Assignment 4, Specification

Hosty Khurana, Khurah2

April 12, 2021

This Module Interface Specification (MIS) document contains modules, types and methods for implementing the game *2048*. At the start of each game,there occur two random numbers on the command line in a matrix of 4 X 4, the numbers are randomized between 2 or 4 with a 10% chance that a 4 can occur. There are certain operations that could be applied by the user to play this game which include going up,down,left or right depending on the requirement at that time. The goal of this game is to achieve a 2048 on any of the tile by performing the mentioned operations. The rules of the game are :

- When two tiles with the same number on them collide with one another as you move them, they will merge into one tile with the sum of the numbers written on them initially.
- A random number(a 2 or a 4) pops up everytime you perform the mentioned operations only if there exist a tile with value Zero(Empty Time).

The game can be launched and play by typing `make run` in terminal.

View Module

Template Module

View

Uses

None

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
new View		View	
runGame			
Instructions			

Semantics

State Variables

gameBoard : Model
scanner : Scanner
continueGame : $\mathbb{B}$

State Invariant

None

Assumptions

None

Access Routine Semantics

new View():

- transition: $gameBoard, scanner, continueGame :=$ new Model, new Scanner, true
- output: none
- exception: none

runGame():

- transition: Method for running the game. The game starts by displaying a welcome message and
- exception: none

Instructions():

- transition: Displays Welcome Message and rules when user first runs the game. While continueGame variable stays true, it asks the user for a character to make the corresponding move. After every move it checks if the user won or lose the game.

Input Char	**Action**
'w' || 'W'	gameBoard.action("Up")
'a' || 'A'	gameBoard.action("Left")
's' || 'S'	gameBoard.action("Down")
'd' || 'D'	gameBoard.action("Right")
'e' || 'E'	continueGame := false
'r' || 'R'	gameBoard = new Module()

- exception: none

Model Module

Module

Model

Uses

None

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Model		Self	
newGame			
placeNumber			
isEmpty		$\mathbb{B}$	
rotation	$\mathbb{N}$, $\mathbb{B}$		
action	String		
calculateMerge			
checkLucky		$\mathbb{B}$	
checkUnlucky		$\mathbb{B}$	
toString		String	
isLucky		$\mathbb{B}$	
isUnlucky		$\mathbb{B}$	
getGame		int seq[$\mathbb{N}$,$\mathbb{N}$]	

Semantics

State Variables

game: seq of $[\mathbb{N}, \mathbb{N}]$
randomNumber: Random
lucky: $\mathbb{B}$
unlucky: $\mathbb{B}$

State Invariant

None

Assumptions

None

Access Routine Semantics

new Model():

- transition: game, randomNumber, lucky, Unlucky := new int[4][4], new Random, true, false
  This calls newGame.
- exception: none

newGame():

- transition: choose two different random points for x and y between 0 and 3 (inclusive) then put a 2 or a 4 at two random places. Chance of a 4 occuring is 10 percent.
- exception: none

placeNumber():

- transition: choose random points for x and y between 0 and 3 (inclusive) then Put a 2 or a 4 at empty random places after an action. Chance of a 4 occuring is 10 percent.
- exception: none

isEmpty():

- transition: Traverse game (2D array) to find positions that carry 0, if any one position that have 0 exist then return true else false.
- output: $\mathbb{B}$
- exception: none

rotation(int angle, boolean rightRotate):

- transition: manipulate the game (2D array) by shifting the values vertically or horizontally. The array moves by 90^* angle everytime.
- exception: none

action(String move):

- transition:

move	**Action**
"Up"	rotation(90, false) then calculateMerge then rotation(90, true)
"Down"	rotation(90, true) then calculateMerge then rotation(90, false)
"Left"	calculteMerge
"Right"	rotation(180, true) then calculateMerge() then rotation(180, false)

- exception: none

calculateMerge():

- transition: basic concepts to add the digits using linkedList.
- exception: none

checkLucky():

- transition: Traverses game (2D array) to find a place with value 2048, If such place is found return true else return false.
- output: $\mathbb{B}$
- exception: none

checkUnlucky():

- transition: Intialize unlucky to be true. Create a copy of game(2D array) and perform all moves on copy of game and if they are equal unlucky remains same but if they are different unlucky becomes false.
- output: $\mathbb{B}$
- exception: none

toString():

- output: Prints game (2D array) in form of a string.

islucky():

- output: return value of lucky

isunlucky():

- output: return value of unlucky

getGame():

- output: return game (2D array)

Critique Of Design

- As the game only runs on command line, this specification miss out the oppurtunity to provide lively experience.

- This specification doesn't display or store high score and current score of the game, making it difficult for user to track his progress.

- As this specification demands the user to give commands on the console, it becomes difficult to reach the goal that is to achieve a 2048 on any tile.

- The dependence of modules on each other shows lack of low coupling.

- Although this specification obeys the laws of generality there still are some functions like $islucky(), isunlucky()$ that could have been avoided.

UML Diagram For Assignment 3: