Assignment 4, Design Specification

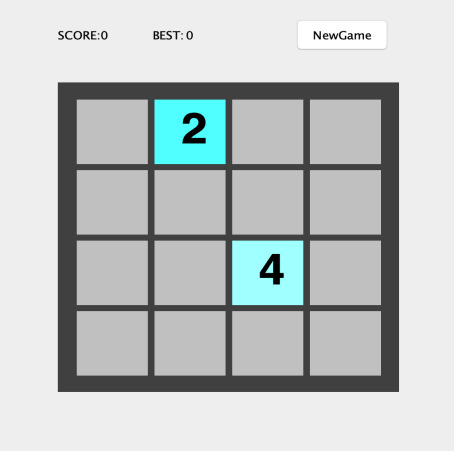
SFWRENG 2AA4

April 12, 2021

This Module Interface Specification (MIS) document contains modules, types and methods for implementing the game *2048*. At the start of the game, there are two numbers displayed on the Board usually randomized between 2 and 4, then the user can perform some operations via up/down/left or right arrow key in order to achieve a tile containing 2048.

The game works on the principle that every time a tile is moved, another tile pops up in a random manner anywhere on the board, when two tiles with the same number on them collide with one another as they are moved, they will merge into one tile with the sum of the numbers written on them initially. The target is always to make a tile having 2048 but if the user wants the game can be continued even after achieving 2048 on any tile.

The game can be launched and played by typing `make game` on Virtual Machine. The test cases could be run by typing `make test` on Virtual Machine.



GUI Module

Module

GUI Module

Uses

JPanel
KeyListener
ActionListener

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
new GUI		self	
TextOnSquare	$\mathbb{N}$		
startgame	GameCal		
keyTyped	KeyEvent		
keyPressed	KeyEvent		
keyReleased	KeyEvent		
actionPerformed	ActionEvent		

Semantics

State Variables

gameover: $\mathbb{N}$
frame: JFrame
calc: GameCal

score: JLabel
best: JLabel
completed: JLabel
newgame: JButton
b: seq [ℕ,ℕ] of JLabel

State Invariant

None

Assumptions

None

Access Routine Semantics

new GUI():

- output: out := self
- transition :=

	:=
b	new JLabel[4][4]
frame	new JFrame()
completed	new JLabel()
score	new JLabel()
best	new JLabel()
newGame	new JButton
y_coordinate	20
x_coordinate	20

  All the display components of GUI can be edited using in-built functions like .setBounds, .setBackground or .setOpaque etc.

- exception := None

TextOnSquare(value, i , j):

- transition := checks for the length of the digits in the tile and depending upon their lengths their font sizes are selected to fit them in the tile. For different values color are chosen by the log function imported via math library.

length of the digit	Font Size
value.length == 1	20
value.length == 2	20
value.length == 3	20
value.length == 4	20
value.length == 5	20

- output: out := None
- exception := None

startgame(GameCal c):

- transition: calc := c
- output:*out* := None
- exception := None

keyTyped(KeyEvent e): *//overriden function*

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

keyPressed(KeyEvent e): *//overriden function*

- transition := checks which key is pressed from the keyboard out of 'w','s','a' and 'd'. *//w is for up , s is for down , a is for left and d is for right.*
- output:*out* := None
- exception := None

keyReleased(KeyEvent e): *//overriden function*

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

actionPerformed(ActionEvent e): *//overriden function*

- transition := spawns a 2 or a 4 when a new game is started or whenever a directional key is pressed.
- output:*out* := None
- exception := None

GameCal Module

Module

GameCal

Uses

None

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
GameCal		new GameCal	
getResult		$\mathbb{N}$	
spawn			
getMaxOnSquare		$\mathbb{N}$	
newGameStart			
allSquareFull		$\mathbb{B}$	
gameCompleted		$\mathbb{B}$	
up			
down			
left			
right			
verticalMove	$\mathbb{N}$,$\mathbb{N}$,String		
horizontalMove	$\mathbb{N}$,$\mathbb{N}$,String		

State Variables

boundry: $\mathbb{N}$
result: $\mathbb{N}$

State Invariant

None

Assumptions

None

Access Routine Semantics

new GameCal():

- output: out := self
- transition : boundry, result := 0, 0
- exception := None

getResult():

- transition := None
- output: out := result
- exception := None

spawn():

- transition := spawns a 2 or a 4 at an empty tile whenever a move is made or whenever it is clicked on new game.
  $x < 0.1 \implies$ a 4 will be spawn || $x \geq 0.1 \implies$ 2 will be spawn
- output: out := None
- exception := None

newGameStart():

- transition := refreshes the game everytime it is clicked on new game

- output: out := None
- exception := None

allSquareFull():

- transition: count := 0
- output:*out* := returns true when all the tiles are filled and there is no move possible and false otherwise.
- exception := None

gameCompleted():

- transition: count := 0
  checks if there is a move possible among any row or column corresponding to a particular tile.
- output:*out* := returns true when any of the tile reaches 2048 and false otherwise.
- exception := None

up():

- transition: boundry := 0
  checks if there is a move possible among any row particularly in up direction . Stops responding when the value or the tile is in the top most row that is the 0th row.
- output:*out* := None
- exception := None

down():

- transition: boundry := 3
  checks if there is a move possible among any row particularly in down direction . Stops responding when the value or the tile is in the bottom most row that is the 3rd row.
- output:*out* := None
- exception := None

left():

- transition: boundry := 0
  checks if there is a move possible among any column particularly in left direction . Stops responding when the value or the tile is in the left most column that is the 0th column.
- output:*out* := None
- exception := None

right():

- transition: boundry := 3
  checks if there is a move possible among any column particularly in right direction. Stops responding when the value or the tile is in the right most column that is the 3rd column.
- output:*out* := None
- exception := None

verticalMove(row,col,direction):

- transition := None
  Compares two tile's values together and if they are the same or if one is equal to 0 (plain tile) - their values are added (provided that the tiles we are comparing are two different tiles and they are moving towards the appropriate direction),this is done through the entire column.
- output:*out* := None
- exception := None

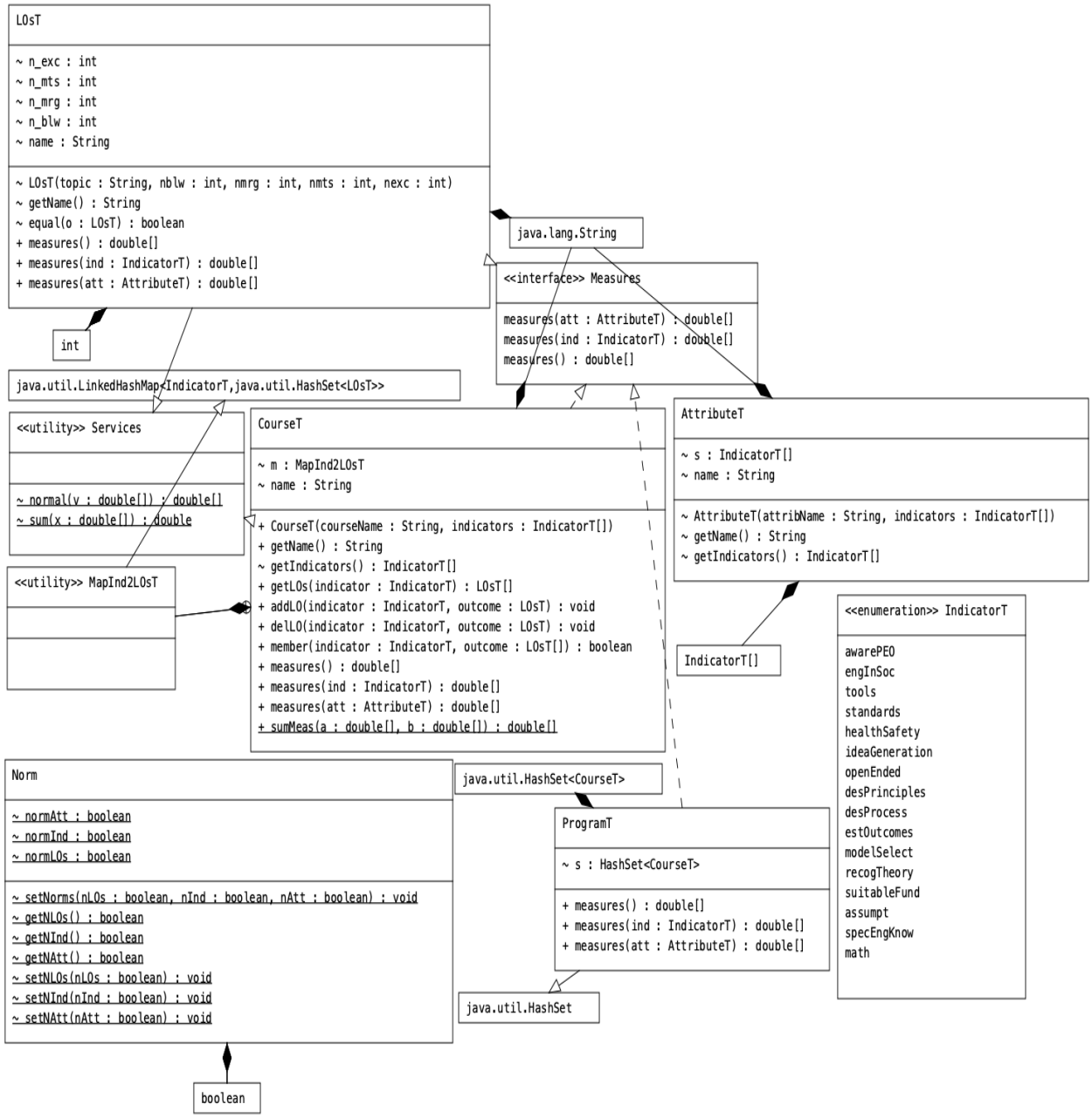
horizontalMove(row,col,direction):

- transition: boundry := 3
  Compares two tile's values together and if they are the same or if one is equal to 0 (plain tile) - their values are added (provided that the tiles we are comparing are two different tiles and they are moving towards the appropriate direction) ,this is done through the entire row.
- output:*out* := None
- exception := None

Critique of Design

- I did not make my GameCal as an ADT module as I thought it could be a little time consuming and complicated to implement although it can be more convenient in terms of essentiality.
- Once the game is played and a best is stored it refreshes when the function is run again although its data could have been stored to give it a moe lively look, but just to save time it as avoided in this specification.
- In the GameCal Module there is function named $getMaxOnSquare()$ although it could have been avoided but it was made in this specification test some edge cases .
- This specification might not be as efficient as it could be because there are certain conditions in the GameCal module that are complicated and might take a little longer time than usual.

UML diagram for modules of Assignment 3:


LOsT
~ n_exc : int
~ n_mts : int
~ n_mrg : int
~ n_blw : int
~ name : String
~ LOsT(topic : String, nblw : int, nmrg : int, nmts : int, nexc : int)
~ getName() : String
~ equal(o : LOsT) : boolean
+ measures() : double[]
+ measures(ind : IndicatorT) : double[]
+ measures(att : AttributeT) : double[]
java.lang.String
<<interface>> Measures
measures(att : AttributeT) : double[]
measures(ind : IndicatorT) : double[]
measures() : double[]
int
java.util.LinkedHashMap<IndicatorT,java.util.HashSet<LOsT>>
<<utility>> Services
~ normal(v : double[]) : double[]
~ sum(x : double[]) : double
CourseT
~ m : MapInd2LOsT
~ name : String
+ CourseT(courseName : String, indicators : IndicatorT[])
+ getName() : String
~ getIndicators() : IndicatorT[]
+ getLOs(indicator : IndicatorT) : LOsT[]
+ addLO(indicator : IndicatorT, outcome : LOsT) : void
+ delLO(indicator : IndicatorT, outcome : LOsT) : void
+ member(indicator : IndicatorT, outcome : LOsT[]) : boolean
+ measures() : double[]
+ measures(ind : IndicatorT) : double[]
+ measures(att : AttributeT) : double[]
+ sumMeas(a : double[], b : double[]) : double[]
AttributeT
~ s : IndicatorT[]
~ name : String
~ AttributeT(attribName : String, indicators : IndicatorT[])
~ getName() : String
~ getIndicators() : IndicatorT[]
<<utility>> MapInd2LOsT
IndicatorT[]
<<enumeration>> IndicatorT
awarePEO
engInSoc
tools
standards
healthSafety
ideaGeneration
openEnded
desPrinciples
desProcess
estOutcomes
modelSelect
recogTheory
suitableFund
assumpt
specEngKnow
math
Norm
~ normAtt : boolean
~ normInd : boolean
~ normLOs : boolean
~ setNorms(nLOs : boolean, nInd : boolean, nAtt : boolean) : void
~ getNLOs() : boolean
~ getNInd() : boolean
~ getNAtt() : boolean
~ setNLOs(nLOs : boolean) : void
~ setNInd(nInd : boolean) : void
~ setNAtt(nAtt : boolean) : void
java.util.HashSet<CourseT>
ProgramT
~ s : HashSet<CourseT>
+ measures() : double[]
+ measures(ind : IndicatorT) : double[]
+ measures(att : AttributeT) : double[]
java.util.HashSet
boolean