Game of Life

Behdad Khamneli

April, 13, 2019

This Module Interface Specification (MIS) document contains modules, types and methods for implementing the state of a game of John Conway's game of life.

Game Board ADT Module

Template Module

BoardT

Uses

N/A

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
new BoardT	string	BoardT	invalid_argument, out_of_range
is_cell_alive	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	out_of_range
set_cell_alive	$\mathbb{N}$, $\mathbb{N}$		out_of_range
set_cell_dead	$\mathbb{N}$, $\mathbb{N}$		out_of_range
next_state			
getGrid		seq of (seq of $\mathbb{B}$)	

Semantics

Environmental Variables

$FileName$: The input file containing the grid representing the first state of the game.

State Variables

$gird$: seq of (seq of $\mathbb{B}$)

State Invariant

$|grid| = 20 \times 20$

Assumptions & Design Decisions

- The boardT constructor is called before any other access routine is called on that instance. Once a BoardT has been created, the constructor will not be called on it

again.

- The input file is in a grid format with dead cells being 0 and alive cells being 1. The first row and column start from 0.

- The game board's grid has 20 rows and 20 columns.

- For better scalability, this module is specified as an Abstract Data Type (ADT) instead of an Abstract Object. This would allow multiple games to be created and tracked at once by a client.[refrence: Assignment 3 spec]

- The getter function is provided, though violating the property of being essential, to give a would-be view function easy access to the game status. This ensures that the model is able to be easily displayed(output) and integrated with a game system in the future. [reference: Assignment 3 spec]

Access Routine Semantics

BoardT(*FileName*):

- transition: Read the data from the input file associated to the string FileName. Initialize the array grid using the data from the file. The text file contains "0"s and "1"s. All the chars in a row are put together with no spaces. "1" represents an alive cell and "0" represents a dead cell. The following is an example of a valid input file(20×20 grid).

00000010000000000000
00000001000000000000
00000111000000000000
00000000000000000000
.
.
.
.
00000000000000000000
00000000000000000000
00000000000000000000

$grid$:= new seq of (seq of $\mathbb{B}$) such that $(\forall row, col : \mathbb{N} | is\_valid\_range(row, col) : (FileName[row][col] = 1 \Rightarrow grid[row][col] = true \,|\, FileName[row][col] = 0 \Rightarrow grid[row][col] = false))$

- exception: exc := (File does not exist $\Rightarrow$ invalid_argument | $\neg$ is_valid_range(num of row $\in$ File, num of column $\in$ File) $\Rightarrow$ out_of_range)

is_cell_alive(row, col):

- output: $out := grid[row][col]$
- exception: exc := ($\neg$is_valid_range(row,col) $\Rightarrow$ out_of_range)

set_cell_alive(row, col):

- transition: $grid[row][col]$:= true
- exception: exc := ($\neg$is_valid_range(row,col) $\Rightarrow$ out_of_range)

set_cell_dead(row, col):

- transition: $grid[row][col]$:= false
- exception: exc := ($\neg$is_valid_range(row,col) $\Rightarrow$ out_of_range)

next_state():

- transition: $grid$:= new seq of (seq of $\mathbb{B}$) G such that $(\forall row, col : \mathbb{N} \,|\, is\_valid\_range(row, col) : (count\_alive\_around(row, col) = 3 \Rightarrow G[row][col] = true \,|\, count\_alive\_around(row, col) < 2 \Rightarrow G[row][col] = false \,|\, count\_alive\_around(row, col) > 3 \Rightarrow G[row][col] = false))$

getGrid():

- $out := grid$
- exception: $none$

Local Types

Local Functions

is_valid_range : $\mathbb{N} \times \mathbb{N} \to \mathbb{B}$
is_valid_range$(row, col) \equiv (0 \leq row < 20 \wedge 0 \leq col < 20)$

count_alive_around: $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$
count_alive_around(row, col) $\equiv$ point such that
$(+point : \mathbb{N} \mid check\_around\_alive(row, col) : 1)$

check_around_alive: $\mathbb{N} \times \mathbb{N} \to \mathbb{B}$
check_around_alive(row,col) $\equiv (\exists i, j : \mathbb{N} \mid i \in [row - 1..row + 1] \wedge j \in [col - 1..col + 1] :$
$gridCopy[i][j] = true \wedge is\_valid\_range(row + i, col + j) \wedge \neg(i = 0 \wedge j = 0))$
where
$gridCopy \equiv$ new seq of (seq of $\mathbb{B}$) N such that $(\forall i, j : \mathbb{N} \mid is\_valid\_range(i, j) : N[i][j] =$
$grid[i][j])$
#new copy of the grid

Display module

Module

view

Uses

N/A

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
display	seq of (seq of $\mathbb{B}$)		
output	seq of (seq of $\mathbb{B}$), string	new File	

Semantics

Environmental Variables

FileName : The output file for the game grid.

State Variables

none

State Invariant

none

Assumptions & Design Decisions

- The grid being printed can have any size.

Access Routine Semantics

display(grid):

- out: Read the grid and display it on the screen. Print the grid using the following format: for each true in the array print a 1 and for each false in the array print a 0. An example:

00000010010000000000
00000000000010000000
.
.
.
00000000000000000000

- exception: $exc := none$

output(grid, FileName):

- output: $out :=$ new file with the name,FileName. Export the grid's data to the file. For each false in the grid print a 0 to the file and for each true in the grid print a 1 to the file. An example of an output:

00000010010000000000
00000000000010000000
.
.
.
00000000000000000000

- exception: $exc := none$

Local Types

none

Local Functions

none