Assignment 4

COMP SCI 2ME3 and SFWR ENG 2AA4

Yaminah Qureshi

April 9, 2018

Four modules, CardTypes, CardADT, CardStackADT and GameModule, make up the specification that models the state of the Freecell game. The CardADT constructs objects of type CardT that represent the 52 unique cards in a deck of cards. Sequences of the CardT objects exported by this module are used to create CardStackT objects to represent a stack of cards in the CardStackADT module. These CardStackT objects are by the GameModule to store the board used in Freecell. This board contains four stacks of seven randomly generated unique cards and four stacks of six randomly generated unique cards as well as four containers, one for each card suit, that accept cards of a suit in increasing order and further, four empty containers that can hold one card each. These stacks and containers are represented by a sequence of CardStackT objects. The GameModule allows for the construction of an instance of a game. This includes setting up the game, moving cards and ending the game if the board enters a winning state.

Card Types Module

Module

CardTypes

Uses

N/A

Syntax

Exported Constants

None

Exported Types

SuitsT = {Clubs, Clover, Hearts, Diamonds}

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

CardADT Module

Template Module

CardADT

Uses

CardTypes

Syntax

Exported Constants

None

Exported Types

CardT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new CardT	$\mathbb{N}$, SuitsT	CardT	invalid_card
suit		SuitsT	
rank		$\mathbb{N}$	

Semantics

State Variables

s: SuitsT //Suit of the card
r: $\mathbb{N}$ //Rank of the card

State Invariant

None

Assumptions

The Ace, Jack, Queen and King card ranks are represented by the numbers 1, 11, 12 and 13 respectively.

Access Routine Semantics

new CardT(S, R):

- transition: $s, r := S, R$
- output: $out := self$
- exception: $exc := (r < 1 \vee r > 13 \Rightarrow \text{invalid\_card})$

suit():

- output: $out := s$

rank():

- output: $out := r$

CardStackADT Module

Template Module

CardStackADT

Uses

CardADT, CardTypes

Syntax

Exported Constants

None

Exported Types

CardStackT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new CardStackT	seq of CardT	CardStackT	
top		CardT	empty_stack
size		$\mathbb{N}$	
push	CardT	CardStackT	
pop		CardStackT	empty_stack

Semantics

State Variables

S: sequence of CardT //Stack of cards

State Invariant

None

Assumptions

None

Access Routine Semantics

new CardStackT(s):

- transition: $S := s$
- output: $out := self$

top():

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

size():

- output: $out := |S|$

push($card$)

- output: $out :=$ new CardStackT($S \mathbin{||} \langle card \rangle$)

pop():

- output: $out :=$ new CardStackT($S[0..|S| - 2]$)
- exception: $exc := (|S| = 0 \Rightarrow \text{empty\_stack})$

Freecell Game Module

Template Module

GameModule

Uses

CardStackADT, CardADT, CardTypes

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
init	sequence of CardT		setup_error
moveCard	ℕ, ℕ		invalid_index, invalid_move
win		boolean	
noMoreMoves		boolean	
getColumn	ℕ	CardStackT	invalid_index

Semantics

State Variables

CardS: sequence of CardStackT //Contains the eight columns of tableau piles, four foundation piles and four free cells respectively

State Invariant

None

Assumptions

The *init* access routine is called first, . win is called after every moveCard. Game ends if the output is true (the player has won). The first four elements of *CardS* hold four tableau piles initially containing seven cards each, the next four elements hold the next four tableau piles of initially six cards each, the next four elements hold the four foundation piles and the final four elements contain the four free cells respectively, holding a maximum of one card each. Cards added to the tableau piles in the order they appear in the passed sequence of cards, such that the first four tableau piies hold the first four groups of seven cards each and the following four tableau piles hold the following four groups of six cards each

Access Routine Semantics

init(*cards*):

- transition: $CardS :=$
  $\langle$new CardStackT$(cards[0..6])$, new CardStackT$(cards[7..13])$, new CardStackT$(cards[14..20])$, new CardStackT$(cards[21..27])$, new CardStackT$(cards[28..33])$, new CardStackT$(cards[34..39])$, new CardStackT$(cards[40..45])$, new CardStackT$(cards[46..51])$, new CardStackT$(<>)$, new CardStackT$(<>)$, new CardStackT$(<>)$, new CardStackT$(<>)$, new CardStackT$(<>)$, new CardStackT$(<>)$, new CardStackT$(<>)$, new CardStackT$(<>)$
- exception: $exec := (|cards| \neq 52 \ \vee \ \exists(card1,\ card2 : CardT \mid card1.\text{suit}() = card2.\text{suit}() \wedge card1.\text{rank}() = card2.\text{rank}()) \Rightarrow \text{setup\_error})$

moveCard(i, j)

- transition: $CardS := \text{removeCard}(j, CardS[0..i{-}1] \mid\mid\ < CardS[i].\text{push}(CardS[j].\text{top}()) > \mid\mid CardS[i+1..|CardS|-1])$
- exception: $exec := (|CardS| <= i < 0 \vee |CardS| <= j < 0 \Rightarrow \text{invalid\_index}$
  $\mid \text{invalidMove}(i, j) \Rightarrow \text{invalid\_move})$

win()

- output: $out := (\forall\ i : \mathbb{N} \mid 8 <= i <= 11 : CardS[i].\text{size}() = 13)$

noMoreMoves()

- output: $out := (\forall\ i, j : \mathbb{N} \mid 0 <= i <= 15 \wedge 0 <= j <= 15 : \text{invalidMove}(i, j))$

getColumn(i)

- output: $out := CardS[i]$
- exception: $exec := (|CardS| <= i < 0 \Rightarrow \text{invalid\_index}$

Local Functions

removeCard: $\mathbb{N}$, sequence of CardStackT $\rightarrow$ sequence of CardStackT
removeCard$(j, cards) \equiv$

$$cards[0..j-1] || < cards[j].\text{pop}() > || cards[j+1..|cards|-1]$$

nextCard: CardT, CardT $\rightarrow$ boolean
nextCard(*card1*, *card2*)

$$(card2.\text{rank}() = card1.\text{rank}() + 1 \wedge card2.\text{suit}() = card1.\text{suit}())$$

validCard: CardT, CardT $\rightarrow$ boolean
validCard(*card1*, *card2*)

$$(card2.\text{rank}() = card1.\text{rank}() - 1 \wedge \neg\text{sameColour}(card2, card1))$$

sameColour: CardT, CardT $\rightarrow$ boolean
sameColour(*card1*, *card2*)

$$(card2.\text{suit}() = \text{Clubs} \vee card2.\text{suit}() = Clover \Rightarrow$$
$$card1.suit() = Diamonds \vee card1.suit() = Hearts$$
$$| \; card2.suit() = Diamonds \vee card2.suit() = Hearts \Rightarrow$$
$$card1.suit() = Clubs \vee card1.suit() = Clover)$$

invalidMove: $\mathbb{N}, \mathbb{N} \rightarrow$ boolean
invalidMove(i, j)

$$(i == j$$
$$\vee \; (i <= 7 \wedge \text{invalidMoveToTableau}(i, j))$$
$$\vee \; (8 <= i <= 11 \wedge \text{invalidMoveToFoundation}(i, j))$$
$$\vee \; (12 <= i <= 15 \wedge \text{invalidMoveToFreeCell}(i, j)))$$

invalidMoveToTableau: $\mathbb{N}, \mathbb{N} \rightarrow$ boolean
invalidMoveToTableau(i, j)

$$(CardS[i].\text{size}() \neq 0 \Rightarrow \neg(\text{validCard}(CardS[i].\text{top}(), CardS[j].\text{top}())) \; | \; true$$

invalidMoveToFoundation: $\mathbb{N}, \mathbb{N} \rightarrow$ boolean
invalidMoveToFoundation(i, j)

$$(CardS[i].\text{size}() = 0 \Rightarrow \neg(CardS[j].\text{top}().\text{rank}() = 1)$$
$$| \; \neg(\text{nextCard}(CardS[i].\text{top}(), CardS[j].\text{top}())))$$

invalidMoveToFreeCell: $\mathbb{N}, \mathbb{N} \rightarrow$ boolean
invalidMoveToFreeCell(i, j)

$$(CardS[i].\text{size}() \neq 0)$$