Assignment 3, Part 1, Specification

SFWR ENG 2AA4

April 5, 2019

This Module Interface Specification (MIS) document contains modules, types and methods for implementing the state of a game of Conway's Game of Life. The game involves a grid of size N x N, of cells that can be "ALIVE" or "DEAD". The player can set the start state of the game through a txt file that will be read by the program. The program will then construct the game according to the txt file and display the changes from one game state to the next game state.

Board Types Module

Module

BoardTypes

Uses

N/A

Syntax

Exported Constants

SIZE = 4

Exported Types

cellT = {ALIVE, DEAD}

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

Generic Grid Module

Generic Template Module

Grid(T)

Uses

BoardTypes

Syntax

Exported Types

Grid(T) = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
Seq2D	seq of (seq of T), $\mathbb{R}$	Seq2D	invalid_argument
set	CellT, T		outside_bounds
get	PointT	T	outside_bounds
getNumRow		$\mathbb{N}$	
getNumCol		$\mathbb{N}$	
getScale		$\mathbb{R}$	
count	T	$\mathbb{N}$	
count	LineT, T	$\mathbb{N}$	invalid_argument
count	PathT, T	$\mathbb{N}$	invalid_argument
length	PathT	$\mathbb{R}$	invalid_argument
connected	PointT, PointT	$\mathbb{B}$	invalid_argument

Semantics

State Variables

s: seq of (seq of T)
scale: $\mathbb{R}$

nRow: $\mathbb{N}$
nCol: $\mathbb{N}$

State Invariant

None

Assumptions

- The Seq2D(T) constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once.
- Assume that the input to the constructor is a sequence of rows, where each row is a sequence of elements of type T. The number of columns (number of elements) in each row is assumed to be equal. That is each row of the grid has the same number of entries. $s[i][j]$ means the ith row and the jth column. The 0th row is at the bottom of the map and the 0th column is at the leftmost side of the map.

Access Routine Semantics

Seq2D(S, scl):

- transition: [Fill in the transition. —SS]$s, \text{scale}, \text{nCol}, \text{nRow} := S, \text{scl}, |S[0]|, |S|$
- output: $out := self$
- exception: [Fill in the exception. One should be generated if the scale is less than zero, or the input sequence is empty, or the number of columns is zero in the first row, or the number of columns in any row is different from the number of columns in the first row. —SS]$exc :=$ (scale $\leq 0 \vee |S| = 0 \vee |S[0]| = 0 \Rightarrow$ invalid_argument| $\neg\forall(l : \text{seq of T} | l \in S : |l| = |S[0]|) \Rightarrow$ invalid_argument)

set(p, v):

- transition: [? —SS]$s[p.y][p.x] := v$
- exception: [Generate an exception if the point lies outside of the map. —SS]$exc :=$ ($\neg$validPoint(p) $\Rightarrow$ outside_bounds)

get(p):

- output: [? —SS]$out := s[p.y][p.x]$

- exception: [Generate an exception if the point lies outside of the map. —SS]$exc :=$ ($\neg$validPoint(p) $\Rightarrow$ outside_bounds)

getNumRow():

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

getNumCol():

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

getScale():

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

count(t: T):

- output: [Count the number of times the value t occurs in the 2D sequence. —SS]$out := +(i, j : \mathbb{N} | \text{validRow}(i) \wedge \text{validCol}(j) \wedge s[i][j] = t : 1)$
- exception: None

count(l: LineT, t: T):

- output: [Count the number of times the value t occurs in the line l. —SS]$out :=$ $+(p : \text{PointT} | p \in \text{pointsInLine}(l) \wedge s[p.y][p.x] = t : 1)$
- exception: [Exception if any point on the line lies off of the 2D sequence (map) —SS]$exc :=$ ($\neg$validLine(l) $\Rightarrow$ invalid_argument)

count(pth: PathT, t: T):

- output: [Count the number of times the value t occurs in the path pth. —SS]$out :=$ $+(p : \text{PointT} | p \in \text{pointsInPath}(pth) \wedge s[p.y][p.x] = t : 1)$
- exception: [Exception if any point on the path lies off of the 2D sequence (map) —SS]$exc :=$ ($\neg$validPath(pth) $\Rightarrow$ invalid_argument)

length(pth: PathT):

- output: [Use the scale to find the length of the path. —SS]$out := pth.\text{len} \cdot \text{scale}$
- exception: [Exception if any point on the path lies off of the 2D sequence (map) —SS]$exc := (\neg \text{validPath}(pth) \Rightarrow \text{invalid\_argument})$

connected(p_1: PointT, p_2: PointT):

- output: [Return true if a path exists between p_1 and p_2 with all of the points on the path being of the same value. p_1 and p_2 are considered to be part of the path. —SS] $out := \exists(pth : \text{PathT} | \text{validPath}(pth) \wedge pth.\text{strt} = p_1 \wedge pth.\text{end} = p_2 : \text{count}(pth, s[p_1.y][p_1.x]) = pth.\text{len})$
- exception: [Return an exception if either of the input points is not valid. —SS]$exc := (\neg \text{validPoint}(p_1) \vee \neg \text{validPoint}(p_1) \Rightarrow \text{invalid\_argument})$

Local Functions

validRow: $\mathbb{N} \to \mathbb{B}$
[returns true if the given natural number is a valid row number. —SS]$\text{validRow}(i) \equiv 0 \leq i \leq (\text{nRow} - 1)$

validCol: $\mathbb{N} \to \mathbb{B}$
[returns true if the given natural number is a valid column number. —SS]$\text{validCol}(j) \equiv 0 \leq j \leq (\text{nCol} - 1)$

validPoint: PointT $\to \mathbb{B}$
[Returns true if the given point lies within the boundaries of the map. —SS]$\text{validPoint}(p) \equiv \text{validRow}(p.y) \wedge \text{validCol}(p.x)$

validLine: LineT $\to \mathbb{B}$
[Returns true if all of the points for the given line lie within the boundaries of the map. —SS]$\text{validLine}(l) \equiv \forall(p : \text{PointT} | p \in \text{pointsInLine}(l) : \text{validPoint}(p))$

validPath: PathT $\to \mathbb{B}$
[Returns true if all of the points for the given path lie within the boundaries of the map. —SS]$\text{validPath}(pth) \equiv \forall(p : \text{PointT} | p \in \text{pointsInPath}(pth) : \text{validPoint}(p))$

pointsInLine: LineT $\to$ (set of PointT)

pointsInLine (l) [The same local function as given in the Path module. —SS]

$\equiv \{i : \mathbb{N} | i \in [0..(l.\text{len} - 1)] : l.\text{strt.translate}($
$(l.\text{orient} = \text{W} \Rightarrow -i | l.\text{orient} = \text{E} \Rightarrow i | \text{True} \Rightarrow 0), (l.\text{orient} = \text{N} \Rightarrow i | l.\text{orient} = \text{S} \Rightarrow -i | \text{True} \Rightarrow 0))\}$

pointsInPath: PathT $\rightarrow$ (set of PointT)
[Return the set of points that make up the input path. —SS] pointsInPath(p) $\equiv \cup(i :$
$\mathbb{N} | i \in [0..p.\text{size}] : \text{pointsInLine}(p.\text{line}(i)))$

CardStack Module

Template Module

CardStackT is Stack(cardT)[What should go here? —SS]

Game Board ADT Module

Template Module

BoardT

Uses

BoardTypes

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
new BoardT		BoardT	invalid_argument
is_valid_tab_mv	CategoryT, $\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	out_of_range
is_valid_waste_mv	CategoryT, $\mathbb{N}$	$\mathbb{B}$	invalid_argument, out_of_range
is_valid_deck_mv		$\mathbb{B}$	
tab_mv	CategoryT, $\mathbb{N}$, $\mathbb{N}$		invalid_argument
waste_mv	CategoryT, $\mathbb{N}$		invalid_argument
deck_mv			invalid_argument
get_tab	$\mathbb{N}$	CardStackT	out_of_range
valid_mv_exists		$\mathbb{B}$	
is_win_state		$\mathbb{B}$	

Semantics

State Variables

T: SeqCrdStckT # *Tableau*
F: SeqCrdStckT # *Foundation*
D: CardStackT # *Deck*
W: CardStackT # *Waste*

State Invariant

$|T| = 10$[What goes here? $---SS$]
$|F| = 8$[What goes here? $---SS$]

cnt_cards(T, F, D, W, $\lambda t \rightarrow True$ [What goes here? —SS]) = TOTAL_CARDS
two_decks(T, F, D, W) # *each card appears twice in the combined deck*

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 Foundation stacks must start with an ace, but any Foundation stack can start with any suit. Once an Ace of that suit is placed there, this Foundation stack becomes that type of stack and only those type of cards can be placed there.
- Once a card has been moved to a Foundation stack, it cannot be moved again.
- 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 getter function is provided, though violating the property of being essential, to give a would-be view function easy access to the state of the game. This ensures that the model is able to be easily integrated with a game system in the future. Although outside of the scope of this assignment, the view function could be part of a Model View Controller design pattern implementation (`https://blog.codinghorror.com/understanding-model-view-controller/`)
- A function will be available to create a double deck of cards that consists of a random permutation of two regular decks of cards (TOTAL_CARDS cards total). This double deck of cards can be used to build the game board.

Access Routine Semantics

GameBoard(*deck*):

- transition:

  $T, F, D, W :=$ tab_deck($deck[0..39]$), init_seq(8), CardStackT($deck[40..103]$), CardStackT($\langle\rangle$)

- exception: $exc :=$ ($\neg$two_decks(init_seq(10), init_seq(8), CardStackT(*deck*), CardStackT($\langle\rangle$))) $\Rightarrow$ invalid_argument)

is_valid_tab_mv(c, n_0, n_1):

- output:

	$out :=$
$c =$ Tableau	valid_tab_tab(n_0,n_1)
$c =$ Foundation	valid_tab_foundation(n_0,n_1)
$c =$ Deck	False [What goes here? —SS]
$c =$ Waste	False [What goes here? —SS]

- exception:

	$exc :=$
$c =$ Tableau $\wedge \neg$(is_valid_pos(Tableau, n_0) $\wedge$ is_valid_pos(Tableau, n_1))	out_of_range
$c =$ Foundation $\wedge \neg$(is_valid_pos(Tableau, n_0) $\wedge$ is_valid_pos(Foundation, n_1))	out_of_range

is_valid_waste_mv(c, n):

- output:

	$out :=$
$c =$ Tableau	valid_waste_tab(n)
$c =$ Foundation	valid_waste_foundation(n)
$c =$ Deck	False[What goes here? —SS]
$c =$ Waste	False [What goes here? —SS]

- exception:

	$exc :=$
W.size() $= 0$	invalid_argument
$c =$ Tableau $\wedge \neg$is_valid_pos(Tableau, n)	out_of_range
$c =$ Foundation $\wedge \neg$is_valid_pos(Foundation, n)	out_of_range

is_valid_deck_mv():

- output:

	$out :=$
$c =$ Tableau	False
$c =$ Foundation	False
$c =$ Deck	False
$c =$ Waste	$\lvert D\rvert > 0$

[What goes here? The deck moves involves moving a card from the deck stack to the waste stack. —SS]

- exception: None

tab_mv(c, n_0, n_1):

- transition:

c = Tableau	T[n_0], T[n_1] := T[n_0].pop(),T[n_1].push(T[n_0].top()) [What goes here? —SS]
c = Foundation	T[n_0], F[n_1] := T[n_0].pop(),F[n_1].push(T[n_0].top())[What goes here? —SS]

- exception: exc := ($\neg$is_valid_tab_mv(c, n_0, n_1) $\Rightarrow$ invalid_argument)

waste_mv(c, n):

- transition:

c = Tableau	W, T[n] := W.pop(),T[n].push(W.top())[What goes here? —SS]
c = Foundation	W, F[n] := W.pop(),F[n].push(W.top())[What goes here? —SS]

- exception: exc := ($\neg$is_valid_waste_mv(c, n) $\Rightarrow$ invalid_argument)

deck_mv():

- transition: D, W := D.pop(),W.push(D.top())[What goes here? —SS]
- exception: exc := ($\neg$is_valid_deck_mv() $\Rightarrow$ invalid_argument)

get_tab(i):

- output: $out := T[i]$
- exception: exc : ($\neg$is_valid_pos(Tableau, i) $\Rightarrow$ out_of_range)

get_foundation(i):

- output: $out := F[i]$
- exception: exc : ($\neg$is_valid_pos(Foundation, i) $\Rightarrow$ out_of_range)

get_deck():

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

get_waste():

- output: $out := W$

- exception: None

valid_mv_exists():

- output: *out* := valid_tab_mv $\vee$ valid_waste_mv $\vee$ is_valid_deck_mv() where

  valid_tab_mv $\equiv$ ($\exists c$: CategoryT, n_0 : $\mathbb{N}$, n_1 : $\mathbb{N}$|is_valid_pos($Tableau$,n_0)$\wedge$is_valid_pos(c,n_1) [What goes here? $---SS$] : is_valid_tab_mv(c, n_0, n_1))

  valid_waste_mv $\equiv$ ($\exists c$: CategoryT, n : $\mathbb{N}$|is_valid_pos(c, n)[What goes here? $---SS$] : is_valid_waste_mv(c, n))

- exception: None

is_win_state():

- output: $cnt\_cards\_seq(F, \lambda t \rightarrow$ True) = TOTAL_CARDS [What goes here? —SS]
- exception: None

Local Types

SeqCrdStckT = seq of CardStackT

Local Functions

two_decks : SeqCrdStckT $\times$ SeqCrdStckT $\times$ CardStackT $\times$ CardStackT $\rightarrow \mathbb{B}$
two_decks(T, F, D, W) $\equiv$ [This function returns True if there is two of each card in the game —SS]

($\forall st$: SuitT, rk : RankT|$st \in$ SuitT $\wedge$ $rk \in$ RankT :
($\exists a, b$: $CardT|a, b \in (T[0..9].toSeq()||F[0..7].toSeq()||D.toSeq()||W.toSeq())$: $a.st = b.st \wedge a.rk = b.rk$)[What goes here? $---SS$])

cnt_cards_seq : SeqCrdStckT $\times$ (CardT $\rightarrow \mathbb{B}$) $\rightarrow \mathbb{N}$
cnt_cards_seq(S, f) $\equiv$ ($+s$: CardStackT|$s \in S$: cnt_cards_stack(s, f))

cnt_cards_stack : CardStackT $\times$ (CardT $\rightarrow \mathbb{B}$) $\rightarrow \mathbb{N}$
cnt_cards_stack(S, f) $\equiv$ ($+s$: CardT|$s \in S.toSeq() \wedge f(s) : 1$)
[What goes here? —SS]

cnt_cards : SeqCrdStckT × SeqCrdStckT × CardStackT × CardStackT × (CardT → $\mathbb{B}$) → $\mathbb{N}$
cnt_cards(T, F, D, W, f) ≡ cnt_cards_seq(T, f)+cnt_cards_seq(F, f)+cnt_cards_stack(D, f)+
cnt_cards_stack(W, f)

init_seq : $\mathbb{N}$ → SeqCrdStckT
init_seq(n) ≡ s such that $(|s| = n \wedge (\forall\, i \in [0..n-1] : s[i] = \text{CardStackT}(\langle\rangle))$

tab_deck : (seq of CardT) → SeqCrdStckT
tab_deck$(deck)$ ≡ T such that $(\forall i : \mathbb{N} | i \in [0..9] : T[i].\text{toSeq}() = deck[i * 4...4 * (i + 1) -$
1[What goes here? $---SS$]])

is_valid_pos: CategoryT × $\mathbb{N}$ → $\mathbb{B}$
is_valid_pos(c, n) ≡ $(c = \text{Tableau} \Rightarrow n \in [0..9] | c = \text{Foundation} \Rightarrow n \in [0..7] | \text{True} \Rightarrow \text{True})$

valid_tab_tab: $\mathbb{N} \times \mathbb{N} \to \mathbb{B}$
valid_tab_tab (n_0, n_1) ≡

$T[n_0]$.size() > 0	T$[n_1]$.size() > 0	T$[n_0]$.top().s = T$[n_1]$.top().s ∧ T$[n_0]$.top().r = T$[n_1]$.top().r- 1 [What goes here? —SS]
	T$[n_1]$.size() = 0	True[What goes here? —SS]
T$[n_0]$.size() = 0	T$[n_1]$.size() > 0	False[What goes here? —SS]
	T$[n_1]$.size() = 0	False[What goes here? —SS]

valid_tab_foundation: $\mathbb{N} \times \mathbb{N} \to \mathbb{B}$
valid_tab_foundation(n_0, n_1) ≡

$T[n_0]$.size() > 0	F$[n_1]$.size() > 0	T$[n_0]$.top().s = T$[n_1]$.top().s ∧ T$[n_0]$.top().r = T$[n_1]$.top().r-1
	F$n_1]$.size() = 0	T$[n_0]$.top().r = ACE
T$[n_0]$.size() = 0	F$[n_1]$.size() > 0	False
	F$[n_1]$.size() = 0	False

[What goes here? You may need a table? —SS]

valid_waste_tab: $\mathbb{N} \to \mathbb{B}$
valid_waste_tab (n) ≡

T$[n]$.size() > 0	tab_placeable(W.top(), T$[n]$.top())
T$[n]$.size() = 0	True

valid_waste_foundation: $\mathbb{N} \to \mathbb{B}$
valid_waste_foundation $(n) \equiv$

F[n].size() > 0	foundation_placeable(W.top(), F[n].top())
F[n].size() $= 0$	W.top().r = ACE

tab_placeable: CardT $\times$ CardT $\to \mathbb{B}$
tab_placeable$(a, b) \equiv a.s = b.s \wedge a.r = b.r - 1 \Rightarrow$ True
[Complete this specification —SS]

foundation_placeable: CardT $\times$ CardT $\to \mathbb{B}$
foundation_placeable$(a, b) \equiv a.s = b.s \wedge a.r = b.r - 1 \Rightarrow$ True
[Complete this specification —SS]

Read Module

Module

Read

Uses

BoardTypes

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
load_start_data	s : string		

Semantics

Environment Variables

start_data: File listing game initialization data

State Variables

None

State Invariant

None

Assumptions

The input file will match the given specification.

Access Routine Semantics

load_start_data(s)

- transition: read data from the file stdnt_data associated with the string s. Use this data to initialize a gameboard. Load will first initialize (SALst.init()) before populating SALst with student data that follows the types in StdntAllocTypes.

  The text file has the following format, where id_i, fn_i, ln_i, g_i, gpa_i, $[ch_i^0, ch_i^1, ..., ch_i^{n-1}]$ and fc_i stand for strings that represent the ith student's macid, first name, last name, gender, grade point average, list of choices and free choice, respectively. The gender is represented by either the string "male" or "female." The list of choices comes from strings following the department names in the type DeptT. The list of choices has length n. fc_i is either the string "True" or the string "False." All data values in a row are separated by commas. Rows are separated by a new line. The data shown below is for a total of m students.

$$\begin{array}{cccccccc} id_0, & fn_0, & ln_0, & g_0, & gpa_0, & [ch_0^0, ch_0^1, ..., ch_0^{n-1}], & fc_0 & \\ id_1, & fn_1, & ln_1, & g_1, & gpa_1, & [ch_1^0, ch_1^1, ..., ch_1^{n-1}], & fc_1 & \\ id_2, & fn_2, & ln_2, & g_2, & gpa_2, & [ch_2^0, ch_2^1, ..., ch_2^{n-1}], & fc_2 & (1) \\ ..., & ..., & ..., & ..., & ..., & [..., ...,], & ... & \\ id_{m-1}, & fn_{m-1}, & ln_{m-1}, & g_{m-1}, & gpa_{m-1}, & [ch_{m-1}^0, ch_{m-1}^1, ..., ch_{m-1}^{n-1}], & fc_{m-1} & \end{array}$$

- exception: none

load_dcap_data (s)

- transition: read data from the file dept_capacity associated with the string s. Use this data to update the state of the DCapALst module. Load will first initialize DCapALst (DCapALst.init()) before populating DCapALst with department capacity data.

  The text file has the following format. Each department is identified by a string with the department name, and then a string for the natural number that represents the department's capacity. All data values in a row are separated by commas. Rows are separated by a new line.

civil,	n_{civil}
chemical,	n_{chemical}
electrical,	$n_{\text{electrical}}$
mechanical,	$n_{\text{mechanical}}$
software,	n_{software}
materials,	$n_{\text{materials}}$
engphys,	n_{engphys}

- exception: none

View Module

Module

View

Uses

BoardTypes

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
display	s : BoardT		

Semantics

State Variables

None

State Invariant

None

Assumptions

The input file will match the given specification.

Access Routine Semantics

load_start_data(s)

- displays the state of the gameboard using text-based ASCII graphics. Dead cells will be represented using blank space characters and live cells will be represented using the "o" character.

$$\begin{array}{cccccccc} id_0, & fn_0, & ln_0, & g_0, & gpa_0, & [ch_0^0, ch_0^1, ..., ch_0^{n-1}], & fc_0 & \\ id_1, & fn_1, & ln_1, & g_1, & gpa_1, & [ch_1^0, ch_1^1, ..., ch_1^{n-1}], & fc_1 & \\ id_2, & fn_2, & ln_2, & g_2, & gpa_2, & [ch_2^0, ch_2^1, ..., ch_2^{n-1}], & fc_2 & (2) \\ ..., & ..., & ..., & ..., & ..., & [..., ...,], & ... & \\ id_{m-1}, & fn_{m-1}, & ln_{m-1}, & g_{m-1}, & gpa_{m-1}, & [ch_{m-1}^0, ch_{m-1}^1, ..., ch_{m-1}^{n-1}], & fc_{m-1} & \end{array}$$

- exception: none

Critique of Design

The interface for the modules has rigour and formality. It uses language from discrete math, which has predefined symbols formal syntax, and precise semantics, removing all ambiguity. As well, arguments are checked to see if they are valid, and if not, an exception is called. Modules exercise a proper amount of separation of concerns, with high cohesion and low coupling. All modules are components of a game, so they are closely related. However, modules only call upon each other when necessary. For example, the StackT object does not call upon any other modules , but only provides the necessary functions for itself. The StackT module exhibits generality, as the T value can be of any type. As a result, StackT can be used to represent stacks of other types. Each function only performs one task, so it is clear how each function could be, or will be used. For example, there is a function "is_valid_tab_mv," which checks to make sure the cardT values are correct, and then a "is_valid_pos" function which checks to make sure the arguments are valid, and then a "tab_mv" function which actually moves the values.

A possible improvement would be making the "valid_tab_tab" and other valid functions more general, so that the functions could be reused for other types of solitare . Another related improvement would be the sizes for the tableau and foundation could be a exported constant, so that the number of tableaus and foundation could be changed. Possible considerations would be limiting the size, or setting a constant for a maximum size of the stack of tableaus, as when the game is actually implemented, the tableau will show an image of the cards, and the number of cards in a tableau stack shouldn't be able to extend out of the game window. This will also limit ways the user could use the applications, and prevent possible bugs or errors

[Write a critique of the interface for the modules in this project. Is there anything missing? Is there anything you would consider changing? Why? —SS]