Assignment 4, Specification

SFWR ENG 2AA4

April 9, 2018

The purpose of this software design exercise is to design and implement a portion of the specification for the FreeCell Solitaire. This document shows the complete specification, which will be the basis for your implementation and testing.

Card ADT Module

Template Module

CardT

Uses

N/A

Syntax

Exported Types

CardT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
CardT	string, string	CardT	
rank		string	
suit		string	
getCard		string	

Semantics

State Variables

r: string
s: string

State Invariant

None

Assumptions

The constructor CardT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

CardT(r, s):

- transition: $r, s := rank, suit$
- output: $out := self$
- exception: None

rank():

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

suit():

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

getCard():

- output: $out := rank() + \text{“ ”} + suit()$
- exception: None

Deck ADT Module

Template Module

DeckT

Uses

CardT

Syntax

Exported Types

DeckT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
DeckT		DeckT	invalid_argument
shuffle			
get	$\mathbb{N}$	string	invalid_argument

State Variables

deck: sequence of CardT

State Invariant

$MAX\_CARD = 52$

Assumptions

The constructor CardT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

DeckT():

- transition: $\{i : \mathbb{N} | i \in [0..MAX\_CARDS] : deck[i] = CardT(rank[i], suit[i])\}$

- output: $out := self$
- exception: exc:= $MAX\_CARDS > 52 \implies invalid\_argument$

shuffle():

- transition: $i : \mathbb{N} | i \in [0..MAX\_CARDS] : deck[i], deck[i+1] = deck[i+1], deck[i]$
- exception: None

get(i):

- output: $out := deck[i]$
- exception: exc:= $i > MAX\_CARD \implies invalid\_argument$

Board ADT Module

Template Module

BoardT

Uses

CardT, DeckT

Syntax

Exported Types

BoardT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
BoardT	cellSize		invalid_argument
dealGameCards			
moveCardTtoT	$\mathbb{N}, \mathbb{N}$		invalid_move
moveCardTtoFree	$\mathbb{N}, \mathbb{N}$		invalid_move
moveCardFreetoT	$\mathbb{N}, \mathbb{N}$		invalid_move
moveCardTtoFoundation	$\mathbb{N}, \mathbb{N}$		invalid_move
isGameOver		$\mathbb{B}$	

State Variables

gameColumns: sequence of (sequence of CardT)
foundationRow: sequence of (sequence of CardT)
freeRow: sequence of CardT

State Invariant

$Max\_Cell\_Size = 4$
$Max\_Cards\_inColumn = 13$
$Number\_of\_Columns = 8$

Assumptions

The constructor BoardT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

BoardT(cellSize):

- transition: $\{i : \mathbb{N} | i \in [0...Max\_Cell\_Size] \wedge freeRow[i] = CardT(\text{“”}, \text{“”})\}$

  $(i, j : \mathbb{N} | i \in [0...Max\_Cell\_Size] \wedge j \in [0...Max\_Cards\_inColumn] \wedge foundationRow[i][j] = CardT(\text{“”}, \text{“”}))$

- output: $out := self$

- exception: exc:= $Max\_Cell\_Size \neq 4 \implies invalid\_argument$

dealGameCards():

- transition: $\{i, j : \mathbb{N} | i \in [0...(Number\_of\_Columns)] \wedge j \in [0...52] : gameColumns[i][j]($

i	**j**
i = 0	deck[0...7]
i = 1	deck[7...14]
i = 2	deck[14...21]
i = 3	deck[21...28]
i = 4	deck[28...34]
i = 5	deck[34...40]
i = 6	deck[40...46]
i = 7	deck[46...52]

$)\}$

- exception: None

moveCardTtoT(column1, column2):

- transition: $gameColumns[column2].at(|gameColumns|) := gameColumns[column1][|gameColumns| - 1]$

- exception: exc:= $(\neg validMove(column1, column2))$

moveCardTtoFree(column1, column2):

- transition:
  $freeRow[column2] := gameColumns[column1][|gameColumns| - 1]$
- exception: exc:= $(\neg validMove(column1, column2))$

moveCardFreetoT(column1, column2):

- transition:
  $gameColumns[column1][|gameColumns|] := freeRow[column2]$
- exception: exc:= $(\neg validMove(column1, column2))$

moveCardTtoFoundation(column1, column2):

- transition:
  $foundationRow[column1][|gameColumns|] := gameColumn[column2][|gameColumns| - 1]$
- exception: exc:= $(\neg validMove(column1, column2))$

isGameOver():

- output: out:=$(i, j : \mathbb{N} | i \in [0...|foundationRow|] \wedge j \in [0...|foundationRow[i]|] \wedge foundationRow[i][j] =$ "K") $\implies TRUE$
- exception: exc:= None

Local Functions

validMove(column1, column2):

- transition:
  $gameColumns[column2].at(|gameColumns|) := gameColumns[column1][|gameColumns| - 1]$
- exception: exc:= $(\neg validMove(column1, column2))$

isAscendingTtoT(column1, column2):

- output: out:=
  $assignIndex(gameColumns[column1][|gameColumns| - 1]) <$
  $assignIndex(gameColumns[column2][|gameColumns| - 1]) \implies TRUE$

- exception: exc:= None

assignIndex(CardT card):

- transition: $gameColumns[column2].at(|gameColumns|) := gameColumns[column1][|gameColumns|-1]$
- exception: exc:= $(\neg validMove(column1, column2))$