2048 Game - Specification

Rishi Vaya

May 2, 2021

This Module Interface Specification (MIS) document contains modules, types and methods used to support the 2048 Game. The MIS document follows the Model and View portion of the MVC format. The game is played on a BoardT objects which is a grid of the game. the printer method from View can be called on the BoardT object to view the game grid. The functions swipe_left, swipe_right, swipe_up and swipe_down can be used form the Moves module to perform tasks in the game. The game also contains methods to check the score and win or lose status. Since the specification below does not contain a Controller module from the MVC, every function required to develop the module from the Model and View modules are present. These functions can allow for a game with GUI to be built fairly easily.

The game described below contains the exact functionality of the 2048 game with some generality. The specification is divided into a template Module BoardT, which consists of the game grid, score and win/lose status. The Moves module is used to perform the swiping action on the game board, and the View Interface module is used to display the game board on the screen. By adding the Controller module from the MVC format, and using the described modules above, the 2048 game can be reproduced with a user interface.

Likely changes my design considers are:

- A game board of any size can be used. This is done by specifying the desired size when constructing a new BoardT object.
- A game board of a different size (Square or rectangle) can be implemented. This can be done through using a sequence of sequences of integers as an input in the BoardT constructor.
- A pre-set game board can be used. This means the user can enter a board as a sequence of sequences of integers which contains the numbers they require in each cell. This can be passed as an argument to the constructor of the BoardT object.

- The user can play the game in endless mode, meaning the game does not terminate after the value of 2048 has been reached.

Board Module

Template Module

BoardT

Uses

None

Syntax

Exported Constants

None

Exported Types

BoardT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new BoardT	$\mathbb{Z}$	BoardT	
new BoardT	seq of (seq of $\mathbb{Z}$)	BoardT	Illegal Argument Exception
find_empty		seq of (seq of $\mathbb{Z}$)	
equal_board	BoardT, $\mathbb{Z}$	Boolean	
lost			
push_left			
push_right			
push_up			
push_down			
get_score		$\mathbb{Z}$	
get_win		Boolean	
get_lose		Boolean	
reset			

Semantics

State Variables

board : seq of (seq of $\mathbb{Z}$)
board_size : $\mathbb{Z}$
score : $\mathbb{Z}$
win : Boolean
lose : Boolean

State Invariant

None

Assumptions

None

Access Routine Semantics

new BoardT(s):

- transition: $board\_size, score, win, lose, board := s, 0, false, false, \langle i : \mathbb{Z} | 0 \leq i < board\_size : \langle 0 \rangle * board\_size \rangle \wedge board[random(board\_size)][random(board\_size)] := random([2,4]) \wedge board[random(board\_size)][random(board\_size)] := random([2,4])$
- output: *out* := self
- exception: none

new BoardT(*given_board*):

- transition: $board\_size, score, win, lose, board := |given\_board|, 0, false, false, given\_board$
- output: *out* := self
- exception: exc := $\forall(i : seqof\mathbb{Z} | i \in given\_board.(|i| \neq board\_size \Rightarrow$ IllegalArgumentException))

find_empty():

- transition: $win := \exists(\forall(i, x : \mathbb{Z} | 0 \leq i < board\_size \wedge 0 \leq x < board\_size.board[i][x] = 2048))$

- output: $out := \langle(\forall(i, x : \mathbb{Z}|0 \leq i < board\_size \wedge 0 \leq x < board\_size.board[i][x] = 0 \Rightarrow \langle i, x\rangle))\rangle$
- exception: none

equal_board(s, n):

- output: $out := (|board| \neq |s| \Rightarrow False|(+(i, j : \mathbb{N}|i, j \in [0..board\_size] \wedge board[i][j] \neq s[i][j] : 1) > n) \Rightarrow False|True \Rightarrow True)$
- exception: none

lost():

- transition: $lose := (|find\_empty()| > 0 \Rightarrow True|\exists(i, j : \mathbb{N}|i, j \in [0..board\_size - 1].board[i][j] = board[i+1][j] \vee board[i][j] = board[i][j+1]) \Rightarrow True|True \Rightarrow False)$
- exception: none

push_left():

- transition: $board := \forall(i : seqof\mathbb{Z}|i \in board.(j : \mathbb{N}|j \in i \wedge j = 0 : i[0..j-1] \cup i[j+1..board\_size-1] \cup \langle 0\rangle))$
- exception: none

push_right():

- transition: $board := \forall(i : seqof\mathbb{Z}|i \in board.(j : \mathbb{N}|j \in i \wedge j = 0 : \langle 0\rangle \cup i[0..j-1] \cup i[j+1..board\_size-1]))$
- exception: none

push_up():

- transition: $board := (i : \mathbb{N}|i \in [0..board\_size - 1].(j : \mathbb{N}|j \in [0..board\_size - 1].(board[j][i] = 0 \Rightarrow (x : \mathbb{N}|j \leq x < board\_size.(board[j][i] := board[j-1][i]) \wedge board[board\_size - 1][i] := 0))))$
- exception: none

push_down():

- transition: $board := (i : \mathbb{N}|i \in [0..board\_size - 1].(j : \mathbb{N}|j \in [0..board\_size - 1].(board[j][i] = 0 \Rightarrow (x : \mathbb{N}|j \leq x < board\_size.(board[j+1][i] := board[j][i]) \wedge board[0][i] := 0))))$

- exception: none

get_score():

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

get_win():

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

get_lose():

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

reset():

- transition: $score, win, lose, board := 0, false, false, \langle i : \mathbb{Z} | 0 \leq i < board\_size : \langle 0 \rangle * board\_size \rangle$
- exception: none

Moves Module

Module

Moves

Uses

BoardT, Random

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
swipe_left	BoardT		
swipe_right	BoardT		
swipe_up	BoardT		
swipe_down	BoardT		

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

swipe_left(B):

- transition: $B.board := B.push\_left() \wedge \forall(i : seqof\mathbb{Z} | i \in board.(j : \mathbb{N} | 0 \leq j < board\_size - 1 \wedge i[j] = i[j+1] : i[j] := i[j] * 2 \wedge i[j+1] := 0 \wedge B.score := B.score + i[j]) \wedge B.push\_left() \wedge randomizer(B.find\_empty(), B)$
- exception: none

swipe_right(B):

- transition: $B.board := B.push\_right() \wedge \forall(i : seqof\mathbb{Z} | i \in board.(j : \mathbb{N} | 1 \leq j < board\_size \wedge i[j] = i[j-1] : i[j] := i[j] * 2 \wedge i[j-1] := 0 \wedge B.score := B.score + i[j]) \wedge B.push\_right() \wedge randomizer(B.find\_empty(), B)$
- exception: none

swipe_up(B):

- transition: $B.board := B.push\_up() \wedge (i : \mathbb{N} | i \in [0..board\_size - 1].(j : \mathbb{N} | 0 \leq j < board\_size - 1.(board[j][i] = board[j+1][i] \Rightarrow (board[j][i] := board[j][i] * 2 \wedge board[j+1][i] := 0 \wedge B.score := B.score + board[j][i]) \wedge board[board\_size - 1][i] := 0)))) \wedge B.push\_up() \wedge randomizer(B.find\_empty(), B)$
- exception: none

swipe_down(B):

- transition: $B.board := B.push\_down() \wedge (i : \mathbb{N} | i \in [0..board\_size - 1].(j : \mathbb{N} | 1 \leq j < board\_size.(board[j][i] = board[j+1][i] \Rightarrow (board[j][i] := board[j][i] * 2 \wedge board[j+1][i] := 0 \wedge B.score := B.score + board[j][i]) \wedge board[board\_size - 1][i] := 0)))) \wedge B.push\_down() \wedge randomizer(B.find\_empty(), B)$
- exception: none

Local Functions

randomizer: seq of (seq of $\mathbb{N}$)$XBoardT$
randomizer($pick, B$):

- transition: $B.board := (|pick| > 0 \Rightarrow B.board[Random(pick)[0]][Random(pick)[1]] := Random(\langle 2, 4 \rangle))$
- exception: none

View Module

Interface Module

View

Uses

BoardT

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
printer	BoardT		

Semantics

State Variables

window: A portion of the screen to display the game and its messages.

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

printer(B):

- transition: *window* := Draws the board on the screen. This is done by printing a grid of the board and entering the number in each element of the grid with its respective number from B.board. The zero numbers are printed as blanks.
- exception: none

Critique of Design

- The design above is Consistent as all the method and test naming follow the same pattern. Furthermore, the specification of the methods in the access routines are also kept consistent in their mathematical description to allow for easier understanding by the user.

- The above design is follows the concept of essentiality as no method has the functionality of any other method. Some of the methods such as the push methods in BoardT do follow similar logic, but the cannot be compressed to one single method as the logic will become complex and there will be a lot of inputs to take care of.

- The concept of generality was used in the design. This was mainly implemented in the BoardT module where the constructor allows for any board size, instead of the conventional 4 by 4 grid game. This allows for easy updates to the game if required.

- The BoardT module contains 2 different kinds of constructors. One requires an integer as the input where the integer is the size of the game board and the game contains 2 random elements. The other takes in a pre-defined game board as the start of the game. This allows for multiple options for the user.

- The design contains an equal_board method which is not essential, but was implemented due to its importance in testing.

- The design can be considered to follow the concepts of minimality. Although the design breaks down functions to multiple methods, all the methods are unique in their respective functionality. Included with their descriptive names, this allows the user to call individual functions instead of calling a function that performs multiple unnecessary tasks.

- The above design has high cohesion and low coupling due to the use of the MVC design format. This ensures that the model, view and controller are all independent (each module has related functions). It is also low coupling as model, view and controller are only connected as part of the game and do not require each other.

- The design does not implement information hiding completely due to the fact that the BoardT module contains all public elements. This means that any other module can use or modify the components of BoardT. This was done to allow for easy access to the game board in Moves and View, which allowed for fast and efficient updating of the required components.