Assignment 4, Design Specification

COMPSCI 2ME3

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, an *4x4* grid with 2 values are given. The user can move the board either up, down, left, or right. Depending on the move, the values in the tiles of the board move respectively. like-numbered values present in the tile when bumped into one-another are merged and added. The main objective of game is to get the value "2048" by moving the board in different directions and adding like-numbered values. The Game ends when the board is filled with values except 2048.

The above board visualization is from https://play2048.co/

1 Overview of the design

This design applies Module View Specification (MVC) design pattern. The MVC components are *BoardT* (model module), and *Views* (view module).

The MVC design pattern is specified and implemented in the following way:

The module *BoardT* stores the state of the game board and the status of the game.
A view module *Views* can display the state of the game board and game using a text-based graphics.

For Views, use the getBoard() method to obtain the abstract object.

Likely Changes my design considers:

- Data structure used for storing the game board
- The visual representation of the game such.

Board ADT Module

Template Module inherits EndCondition

BoardT (with EndCondition)

Uses

None

Syntax

Exported Types

None

Exported Constant

Size = 4 // Size of the board 4 x 4

Exported Access Programs

Routine name	In	Out	Exceptions
BoardT		BoardT	
getBoard		seq[size] of seq[size] of $\mathbb{N}$	
getScore		$\mathbb{N}$	
Insert_Random_Beginning			
Insert_Random			
end_condition_2048		$\mathbb{B}$	
end_condition_filled_board		$\mathbb{B}$	
moveUp			
moveDown			
moveLeft			
moveRight			
is_moveUp		$\mathbb{B}$	
is_moveDown		$\mathbb{B}$	
is_moveLeft		$\mathbb{B}$	
is_moveRight		$\mathbb{B}$	

Semantics

State Variables

board: seq[size] of seq[size] of $\mathbb{N}$
score: $\mathbb{N}$

State Invariant

None

Assumptions

- 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 that generates a random value between 0 and 1.

Access Routine Semantics

BoardT():

- transition:

  $$\text{board} := \langle \begin{matrix} \langle 0, 0, 0, 0 \rangle \\ \langle 0, 0, 0, 0 \rangle \\ \langle 0, 0, 0, 0 \rangle \\ \langle 0, 0, 0, 0 \rangle \end{matrix} \rangle$$

  score = 0
- output: *out* := *self*
- exception: None

getBoard():

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

getScore():

- transition: none

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

Insert_Random_Beginning():

- transition: (a,b : $\mathbb{N}$ | a, b, c, d $\in$ [0, size - 1] $\implies$ ((board[a][b] = 2 $\vee$ 4) $\wedge$ (board[c][d] = 2 $\vee$ 4)))

  # todo
- output: *out* := none
- exception: None

Insert_Random():

- transition: (a,b : $\mathbb{N}$ | a, b $\in$ [0, size - 1] $\implies$ (board[a][b] = 2 $\vee$ 4))

  # todo
- output: *out* := None
- exception: None

end_condition_2048():

- transition: None
- output: *out* := ($\forall$ a,b : $\mathbb{N}$ | a, b $\in$ [0, size - 1] : (board[a][b] = 2048))

  # todo
- exception: None

end_condition_filled():

- transition: None
- output: *out* := $\neg$($is\_moveUp() \vee is\_moveDown() \vee is\_moveLeft() \vee is\_moveRight()$)

  # todo

- exception: None

moveUp():

- transition: $(\forall$ a,b : $\mathbb{N}$ | $(1 \leq$ a $<$ size$) \wedge (0 \leq$ b $<$ size$)$: ((board[a - 1][b] = 0) $\Rightarrow$ (board[a - 1][b] =(board[a][b])) $\vee$ ((board[a - 1][b] $\neq$ 0) $\wedge$ (board[a - 1][b] % 2 = 0) $\wedge$ ((board[a - 1][b]) = (board[a][b])) $\Rightarrow$ (board[a - 1][b] += board[a][b])))

  # todo

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

moveDown():

- transition: $(\forall$ a,b : $\mathbb{N}$ | $(0 \leq$ a $<$ size - 1$) \wedge (0 \leq$ b $<$ size$)$: ((board[a + 1][b] = 0) $\Rightarrow$ (board[a + 1][b] =(board[a][b])) $\vee$ ((board[a+1][b] $\neq$ 0) $\wedge$ (board[a+1][b] % 2 = 0) $\wedge$ ((board[a + 1][b]) = (board[a][b])) $\Rightarrow$ (board[a + 1][b] += board[a][b])))

  # todo

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

moveLeft():

- transition: $(\forall$ a,b : $\mathbb{N}$ | $(0 \leq$ a $<$ size$) \wedge (1 \leq$ b $<$ size$)$: ((board[a][b - 1] = 0) $\Rightarrow$ (board[a][b - 1] =(board[a][b])) $\vee$ ((board[a][b - 1] $\neq$ 0) $\wedge$ (board[a][b - 1] % 2 = 0) $\wedge$ ((board[a][b - 1]) = (board[a][b])) $\Rightarrow$ (board[a][b - 1] += board[a][b])))

  # todo

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

moveRight():

- transition: ($\forall$ a,b : $\mathbb{N}$ | (0 $\leq$ a $<$ size) $\wedge$ (0 $\leq$ b $<$ size - 1) : ((board[a][b + 1] = 0) $\Rightarrow$ (board[a][b + 1] =(board[a][b])) $\vee$ ((board[a][b + 1] $\neq$ 0) $\wedge$ (board[a][b + 1] % 2 = 0) $\wedge$ ((board[a][b + 1]) = (board[a][b])) $\Rightarrow$ (board[a][b + 1] += board[a][b])))

  # todo

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

Local Functions

1) is_moveUp : seq[size] of seq[size] of $\mathbb{N} \rightarrow \mathbb{B}$

is_moveUp() $\equiv$ ($\forall$ a,b : $\mathbb{N}$ | (1 $\leq$ a $<$ size) $\wedge$ (0 $\leq$ b $<$ size) : ((board[a - 1][b] = 0) $\vee$ ((board[a - 1][b]) = (board[a][b]))))

2) is_moveDown : seq[size] of seq[size] of $\mathbb{N} \rightarrow \mathbb{B}$

is_moveDown() $\equiv$ ($\forall$ a,b : $\mathbb{N}$ | (0 $\leq$ a $<$ size - 1) $\wedge$ (0 $\leq$ b $<$ size) : ((board[a + 1][b] = 0) $\vee$ ((board[a + 1][b]) = (board[a][b]))))

3) is_moveLeft : seq[size] of seq[size] of $\mathbb{N} \rightarrow \mathbb{B}$

is_moveLeft() $\equiv$ ($\forall$ a,b : $\mathbb{N}$ | (0 $\leq$ a $<$ size) $\wedge$ (1 $\leq$ b $<$ size) : ((board[a][b - 1] = 0) $\vee$ ((board[a][b - 1]) = (board[a][b]))))

4) is_moveRight : seq[size] of seq[size] of $\mathbb{N} \rightarrow \mathbb{B}$

is_moveRight() $\equiv$ ($\forall$ a,b : $\mathbb{N}$ | (0 $\leq$ a $<$ size) $\wedge$ (0 $\leq$ b $<$ size - 1) : ((board[a][b + 1] = 0) $\vee$ ((board[a][b + 1]) = (board[a][b]))))

Views Module

Views Module

Uses

BoardT

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
printBoard	BoardT		

Semantics

Environment Variables

window: A portion of computer screen is used to print the board and the score.

State Variables

visual: UserInterface

State Invariant

None

Access Routine Semantics

printBoard(*board*):

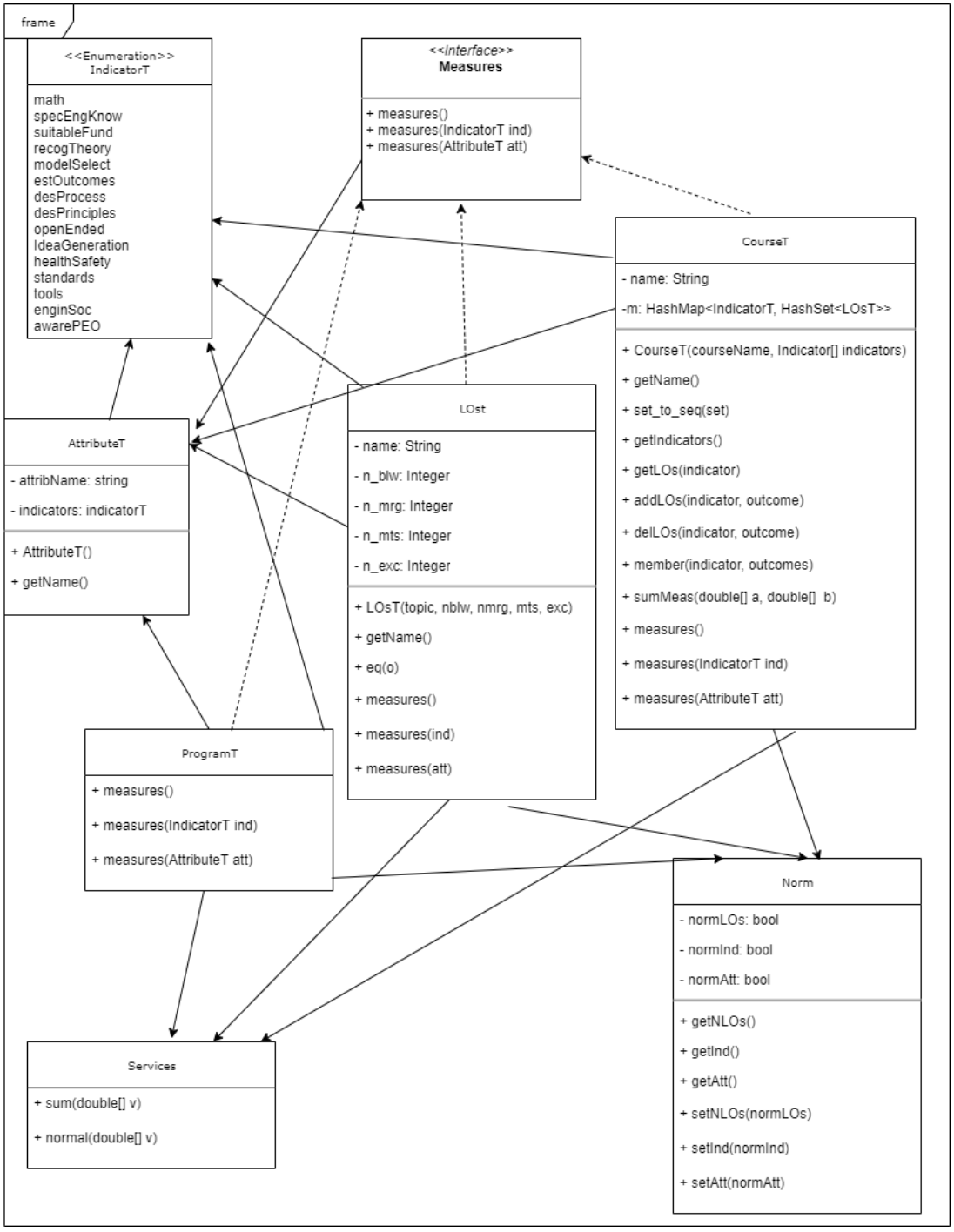
- transition: window := Prints the game board onto the screen. Each element is accessed from *BoardT*. The board is displayed as a grid of dimensions 4x4.

Critique of Design

- The BoardT module is implemented as an ADT over abstract object. this is mainly because it is more convenient to create a new object each time the game is restarted and for unit testing.
- The Views module which represents the View of the MVC design pattern is specefied as an abstract object. This is done so that any unexpected change in the state varibales can be avoided as only one instance of the class is created.
- The local functions is_moveUp(), is_moveDown(), is_moveLeft(), is_moveRight() are not essential we can get the same results from the moveUp(), moveDown(), moveLeft() and moveRight() methods respectively. It was included here to for usability in end_condition_filled_board() method.
- The local functions are made public instead of it usually being private. This was done to make testing of local functions with junit possible.
- The moveUp(), moveDown(), moveLeft() and moveRight() methods in BoardT module violates the principle of minimality as these methods also update the score as well as the state of the board. This was done to make it convenient to calculate the score after each move.
- Test cases generated are used to check the validity and correctness of the the program based on the requirements. Junit was used to impement unit testing due to its friendly and usefull framework.
- Here the MVC design pattern is used. It is decompsed into model and view and exhibits the principle of seperation of concern as all the two modules have different functionality.
- The design used here achieves high cohesion as it related functionalities are grouped with in the same module like the boardT module. low coupling can be observed because each module doesnt depend on each other so a change in my Views model would affect the BoardT module much.

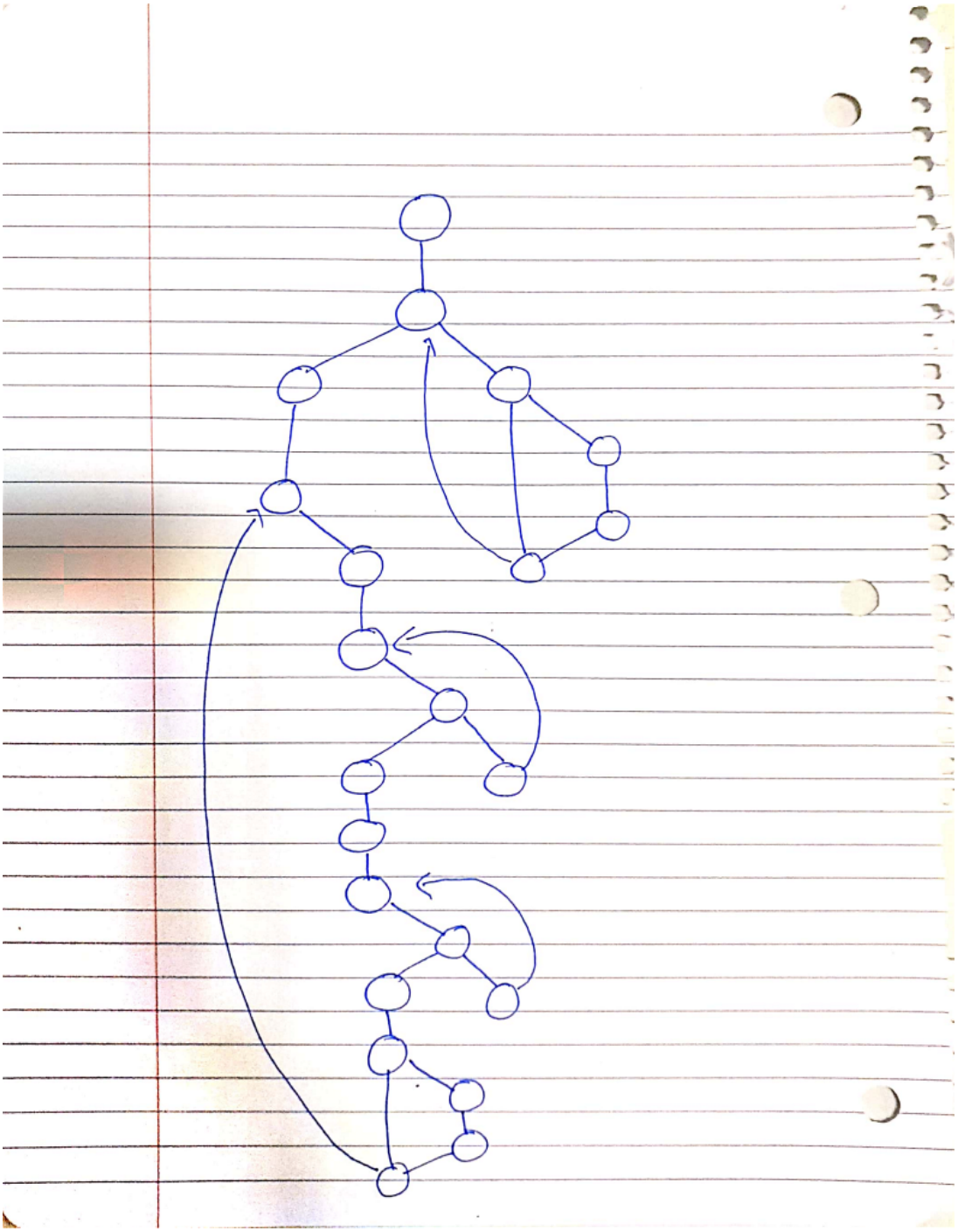
Answers to Questions:

Q1: Draw a UML diagram for the modules in A3.



Q2: Draw a control flow graph for the convex hull algorithm. The graph should follow the

approach used by the Ghezzi et al. textbook. In particular, the code statements should be edges of the graph, not nodes. Code for the convex hull algorithm can be found at: https://startupnextdoor.com/computing-convex-hull-in-python/. To match the diagrams available from Ghezzi, replace the for loop in the code with a while loop.



Citations:

- Code reference: https://github.com/gn03249822/Game-2048%7D%7B https://github.com/gn03249822/Game-2048%7D
- MIS: $https://gitlab.cas.mcmaster.ca/smiths/se2aa4_c s2me3/-/tree/master/Assignments/PreviousYears/2020$