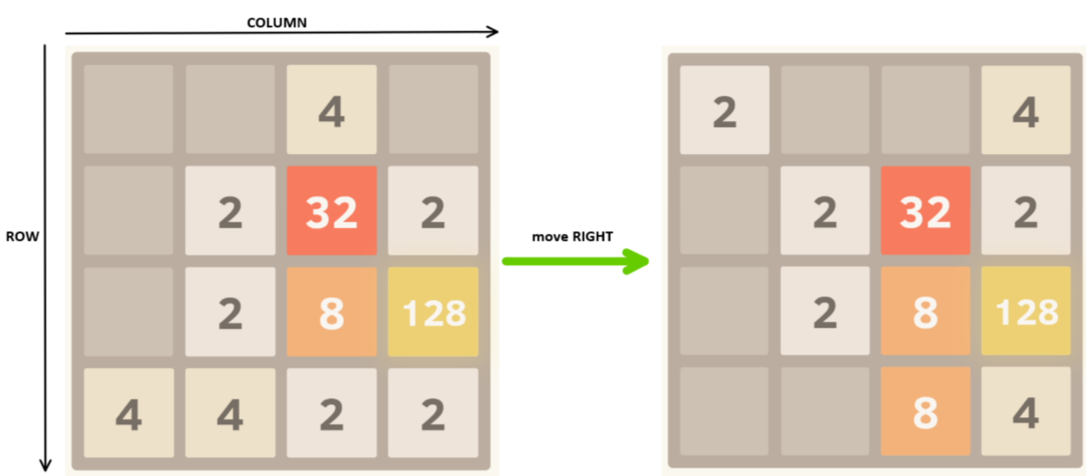
2048, Design Specification

Aamina Hussain

April 16, 2021

This Module Interface Specification (MIS) document contains the modules necessary to implement the model and the view of the game *2048*. At the start of the game, the player will see a 4x4 board containing 16 cells. Two random cells will contain either two `2` tiles, or one `2` tile and one `4` tile. The player can then move the tiles either up, down, left, or right, and the tiles slide over in that direction as far as they can go, unless they are blocked by the edge of the board, or another tile. If the tiles that slide together are the same, then they combine to create a single tile whose value is two times its original value. Therefore, all cells are either empty, or contain a tile whose value is 2^n, where n is a natural number not including 0. After that, a random empty cell will be replaced by a `2` tile. For this implementation, the empty cells will be represented by a `0` tile. A visualization of the game board and an example of a move `right` is shown below:



1 Overview of the design

This design applies Model View Controller (MVC) design pattern, where *BoardT* is the model module and *View* is the view module. The module *BoardT* stores the state of the game board and the status of the game, while the view module *View* displays the current state of the game board and the overall game using text-based (ASCII) graphics. There is no controller module in this implementation.

Likely Changes my design considers:

- The interface that enables the player to play the game.
- The peripheral devices used to take input from the player.
- The data structure used for storing the board.
- The interface that provides a visual display of the game to the player.

Board ADT Module

Template Module

BoardT

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 of (seq of $\mathbb{N}$)	
getScore		$\mathbb{N}$	
getHighScore		$\mathbb{N}$	
emptyCells		set of (seq of $\mathbb{N}$)	
isGameWon		$\mathbb{B}$	
isGameLost		$\mathbb{B}$	
setBoard	seq of (seq of $\mathbb{N}$)		IllegalArgumentException
resetBoard			
isValidMoveRight		$\mathbb{B}$	
isValidMoveLeft		$\mathbb{B}$	
isValidMoveUp		$\mathbb{B}$	
isValidMoveDown		$\mathbb{B}$	
moveRight			
moveLeft			
moveUp			
moveDown			

Semantics

State Variables

board: sequence [size, size] of $\mathbb{N}$
score: $\mathbb{N}$
highscore: $\mathbb{N}$
empty: set of (sequence of $\mathbb{N}$)
win: $\mathbb{B}$
lose: $\mathbb{B}$

State Invariant

$0 \leq |empty| \leq 14$
$0 \leq score \leq highscore$

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.
- *highscore* is a static variable.
- The seq of (seq of $\mathbb{N}$) provided as input for the *setBoard* method will consist of correct numbers. This means that the numbers will be either 0 to represent an empty cell, or 2^n, where $n \neq 0$.
- The set of (sequence of $\mathbb{N}$) used to represent the state variable *empty* will contain the coordinates for the empty cells of the game board. The sequence of $\mathbb{N}$ will only contain two numbers to represent the row and column of an empty cell. For example, *empty* will be a set of $[row, column]$.

Access Routine Semantics

BoardT():

- transition:

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

where two random empty cells are replaced with two `2` tiles or one `2` tile and one `4` tile.

score, win, lose = 0, False, False
empty := $\forall i, j : [0...size-1] | (board[i][j] = 0 \Rightarrow empty \cup \{\langle i, j\rangle\} \mid True \Rightarrow empty - \{\langle i, j\rangle\})$

- output: $out := self$
- exception: None

getBoard():

- output: $out :=$ board
- exception: None

getScore():

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

getHighScore():

- output: $out :=$ highscore
- exception: None

emptyCells():

- output: $out :=$ empty
- exception: None

isGameWon():

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

isGameLost():

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

setBoard(b):

- transition: board := b
- output: None
- exception: $exc := ((\neg\ (|b| = 4) \Rightarrow \text{IllegalArgumentException}) \mid (\forall(i : [0...size-1] \mid \neg(|b[i]| = 4) \Rightarrow \text{IllegalArgumentException})))$

resetBoard():

- transition:
  $$\text{board} := \langle \begin{matrix} \langle 0,0,0,0\rangle \\ \langle 0,0,0,0\rangle \\ \langle 0,0,0,0\rangle \\ \langle 0,0,0,0\rangle \end{matrix} \rangle$$
  where two random empty cells are replaced with two 2 tiles or one 2 tile and one 4 tile.

  score, win, lose = 0, False, False
  empty := $\forall i,j : [0...size-1] | (board[i][j] = 0 \Rightarrow empty \cup \{\langle i,j\rangle\} \mid True \Rightarrow empty - \{\langle i,j\rangle\})$
- output: None
- exception: None

isValidMoveRight():

- output: $\forall i : [0...size-1]\ (\forall j : [0...size-2] \mid (board[i][j] \neq 0 \wedge board[i][j+1] = 0 \Rightarrow True \mid board[i][j] = board[i][j+1] \wedge board[i][j], board[i][j+1] \neq 0 \Rightarrow True \mid True \Rightarrow False))$
- exception: None

isValidMoveLeft():

- output: $\forall i : [0...size-1]\ (\forall j : [size-1...1] \mid (board[i][j] \neq 0 \wedge board[i][j-1] = 0 \Rightarrow True \mid board[i][j] = board[i][j-1] \wedge board[i][j], board[i][j-1] \neq 0 \Rightarrow True \mid True \Rightarrow False))$
- exception: None

isValidMoveUp():

- output: $\forall i : [size - 1...1] \ (\forall j : [0...size - 1] \mid (board[i][j] \neq 0 \land board[i-1][j] = 0 \Rightarrow True \mid board[i][j] = board[i-1][j] \land board[i][j], board[i-1][j] \neq 0 \Rightarrow True \mid True \Rightarrow False))$

- exception: None

isValidMoveDown():

- output: $\forall i : [0...size - 2] \ (\forall j : [0...size - 1] \mid (board[i][j] \neq 0 \land board[i+1][j] = 0 \Rightarrow True \mid board[i][j] = board[i+1][j] \land board[i][j], board[i+1][j] \neq 0 \Rightarrow True \mid True \Rightarrow False))$

- exception: None

moveRight():

- transition:
  board := $\forall i : [0...size - 1] \ (\forall j : [0...size - 2] \mid$

<table>
<tr><td colspan="2"></td><td>transition</td></tr>
<tr><td colspan="2">$board[i][j] \neq 0 \land board[i][j+1] = 0$</td><td>board := shiftRight(i, j, board)</td></tr>
<tr><td rowspan="4">$(board[i][j], board[i][j+1] \neq 0$
$\land board[i][j] = board[i][j+1]$</td><td>$2 \times board[i][j] = 2048 \land$
$\neg$(score $+ 2 \times board[i][j] >$ highscore)</td><td>board := combineRight(i, j, board)
score := score $+ 2 \times board[i][j]$
win := True</td></tr>
<tr><td>$2 \times board[i][j] \neq 2048 \land$
(score $+ 2 \times board[i][j] >$ highscore)</td><td>board := combineRight(i, j, board)
score := score $+ 2 \times board[i][j]$
highscore := score $+ 2 \times board[i][j]$</td></tr>
<tr><td>$2 \times board[i][j] = 2048 \land$
(score $+ 2 \times board[i][j] >$ highscore)</td><td>board := combineRight(i, j, board)
score := score $+ 2 \times board[i][j]$
highscore := score $+ 2 \times board[i][j]$
win := True</td></tr>
<tr><td>True</td><td>board := combineRight(i, j, board)
score := score $+ 2 \times board[i][j]$</td></tr>
</table>

) // *And also replaces one random empty cell (if it exists) with a 2 tile.*
  lose := $\neg$ (isValidMoveRight() $\lor$ isValidMoveLeft() $\lor$ isValidMoveUp $\lor$ isValidMoveDown()) $\Rightarrow True$
  empty := $\forall i, j : [0...size - 1] | (board[i][j] = 0 \Rightarrow empty \cup \{\langle i, j \rangle\} \mid True \Rightarrow empty - \{\langle i, j \rangle\})$

- output: None

- exception: None

moveLeft():

- transition:
  board := $\forall i : [0 ... size-1]$ $(\forall j : [size-1 ... 1]$ |

<table>
<tr><td></td><td></td><td>transition</td></tr>
<tr><td colspan="2">$board[i][j] \neq 0 \wedge board[i][j-1] = 0$</td><td>board := shiftLeft(i, j, board)</td></tr>
<tr><td rowspan="4">$(board[i][j], board[i][j-1] \neq 0$
$\wedge board[i][j] = board[i][j-1]$</td><td>$2 \times board[i][j] = 2048 \wedge$
$\neg(\text{score} + 2 \times board[i][j] > \text{highscore})$</td><td>board := combineLeft(i, j, board)
score := score + $2 \times board[i][j]$
win := True</td></tr>
<tr><td>$2 \times board[i][j] \neq 2048 \wedge$
$(\text{score} + 2 \times board[i][j] > \text{highscore})$</td><td>board := combineLeft(i, j, board)
score := score + $2 \times board[i][j]$
highscore := score + $2 \times board[i][j]$</td></tr>
<tr><td>$2 \times board[i][j] = 2048 \wedge$
$(\text{score} + 2 \times board[i][j] > \text{highscore})$</td><td>board := combineLeft(i, j, board)
score := score + $2 \times board[i][j]$
highscore := score + $2 \times board[i][j]$
win := True</td></tr>
<tr><td>True</td><td>board := combineLeft(i, j, board)
score := score + $2 \times board[i][j]$</td></tr>
</table>

) // *And also replaces one random empty cell (if it exists) with a 2 tile.*
  lose := ¬ (isValidMoveRight() ∨ isValidMoveLeft() ∨ isValidMoveUp ∨ isValidMoveDown()) ⇒ $True$
  empty := $\forall i, j : [0 ... size-1] | (board[i][j] = 0 \Rightarrow empty \cup \{\langle i, j \rangle\} \mid True \Rightarrow empty - \{\langle i, j \rangle\})$

- output: None

- exception: None

moveUp():

- transition:
  board := $\forall i : [size - 1 ... 1]$ ($\forall j : [0 ... size - 1]$ |

<table>
<tr><td colspan="2"></td><td>transition</td></tr>
<tr><td colspan="2">$board[i][j] \neq 0 \wedge board[i-1][j] = 0$</td><td>board := shiftUp(i, j, board)</td></tr>
<tr><td rowspan="4">$(board[i][j], board[i-1][j] \neq 0$
$\wedge board[i][j] = board[i-1][j]$</td><td>$2 \times board[i][j] = 2048 \wedge$
$\neg(\text{score} + 2 \times board[i][j] > \text{highscore})$</td><td>board := combineUp(i, j, board)
score := score + $2 \times board[i][j]$
win := True</td></tr>
<tr><td>$2 \times board[i][j] \neq 2048 \wedge$
$(\text{score} + 2 \times board[i][j] > \text{highscore})$</td><td>board := combineUp(i, j, board)
score := score + $2 \times board[i][j]$
highscore := score + $2 \times board[i][j]$</td></tr>
<tr><td>$2 \times board[i][j] = 2048 \wedge$
$(\text{score} + 2 \times board[i][j] > \text{highscore})$</td><td>board := combineUp(i, j, board)
score := score + $2 \times board[i][j]$
highscore := score + $2 \times board[i][j]$
win := True</td></tr>
<tr><td>True</td><td>board := combineUp(i, j, board)
score := score + $2 \times board[i][j]$</td></tr>
</table>

) // *And also replaces one random empty cell (if it exists) with a 2 tile.*
  lose := $\neg$ (isValidMoveRight() $\vee$ isValidMoveLeft() $\vee$ isValidMoveUp $\vee$ isValidMoveDown()) $\Rightarrow True$
  empty := $\forall i, j : [0 ... size - 1] | (board[i][j] = 0 \Rightarrow empty \cup \{\langle i, j \rangle\} \mid True \Rightarrow empty - \{\langle i, j \rangle\})$

- output: None

- exception: None

moveDown():

- transition:
  board := $\forall i : [0 ... size - 2]$ $(\forall j : [0 ... size - 1]$ |

		transition
$board[i][j] \neq 0 \wedge board[i+1][j] = 0$		board := shiftDown(i, j, board)
$(board[i][j], board[i+1][j] \neq 0$ $\wedge board[i][j] = board[i+1][j]$	$2 \times board[i][j] = 2048 \wedge$ $\neg$(score $+ 2 \times board[i][j] >$ highscore)	board := combineDown(i, j, board) score := score $+ 2 \times board[i][j]$ win := True
	$2 \times board[i][j] \neq 2048 \wedge$ (score $+ 2 \times board[i][j] >$ highscore)	board := combineDown(i, j, board) score := score $+ 2 \times board[i][j]$ highscore := score $+ 2 \times board[i][j]$
	$2 \times board[i][j] = 2048 \wedge$ (score $+ 2 \times board[i][j] >$ highscore)	board := combineDown(i, j, board) score := score $+ 2 \times board[i][j]$ highscore := score $+ 2 \times board[i][j]$ win := True
	True	board := combineDown(i, j, board) score := score $+ 2 \times board[i][j]$

) *// And also replaces one random empty cell (if it exists) with a 2 tile.*
  lose := $\neg$ (isValidMoveRight() $\vee$ isValidMoveLeft() $\vee$ isValidMoveUp $\vee$ isValidMoveDown()) $\Rightarrow True$
  empty := $\forall i, j : [0 ... size - 1] | (board[i][j] = 0 \Rightarrow empty \cup \{\langle i, j \rangle\} \mid True \Rightarrow empty - \{\langle i, j \rangle\})$

- output: None

- exception: None

Local Functions

shiftRight: $\mathbb{N} \times \mathbb{N} \times$ seq of (seq of $\mathbb{N}$) $\rightarrow$ seq of (seq of $\mathbb{N}$)

shiftRight$(i, j, board) \equiv \forall x : [j + 1 ... 0] \mid (x = 0 \Rightarrow board[i][x] = 0 \mid True \Rightarrow board[i][x] = board[i][x-1])$

combineRight: $\mathbb{N} \times \mathbb{N} \times$ seq of (seq of $\mathbb{N}$) $\rightarrow$ seq of (seq of $\mathbb{N}$)

combineRight$(i, j, board) \equiv \forall x : [j + 1 ... 0] \mid (x = j + 1 \Rightarrow board[i][x] = 2 \times board[i][x] \mid x = 0 \Rightarrow board[i][x] = 0 \mid True \Rightarrow board[i][x] = board[i][x-1])$

shiftLeft: $\mathbb{N} \times \mathbb{N} \times$ seq of (seq of $\mathbb{N}$) $\rightarrow$ seq of (seq of $\mathbb{N}$)

shiftLeft$(i, j, board) \equiv \forall x : [j-1 ... size-1] \mid (x = 0 \Rightarrow board[i][x] = 0 \mid True \Rightarrow board[i][x] = board[i][x+1])$

combineLeft: $\mathbb{N} \times \mathbb{N} \times$ seq of (seq of $\mathbb{N}$) $\rightarrow$ seq of (seq of $\mathbb{N}$)

combineLeft$(i, j, board) \equiv \forall x : [j-1 ... size-1] \mid (x = j-1 \Rightarrow board[i][x] = 2 \times board[i][x] \mid x = size-1 \Rightarrow board[i][x] = 0 \mid True \Rightarrow board[i][x] = board[i][x+1])$

shiftUp: $\mathbb{N} \times \mathbb{N} \times$ seq of (seq of $\mathbb{N}$) $\rightarrow$ seq of (seq of $\mathbb{N}$)

shiftUp$(i, j, board) \equiv \forall x : [i-1 ... size-1] \mid (x = size-1 \Rightarrow board[x][j] = 0 \mid True \Rightarrow board[x][j] = board[x+1][j])$

combineUp: $\mathbb{N} \times \mathbb{N} \times$ seq of (seq of $\mathbb{N}$) $\rightarrow$ seq of (seq of $\mathbb{N}$)

combineUp$(i, j, board) \equiv \forall x : [i-1 ... size-1] \mid (x = i-1 \Rightarrow board[x][j] = 2 \times board[x][j] \mid x = size-1 \Rightarrow board[x][j] = 0 \mid True \Rightarrow board[x][j] = board[x+1][j])$

shiftDown: $\mathbb{N} \times \mathbb{N} \times$ seq of (seq of $\mathbb{N}$) $\rightarrow$ seq of (seq of $\mathbb{N}$)

shiftDown$(i, j, board) \equiv \forall x : [i+1 ... 0] \mid (x = 0 \Rightarrow board[x][j] = 0 \mid True \Rightarrow board[x][j] = board[x-1][j])$

combineDown: $\mathbb{N} \times \mathbb{N} \times$ seq of (seq of $\mathbb{N}$) $\rightarrow$ seq of (seq of $\mathbb{N}$)

combineDown$(i, j, board) \equiv \forall x : [i+1 ... 0] \mid (x = i+1 \Rightarrow board[x][j] = 2 \times board[x][j] \mid x = 0 \Rightarrow board[x][j] = 0 \mid True \Rightarrow board[x][j] = board[x-1][j])$

View Module

Module

View

Uses

BoardT

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
printWelcomeMessage			
printBoard	BoardT		
printMovePrompt			
printScore	BoardT		
printHighScore	BoardT		
printLosingMessage			
printWinningMessage			
printFarewellMessage			

Semantics

Environment Variables

terminal: displays the game, messages, and prompts to the player

State Variables

None

State Invariant

None

Access Routine Semantics

printWelcomeMessage():

- transition: terminal := Displays a welcome message when the player begins the game for the first time.

printBoard():

- transition: terminal := Displays the game board. The cells can be accessed by using the *getBoard* method from *BoardT*. The board is displayed as a 4x4 matrix, so each row of cells is on a separate line. The board[x][y] is displayed so that x increases as you go right across the screen, and y increases as you go down the screen.

printMovePrompt():

- transition: terminal := Displays a prompt asking the player to select which direction they want to move the tiles.

printScore(b):

- transition: terminal := Displays the current score of the game, which can be accessed by using the *getScore* method from *BoardT*.

printHighScore(b):

- transition: terminal := Displays the best/highest score of the game, which can be accessed by using the *getHighScore* method from *BoardT*.

printLosingMessage():

- transition: terminal := Displays a message telling the player they lost, then displays a prompt asking the player if they would like to try again or exit the game.

printWinningMessage():

- transition: terminal := Displays a message telling the player they won, then displays a prompt asking the player if they would like to continue the current game, start a new game, or exit the game.

printFarewellMessage():

- transition: terminal := Displays a farewell message after the player decides to exit the game.

Critique of Design

- The BoardT module is an abstract data type instead of an abstract object. This is so multiple instances can be created at a time, and therefore, multiple games can be played. Player A does not have to reset their game if Player B wants to start their own game. Also, this makes it easier to implement the $resetBoard()$ method for if the player would like to reset the game.

- The View module is implemented as a library of methods. This is because the View module only contains methods that print out different messages along with the current state of the board. That means you don't need to create an instance of a View to use the methods, you can just call them when needed.

- The $resetBoard()$ method from the BoardT module is not essential, since instead of resetting the board/game, you can also just create a new instance of the board.

- The $emptyCells()$ method from the BoardT module is also not essential. You can figure out which cells are empty by looking at the board, which you can see by using the $getBoard()$ method.

- The BoardT module is not minimal. This is because $resetBoard()$, $moveRight()$, $moveLeft()$, $moveUp()$, and $moveDown()$ all make changes to multiple state variables at once. For example, $moveRight()$ changes the state of the board, the score, the empty cells, and possibly even win, lose, and the highscore.

- The $setBoard()$ method is mostly for the programmer and can be used for testing purposes. This way the programmer knows what the board looks like, and it is not randomly generated. A randomly generated board makes it difficult to use automated testing.

- My implementation of the $moveRight()$, $moveLeft()$, $moveUp()$, and $moveDown()$ methods were not very abstract or general, since they only account for a 4x4 game board.

- Applying the MVC (or MV in this case) design pattern promotes the maintainability of the interface. It also applies separation of concerns since the model module and view module contain methods mostly independent of each other; the model module handles storing the state and status of the game, while the view module displays the current status of the game.

- The MVC design pattern promotes high cohesion since BoardT and View are closely related. The methods of View are based entirely on the model of the game implemented in BoardT.

- The MVC design pattern also promotes low coupling since the model and view modules are almost entirely independent of each other. This way, changing one module will not greatly affect the other module.