

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 The Game of Life. In each state of the game, the cell on the game board can be populated or empty and according to the number of neighbours, the population of cells can be changed in the next state.

File Module

Module

File

Uses

N/A

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
File	string	File	
read		seq of (seq of \mathbb{Z})	
write	seq of (seq of \mathbb{Z})		

Semantics

State Variables

filename

State Invariant

None

Access Routine Semantics

File(*filename*):

- transition: $filename := filename$
- exception: None

read():

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

write(total):

- output: None
- exception: None

Assumptions & Design Decisions

- This module is used to read an input file and output as a matrix or output the state. The characters in the file are 0 or 1 representing if a cell is empty.
- The read() function reads the input file and turn the characters to integers and output as a 2 dimensional sequence.
- The write function takes a 2-dimensional sequence as input and write it into the file. When finish writing one row, it goes to next line.

Local Functions

split: $string \times char \rightarrow (\text{seq of string})$
 split ($str, delimiter$) = $internal$

Game Board Display Module

Module

Display

Uses

N/A

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
display	seq of (seq of \mathbb{Z})		

Semantics

State Variables

None

State Invariant

None

Access Routine Semantics

display(*total*):

- output: None
- exception: None

Assumptions & Design Decisions

- This module can display the state of game board on the console. The state of game board is represented as 2-dimentional sequence of integers. And according to this sequence, the display function can show each cell by printing the ASCII symbols.

Game Board ADT Module

Template Module

Board

Uses

None

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
Board	seq of (seq of \mathbb{Z})	GameBoardT	
get	\mathbb{Z} , \mathbb{Z}	\mathbb{Z}	
next			
toSeq		seq of (seq of \mathbb{Z})	

Semantics

State Variables

None

State Invariant

None

Assumptions & Design Decisions

- The Board constructor is called before any other access routine is called on that instance. Once a Board has been created, the constructor will not be called on it again.
- `get()` is called when checking one of the cell. It takes two integers as the coordinate of a cell and output the integer representing its population.
- The `next()` function is used to move the game board to the next stage. It checks the neighbours for every cell and determines if the cell should be populated in the next state. The rule is according to the population of neighbours.

- The 2-dimentional sequence is used to represent for the game board. `toSeq()` can be used to get the sequence of the board, used to show on the console or write into file.

Access Routine Semantics

`Board(total):`

- transition: $total := total$
- exception: none

`get(row, col):`

- output: $out := total[*row*][*col*]$
- exception: none

`next():`

- output: none

• transition:	$total[i][j] = 1$	$count(i, j) < 2$	$total[i][j] := 0$
		$count(i, j) > 3$	$total[i][j] := 0$
		$count(i, j) = 2$	$total[i][j] := 1$
		$count(i, j) = 3$	$total[i][j] := 1$
	$total[i][j] = 0$	$count(i, j) > 3$	$total[i][j] := 0$
		$count(i, j) < 3$	$total[i][j] := 0$
		$count(i, j) = 3$	$total[i][j] := 1$

- exception: none

`toSeq():`

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

Local Functions

`count: $\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$`

`count (row, col) = sum` such that ($sum = total[*row* - 1][*col* - 1] + total[*row* - 1][*col*] + total[*row* - 1][*col* + 1] + total[*row*][*col* - 1] + total[*row*][*col* + 1] + total[*row* + 1][*col* - 1] + total[*row* + 1][*col*] + total[*row* + 1][*col* + 1]$)