Assignment 4 Specification

SFWR ENG 2AA4

April 6, 2020

This Module Interface Specification (MIS) document contains modules, types and methods for implementing the Dots game. The spec utilizes the Model-View-Controller design pattern.

The model creates and sets up the Dots Game board and its arrangement, as well as modifying the board's state. The view displays the Dots Game Board & game messages to the user, and also receives the input from the user. The controller contains the logic and action of the Dots game, calling both the model and view to modify both the game board and display messages based on the game logic.

Colour Type Module

Module

ColourT

Uses

N/A

Syntax

Exported Constants

None

Exported Types

Colours = {R, G, B, Y, P}

//R stands for Red, G for Green, B for Blue, Y for Yellow, P for Pink

Exported Access Programs

Routine name	In	Out	Exceptions
getRandomColour		ColourT	

Semantics

State Variables

None

State Invariant

None

Considerations

Access Routine Semantics

getRandomColour():

- output: *out* :=
- exception: None

Dots ADT Module

Template/Model Module

Dots

Uses

ColourT

Syntax

Exported Types

Dots = ?

Exported Access Programs

Routine name	In	Out	Exceptions
Dots	$\mathbb{Z}$	Dots	
matrix		Seq(Seq(ColourT))	
n		$\mathbb{Z}$	
getColour	$\mathbb{Z}$, $\mathbb{Z}$	ColourT	
setColour	$\mathbb{Z}$, $\mathbb{Z}$		
addRandomColour	$\mathbb{Z}$		
setRandomColour	$\mathbb{Z}$, $\mathbb{Z}$		
initializeDots			
isValidPath	Seq(string)	$\mathbb{B}$	
dropDots			
processDots			
hasValidCombo		$\mathbb{B}$	

Semantics

State Variables

n: $\mathbb{Z}$
$matrix$: Seq of (Seq of ColourT)

State Invariant

None

Assumptions

...

Access Routine Semantics

Dots(num):

- transition: n, matrix := num, new Seq of (Seq of ColourT)
- output: *out* := *self*
- exception: None

matrix():

- output: *out* := Seq of (Seq of ColourT)
- exception: None

n():

- [What should go here? —SS]output: *out* := n
- exception: None

getColour(i, j):

- output: *out* := matrix[i][j]
- exception: None

setColour(i, j, c):

- transition: matrix[i][j] := c
- exception: None

addRandomColour(i):

- transition: matrix[i].*append*()
- exception: None

setRandomColour(i, j):

- transition: matrix[i][j] :=
- exception: None

initializeDots():

- transition: $\forall i, j$: matrix[i][j] :=
- exception: None

isValidPath(input):

- out: $\forall i, j$: matrix[i][j] :=
- exception: None

dropDots():

- transition: matrix[i][j] :=
- exception: None

processDots():

- transition: matrix[i][j] :=
- exception: None

hasValidPath():

- out: matrix[i][j] :=
- exception: None

Dots View Module

View Module

DotsView

Uses

N/A

Syntax

Exported Types

?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
startMenu			
printEnterNewInput			
displayScore	$\mathbb{Z}$		
displayTarget	$\mathbb{Z}$		
displayMovesLeft	$\mathbb{Z}$		
printInvalidMove			
printReshuffled			
printScoreReached			
printMovesOut			
renderDots	Seq(Seq(ColourT))		
getInput		string	

Semantics

State Variables

None

State Invariant

None

Assumptions

Since most of the routines in the View are just displaying print statements, the routines are ignored in the Access Routine Semantics. The purpose of each View routine is listed below:

- startMenu(): display the Game Menu message
- printEnterNewInput(): displays Enter New Input message
- displayScore(n): display Score of n
- displayTarget(n): display Target of n
- displayMovesLeft(n): display Moves of n
- printInvalidMove(): display Invalid Move message
- printReshuffled(): displays Board Reshuffled message
- printScoreReached(): displays Score Reached message
- renderDots(board): display Game Board by: $\forall i, j$: display each board$[i][j]$
- getInput(): read and return input message

Access Routine Semantics

N/A

Dots Controller Module

Controller Module

DotsController

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
DotsController	Dots, DotsView	DotsController	
isValidInput	string	$\mathbb{B}$	
startGame			
infiniteMode			
targetMode			

Semantics

Access Routine Semantics

total():

- output: [Total of all the values in all of the cells. —SS]$out := +(i, j : \mathbb{N} | \text{validRow}(i) \wedge \text{validCol}(j) : s[i][j])$
- exception: None

max():

- output: $out := s[mi][mj]$ such that $\forall(i, j : \mathbb{N} | \text{validRow}(i) \wedge \text{validCol}(j) : s[i][j] \leq s[mi][mj])$
- exception: None

ascendingRows():

- output: [Returns True if the sum of all values in each row increases as the row number increases, otherwise, returns False. —SS]$out := \forall(i : \mathbb{N} | 0 \leq i \leq \text{nRow} - 2 : +(j : \mathbb{N} | \text{validCol}(j) : s[i+1][j]) > +(j : \mathbb{N} | \text{validCol}(j) : s[i][j]))$
- exception: None

Local Functions

validRow: $\mathbb{N} \to \mathbb{B}$
[returns true if the given natural number is a valid row number. —SS]validRow$(i) \equiv 0 \leq i \leq (\text{nRow} - 1)$

validCol: $\mathbb{N} \to \mathbb{B}$
validCol$(j) \equiv 0 \leq j \leq (\text{nCol} - 1)$

Answer to Design Questions

1. The
2. Although