Assignment 4, Specification

SFWR ENG 2AA4

April 12, 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

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 $p(0) = \text{False}$. In the language of our specification, if evaluating an expression generates an exception, then the value of the expression is undefined.

Cell 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

None so far.

Access Routine Semantics

new Cell():

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

new Cell(s):

- $S, N := s, 0$
- 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: $exc := (n > 6 \Rightarrow out\_of\_range)$

6

Generic Stack Module

Generic Template Module

Stack(T)

Uses

N/A

Syntax

Exported Types

Stack(T) = ?

Exported Constants

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new Stack	seq of T	Stack	none
push	T	Stack	none
pop		Stack	out_of_range
top		T	out_of_range
size		$\mathbb{N}$	
toSeq		seq of T	

Semantics

State Variables

S: seq of T

State Invariant

None

Assumptions & Design Decisions

- The Stack(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.
- Though the toSeq() method violates the essential property of the stack object, since this could be achieved by calling top and pop many times, this method is provided as a convenience to the client. In fact, it increases the property of separation of concerns since this means that the client does not have to worry about details of building their own sequence from the sequence of pops.

Access Routine Semantics

new Stack(s):

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

push(e):

- output: $out :=$ new Stack($S \mid\mid \langle e \rangle$)
- exception: none

pop():

- output: $out :=$ new Stack($S[0..|S|-2]$)
- exception: $exc := (|S| = 0 \Rightarrow \text{out\_of\_range})$

top():

- output: $out := S[|S|-1]$
- exception: $exc := (|S| = 0 \Rightarrow \text{out\_of\_range})$

size():

- output: $out := |S|$
- exception: None

toSeq():

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

CardStack Module

Template Module

CardStackT is Stack(CardT)

Game Board ADT Module

Template Module

BoardT

Uses

CardTypes
CardStack

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
new BoardT	seq of CardT	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
get_foundation	$\mathbb{N}$	CardStackT	out_of_range
get_deck		CardStackT	
get_waste		CardStackT	
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

$|T| = 10$
$|F| = 8$
$\text{cnt\_cards}(T,\ F,\ D,\ W,\ \lambda c : \text{True}) = \text{TOTAL\_CARDS}$
$\text{two\_decks}(T,\ F,\ D,\ W)$ # *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

BoardT(*deck*):

- transition:

  $T, F, D, W := \text{tab\_deck}(deck[0..39]), \text{init\_seq}(8), \text{CardStackT}(deck[40..103]), \text{CardStackT}(\langle\rangle)$

- exception: $exc := (\neg\text{two\_decks}(\text{init\_seq}(10), \text{init\_seq}(8), \text{CardStackT}(deck), \text{CardStackT}(\langle\rangle)) \Rightarrow \text{invalid\_argument})$

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
c = Waste	False

- 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
c = Waste	False

- 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* := D.size() > 0
- 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())
c = Foundation	T[n_0], F[n_1] := T[n_0].pop(), F[n_1].push(T[n_0].top())

- 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())
$c =$ Foundation	W, F[n] := W.pop(), F[n].push(W.top())

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

deck_mv():

- transition: D, W := D.pop(), W.push(D.top())
- 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 : $\mathbb{N}|c \in$ {Tableau, Foundation} $\wedge$ is_valid_pos(Tableau, n_0) $\wedge$ is_valid_pos(c, n_1) : is_valid_tab_mv(c, n_0, n_1))

  valid_waste_mv $\equiv$ ($\exists c$: CategoryT, n : $\mathbb{N}|c \in$ {Tableau, Foundation} $\wedge$ is_valid_pos(c, n) : is_valid_waste_mv(c, n))

- exception: None

is_win_state():

- output: $out := (\forall\, i : \mathbb{N} | i \in [0..7] : F[i].\text{size}() > 0 \land F[i].\text{top}().r = \text{KING})$
- exception: None

Local Types

SeqCrdStckT = seq of CardStackT

Local Functions

two_decks : SeqCrdStckT × SeqCrdStckT × CardStackT × CardStackT → $\mathbb{B}$
two_decks$(T, F, D, W) \equiv$

$(\forall st : \text{SuitT}, rk : \text{RankT} | st \in \text{SuitT} \land rk \in \text{RankT} : \text{cnt\_cards}(T, F, D, W, \lambda c : c.s = st \land c.r = rk) = 2)$

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

cnt_cards_stack : CardStackT × (CardT → $\mathbb{B}$) → $\mathbb{N}$
cnt_cards_stack$(s, f) \equiv (+c : \text{CardT} | c \in s.\text{toSeq}() \land f(c) : 1)$

cnt_cards : SeqCrdStckT×SeqCrdStckT×CardStackT×CardStackT×(CardT → $\mathbb{B}$) → $\mathbb{N}$
cnt_cards$(T, F, D, W, f) \equiv$ 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$(n) \equiv s$ such that $(|s| = n \land (\forall\, i \in [0..n-1] : s[i] = \text{CardStackT}(\langle\rangle))$

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

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

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

$T[n_0]$.size() > 0	T$[n_1]$.size() > 0	tab_placeable(T$[n_0]$.top(), T$[n_1]$.top())
	T$[n_1]$.size() $= 0$	True
T$[n_0]$.size() $= 0$	T$[n_1]$.size() > 0	False
	T$[n_1]$.size() $= 0$	False

valid_tab_foundation: $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$
valid_tab_foundation$(n_0, n_1) \equiv$

T$[n_0]$.size() > 0	F$[n_1]$.size() > 0	foundation_placeable(T$[n_0]$.top(), F$[n_1]$.top())
	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

valid_waste_tab: $\mathbb{N} \rightarrow \mathbb{B}$
valid_waste_tab $(n) \equiv$

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

valid_waste_foundation: $\mathbb{N} \rightarrow \mathbb{B}$
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 $\rightarrow \mathbb{B}$
tab_placeable$(c, d) \equiv c.s = d.s \wedge c.r = d.r - 1$

foundation_placeable: CardT $\times$ CardT $\rightarrow \mathbb{B}$
foundation_placeable$(c, d) \equiv c.s = d.s \wedge c.r = d.r + 1$

Critique of Design

[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]

Potential discussion points:

- The stack module provides a toSeq module that violates essentiality. To address this, another module could be built to provide the toSeq service through a function that takes a stack as input and return a sequence.