

Assignment 3, Part 1, Specification

SFWR ENG 2AA4

April 13, 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]

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 = ?

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		N	
toSeq		seq of T	

Semantics

State Variables

S : seq of CardT [[What goes here? —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: $out := new\ Stack(S \parallel \langle e \rangle)$
- exception: none

`pop()`:

- output: $out := new\ Stack(S[0..|S| - 2])$ [What goes here? —SS]
- exception: $((|S| = 0) \implies out_of_range)$ [What goes here? —SS]

`top()`:

- output: $out := S[|S| - 1]$
- exception: $((|S| = 0) \implies out_of_range)$ [What goes here? —SS]

`size()`:

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

`toSeq()`:

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

CardStack Module

Template Module

CardStackT is Stack(CardT) [\[What goes here? —SS\]](#)

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

$|T| = 10$ [What goes here? —SS]

$|F| = 8$ [What goes here? —SS]

$\text{cnt_cards}(T, F, D, W, f \text{ [What goes here? —SS]}) = \text{TOTAL_CARDS}$

$\text{two_decks}(T, F, D, W) \# \text{ each card appears twice in the combined deck}$

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:

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

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

is_valid_tab_mv(c, n_0, n_1):

- output:

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

- exception:

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

is_valid_waste_mv(c, n):

- output:

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

- exception:

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

is_valid_deck_mv():

- output: $out := \neg(D.\text{size}() = 0)$ [What goes here? —SS]
- exception: None

tab_mv(c, n_0, n_1):

- transition:

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

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

waste_mv(c, n):

- transition:

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

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

deck_mv():

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

get_tab(i):

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

get_foundation(i):

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

get_deck():

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

get_waste():

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

valid_mv_exists():

- output: $out := \text{valid_tab_mv} \vee \text{valid_waste_mv} \vee \text{is_valid_deck_mv}()$ where

$\text{valid_tab_mv} \equiv (\exists c : \text{CategoryT}, n_0 : \mathbb{N}, n_1 : \mathbb{N} | c \in \text{CategoryT} \wedge n_0, n_1 \in [0 \dots 9] [\text{What goes here? —SS}] : \text{is_valid_tab_mv}(c, n_0, n_1))$

$\text{valid_waste_mv} \equiv (\exists c : \text{CategoryT}, n : \mathbb{N} | c \in \text{CategoryT} \wedge n \in [0 \dots 9] [\text{What goes here? —SS}] : \text{is_valid_waste_mv}(c, n))$

- exception: None

is_win_state():

- output: $(\forall i : \mathbb{N} | i \in [0..7] : F[i].\text{top}().r = \text{KING})$ [What goes here? —SS]
- exception: None

Local Types

SeqCrdStckT = seq of CardStackT

Local Functions

two_decks : SeqCrdStckT \times SeqCrdStckT \times CardStackT \times CardStackT $\rightarrow \mathbb{N}$

two_decks(T, F, D, W) \equiv [This function returns True if there is two of each card in the game —SS] [What goes here? —SS]

$(\forall st : \text{SuitT}, rk : \text{RankT} | st \in \text{SuitT} \wedge rk \in \text{RankT} : \text{cnt_cards}(T, F, D, W, \lambda x : (x.s = st) \wedge (x.r = rk)) = 2)$

cnt_cards_seq : SeqCrdStckT \times (CardT $\rightarrow \mathbb{B}$) $\rightarrow \mathbb{N}$

cnt_cards_seq(S, f) $\equiv (+s : \text{CardStackT} | s \in S : \text{cnt_cards_stack}(s, f))$

cnt_cards_stack : CardStackT \times (CardT $\rightarrow \mathbb{B}$) $\rightarrow \mathbb{N}$

cnt_cards_stack(C, f) $\equiv (+x : \text{CardT} | x \in C \wedge f(x) : 1)$ [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) $\equiv \text{cnt_cards_seq}(T, f) + \text{cnt_cards_seq}(F, f) + \text{cnt_cards_stack}(D, f) + \text{cnt_cards_stack}(W, f)$

init_seq : $\mathbb{N} \rightarrow \text{SeqCrdStckT}$

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) $\rightarrow \text{SeqCrdStckT}$

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

is_valid_pos: CategoryT $\times \mathbb{N} \rightarrow \mathbb{B}$

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: $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$

valid_tab_tab (n_0, n_1) \equiv

$T[n_0].size() > 0$	$T[n_1].size() > 0$	tab_placeable($T[n_0].top()$, $T[n_1].top()$) [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: $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$

valid_tab_foundation(n_0, n_1) \equiv [What goes here? You may need a table? —SS]

$T[n_0].size() > 0$	$F[n_1].size() > 0$	foundation_placeable($T[n_0].top()$, $F[n_0].top()$) [What goes here? —SS]
	$F[n_1].size() = 0$	$T[n_0].top().r = \text{ACE}$ [What goes here? —SS]
$T[n_0].size() = 0$	$F[n_1].size() > 0$	False [What goes here? —SS]
	$F[n_1].size() = 0$	False [What goes here? —SS]

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($W.top()$, $F[n].top()$)
$F[n].size() = 0$	$W.top().r = \text{ACE}$

tab_placeable: $\text{CardT} \times \text{CardT} \rightarrow \mathbb{B}$

tab_placeable(c_0, c_1) $\equiv (c_0.s = c_1.s) \wedge (c_0.r + 1 = c_1.r)$ [Complete this specification —SS]

foundation_placeable: $\text{CardT} \times \text{CardT} \rightarrow \mathbb{B}$

[Complete this specification —SS] foundation_placeable(c_0, c_1) $\equiv (c_0.s = c_1.s) \wedge (c_0.r = c_1.r + 1)$

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]

The overall interface of the modules is well designed as it takes all the rules and movements of the card game into consideration. Although, I would add some modules that would enhance the game for players. First, I would include an interface that allows a stack of cards with the same suit and correct rank order to be transferred to the foundation if possible. For example, a foundation contains an ACE of HEARTS. If the tableau contains a stack with the top two cards being TWO and THREE of HEARTS respectively, then both can be moved at the same time to the foundation. For larger stacks, moving the cards one at a time would be tedious and strenuous. This module would make the movement of cards easier and increase the user-friendliness of the program.

Another module that could be added to the project, although advanced, could calculate the minimum of number of moves to solve the deck. While this game is in itself is challenging, this feature would encourage users to complete a deck in a certain amount of moves. It would set a goal for users and outline the best possible outcome for a deck. As well, this module could be used to state if a deck is solvable or not. The actual specification of this module would be very difficult, but could still be an interesting addition to the program.

Lastly, similar to the website, additional game use features can be added. For example, an undo module would reverse what ever the previous step is. Another module would record the latest move, and this module would reverse the move. As well, a module can be included that refreshes the game and creates a new deck (New Game feature).