge playerBoard[row][column] = 0) \wedge (percentPlayerShips() > 0) \Rightarrow switchTurn()))$

- exception: $exc := \neg isValidShot \Rightarrow InvalidLocationException | \neg readyToFire \Rightarrow PrematureShotException | (percentPlayerShips = 0 \vee percentOpponentShips = 0) \Rightarrow GameOverException)$

//percentPlayerShips() and percentOpponentShips() are seperate methods because both should be accessed at any time by any player.
percentPlayerShips():

- output: $out :=$ ((remainingPlayerHitPoints()/HITPOINTS) $\times$ 100)

- exception: none

percentOpponentShips():

- output: $out :=$ ((remainingOpponentHitPoints()/HITPOINTS) $\times$ 100)

- exception: none

//numShotsFired() and numShotsTaken() are seperate methods because both should be accessed at any time by any player.
numShotsFired():

- output: $out := +(i, j : \mathbb{N} | i \in [0..SIZE-1] \wedge j \in [0..SIZE-1] : shotsFired[i][j])$

- exception: none

numShotsTaken():

- output: $out := +(i, j : \mathbb{N} | i \in [0..SIZE-1] \wedge j \in [0..SIZE-1] : shotsTaken[i][j])$

- exception: none

isWinning():

- output: $out := (playerTurn \Rightarrow ((percentPlayerShips() > percentOpponentShips()) \Rightarrow True | percentPlayerShips \leq percentOpponentShips() \Rightarrow False) | \neg playerTurn \Rightarrow ((percentOpponentShips() > percentPlayerShips()) \Rightarrow True | percentOpponentShips() \leq percentPlayerShips() \Rightarrow False))$
- exception: $exc := (percentPlayerShips = 0 \vee percentOpponentShips = 0 \Rightarrow GameOverException)$

isWinner():

- output: $out := (playerTurn \Rightarrow ((percentOpponentShips() = 0) \Rightarrow True | percentOpponentShips > 0 \Rightarrow False) | \neg playerTurn \Rightarrow ((percentPlayerShips() = 0) \Rightarrow True | percentPlayerShips > 0 \Rightarrow False))$
- exception: none

Local Functions

isValidPlacement: integer × integer × integer × boolean → boolean
isValidPlacement(row, column, shipType, orientation) ≡ $(i : \mathbb{N} | i \in [0..SHIPS[shipType]] : (playerTurn \Rightarrow (orientation \Rightarrow (0 \leq column{+}i \leq SIZE{-}1)) \wedge \neg(playerBoard[row][column+i] = 1) \Rightarrow True) | (\neg orientation \Rightarrow (0 \leq row + i \leq SIZE - 1)) \wedge \neg(playerBoard[row+i][column] = 1) \Rightarrow True) |$
$(\neg playerTurn \Rightarrow (orientation \Rightarrow (0 \leq column{+}i \leq SIZE{-}1)) \wedge \neg(opponentBoard[row][column+i] = 1) \Rightarrow True) | (\neg orientation \Rightarrow (0 \leq row + i \leq SIZE - 1)) \wedge \neg(opponentBoard[row+i][column] = 1) \Rightarrow True))$

isValidShot: integer × integer → boolean
isValidShot(row, column) ≡ $(0 \leq row \leq SIZE - 1 \wedge 0 \leq column \leq SIZE - 1 \wedge (playerTurn \Rightarrow \neg(shotsFired[row][column] = 1) | \neg playerTurn \Rightarrow \neg(shotsTaken[row][column] = 1)))$

allShipsPlaced: () → boolean
allShipsPlaced() ≡ $((percentPlayerShips = 100 \wedge percentOpponentShips = 100) \Rightarrow True | True \Rightarrow True)$
remainingPlayerHitPoints: () → boolean
remainingHitPoints() ≡ $+(i, j : \mathbb{N} | i \in [0..SIZE{-}1] \wedge j \in [0..SIZE{-}1] : playerBoard[i][j])$

remainingOpponentHitPoints: () → boolean
remainingHitPoints() ≡ $+(i, j : \mathbb{N} | i \in [0..SIZE{-}1] \wedge j \in [0..SIZE{-}1] : opponentBoard[i][j])$