Assignment 4

COMP SCI 2ME3 and SFWR ENG 2AA4

Harry Fu

April 9, 2018

Specifications of this freecell game are split across six modules: Card, Cell, Pile, Foundation, Mover and Deck. Card modules provide basic access to starting a game. Card provides the card class declaration, and make sure all the cards' suit and rank can fit this game. Cell and Pile modules are taking card type to do all the conditions and return bool types to construct cells and piles. The foundation module, taking card type as well and accept a card to construct foundation. Cell, Pile and foundation modules both based on the card module. The Mover module uses cell, pile and foundation modules, it takes card type and return bool type to check can a card be moved between cell and pile and can this card be moved between pile and foundation. After the checking process, mover module will move this card between pile and cell or between pile and foundation and it returns bool type if the movement is successful or failing. The last one is Deck module, it checks wheather game is over. The cards adn their order in foundations must meet the corresponding game rules. The cards and their order in piles must meet the corresponding game rules. The number of all cards must be 52 and no duplicate card allowed. Deck module takes pile, cell and foundation types and check if there has any movement can be done between pile, cell and foundation. Then Deck module will show that the player is won or not.

Card Module

Template Module

Card

Uses

N/A

Syntax

Exported Constants

$Suits = \{SC, SD, SH, SS\}$
$Ranks = \{RA, R2, R3, R4, R5, R6, R7, R8, R9, R10, RJ, RQ, RK\}$

Exported Types

Card = ?

Exported Access Programs

Routine name	In	Out	Exceptions
Card	Suits, Ranks	Card	
getSuit		Suit	
getRank		Rank	
sameColor		boolean	
smallerByOne		boolean	

Semantics

State Variables

s: Suits
r: integer

State Invariant

None

Assumptions

None

Access Routine Semantics

Card ($SuitS, RanksR$):

- transition: $s, r := S, R$
- output: $out := self$
- exception: None

getSuit ()

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

getRanks ()

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

Local Functions

sameColor (c): Card $\rightarrow$ boolean

- sameColor(Card c) $\equiv \{s : Suit | switch(s)(s = SC \Rightarrow c.s \equiv SC \vee c.s \equiv SS | s = SD \Rightarrow c.s \equiv SD \vee c.s \equiv SH | s = SH \Rightarrow c.s \equiv SD \vee c.s \equiv SH | s = SS \Rightarrow c.s \equiv SC \vee c.s \equiv SS)\}$

smallerByOne (c): Card $\rightarrow$ boolean

- smallerByOne (c) $\equiv \{r : Rank | (c.r - r = 1) \Rightarrow True\}$

Cell Module

Template Module

Cell

Uses

Card

Syntax

Exported Types

Cell = ?

Exported Access Programs

Routine name	In	Out	Exceptions
Cell		Card	
empty		boolean	
gerCard		Card	emtpy_source raises if cell is empty
chkAcceptCard	Card	boolean	
acceptCard	Card	boolean	
popCard		Card	emtpy_source raises for cell is empty

Semantics

State Variables

c: Card

State Invariant

e: boolean // True

Access Routine Semantics

Cell ():

- output: $out := e(true), c(\{Card.SC, Card.RA\})$

empty ()

- output: $out := (e : True | Cell \equiv \epsilon \Rightarrow e)$

getCard ()

- output: $out := Card$
- exception: $exec : Card \equiv \epsilon \Rightarrow emtpy\_source$

chkAcceptCard(C)

- output: $out := (e : True | Cell \equiv \epsilon \Rightarrow e)$

acceptCard(C)

- output: $out := \{e : Boolean | ((Cell \equiv \epsilon \Rightarrow (e = false)) \Rightarrow c = C) \Rightarrow True\}$

popCard()

- output: $out := e$
- exception: $exec : Cell \equiv \epsilon \Rightarrow emtpy\_source$

Pile

Module

Pile

Uses

Card

Syntax

Exported Types

Pile = ?

Exported Access Programs

Routine name	In	Out	Exceptions
Pile			
size		integer	
empty		boolean	
assignCard	Card		
chkAcceptCard	Card	boolean	
acceptCard	Card	boolean	
getCard	integer	Card	emtpy_source,invalid_index
popCard			emtpy_source

Semantics

State Variables

$cards$:sequence of Card

Access Routine Semantics

empty()

- output: $out := (pile \equiv \epsilon \Rightarrow True)$

assignCard(c):

- transition: $c := cards$

chkAcceptCard(c):

- output: $out := (cards \equiv \epsilon \Rightarrow True)$

acceptCard(c):

- output: $\{(cards \equiv \epsilon \Rightarrow (cards = c)) \Rightarrow True)\}$

getCard(i):

- output: $cards[i]$
- exception: $exec := \{(pile \equiv \epsilon \Rightarrow emtpy\_source), (i < 0 \vee i > |cards| \Rightarrow invalid\_index)\}$

popCard():

- output: cards
- exception :$exec := (pile \equiv \epsilon \Rightarrow emtpy\_source)$

Foundation

Module

Foundation

Uses

Card

Syntax

Exported Types

Foundation = ?

Exported Access Programs

Routine name	In	Out	Exceptions
Foundation			
size		integer	
empty		boolean	
chkAcceptCard	Card	boolean	
acceptCard	Card	boolean	
getCard	integer	Card	emtpy_source,invalid_index
popCard			emtpy_source

Semantics

State Variables

cards :seqence of Card

Access Routine Semantics

: empty()

- output: $out := (foundation \equiv \epsilon \Rightarrow True)$

chkAcceptCard(c):

- output: $out := (cards \equiv \epsilon \Rightarrow True)$

acceptCard(c):

- output: $\{(cards \equiv \epsilon \Rightarrow (cards = c)) \Rightarrow True)\}$

getCard(i):

- output: $cards[i]$
- exception:$exec := \{(foundation \equiv \epsilon \Rightarrow emtpy\_source), (i < 0 \vee i > |cards| \Rightarrow invalid_index)\}$

popCard():

- output: cards
- exception :$exec := (foundation \equiv \epsilon \Rightarrow emtpy\_source)$

Mover

Module

Mover

Uses

Cell, Pile, Foundation

Syntax

Exported Types

Mover = ?

Exported Access Programs

Routine name	In	Out	Exceptions
chkPileToPile	Pile	boolean	
pileToPile	Pile	boolean	
chkPileToCell	Pile, Cell	boolean	
pileToCell	Pile, Cell	boolean	
chkCellToPile	Cell,Pile	boolean	
cellToPile	Cell, Pile	boolean	
chkPileToFoundation	Pile, Foundation	boolean	
chkPileToFoundation	Pile, Foundation	boolean	
pileToFoundation	Pile, Foundation	boolean	
chkFoundationToPile	Pile, Foundation	boolean	
foundationToPile	Pile, Foundation	boolean	
chkCellToFoundation	Cell, Foundation	boolean	
cellToFoundation	Cell, Foundation	boolean	
chkFoundationToCell	Cell, Foundation	boolean	
foundationToCell	Cell, Foundation	boolean	

Semantics

Access Routine Semantics

chkPileToPile(s, d)

- output: $out := (s : Pile | s \equiv \epsilon \Rightarrow false)$

pileToPile(s, d):

- transition s, d : = Pile, Pile
- output: $out := (d : Pile | (d \equiv \epsilon \Rightarrow d = s) \Rightarrow true)$

chkPileToCell(s, d):

- output:$out := (s : Pile | s \equiv \epsilon \Rightarrow false)$

pileToCell(s, d):

- transition s, d : = Pile, Cell
- output: $out := \{(d : Cell | (d \equiv \epsilon \Rightarrow d = s) \Rightarrow true) \Rightarrow false\}$

chkCellToPile(s, d):

- output: $out := (s : Cell | s \equiv \epsilon \Rightarrow false)$

cellToPile(s, d):

- transition s, d : = Cell, Pile
- output: $out := \{(d : Pile | (d \equiv \epsilon \Rightarrow d = s) \Rightarrow true) \Rightarrow false\}$

chkPileToFoundation(s, d):

- output: $out := (s : Pile | s \equiv \epsilon \Rightarrow false)$

pileToFoundation(s, d):

- transition s, d : = Pile,Foundation
- output: $out := \{(d : Foundation | (d \equiv \epsilon \Rightarrow d = s) \Rightarrow true) \Rightarrow false\}$

chkFoundationToPile(s, d):

- output: $out := (s : Foundation | s \equiv \epsilon \Rightarrow false)$

foundationToPile(s, d):

- transition: s, d := Foundation, Pile
- output: $out := \{(d : Pile | (d \equiv \epsilon \Rightarrow d = s) \Rightarrow true) \Rightarrow false\}$

chkCellToFoundation(s, d):

- output: $out := (s : Cell | s \equiv \epsilon \Rightarrow false)$

cellToFoundation(s, d):

- transition: s, d := Cell, Foundation
- output: $out := \{(d : Foundation | (d \equiv \epsilon \Rightarrow d = s) \Rightarrow true) \Rightarrow false\}$

chkFoundationToCell(s, d):

- output: $out := (s : Foundation | s \equiv \epsilon \Rightarrow false)$

foundationToCell(s, d):

- transition: s, d := Foundation, Cell
- output: $out := \{(d : Cell | (d \equiv \epsilon \Rightarrow d = s) \Rightarrow true) \Rightarrow false\}$

Deck

Module

Deck

Uses

Card, Cell, Foundation, Mover, Pile

Syntax

Exported Constants

NUM_P = 8 NUM_C = 4 NUM_F = 4

Exported Types

Deck = ?

Exported Access Programs

Routine name	In	Out	Exceptions
Deck	Cell, Foundation, Pile		
isPlayerWon		boolean	
hasValidMoves		boolean	invalid_index
isCellEmpty	integer	boolean	invalid_index
isPileEmpty	integer	boolean	invalid_index
isFoundationEmpty	integer	boolean	invalid_index
getPileSize	integer	integer	invalid_index
getFoundationSize	integer	integer	invalid_index
getCardFromCell	integer	Card	invalid_index
getCardFromPile	integer	Card	invalid_index
getCardFromFoundation	integer	Card	invalid_index
chkPileToPile	integer	boolean	invalid_index
pileToPile	integer	boolean	invalid_index
chkPileToCell	integer	boolean	invalid_index
pileToCell	integer	boolean	invalid_index
chkCellToPile	integer	boolean	invalid_index
cellToPile	integer	boolean	invalid_index
chkPileToFoundation	integer	boolean	invalid_index
pileToFoundation	integer	boolean	invalid_index
chkCellToFoundation	integer	boolean	invalid_index
cellToFoundation	integer	boolean	invalid_index

Semantics

State Variables

cs :sequence of Cell fs :sequence of Foundation ps :sequence of Pile m : Mover

Access Routine Semantics

Deck()

- transition: cs, fs, ps: Cell, Foundation, Pile

isPlayerWon():

- output: $\{(\forall(i : \mathbb{N} | i \in 0..(len(NUM\_P)) : ps[i] \vee cs[i] \not\equiv \epsilon \Rightarrow false)) \Rightarrow ture)\}$

hasValidMoves():

- output:$out :=$
- exception: $exce : (NUM\_P, NUM\_C, NUM\_F = 0 \vee NUM\_P, NUM\_C, NUM\_F < 0 \Rightarrow invalid\_index)$

isCellEmpty(i):

- output: $out := (cs : Cell | cs[i] \equiv \epsilon \Rightarrow true)$
- exception: $exce : (NUM\_C = 0 \vee NUM\_C < 0 \Rightarrow invalid\_index)$

isPileEmpty(i)

- output: $out := (ps : Pile | ps[i] \equiv \epsilon \Rightarrow true)$
- exception: $exce : (NUM\_P = 0 \vee NUM\_P < 0 \Rightarrow invalid\_index)$

isFoundationEmpty(i):

- output: $out := (fs : Foundation | fs[i] \equiv \epsilon \Rightarrow true)$
- exception: $exce : (NUM\_F = 0 \vee NUM\_F < 0 \Rightarrow invalid\_index)$

getPileSize(i):

- output: $out := len(fs[i])$
- exception: $exce : (NUM\_P = 0 \vee NUM\_P < 0 \Rightarrow invalid\_index)$

getFoundationSize(i):

- output: $out := len(fs[i])$
- exception: $exce : (NUM\_F = 0 \vee NUM\_F < 0 \Rightarrow invalid\_index)$

getCardFromCell(i):

- output: $out := cs[i]$
- exception: $exce : (NUM\_C = 0 \vee NUM\_C < 0 \Rightarrow invalid\_index)$

getCardFromPile(i, j):

- output: $out := ps[i][j]$
- exception: $exce : (NUM\_P = 0 \vee NUM\_P < 0 \Rightarrow invalid\_index)$

getCardFromFoundation(i, j):

- output: $out := fs[i][j]$
- exception: $exce : (NUM\_F = 0 \vee NUM\_F < 0 \Rightarrow invalid\_index)$

Local Functions

dealCards()