Assignment 4 Specification

SFWR ENG 2AA4

April 13, 2019

This Module Interface Specification (MIS) document contains modules, types and methods for implementing the game state of a game of FreeCell solitaire. The game involves a standard 52-card deck of cards shuffled into 8 piles called Tableau cascades, four having 7 cards each and four having 6 cards each at the beginning. The player has 4 FreeCells where they may choose to place any card as part of their moves. Tableaux can be built by placing alternately-coloured cards of descending suits on top of one another. The goal is to fill all four HomeCells, each stacked from Ace to King of the four different suits. Throughout this document, each of these will be referred to as a different type of "cell", following naming conventions from the following gameboard visualization from http://www.solitairecity.com/Help/FreeCell.shtml:

Life Module

Template Module

Life

Uses

N/A

Syntax

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
Life	String	Life	runtime_error, invalid_argument
display			
writeOutput	String		
update			
simulate	$\mathbb{N}$		

Semantics

State Variables

L: seq of seq of bool

State Invariant

$|L| = 20x20$

Assumptions & Design Decisions

- The Life constructor is called before any other access routine is called on that instance.
- The format of the input textfile must be twenty lines of twenty 1s or 0s separated by spaces.

- The Life board is sized at 20 by 20. There are no cells beyond the board, so cells on the edges are only neighbouring 5 cells, and cells on the corners are only neighbouring 3 cells.

Access Routine Semantics

Life($txtfile$):

- Initializes the Life board object, and it configures the board, L, to the input file $txtfile$. Based on the input file, $txtfile$, $L[i][j]$ is set to true if the character in row i and column j is a 1, and $L[i][j]$ is set to false if the character in row i and column j is a 0.
- exceptions
  - runtime_error: If $txtfile$ cannot be found, throw runtime_error
  - invalid_argument: If $txtfile$ is not in the correct format (too many/little rows or columns, characters not 1 or 0), throw invalid_argument

display():

- Output to console a visual reprentation of the Life board. Format is similar to the input textfile, but with a filled square instead of a 1 and an empty square instead of a 0.

writeOutput($outfile$):

- Creates a textfile with name $outfile$ of the current state of the Life board. Format of $outfile$ follows the same format as input files.

update():

- Updates the entire Life board after one round of life.
  - Each boolean or 'cell' in the board should evaluated based on its current state and the number of 'neighbours' it has to determine its next state.
  - A cell is a neighbour to another cell if it alive and adjacent or diagonal to the cell. A cell is 'alive' if it is true, and it is 'dead' if it is false.
  - A cell that is alive will only stay alive in the next round if it has 2 or 3 neighbours, otherwise it dies. A cell that is dead will become alive in the next round only if it has 3 neighbours.

$L[i][j] = \text{true}$	$neighbours(L[i][j]) = 2\|\|3$	L[i][j] := true
	$neighbours(L[i][j]) \neq 2\|\|3$	L[i][j] := false
$L[i][j] = \text{false}$	$neightbours(L[i][j]) = 3$	L[i][j] := true
	$neightbours(L[i][j]) \neq 3$	L[i][j] := false

simulate(n):

- Calls update() an n amount of times to simulate rounds.

Local Functions

readInput: String
readInput($infile$):

- Configures the board L to the input file. Based on the input file, $txtfile$, $L[i][j]$ is set to true if the character in row i and column j is a 1, and $L[i][j]$ is set to false if the character in row i and column j is a 0. This local function is used in the Life constructor when initializing the Life object, and it is used in update to read and configure from the output file.

inBoard: $\mathbb{N} \times \mathbb{N} \rightarrow$ bool
inBoard(x, y):

- Returns true if $L[x][y]$ is a valid position (within row and column 1 to 20) on the board, and false if it is not a valid position. This local function is used in neighbours to ensure positions outside of the board are not considered.

neighbours: $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$
neighbours(x, y):

- Returns the number of neighbours the cell at $L[x][y]$ has. If there is a cell that is alive and adjacent or diagonal to the cell, then it is a neighbour.

updateCell: $\mathbb{N} \times \mathbb{N} \times$ bool $\rightarrow$ bool
updateCell(x, y, s):

- Returns the next state of the cell $L[x][y]$. s is the current state of $L[x][y]$.

$s = \text{true}$	$neighbours(L[x][y]) = 2\|\|3$	output = true
	$neighbours(L[x][y]) \neq 2\|\|3$	output = false
$s = \text{false}$	$neightbours(L[x][y]) = 3$	output = true
	$neightbours(L[x][y]) \neq 3$	output = false

Critique

Consistency

The Life module is completely consistent in following the rules of the game of life. There are exceptions in place in the readInput function to ensure that the input files follow suit of the Life board. The update function follows strict rules to determine whether a cell will be dead or alive in the next round, so there is no room for the output to be incorrect.

Essentiality

My design is fairly essential, however the update function could be done differently. The update function currently determines the next state and creates an output text file, then it updates the board using readInput from that formed output file. This is somewhat unnecessary as it creates an auxillary file that remains after the function call. Using a temporary 2D array to represent the new board state would be preferably as the temporay board will be gone after the function call.

Generality

I believe that the Life module is somewhat general. The readInput function takes in an input file and configures the board to it. Since this function is general, it is able to be used in multiple functions (the constructor and the update function). However, these functions are made specific to a square board of size 20 by 20. To make it more general, I could have used variables to reprent the number of rows and columns. Also, I believe using a 2D array of integers over booleans would have made my design even more general by allowing the option for more than 2 states of cells.

Minimality

I think that my design is quite minimal since it only involves one module. Each public function represents one task that the Life board can do, and there are a few local functions, like readInput, which make things simpler by avoiding repetition.

Cohesion

This design is very cohesive since it just consists of one module. The functions work together seemlessly to produce the functionality of the Life board. For example, the readInput function is entwined in the update function to read the output function it creates.

Information Hiding

In terms of information hiding, everything is hidden from the user. The only state variable is the 2D array of boolean which represents the board, and this is inaccessible. Everything else that is accessible to the user is meant to be, such as the output textfiles and the display of the board.