Assignment 4, Specification

SFWR ENG 2AA4

April 8, 2019

This Module Interface Specification (MIS) document contains modules, types and methods for implementing the state of a game called Game of Life.

Point Module

Generic Template Module

point(l)

Uses

N/A

Syntax

Exported Types

point = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new point		point	none
new point	bool	point	none
state		bool	none
turn		point	none

Semantics

State Variables

live: state of point (true or false representing alive or dead)

State Invariant

None

Assumptions & Design Decisions

- The point(live) constructor is called for each object instance before any other access routine is called for that object.
- The point() constructor create a dead point to make the coding for other modules easier to write. Though it violates the essential property of the point object, since this could be achieved by calling point(false), this method is provided as a convenience to write the code.

Access Routine Semantics

new point():

- transition: $live := false$
- output: $out := self$
- exception: none

new point(l):

- transition: $live := l$
- output: $out := self$
- exception: none

state():

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

turn():

- transition: $live := \neg live$
- output: $out := self$
- exception: none

map ADT Module

Template Module

map

Uses

point

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
new map		map	
new map	seq of (seq of point)	map	bad size
new map	name of the file in string	map	invalid argument, bad size
generate		map	
tab	$\mathbb{N}$, $\mathbb{N}$	map	out of range
output		file	invalid_argument

Semantics

State Variables

$board$: seq of (seq of point) # *game board*

State Invariant

$min\_size = 25$
$max\_size = 50$

Assumptions & Design Decisions

- The map constructor is called before any other access routine is called on that instance.
- The points on the edge of the map dies in next generation because of the void that consumes them

- The map(string fn) function reads file that named fn, the file only contains ”0” and ”*” representing $true$ and $false$ respectively. The function throws bad_size, which is a custom exception.

- The $output()$ function is a void type function. However, it does output a file named ”output.txt”, which is the text based graph of the gameboard in this generation.

- The map can be a square or a rectangle. As long as the length of sides are in beween (min_size , max_size)

- There are three functions that can initialize a new map. Though violating the preperty of being essential, it provides different ways get a new board of game. The map(vector(vector(point)) and map(filename) are neccesary because those are the only two way to create a new board with inputs. The map() function is just for users to get a empty board in the easiest way.

Access Routine Semantics

map():

- transition: $board := s$ such that $(|s| = max\_size) \wedge (\forall\, i \in [0...max\_size - 1] : s[i] =$ t such that $(|t| = max\_size \wedge (\forall\, j \in [0...max\_size - 1] : t[j] =$ point(false)

- output: $out := self$

- exception: None

map(*seq_of*(*seq_of*(*point*)) *b*):

- transition: $board := b$

- output: $out := self$

- exception: $exc := (|b| < min\_size) \vee (|b| > max\_size) \vee (|b[0]| < min\_size) \vee (|b[0]| > max\_size) \vee (\exists i \in [0...|b[0]| - 2] : |b[i]| \neq |b[0]|) \Rightarrow$ bad_size

map(*string fn*):

- transition: $board := s$ such that $(|s| =$ num of rows in $fn) \wedge$
  $(\forall\, i \in [0...$ (num of rows in) $fn{-}1] : s[i] = t$ such that $(|t| =$ (num of columns in) $fn \wedge$

  $(\forall\, j \in [0...$ (num of columns in($fn - 1] : ($” $*$ ” $\Rightarrow t[j] =$ point(false) $\vee$ ”0” $\Rightarrow t[j] =$ point(true))

- exception: $exc := |b| < min\_size \lor |b| > max\_size \lor |b[0]| < min\_size \lor |b[0]| > max_size \lor (\exists i \in [0...|b[0]| - 2] : |b[i]| \neq |b[0]|) \Rightarrow$ bad_size

generation():

- transition: $x, y \in \mathbb{N} | \forall x \in [0...|board| - 1] \land \forall y \in [0...|board[0]| - 1]$

$board[x][y] = point(false)$	$x = 0 \lor y = 0 \lor neighbor(board, x, y) <= 2 \lor neighbor(board, x, y) > 3$
$board[x][y] = point(true)$	$board[x][y].neighbor = 3$

- output: $out := self$

- exception: None

tab(x, y):

- transaction: $board[x][y] = board[x][y].turn()$

- output: $out := self$

- exception: $(x < 1 \lor y < 1 \lor x > |board.size| - 1 \lor y > |board[0].size| - 1) \Rightarrow out\_of\_range$

output():

- output: a file named ”output.txt” such that:
  $(\forall\, x \in [0...|board| - 1] \land \forall\, y \in [0...|board[0]| - 1]$:
  $(board[x][y].state() = false \Rightarrow$ write ”*” to file) $\lor$
  $(board[x][y].state() = true \Rightarrow$ write ”0” to file))

- exception: None

Local Types

None

Local Functions

neighbor(board,x , y) : $seq\_of(seq\_of(point)) \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

- output:$+(i, j \in int | \exists i \in [-1, 0, 1] \land \exists j \in [-1, 0, 1] : board[x+i][y+j].state() = true)$

Critique of Design

The $point(live)$ module is not neccesary for the project, since we can just construct map with $vector < vector < bool >>$. However, I think it is neccesary to have a function $turn()$, that can easily change the dead point to alive and the alive point to dead. The $point$ medule also gives client a better clarification how the game works than just using $bool$.