MoveT Module

Module

MoveT

Uses

N/A

Syntax

Exported Constants

None

Exported Types

MoveT = {Left, Right, Up, Down}

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

BoardT Module

Template Module

BoardT

Uses

MoveT

Syntax

Exported Types

None

Exported Constant

Size = 4 // The game is played on a plain 4×4 grid

Exported Access Programs

Routine name	In	Out	Exceptions
BoardT			
startBoard			
addCell	$\mathbb{N}$		RuntimeException
getStatus		$\mathbb{B}$	
getMarks		$\mathbb{N}$	
getCell	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{N}$	IndexOutOfBoundsException
setCell	$\mathbb{N}$, $\mathbb{N}$, $\mathbb{N}$		IndexOutOfBoundsException
isEmpty	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	IndexOutOfBoundsException
isFull		$\mathbb{B}$	
horizontalCheck		$\mathbb{B}$	
verticalCheck		$\mathbb{B}$	
isOver		$\mathbb{B}$	
moveBoard	MoveT		
down			RuntimeException
mergeColumnDown	$\mathbb{N}$		
up			RuntimeException
mergeColumnUp	$\mathbb{N}$		
left			RuntimeException
mergeRowLeft	$\mathbb{N}$		
right			RuntimeException
mergeRowRight	$\mathbb{N}$		

Semantics

State Variables

board: sequence [Size, Size] of int
status: $\mathbb{B}$ // false represents the game is over
marks: $\mathbb{N}$

State Invariant

None

Assumptions

- All input types are followed as the specifications.

- The constructor BoardT is called for each object instance before any other access routine is called for that object.
- Assume there is a random function (random()) that generates a random value between 0 (inclusive) and 1 (exclusive).
- Assume the origin is at the left top, so cell(0, 0) is the left top cell on the grid, and cell(3, 3) is at the right bottom corner.

Access Routine Semantics

BoardT():

- transition:
  $$\text{board} := \langle \begin{matrix} \langle 0,0,0,0 \rangle \\ \langle 0,0,0,0 \rangle \\ \langle 0,0,0,0 \rangle \\ \langle 0,0,0,0 \rangle \end{matrix} \rangle$$
  status, marks = true, 0
- exception: None

startBoard():

- transition: board := $(\exists x, y < \text{size} \mid \text{board[x][y]} = 2) \wedge (\exists a, b < \text{size} \mid (\text{a} \neq \text{x} \wedge \text{b} \neq \text{y}) \wedge ((\text{board[a][b]} = 2) \vee (\text{board[a][b]} = 4)))$
- exception: None

addCell(*num*):

- transition: (isEmpty(x, y) $\Rightarrow$ setCell(x, y, num) | True $\Rightarrow$ addCell(num)) where x = randomNum(), y = randomNum()
- exception: *exc* := (isFull() $\Rightarrow$ RuntimeException)

getStatus():

- transition: none
- output: *out* := status
- exception: None

getMarks():

- transition: none
- output: out := marks
- exception: None

getCell(x, y):

- transition: none
- output: out := board$[x][y]$
- exception: $exc := (x \geq size \vee y \geq size \Rightarrow$ IndexOutOfBoundsException)

setCell(x, y, num):

- transition: $board$:= board$[x][y] = num$
- output: none
- exception: $exc := (x \geq size \vee y \geq size \Rightarrow$ IndexOutOfBoundsException)

isEmpty(x, y):

- transition: none
- output: out := board$[x][y] = 0$
- exception: $exc := (x \geq size \vee y \geq size \Rightarrow$ IndexOutOfBoundsException)

isFull():

- transition: none
- output: $out := (\forall x, y : \mathbb{N} | x < size \wedge y < size \wedge$ board[x][y] $\neq 0$)
- exception: None

horizontalCheck():

- transition: none
- output: $out := \neg$(is_Full()) $\vee$ $(\exists x, y : \mathbb{N} | x < size \wedge y < size - 1 \wedge$ board[x][y] = board[x][y + 1])
- exception: None

verticalCheck():

- transition: none
- output: out := ¬(is_Full()) ∨ ($\exists x, y : \mathbb{N} | x < size - 1 \wedge y < size \wedge$ board[x][y] = board[x + 1][y])
- exception: None

isOver():

- transition: status := isOver()
- output: out := ¬ horizontalCheck() ∧¬ verticalCheck() ∧ isFull()
- exception: None

moveBoard($move$):

- transition: ($move = Down \Rightarrow$ down() |$move = Up \Rightarrow$ up() |$move = Left \Rightarrow$ left() |$move = Right \Rightarrow$ right())
- output: none
- exception: None

down():

- transition: ($\forall col : \mathbb{N} | col < size$: mergeColumnDown($col$)
- output: none
- exception: exc := (isFull() ∧ ¬ verticalCheck()) ⇒ RuntimeException)

mergeColumnDown($colIndex$):

- transition: (len(numIndex(row)) = 1 ⇒ row := { 0, 0, 0, x } | len(numIndex(row)) = 2 ⇒ row := { 0, 0, 0, 2x } ∨ { 0, 0, x, y } | len(numIndex(row)) = 3 ⇒ row := { 0, x, y, z } ∨ { 0, 0, 2x, y } ∨ { 0, 0, x, 2y } | len(numIndex(row)) = 4 ⇒ row := { w, x, y, z } ∨ { 0, 2x, y, z } ∨ { 0, x, 2y, z } ∨ { 0, x, y, 2z }) where row = { board[rowIndex][0], board[rowIndex][1], board[rowIndex][2], board[rowIndex][3] }

  marks := marks + change where change = 2x ∨ 2y ∨ 2z
- output: none

- exception: None

up():

- transition: $(\forall col : \mathbb{N} | col < size$: mergeColumnUp($col$)
- output: none
- exception: $exc :=$ ($\neg$ ($\neg$ isFull() $\vee$ verticalCheck()) $\Rightarrow$ RuntimeException)

mergeColumnUp($colIndex$):

- transition: (len(numIndex(row)) = 1 $\Rightarrow$ row := { x, 0, 0, 0 } | len(numIndex(row)) = 2 $\Rightarrow$ row := { 2x, 0, 0, 0 } $\vee$ { x, y, 0, 0 } | len(numIndex(row)) $\Rightarrow$ row := { 0, x, y, z } $\vee$ { 2x, y, 0, 0 } $\vee$ { x, 2y, 0, 0 } | len(numIndex(row)) = 4 $\Rightarrow$ row := { w, x, y, z } $\vee$ { 2x, y, z, 0 } $\vee$ { x, 2y, z, 0 } $\vee$ { x, y, 2z, 0 }) where row = { board[0][colIndex], board[1][colIndex], board[2][colIndex], board[3][colIndex] }

  marks := marks + change where change = 2x $\vee$ 2y $\vee$ 2z
- output: none
- exception: None

left():

- transition: $(\forall row : \mathbb{N} | row < size$: mergeColumnLeft($row$)
- output: none
- exception: $exc :=$ ($\neg$ ($\neg$ isFull() $\vee$ horizontalCheck()) $\Rightarrow$ RuntimeException)

mergeRowLeft($rowIndex$):

- transition: (len(numIndex(row)) = 1 $\Rightarrow$ row := { x, 0, 0, 0 } | len(numIndex(row)) = 2 $\Rightarrow$ row := { 2x, 0, 0, 0 } $\vee$ { x, y, 0, 0 } | len(numIndex(row)) = 3 $\Rightarrow$ row := { 0, x, y, z } $\vee$ { 2x, y, 0, 0 } $\vee$ { x, 2y, 0, 0 } | len(numIndex(row)) = 4 $\Rightarrow$ row := { w, x, y, z } $\vee$ { 2x, y, z, 0 } $\vee$ { x, 2y, z, 0 } $\vee$ { x, y, 2z, 0 }) where row = { board[rowIndex][0], board[rowIndex][1], board[rowIndex][2], board[rowIndex][3] }

  marks := marks + change where change = 2x $\vee$ 2y $\vee$ 2z
- output: none
- exception: None

right():

- transition: $(\forall row : \mathbb{N} | row < size$: mergeColumnRight($row$)
- output: none
- exception: $exc :=$ ($\neg$ ($\neg$ isFull() $\vee$ horizontalCheck()) $\Rightarrow$ RuntimeException)

mergeRowRight($rowIndex$):

- transition: (len(numIndex(row)) = 1 $\Rightarrow$ row := { 0, 0, 0, x } | len(numIndex(row)) = 2 $\Rightarrow$ row := { 0, 0, 0, 2x } $\vee$ { 0, 0, x, y } | len(numIndex(row)) = 3 $\Rightarrow$ row := { 0, x, y, z } $\vee$ { 0, 0, 2x, y } $\vee$ { 0, 0, x, 2y } | len(numIndex(row)) = 4 $\Rightarrow$ row := { w, x, y, z } $\vee$ { 0, 2x, y, z } $\vee$ { 0, x, 2y, z } $\vee$ { 0, x, y, 2z }) where row = { board[rowIndex][0], board[rowIndex][1], board[rowIndex][2], board[rowIndex][3] }

  marks := marks + change where change = 2x $\vee$ 2y $\vee$ 2z
- output: none
- exception: None

Local Functions

randomNum: None $\rightarrow \mathbb{N}$

randomNum() $\equiv$ int(random() * 4)

numIndex: $\mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow$ seq of $\mathbb{N}$

numIndex(a, b, c, d) $\equiv +(\exists x \in \{a, b, c, d\} \wedge x \neq 0$: 1)

UserInterface Module

UI Module

Uses

BoardT

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
UI			
printWelcomeMessage			
printBoard	BoardT		
printRow	seq of $\mathbb{N}$		
printEndingMessage			

Semantics

State Variables

None

State Invariant

None

Assumptions

- The UI constructor is called at the beginning of each game before any other access routine is called for that object. The constructor can only be called once.

Access Routine Semantics

UI():

- transition: none
- output: none
- exception: None

printWelcomeMessage():

- transition: none
- output: none *// Displays a welcome message at the beginning of each game.*

printBoard(*board*):

- transition: none
- output: none *// Displays a game board and current marks.*

printRow(*row*):

- transition: none
- output: none *// Displays a row in the game board.*

printEndingMessage():

- transition: Prints a ending message when the game is over.

PlayT Module

Module

PlayT

Uses

BoardT, UI, MoveT

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
PlayT			

Semantics

State Variables

None

State Invariant

None

Assumptions

- The user will always have a valid input.

Access Routine Semantics

PlayT():

- transition: none
- output: none *// Create a new game, when the game is not over, keep asking for user input and print current board. Game procedure: check if the board is full → (not full) → next move → perform move → add random tile → check if the board is full...*
- exception: None