Assignment 4, Specification

SFWR ENG 2AA4

April 13, 2019

Joshua Guinness, guinnesj, 400134735

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

In applying the specification, there will be cases that involve undefinedness. We will interpret undefinedness following [**?**]:

If $p : \alpha_1 \times .... \times \alpha_n \rightarrow \mathbb{B}$ and any of $a_1, ..., a_n$ is undefined, then $p(a_1, ..., a_n)$ is False. For instance, if $p(x) = 1/x < 1$, then $p(0) =$ False. In the language of our specification, if evaluating an expression generates an exception, then the value of the expression is undefined.

Cell ADT Module

Module

Cell

Uses

N/A

Syntax

Exported Constants

None

Exported Types

Cell = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new Cell		Cell	none
new Cell	boolean	Cell	none
get_life		boolean	none
get_neighbours		int	none
set_life	boolean		none
set_neighbours	int		out_of_range

Semantics

State Variables

S: boolean # *Alive or Dead*
N: int # *Number of neighbors*

State Invariant

$n \leq 8$

Assumptions and Design Decisions

- The Cell(S) or Cell() constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once.

Access Routine Semantics

new Cell():

- transition: $S, N := false, 0$
- output: $out := self$
- exception: none

new Cell(s):

- transition: $S, N := s, 0$
- output: $out := self$
- exception: none

get_life():

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

get_neighbours():

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

set_life(s):

- transition: $S := s$
- output: none
- exception: none

set_neighbours(n):

- transition: $N := n$
- output: none
- exception: $exc := (n > 6 \Rightarrow out\_of\_range)$

Game Board ADT Module

Template Module

BoardT

Uses

Cell
View

Syntax

Exported Constants

None

Exported Types

BoardT

Exported Access Programs

Routine name	In	Out	Exceptions
new BoardT	seq of (seq of C)	BoardT	invalid_argument
next			
view			

Semantics

State Variables

C: seq of (seq of Cell) *#2D Array of Cells*

State Invariant

$|seq\ of\ Cell| = |seq\ of\ (seq\ of\ Cell)|$ *#2D array is a perfect square*

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 seq of (seq of C) that is passed to the constructor is a perfect square. This means that both sequences are of the same length.
- 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.
- The view() function calls the view module which displays the current state of the game.

Access Routine Semantics

new BoardT(c):

- transition: $C := c$
- output: $out := self$
- exception: invalid_argument

next():

- transition: $C := update\_neighbours\_middle(), update\_neighbours\_leftside(), update\_neighbours\_rightside(), update\_neighbours\_top(), update\_neighbours\_bottom(), update\_corners()$ $update\_cells()$
- output: none
- exception: none

view():

- transition: none
- output: none
- exception: none

Local Types

None

Local Functions

update_neighbours_middle:
update_neighbours_middle() $\equiv \forall i : \mathbb{N} \mid i \in [1..|seq\ of\ C|]\ :\ (\forall j : \mathbb{N} \mid j \in [1..|seq\ of\ C|]\ : C[i][j].set\_neighbours(C[i-1][j-1] + C[i-1][j] + C[i-1][j+1] + C[i][j-1] + C[i][j+1] + C[i+1][j-1] + C[i+1][j] + C[i+1][j+1]))$

update_neighbours_leftside:
update_neighbours_leftside() $\equiv \forall i : \mathbb{N}|i \in [1..|seq\ of\ C|-2] : C[i][0].set\_neighbours(C[i-1][0] + C[i-1][1] + C[i][1] + C[i+1][1] + C[i+1][0])$

update_neighbours_rightside:
update_neighbours_rightside() $\equiv \forall i : \mathbb{N}|i \in [1..|seq\ of\ C|-2] : C[i][|seq\ of\ C|-2].set\_neighbours(C[i-1][|seq\ of\ C|-2] + C[i-1][|seq\ of\ C|-3] + C[i][|seq\ of\ C|-3] + C[i+1][|seq\ of\ C|-3] + C[i+1][|seq\ of\ C|-2])$

update_neighbours_top
update_neighbours_top() $\equiv \forall i : \mathbb{N}|j \in [1..|seq\ of\ C|-2] : C[0][j].set\_neighbours(C[0][j-1] + C[1][j-1] + C[1][j] + C[1][j+1] + C[0][j+1|])$

update_neighbours_bottom:
update_neighbours_bottom() $\equiv \forall i : \mathbb{N}|j \in [1..|seq\ of\ C|-2] : C[|seq\ of\ C|-2][j].set\_neighbours(C[|seq\ of\ C|-2][j-1] + C[|seq\ of\ C|-3][j-1] + C[|seq\ of\ C|-3][j] + C[|seq\ of\ C|-3][j+1] + C[|seq\ of\ C|-2][j+1|])$

update_corners:
update_corners() $\equiv$
$C[0][0].set\_neighbours(C[0][1] + C[1][1] + C[1][0])$
$C[0][|seq\ of\ C|-1].set\_neighbours(C[0][|seq\ of\ C|-2]+C[1][|seq\ of\ C|-2]+C[1][|seq\ of\ C|-1])$
$C[|seq\ of\ C|-1][0].set\_neighbours(C[|seq\ of\ C|-2][0]+C[|seq\ of\ C|-2][1]+C[|seq\ of\ C|-1][1])$
$C[|seq\ of\ C|-1][|seq\ of\ C|-1].set\_neighbours(C[|seq\ of\ C|-1][|seq\ of\ C|-2] + C[|seq\ of\ C|-2][|seq\ of\ C|-2] + C[|seq\ of\ C|-2][|seq\ of\ C|-1])$

update_cells:
update_cells() $\equiv$
$\forall x : Cell \,.\, x \in C \wedge x.get\_life = true \mid ((x.get\_neighbours \leq 1 \Rightarrow x.set\_life := false) \vee (x.get\_neighbours \geq 4 \Rightarrow x.set\_life := false) \vee (x.get\_neighbours > 1 \wedge x.get\_neighbours < 4 \Rightarrow x.set\_life := true))$

$\forall x : Cell \,.\, x \in C \wedge x.get\_life = False \mid ((x.get\_neighbours = 3 \Rightarrow x.set\_life := alive) \vee (x.get\_neighbours() > 3 \vee x.get\_neighbours < 3 \Rightarrow x.set\_life() := false))$

Read Module

Module

Read

Uses

BoardT Cell

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
read_state	String	seq of (seq of Cell)	filesystem_error

Semantics

State Variables

None

State Invariant

None

Assumptions and Design Decisions

- The contents of the file are in the right format

Access Routine Semantics

View Module

Module

View

Uses

BoardT Cell

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
view_state	seq of (seq of Cell)		

Semantics

State Variables

None

State Invariant

None

Assumptions and Design Decisions

Access Routine Semantics

Critique of Design