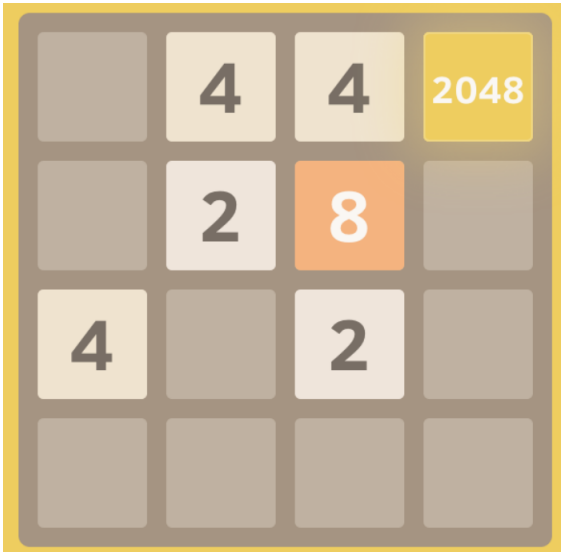
Assignment 4 Specification

SFWR ENG 2AA4

April 12, 2021

This Module Interface Specification (MIS) document contains modules, types and methods used for implementing the game 2048. At the start of the game, a board is created with two tiles in two random cells on the board containing a 2 or a 4 in each. The user must specify the direction that the board shifts in order to merge the tiles that appear. When two tiles merge, their value get added together and leave behind a single tile instead. After a direction has been specified and the tiles have been merged or moved in that direction, a new tile is added in a non-occupied spot containing a 2 or a 4. The possible directions are up ("w"), down ("s"), left ("a"), and right ("d"). This game play will continue until there are no more places to add tiles and there are no possible merges, or when the player merges the tiles to create the value 2048.

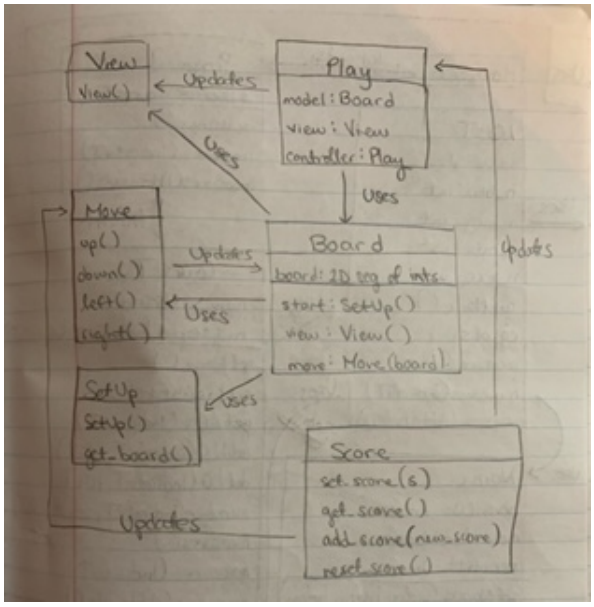


The above board visualization is from http://play2048.co/

Overview of the design

This design applies Module View Specification (MVC) design pattern and Singleton design pattern. The MVC components are *Play* (controller module), *Board* (model module) and *View* (view module). The Singleton pattern is specified and implemented by *Play* and *View*.

A UML diagram is provided below for visualizing the structure of this software architecture:



The MVC design pattern is specified and implemented in the following way: the module *Board* stores the current game board state and the status of the game. It also allows for modifications to the current board state via its methods. A view module *View* can display the state of the game board at a given time using text-based graphics printed to the user's console/screen. The controller *Play* is responsible for handling input actions from the player, as well as displaying the rules and other messages for the user to see and interact with.

For *Play*, they can get the score of the game by using the abstract method $Score.get_score()$ to check if the user has gotten to the value of 2048.

Likely changes my design considers:

Data structure for storing the game board.
Changes to the user input commands (play again ("y"), try again ("t") and keep going ("k")).

Changes to the scoring (input does not have to be power of 2 so could increase/decrease amount of points for high scores)

SetUp Module

Template Module

SetUp

Uses

None

Syntax

Exported Constants

None

Exported Types

SetUp = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new SetUp			

Semantics

State Variables

game_board : grid

State Invariant

None

Assumptions

None

Access Routine Semantics

new SetUp():

- transition: game_board(random_spot() = random_num()), such that there are two unique cells filled once the transition is finished.
- output: *out* := *none*
- exception: none

get_board():

- output: *out* := *gameboard*
- exception: none

Local Functions

random_num: none →N
random_num() ≡ num # *Where num is a random number such that a 2 is 90 percent likely, and 4 is 10 percent likely to occur*

random_entry: none → seq of N
random_entry() ≡ cell # *Where cell is a random set of coordinates from a grid where each space in the grid is equally likely*

Score (Abstract Object)

Module

Score

Uses

None

Syntax

Exported Constants

None

Exported Types

Score = ?

Exported Access Programs

Routine name	In	Out	Exceptions
set_score	$\mathbb{N}$		
get_score		$\mathbb{N}$	
add_score	$\mathbb{N}$		
reset_score			

Semantics

State Variables

score: $\mathbb{N}$

State Invariant

None

Assumptions

The state variable will be set to zero (to start new score for game) before it is modified.
It is also assumed that all inputs will be powers of 2.

Access Routine Semantics

set_score(s):

- transition: $score := s$
- exception: none

get_score( ):

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

add_score(new$_s core$) :

transition: $score := score + new\_score$

out: none

exception: none

reset_score():

- transition: $score := 0$
- out: none
- exception: none

Move

Module

Move

Uses

Score

Syntax

Exported Constants

None

Exported Types

Move = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new Move	grid of $\mathbb{N}$)		
up			
down			
left			
right			

Semantics

State Variables

game_board: grid of $\mathbb{N}$

State Invariant

None

Assumptions

Assumes the Board object that the Move class will be used in will be set up first before engaging in a move.

Access Routine Semantics

Move(board):

- transition: *game_board* := *board*
- out: none
- exception: none

up( ):

- transition: *# Loop through board starting at second highest row and work towards last row. Check if cell above current cell has same value, if so merge(current, above cell). If cell above current cell is* 0*, make above cell value the value of the current cell, current cell value now empty. Then after all cells have been shifted, add_new_block to game_board*
- exception: none

down( ):

- transition: *# Loop through board starting at second lowest row and work towards first row. Check if cell below current cell has same value, if so merge(current, below cell). If cell below current cell is* 0*, make below cell value the value of the current cell, current cell value now empty. Then after all cells have been shifted, add_new_block to game_board*
- exception: none

left( ):

- transition: *# Loop through board starting at second leftmost row and work towards rightmost row. Check if cell to left of current cell has same value, if so merge(current, left cell). If cell left of current cell is* 0*, make left cell value the value of the current cell, current cell value now empty. Then after all cells have been shifted, add_new_block to game_board*

- exception: none

right( ):

- transition: *# Loop through board starting at second rightmost row and work towards leftmost row. Check if cell to right of current cell has same value, if so merge(current, right cell). If cell right of current cell is* 0*, make right cell value the value of the current cell, current cell value now empty. Then after all cells have been shifted, add_new_block to game_board*

- exception: none

Local Functions

merge: seq of $\mathbb{N}$, seq of N $\rightarrow$ *none*
merge(cell1, cell2) $\equiv$ merge *# Where merge changes the cell1 to a value of* 0*, and doubles the value of cell2, while adding the value in cell2 to the Score abstract object*

add_new_block: none $\rightarrow$ *none*
add_new_block() $\equiv$ add *# Where add adds another random number in a random empty spot on the game board, if no spot is free, do not add block*

random_num: none $\rightarrow$N
random_num() $\equiv$ num *# Where num is a random number such that a 2 is 90 percent likely, and 4 is 10 percent likely to occur*

random_entry: none $\rightarrow$ seq of N
random_entry() $\equiv$ cell *# Where cell is a random set of coordinates from a grid where each space in the grid is equally likely*

Board (ADT)

Module

Board

Uses

SetUp, View, Move

Syntax

Exported Constants

None

Exported Types

Board = ?

Exported Access Programs

Routine name	In	Out	Exceptions
start			
view			
move	String		
get_board		seq of (seq of) $\mathbb{N}$	
get_board_value	$\mathbb{N}, \mathbb{N}$	$\mathbb{N}$	

Semantics

State Variables

game_board: sequence [] $\mathbb{N}$

State Invariant

$\text{game}_b oard[i][j] = [4][4]$ *# Size of the board is 4x4*

Assumptions

Move input (direction) assumed to be only "wasd" characters.

Access Routine Semantics

start( ):

- transition: $game\_board := SetUp.get\_board()$
- out: none
- exception: none

view( ):

- transition: *# New instance of View(game_board) to show player the state of the board*
- exception: none

move( ):

- transition: *# New instance of Move(game_board)*

  $(\langle direction = "w" \rightarrow Move.up() | direction = "s" \rightarrow Move.down() | direction = "a" \rightarrow Move.left() | True \rightarrow Move.right())\rangle$
- exception: none

get_board( ):

- out: *game_board*

- exception: none

get_board_value(i, j):

- out: $game\_board[i][j]$
- exception: none

View Module

Module

View

Uses

Score

Syntax

Exported Types

View = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new View	grid of $\mathbb{N}$)		

Semantics

Environmental Variables

window: A portion of the computer screen that displays the game and messages.

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

View(game_board)

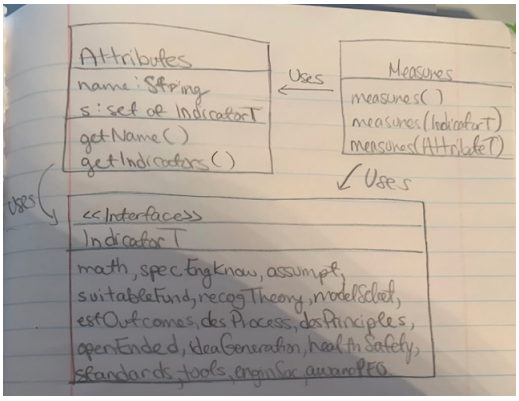
- transition: $window := DisplaysScoreusingScoreabstractobject, displaysallcellsingame\_board$

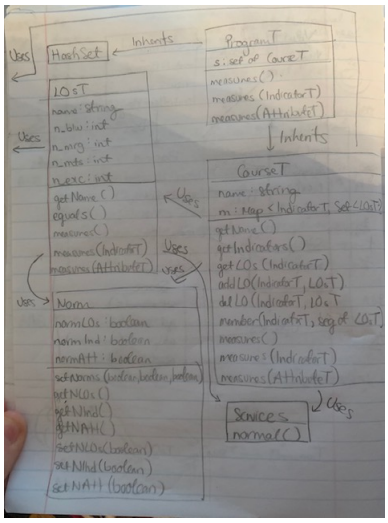
Critique of Design

-

Answers to Questions

1. UML diagram for modules in *A*3





where LOsT and CourseT use Measures (it was hard to try to line up)

2. here too