

Assignment 4, Part 1, Specification

Safwan Hossain, hossam18, 400252391

April 13, 2021

This Module Interface Specification (MIS) document contains modules, types and methods used to run a game of 2048.

Move Module

Module

Move

Uses

None

Syntax

Exported Constants

None

Exported Types

```
IndicatorT = {  
  up, #Player moves up  
  down, #Player moves down  
  left, #Player moves left  
  right; #Player moves right  
}
```

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Consideration

Move is an enum class that represent the possible moves a player can make.

Positions Module

Template Module

Positions

Uses

none

Syntax

Exported Constants

None

Exported Types

Position = ?

Exported Access Programs

Routine Name	In	Out	Exceptions
new Positions	\mathbb{Z}	Position	
rotateAvailablePositions			
getRotatedPosition	seq of \mathbb{Z}	seq of \mathbb{Z}	
addAllAvailablePositions			
positionIsAvailable	\mathbb{Z}, \mathbb{Z}	\mathbb{B}	
addAvailablePosition	\mathbb{Z}, \mathbb{Z}		
removeAvailablePosition	\mathbb{Z}, \mathbb{Z}		
getRandomPosition		seq of \mathbb{Z}	
hasAvailablePosition		\mathbb{B}	
merged	\mathbb{Z}, \mathbb{Z}		
resetMergedPositions			
wasRecentlyMerged	\mathbb{Z}, \mathbb{Z}	\mathbb{B}	

Semantics

State Variables

boardSize : \mathbb{Z}

availablePositions : set of sequences of \mathbb{Z}

recentlyMerged : set of sequences of \mathbb{Z}

State Invariant

None

Assumptions

All inputs made entered are handled by Game or Board and not by user, therefore inputs will have the proper indexes.

Access Routine Semantics

new Positions(*boardSize*):

- transition: *boardSize*, *availablePositions*, *recentlyMerged* := *boardSize*, {}, {}
- output: *out* := self
- exception: none

rotateAvailablePositions():

- transition:
availablePositions := {*i* : seq of \mathbb{Z} | *i* ∈ *availablePositions* : *getRotatedPosition*(*i*)}
- output: *out* := none
- exception: none

getRotatedPosition(*position*):

- output: *out* := $\langle \text{boardSize} - 1 - \text{position}[1], \text{position}[0] \rangle$
- exception: none

addAllAvailablePositions():

- transition:
 $\forall (x, y : \mathbb{N} | x, y < \text{boardSize} : \text{addAvailablePosition}(x, y))$

- output: none
- exception: none

positionIsAvailable(*row*, *col*):

- output: $out := \exists(s : \text{seq of } \mathbb{Z} \mid s \in \text{availablePositions} : s[0] = row \wedge s[1] = col)$
- exception: none

addAvailablePosition(*row*, *col*):

- transition:

$$\text{availablePositions} := \{\neg(\text{positionIsAvailable}(row, col)) \Rightarrow \text{availablePositions} \cup \langle row, col \rangle \mid \text{True} \Rightarrow \text{availablePositions}\}$$
- output: none
- exception: none

removeAvailablePosition(*row*, *col*):

- transition:

$$\text{availablePositions} := \exists(s : \text{seq of } \mathbb{Z} \mid s \in \text{availablePositions} : s[0] = row \wedge s[1] = col) \Rightarrow \text{availablePositions} - s \mid \text{True} \Rightarrow \text{availablePositions}\}$$
- output: none
- exception: none

getRandomPosition():

- output: $out := |\text{availablePositions}| = 0 \Rightarrow \langle -1, -1 \rangle \mid \text{True} \Rightarrow \text{availablePositions}[\lfloor (\text{random}() * |\text{availablePositions}|) \rfloor]$
- exception: none

hasAvailablePosition():

- output: $out := \neg(|\text{availablePositions}| = 0)$
- exception: none

merged(*row*, *col*):

- transition:

$$\text{recentlyMerged} := \text{recentlyMerged} \cup \{\langle row, col \rangle\}$$

- output: none
- exception: none

resetMergedPositions():

- transition:
 $recentlyMerged := \{\}$
- output: none
- exception: none

wasRecentlyMerged(*row*, *col*):

- output:
 $out := \exists(s : \text{seq of } \mathbb{Z} \mid s \in recentlyMerged : s[0] = row \wedge s[1] = col)$
- exception: none

Board Module

Template Module

Board

Uses

Position

Syntax

Exported Constants

None

Exported Types

Board = ?

Exported Access Programs

Routine Name	In	Out	Exceptions
new Board	\mathbb{Z} , Position	Board	IllegalArgumentException
getNumber	\mathbb{Z} , \mathbb{Z}	\mathbb{Z}	
getBoardSize		\mathbb{N}	
getScore		\mathbb{N}	
getRow	\mathbb{Z}	seq of \mathbb{Z}	
wasChangeMade		\mathbb{B}	
setNumber	\mathbb{Z} , \mathbb{Z} , \mathbb{Z}		
resetChangeChecker			
getLargestCurrentNumber		\mathbb{B}	
isMovePossible		\mathbb{B}	
canMove	\mathbb{Z} , \mathbb{Z}	\mathbb{B}	
rotate			
slideUp	\mathbb{Z} , \mathbb{Z}		
slideAllUp			

Semantics

State Variables

$boardSize : \mathbb{Z}$
 $positions : \text{Positions}$
 $wasChangeMade : \mathbb{B}$
 $numbers : \text{a sequence of sequences of } \mathbb{Z}$
 $score : \mathbb{N}$

State Invariant

None

Assumptions

All inputs made entered are handled by Game or Display and not by user, therefore inputs will have the proper indexes.

Access Routine Semantics

new Board($boardSize, positions$):

- transition:
 $boardSize, positions, wasChangeMade, score, numbers :=$
 $boardSize, positions, False, 0, \langle x : \mathbb{N} | x < boardSize : \langle 0, 0, 0, 0 \rangle \rangle$
- output: $out := self$
- exception: $exc := (boardSize = 0) \Rightarrow IllegalArgumentException$

getNumber(row, col):

- output: $out := numbers[row][col]$
- exception: none

getBoardSize():

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

getScore():

- output: $out := score$

- exception: none

getRow(row):

- output: $out := numbers[row]$

- exception: none

wasChangeMade():

- output: $out := wasChangeMade$

- exception: none

setNumber($row, col, number$):

- transition:
 $numbers[row][col] := number$

- output: none

- exception: none

resetChangeChecker():

- transition:
 $wasChangeMade := False$

- output: none

- exception: none

getLargestCurrentNumber():

- output: $out := \text{Max}(numbers)$

- exception: none

isMovePossible():

- output: $out := positions.hasAvailablePosition() \vee \exists (x, y : \mathbb{N} | x, y < boardSize : canMove(x, y))$

- exception: none

canMove(*row*, *col*):

- output: $out := \exists(x, y : \mathbb{Z}) \neg(row - x = 0 \wedge col - y = 0) \wedge (row - x = 0 \vee col - y = 0) \wedge (|row - x| = 1 \vee |col - y| = 1) : numbers[x][y] = numbers[row][col]$
- exception: none

rotate():

- transition: $numbers := \langle \forall(x : \mathbb{N}) x < boardSize : \langle \forall(y : \mathbb{N}) y < boardSize : numbers[y][boardSize - 1 - x] \rangle \rangle$
- output: none
- exception: none

slideUp(*row*, *col*):

Description: If the number above was merged during this move then stop the function. If the number above is the same as the current number then, the number above will double and the current number will be turned to a 0. If the number above is a 0 then it will slide the number up by one square and call the slideUp() function recursively for the number located above.

- transition:
for easier readability let the following be boolean statements:
 $A = row \leq 0 \vee positions.wasRecentlyMerged(row - 1, col)$
 $B = numbers[row][col] = numbers[row - 1][col]$
 $C = numbers[row - 1][col] = 0$

$number[row - 1][col], number[row][col], score, wasChangeMade :=$
 $(A \Rightarrow number[row - 1][col], number[row][col], score, wasChangeMade)|$
 $(B \Rightarrow numbers[row][col], 0, score + number[row][col]*2, True)|$
 $(C \Rightarrow numbers[row][col], 0, score, True)|$
 $(True \Rightarrow number[row - 1][col], number[row][col])$

The bottom transitions could have been added with the ones on top, but for easier readability they are separted. Assume that at each condition first the top instructions are called then the bottom.

$(A \Rightarrow)|$
 $(B \Rightarrow positions.Merged(row - 1, col), positions.addAvailablePosition(row, col)|$
 $(C \Rightarrow positions.removeAvailablePosition(row - 1, col), positions.addAvailablePosition(row, col), slideUp$
 $1, col)$

- output: none
- exception: none

slideAllUp():

- transition:

$$\forall(x : \mathbb{N} | 1 \leq x < \text{boardSize} : \forall(y : \mathbb{N} | y < \text{boardSize} : \neg(\text{numbers}[x, y] = 0) \Rightarrow \text{slideUp}(x, y))$$
- output: none
- exception: none

Consideration

This module is used to get and manipulate values on the board. The numerical values for the board is stored in a square matrix of length boardSize.

Game Module

Template Module

Game

Uses

Board, Positions

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	In	Out	Exceptions
new Game	\mathbb{Z}, \mathbb{Z}	Game	
move	Move		
getScore		\mathbb{N}	
getBoard		Board	
rotateGame	\mathbb{Z}		
isMovePossible		\mathbb{B}	
pushRandomNumber			
resetChangeChecker			
getRandomNumber		\mathbb{N}	

Semantics

State Variables

positions : Positions *board* : Board *boardSize* : \mathbb{Z} *numOfRandomPerMove* : \mathbb{Z} *score* : \mathbb{N}

State Invariant

None

Assumptions

All inputs made entered are handled by Controller, therefore inputs will have the proper indexes.

Access Routine Semantics

new Game(boardSize, numOfRandomPerMove):

- transition: *positions, board, boardSize, numOfRandomPerMove, score, := newPosition(boardSize), new pushRandomNumber() pushRandomNumber()*
- output: none
- exception: none

move(direction):

- transition: if direction is Move.up do not rotate. If the direction is Move.right rotate once now. If the direction is Move.down, rotate twice, if the direction is move.left rotate three times. After rotating slide the board up, reset any merged markers, update the new score, then rotate the game back to its original orientation.
- output: none
- exception: none

Display

Template Module

Display

Uses

Board

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	In	Out	Exceptions
--------------	----	-----	------------

Semantics

State Variables

State Invariant

None

Assumptions

Access Routine Semantics

Controller

Module

Controller

Uses

Display, Game, Board

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	In	Out	Exceptions
tryInt	\mathbb{Z}	\mathbb{Z}	NumberFormatException
start			
newGame			
gameOver			
runGame			

Semantics

State Variables

State Invariant

None

Assumptions

None

Access Routine Semantics

HashSet Module

Generic Template Module inherits Set(E)

HashSet(E)

Considerations

Implemented as part of Java, as described in the Oracle Documentation