Assignment 3, Part 1, Specification

SFWR ENG 2AA4

March 5, 2019

This Module Interface Specification (MIS) document contains modules, types and methods for implementing the state of a game of Forty Thieves solitaire.

[The parts that you need to fill in are marked by comments, like this one. In several of the modules local functions are specified. You can use these local functions to complete the missing specifications. —SS]

[As you edit the tex source, please leave the `wss` comments in the file. Put your answer **before** the comment. This will make grading easier. —SS]

Worked with Kanakabha Choudhri for some of the functions.

Card Types Module

Module

CardTypes

Uses

N/A

Syntax

Exported Constants

TOTAL_CARDS = 104
ACE = 1
JACK = 11
QUEEN = 12
KING = 13

Exported Types

SuitT = {Heart, Diamond, Club, Spade}
RankT = [1..13]
CategoryT = {Tableau, Foundation, Deck, Waste}
CardT = tuple of (s: SuitT, r: RankT)

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

Generic Stack Module

Generic Template Module

Stack(T)

Uses

N/A

Syntax

Exported Types

Stack = ?
[What should be written here? —SS]

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: Sequence of T [What is the type of the state variable? —SS]

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 || $\langle e \rangle$)
- exception: none

pop():

- output: $S := S[0..|S| - 2]$ [What should go here? —SS]
- exception: $exc := (|S| = 0 \Rightarrow \text{EMPTY})$ [What should go here? —SS]

top():

- output: $out := S[|S| - 1]$
- exception: $exc := (|S| = 0 \Rightarrow \text{EMPTY})$ [What should go here? —SS]

size():

- output: $out := |S|$ [What should go here? —SS]
- exception: None

toSeq():

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

CardStack Module

Template Module

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

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

$|T| \leq$ TOTAL_CARDS [What goes here? $---SS$]
$|F| \leq$ TOTAL_CARDS [What goes here? $---SS$]
cnt_cards(T, F, D, W, f [What goes here? —SS]) = TOTAL_CARDS

f = $(\forall c : CardT | c \in (T \cup F \cup D \cup W) : True)$

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: $out := 63 > |D| > 0$ [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][|T[n_0]| - 1]) [What goes here? —SS]
$c =$ Foundation	T[n_0], F[n_1] := T[n_0].pop(), F[n_1].push(T[n_0][|T[n_0]| - 1]) [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[|W| - 1]) [What goes here? —SS]
$c =$ Foundation	W, F[n] := W.pop(), F[n].push(W[|W| - 1]) [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[|D| - 1])[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 : $\mathbb{N}|(c : T \wedge n_0 \in [0..9]) \vee (c : F \wedge n_1 \in [0..12])$[What goes here? $---SS$] : is_valid_tab_mv(c, n_0, n_1))

  valid_waste_mv $\equiv$ ($\exists c$: CategoryT, n : $\mathbb{N}|(c : T \wedge n \in [0..63]) \vee (c : F \wedge n \in [0..63])$[What goes here? $---SS$] : is_valid_waste_mv(c, n))

- exception: None

is_win_state():

- output: out := ($\forall s : CardStackT|s \in F : |s| = 13$) [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 :
($\exists n_0, n_1 : CardT|n_0, n_1 \in (T \cup F \cup D \cup W) : (n_0.st = n_1.st) \wedge (n_0.rk = n_1.rk)$))[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$|s \in S : 1)$
[What goes here? —SS]

cnt_cards : SeqCrdStckT $\times$ SeqCrdStckT $\times$ CardStackT $\times$ CardStackT $\times$ (CardT $\rightarrow \mathbb{B}$) $\rightarrow \mathbb{N}$
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 : $\mathbb{N} \to$ SeqCrdStckT
init_seq(n) $\equiv s$ such that $(|s| = n \wedge (\forall\, i \in [0..n-1] : s[i] = \text{CardStackT}(\langle\rangle))$

tab_deck : (seq of CardT) $\to$ SeqCrdStckT
tab_deck($deck$) $\equiv T$ such that $(\forall i : \mathbb{N} | i \in [0..9] : T[i].\text{toSeq}() = deck[4i..4i+3$[What goes here? $---SS$]$])$

is_valid_pos: CategoryT $\times\, \mathbb{N} \to \mathbb{B}$
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	True [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$(n_0, n_1) \equiv$ [What goes here? You may need a table? —SS]

$T[n_0]$.size() > 0	F$[n_1]$.size() > 0	True
	F$[n_1]$.size() $= 0$	True
T$[n_0]$.size() $= 0$	F$[n_1]$.size() > 0	False
	F$[n_1]$.size() $= 0$	False

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

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

valid_waste_foundation: $\mathbb{N} \to \mathbb{B}$
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 $\times$ Card T $\to \mathbb{B}$
$(\forall n_0, n_1 : CardT | n_0, n_1 \in T : (n_0.s = n_1.s) \wedge (n_0.r + 1 = n_1.r))$
[Complete this specification —SS]

foundation_placeable:$(\forall n_0, n_1 : CardT | n_0, n_1 \in F : (n_0.s = n_1.s) \wedge (n_0.r = n_1.r + 1))$
[Complete this specification —SS]

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]

There are many parts in the critique that are redundant and unused within the specification. For example, in the CardTypes Module, in the exported constants only TOTAL_CARDS is used in the other modules. The values given to the face cards ACE, JACK, QUEEN, KING are largely unused and merely serve as a way to clarify to the user where these cards fall in values because they are not straight forward in rank like the number cards. A way to change this to reduce redundancy would be to specify this in the assumptions instead to limit the use of unused exported constants.

This redundancy extends to the Game Board Module. Many of the local functions are unused in the specification, such as the several count card location functions as well as tab_placeable and foundation_placeable. Because they are unused in the exported functions there is little point in keeping them within the specification, and for the interest of clarity can be removed.

Additionally, the Game Board Module is fairly bloated and only keeps track of the state of the game, details regarding whether a move is valid besides checking if the stack is empty are left out (for example, that tableau moves can only be done with the same suit and according descending rank). It may be preferable to further split up the Game Board Module into several modules, for example a module that handles the rules for moving from the tableau, moving from the waste etc. While this would increase the hierarchy and dependency of the Game Board Module on other modules, it would be easier to maintain and make changes to the rules for a given aspect of the game.