

Assignment 4, Design Specification

SFWRENG 2AA4

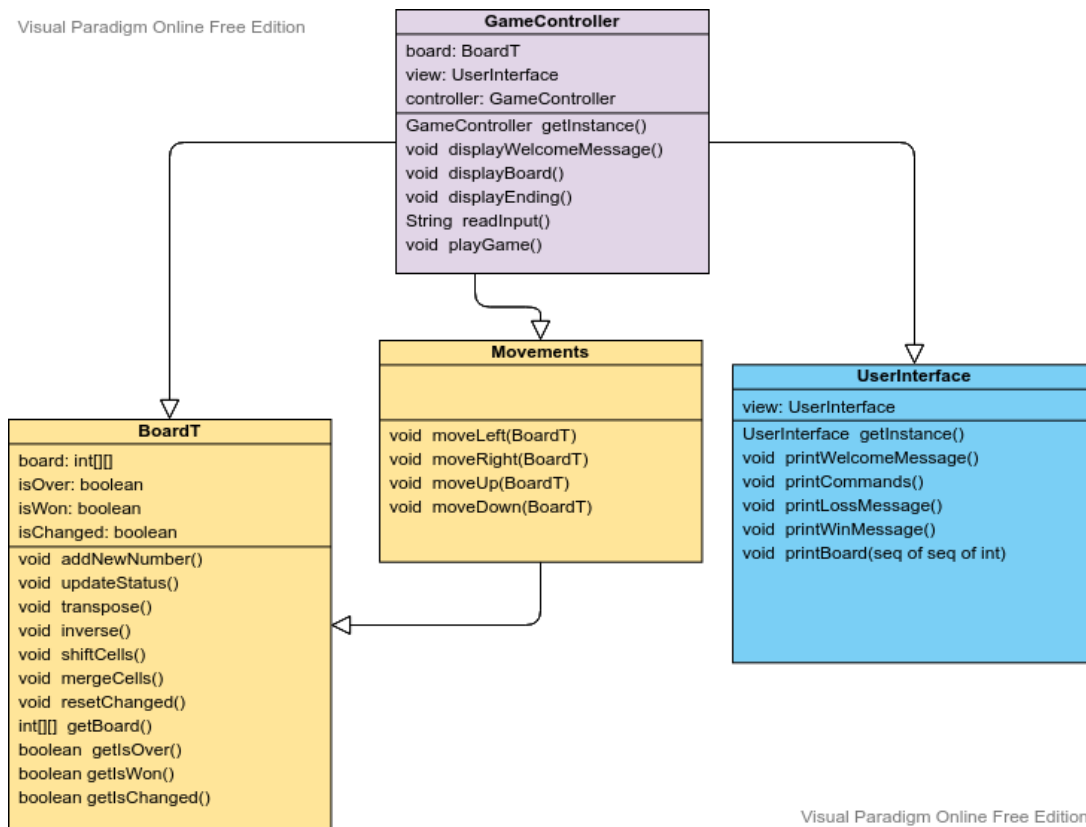
April 16, 2021

This Module Interface Specification (MIS) document contains modules, types and methods for implementing the game *2048*. At the start of each game, the board is randomly filled with two numbers either 2 or 4. The user has four moves in total to play the game. The keyboard keys w, a, s, d are used to move the numbers in the board along the top, left, down and right respectively. The game continues until the entire board is filled up with numbers and cannot merge anymore or if the user achieves the number 2048. The only way to win this game is to achieve 2048 in the board.

1 Overview of the design

This design applies Model-View-Controller (MVC) design pattern and Singleton design pattern. The MVC components are *BoardT*, *Movements* (model module), *UserInterface* (view module) and *GameController* (controller module). Singleton design pattern is specified and implemented for *UserInterface* and *GameController*.

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 *BoardT* stores the state of the game board and the status of the game. The module *Movements* handles all the user input. A view module *UserInterface* displays the state of the game board and game using a test-based graphics. The controller module *GameController* is responsible for handling input actions. The design allows the board size to be anything greater than 3.

For *GameController* and *UserInterface*, use the `getInstance()` method to obtain the abstract object.

Likely Changes my design considers:

Board ADT Module

Template Module

BoardT

Uses

None

Syntax

Exported Types

BoardT = ?

Exported Constant

None

Exported Access Programs

Routine name	In	Out	Exceptions
BoardT	\mathbb{N}	BoardT	IllegalArgumentException
addNewNumber			
updateStatus			
transpose			
inverse			
shiftCells			
resetChanged			
getBoard		seq of (seq of \mathbb{N})	
getIsOver		\mathbb{B}	
getIsWon		\mathbb{B}	
getIsChanged		\mathbb{B}	

State Variables

board: seq of (seq of \mathbb{N})

isOver: \mathbb{B}

isWon: \mathbb{B}

changed: \mathbb{B}

State Invariant

None

Assumptions

- Assume there is a random function that generates a random value between 0 and 1.
- Access routine `addNewNumber` is called two times after constructor is called.

Access Routine Semantics

`new BoardT(size)`:

- transition: `board, isOver, changed` := sequence `[size, size]` of \mathbb{N} , `false`, `false`
- output: `out` := `self`
- exception: `exc` := $(size < 4 \Rightarrow IllegalArgumentException)$

`addNewNumber()`:

- transition: `board` := Adds a number (either 2 or 4) on a random position in the board. The number 2 has a 90% probability of being picked whereas 4 has 10% probability.
- output: `None`
- exception: `None`

`updateStatus()`:

- transition:
$$isWon := \exists(i, j : \mathbb{N} | i, j \in [0..|board| - 1] : board[i][j] = 2048) \Rightarrow true)$$

$$isOver := (\exists(i, j : \mathbb{N} | i, j \in [0..|board| - 1] : board[i][j] = 2048) \Rightarrow true) |$$

$$\exists(i, j : \mathbb{N} | i, j \in [0..|board| - 1] : board[i][j] = board[i + 1][j]) \Rightarrow false) |$$

$$\exists(i, j : \mathbb{N} | i, j \in [0..|board| - 1] : board[i][j] = board[i][j + 1]) \Rightarrow false) |$$

$$\exists(i : \mathbb{N} | i \in [0..|board| - 1] : board[i][|board| - 1] = board[i + 1][|board| - 1]) \Rightarrow false) |$$

$$\exists(i : \mathbb{N} | i \in [0..|board| - 1] : board[|board| - 1][j] = board[|board| - 1][j + 1]) \Rightarrow false) |$$

$$true \Rightarrow true)$$
- output: `None`
- exception: `None`

transpose():

- transition: $\text{board} := \forall(i, j : \mathbb{N} | i, j \in [0..|\text{board}| - 1] : \text{board}[i][j] = \text{board}[j][i])$
- output: None
- exception: None

inverse():

- transition: $\text{board} := \forall(i, j : \mathbb{N} | i, j \in [0..|\text{board}| - 1] : \text{board}[i][j] = \text{board}[i][|\text{board}| - 1 - j])$
- output: None
- exception: None

shiftCells():

- transition: Moves all the numbers present in the board towards their respective left most index that is not filled up with any other number.
- output: None
- exception: None

mergeCells():

- transition:
 $\text{board} := \exists(i, j : \mathbb{N} | i, j \in [0..|\text{board}| - 1] : (\text{board}[i][j] = \text{board}[i][j + 1]) \wedge \text{board}[i][j] \neq 0) \Rightarrow (\text{board}[i][j] = \text{board}[i][j] * 2) \wedge (\text{board}[i][j + 1] = 0)$

 $\text{changed} := \exists(i, j : \mathbb{N} | i, j \in [0..|\text{board}| - 1] : (\text{board}[i][j] = \text{board}[i][j + 1]) \wedge \text{board}[i][j] \neq 0) \Rightarrow \text{true}$
- output: None
- exception: None

resetChanged():

- transition: $\text{changed} := \text{false}$
- output: None
- exception: None

getBoard():

- output: board
- exception: None

getIsOver():

- output: isOver
- exception: None

getIsWon():

- output: isWon
- exception: None

getIsChanged():

- output: changed
- exception: None

Movements Module

Module

Movements

Uses

BoardT

Syntax

Exported Types

None

Exported Constant

None

Exported Access Programs

Routine name	In	Out	Exceptions
moveLeft	BoardT		
moveRight	BoardT		
moveUp	BoardT		
moveDown	BoardT		

State Variables

None

State Invariant

None

Assumptions

- Assume there is already an instance of BoardT called before using any of the access routines.

Access Routine Semantics

`moveLeft(board):`

- transition: Performs actions on the game board when the left button "a" is pressed. This method calls `shiftCells()` to shift all the numbers in the board to the left most index and calls `mergeCells()` to see if any of the number merges or not. It again calls `shiftCells()` in case there are any empty cells in between.
- output: None
- exception: None

`moveRight(board):`

- transition: Performs actions on the game board when the right button "d" is pressed. This method calls `inverse()` to inverse the board and calls `moveLeft()` to perform a left moving action. It again calls `inverse()` to inverse the board to result the action of moving right.
- output: None
- exception: None

`moveUp(board):`

- transition: Performs actions on the game board when the up button "w" is pressed. This method calls the `transpose()` method to interchange the rows and columns of the board and performs a `moveLeft()` method. It again calls the `transpose()` to result in a moving up action.
- output: None
- exception: None

`moveDown(board):`

- transition: Performs actions on the game board when the down button "s" is pressed. This method calls the `transpose()` method to interchange the rows and columns of the board and performs a `moveRight()` method. It again calls the `transpose()` to result in a moving down action.
- output: None
- exception: None

UserInterface Module

Module

UserInterface

Uses

None

Syntax

Exported Types

None

Exported Constant

None

Exported Access Programs

Routine name	In	Out	Exceptions
getInstance		UserInterface	
printWelcomeMessage			
printCommands			
printLossMessage			
printWinMessage			
printBoard	seq of (seq of N)		

State Variables

view: UserInterface

State Invariant

None

Assumptions

- The UserInterface constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once.

Access Routine Semantics

`getInstance()`:

- transition: $\text{view} := (\text{view} = \text{null} \Rightarrow \text{new } \text{UserInterface}())$
- $\text{out} := \text{self}$
- exception: None

`printWelcomeMessage()`:

- transition: Displays a welcome message on the screen when user first enters the game.

`printCommands()`:

- transition: Displays the input commands on the screen to play the game.

`printLossMessage()`:

- transition: Displays the game lost message on the screen when the game ends.

`printWinMessage()`:

- transition: Displays the game won message on the screen when the game ends.

`printBoard(board)`:

- transition: Draws the game board on the screen. The $\text{board}[x][y]$ is displayed in such a way that x is increasing from top of the screen to the bottom, and y value is increasing from left of the screen to the right. For example, $\text{board}[0][0]$ is displayed at the top-left of the screen and $\text{board}[4][4]$ is displayed at the bottom-right of the screen.

Local Function:

`UserInterface`: $\text{void} \rightarrow \text{UserInterface}$

$\text{UserInterface}() \equiv \text{new } \text{UserInterface}()$

GameController Module

Module

GameController

Uses

BoardT, UserInterface

Syntax

Exported Types

None

Exported Constant

None

Exported Access Programs

Routine name	In	Out	Exceptions
getInstance	BoardT, UserInterface	GameController	
displayWelcomeMessage			
displayBoard			
displayEnding			
readInput		String	
playGame			

Semantics

Environment Variables

keyPress: Scanner(System.in) *// reading inputs from keyboard*

State Variables

board: BoardT

view: UserInterface

controller: GameController

State Invariant

None

Assumptions

- The GameController constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once.
- Assume that board and view instances are already initialized before calling GameController constructor

Access Routine Semantics

getInstance(*board*, *view*):

- transition: $\text{controller} := (\text{controller} = \text{null} \Rightarrow \text{new GameController}(\text{board}, \text{view}))$
- output: *self*
- exception: None

displayWelcomeMessage():

- transition: $\text{view} := \text{view.printWelcomeMessage}() \wedge \text{view.printCommands}()$

displayBoard():

- transition: $\text{view} := \text{view.printBoard}(\text{board.getBoard}())$

displayEnding():

- transition: $\text{view} := (\text{board.getIsWon}() \Rightarrow \text{view.printWinMessage}() | \text{true} \Rightarrow \text{view.printLossMessage}())$

readInput():

- output: *input* := String entered from the keyboard by the user

playGame():

- transition: Runs the game in an infinite loop until `board.getIsOver()` is true. Initially runs `displayWelcomeMessage()` and `displayBoard()` to the screen. Then takes input from the user (w, s, a, d) and moves the numbers on the board accordingly. After each user input, the board is displayed and the status of game is updated by running `board.updateStatus()`.

	<i>out</i> :=
<i>input</i> = 'w'	Movements.moveUp(board)
<i>input</i> = 's'	Movements.moveDown(board)
<i>input</i> = 'a'	Movements.moveLeft(board)
<i>input</i> = 'd'	Movements.moveRight(board)

Local Function:

GameController: $\text{BoardT} \times \text{UserInterface} \rightarrow \text{GameController}$

$\text{GameController}(\text{board}, \text{view}) \equiv \text{new GameController}(\text{board}, \text{view})$