Assignment 4 Specification

SFWR ENG 2AA4

April 8, 2019

This Module Interface Specification (MIS) document contains modules, types and methods for implementing the game state of a game of John Conway's Game of Life. The game involves a grid of alive and dead cells placed in an initial state by the player. The player may choose where to place the alive cells and dead cells. The game determines the next state by calculations of population and underpopulation, with characteristics such as solitude and overpopulation. 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 https://bitstorm.org/gameoflife/

Grid Size Module

Module

Grid Size

Uses

N/A

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
GridSize	string		GridSize
getRows		$\mathbb{N}$	
getColumns		$\mathbb{N}$	
readDimensions			

Semantics

State Variables

$rows$: $\mathbb{N}$
$columns$: $\mathbb{N}$
$textFile$: $string$

State Invariant

None

Access Routine Semantics

GridSize($grid, textFile$):

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

getRows():

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

getColumns():

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

readDimensions():

- transition: rows, columns := countRows(getline[0..(inBinaryGrid, Line)]) , countColumns(line.length)
- exception: None

Binary Grid Module

Module

BinaryGrid

Uses

N/A

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
BinaryGrid	seq of (seq of char), seq of (seq of char)	BinaryGrid	
toSeq		seq of (seq of char)	
toNextSeq		seq of (seq of char)	

Semantics

State Variables

grid: seq of (seq of char)
nextGrid: seq of (seq of char)

State Invariant

None

Assumptions

The constructor BinaryGrid is called for each object instance. The constructor cannot be called on a non-existing object.

Access Routine Semantics

BinaryGrid($grid, nextGrid$):

- transition: $grid, nextGrid := grid, nextGrid$
- output: $out := self$
- exception: None

toSeq():

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

toNextSeq():

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

Game State Module

Template Module

GameState

Uses

N/A

Syntax

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
GameState	$\mathbb{N}$, $\mathbb{N}$, string	GameState	
getState		seq of (seq of char)	
getNextState		seq of (seq of char)	
initialState			
getAlive		$\mathbb{N}$	
getDead		$\mathbb{N}$	
getNextAlive		$\mathbb{N}$	
getNextDead		$\mathbb{N}$	
aliveNeighbours	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{N}$	
cellRules	$\mathbb{N}$, $\mathbb{N}$		
aliveVector			
nextState			
writeToFile	seq of (seq of char)		

Semantics

State Variables

grid: seq of (seq of char)
liveGrid: seq of (seq of char)
nextGrid: seq of (seq of char)
textFile: string

alive: $\mathbb{N}$
dead: $\mathbb{N}$
rows: $\mathbb{N}$
columns: $\mathbb{N}$

State Invariant

None

Assumptions & Design Decisions

- The GameState constructors is called before any other access routine is called on that instance. Once a GameState has been created, the constructor will not be called on it again.
- The constructor row and column natural numbers are defined before the constructor is called
- The textFile value exists and includes type char values inside of it
- The functions initialState(), aliveVector(), and nextState() are called as a way to build the next game state in the game. These functions are based on the values in the constructor
- This module is created only for the next state of the game, not for infinitely many states

Access Routine Semantics

GameState($rows, columns, string$):

- transition: grid, textFile, liveGrid, nextGrid, alive, dead, rows, columns :=
  grid, textFile, $\emptyset$, $\emptyset$, 0, 0, rows, columns
- output: $out := self$
- exception: None

getState():

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

getNextState():

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

initialState():

- transition: $(grid = \forall i, j \in textFile : temp[i][j] = grid(n, \langle\rangle))$
- exception: None

getAlive():

- transition: $(+alive : \mathbb{N}|alive \in grid \wedge alive =' @' : 1)$
- output: $out := alive$
- exception: None

getDead():

- transition: $(+dead : \mathbb{N}|dead \in grid \wedge dead =' -' : 1)$
- output: $out := dead$
- exception: None

getNextAlive():

- transition: $(+alive : \mathbb{N}|alive \in nextGrid \wedge alive =' @' : 1)$
- output: $out := alive$
- exception: None

getNextDead():

- transition: $(+dead : \mathbb{N}|dead \in nextGrid \wedge dead =' -' : 1)$
- output: $out := dead$
- exception: None

aliveVector():

- transition: $(liveGrid = \forall i, j \in grid : aliveNeighbours(i, j) = liveGrid(n, \langle\rangle))$ $\wedge$ writeToFile(nextGrid)
- exception: None

nextState():

- transition: $(nextGrid = \forall i, j \in grid : cellRules(i, j) = nextGrid(n, \langle\rangle))$
- exception: None

Local Functions

cellRules: $\mathbb{N} \times \mathbb{N} \rightarrow$ (char)
cellRules (i, j) = s such that it follows the liveGrid and rules of the game

aliveNeighbours: $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$
aliveNeighbours (i, j) = s such that $(+alive : \mathbb{N} | alive \in grid[i][j] \wedge alive =' @' : 1)$

writeToFile: seq of (seq of char) → transition
writeToFile (seq of (seq of char)) = transition such that
$(newfile = \forall i, j \in$ seq of (seq of char) : (seq of (seq of char))[i][j] = new file