Assignment 4, Two Dots game Specification

SFWR ENG 2AA4

April 6, 2020

This Module Interface Specification (MIS) document contains interfaces,ADT's and methods for implementing a game of Two Dots

Color Module

Module

Color

Uses

N/A

Syntax

Exported Constants

None

Exported Types

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

//R stands for Red, G for green, B for blue, P for Purple, Y for yellow

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
randomColor		Color	

Semantics

State Variables

colors: color

State Invariant

None

Access Routine Semantics

randomColor():

- transition: none

- output: $out :=$ randomVal()
- exception: none

Local Functions

randomVal(): Color
randomVal() $\equiv (i = 0 \implies \text{R} \mid i = 1 \implies \text{G} \mid i = 2 \implies \text{B} \mid i = 3 \implies \text{P} \mid i = 4 \implies \text{Y})$
Where i is a uniformly-distributed random number in the range $0 \leq i \leq 4$

Point ADT Module

Template Module

PointT

Uses

N/A

Syntax

Exported Types

PointT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
PointT	$\mathbb{Z}$, $\mathbb{Z}$	PointT	
row		$\mathbb{Z}$	
col		$\mathbb{Z}$	
translate	$\mathbb{Z}$, $\mathbb{Z}$	PointT	

Semantics

State Variables

r: $\mathbb{Z}$
c: $\mathbb{Z}$

State Invariant

None

Assumptions

The constructor PointT 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

PointT(row, col):

- transition: $r := row, c := col$
- output: $out := self$
- exception: None

row():

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

col():

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

Generic Board Module

Generic Template Module

Board(T)

Uses

PointT

Syntax

Exported Types

Board(T) = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
Board	$\mathbb{N}, \mathbb{N}$	Board	IllegalArgumentException
set	PointT, T		IndexOutOfBoundsException
get	PointT	T	IndexOutOfBoundsException
getNumRow		$\mathbb{N}$	
getNumCol		$\mathbb{N}$	

Semantics

State Variables

s: seq of (seq of T)
nRow: $\mathbb{N}$
nCol: $\mathbb{N}$

State Invariant

None

Assumptions

- The Board(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.
- $s[i][j]$ means the ith row and the jth column. The 0th row is at the top of the grid and the 0th column is at the leftmost side of the grid.

Access Routine Semantics

Board(row, col):

- transition (note that the list does not enforce an *order* in which the transitions occur, only the transitions that must occur):
  1. $nRow := row$
  2. $nCol := col$
- output: $out := self$
- exception:
  exc := (row $\leq$ 0) $\vee$ (col $\leq$ 0) $\implies$ IllegalArgumentException

set(p, v):

- transition: $s[p.row()][p.col()] = v$
- exception:
  $\neg$ validPoint(p) $\implies$ IndexOutOfBoundsException

get(p):

- output: $out := s[p.row()][p.col()]$
- exception:
  $\neg$ validPoint(p) $\implies$ IndexOutOfBoundsException

getNumRow():

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

getNumCol():

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

Local Functions

validRow: $\mathbb{N} \rightarrow \mathbb{B}$
validRow$(r) \equiv r \geq 0 \wedge (r <$ nRow$)$

validCol: $\mathbb{N} \rightarrow \mathbb{B}$
validCol$(c) \equiv (c \geq 0) \wedge (c <$ nCol$)$

validPoint: PointT $\rightarrow \mathbb{B}$
validPoint$(p) \equiv$ validCol(p.col()) $\wedge$ validRow(p.row())

BoardMoves Module

Template Module

BoardMoves is seq of PointT

Considerations

When using in Java. Use Arraylist parameterized by PointT

TwoDotsBoard Module

Template Module

TwoDotsBoard is Board(Color)

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
validateMoves	BoardMoves	$\mathbb{B}$	
updateBoard	BoardMoves		

Semantics

Access Routine Semantics

validateMoves(b):

- output : out := $|b| > 1 \wedge \forall(p : PointT | p \in b : validPoint(p)) \wedge validPath(b) \wedge isDistinct(b)$
- exception: None

updateBoard(b):

- output : out := None
- transition : s := $\forall(p : PointT | p \in b \forall(i : \mathbb{N} | i \in [p.row()..1]))$
- exception: None

Local Functions

validRow: $\mathbb{N} \rightarrow \mathbb{B}$
validRow(r) $\equiv r \geq 0 \wedge (r <$ nRow$)$

validCol: $\mathbb{N} \rightarrow \mathbb{B}$
validCol(c) $\equiv (c \geq 0) \wedge (c <$ nCol)

validPoint: PointT $\rightarrow \mathbb{B}$
validPoint(p) $\equiv$ validCol(p.col()) $\wedge$ validRow(p.row())

isDistinct: BoardMoves $\rightarrow \mathbb{B}$
isDistinct(b) $\equiv \forall(i : \mathbb{N}|i \in [0..|b|-1] : \forall(j : \mathbb{N}|j \in [(i+1)..|b|-1]) : \neg(b[i].row() = b[j].row()) \wedge (b[i].col() = b[j].col()))$

validPath: BoardMoves $\rightarrow \mathbb{B}$
validPath(b) $\equiv \forall(i : \mathbb{N}|i \in [0..|b|-2] : isAdjacent(b, i, i+1) \wedge sameColor(b, i, i+1))$

sameColor : BoardMoves $\times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$
sameColor$(b, i, j) \equiv s[b[i].row()][b[i].col()] = s[b[j].row()][b[j].col()]$

isAdjacent: BoardMoves $\times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$
isAdjacent$(b, i, j) \equiv b[i].row() = b[j].row() \wedge b[i].col() = b[j].col() + 1$
$\vee\ b[i].row() = b[j].row() \wedge b[i].col() = b[j].col() - 1$
$\vee\ b[i].row() = b[j].row() - 1 \wedge b[i].col() = b[j].col()$
$\vee\ b[i].row() = b[j].row() + 1 \wedge b[i].col() = b[j].col()$

Strategy Interface Module

Interface Module

Strategy

Syntax

Exported Constants

None

Exported types

None

Exported Access Programs

Routine name	In	Out	Exceptions
play	TwoDotsBoard		

BoardView Module

Template Module

BoardView

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
printBoard	TwoDotsBoard	$\mathbb{B}$	
modePrompt		Strategy	
getInput		BoardMoves	
closeStream			
printMsg	$msg : string$		

Semantics

Environment variables

s : 2D sequence of pixels displayed on a standard Unix Shell/console
r : an object to write text out on a standard Unix Shell/console

Access Routine Semantics

printBoard(b):

- transition $s :=$ Modify the Console so that the TwoDotsBoard b is printed in a tabular manner. The contents of each row from b should be on individual line. Their should be horizontal and vertical numbering indicating each row and column from one upto and including the row and column size of the board
- exception: None

modePrompt():

- transition :

- s := Modify the console to print a message asking the user to enter "T" for the timed version of the game and "M" for the mode of the game with a set number of moves
- r:= read a single line of text from the standard input. Store this value in memory and then determine what to output as follows:
  If the line read in is "T" or "t" output : out := new TimedStrategy()
  If the line read in is "M" or "m" output: out := new MovesStrategy()
  Otherwise keep reading a line from the standard input until one of the above two conditions are met

- exception: None

getInput():

- transition :
  - r:= read a single line of text from the standard input to determine the coordinates of the dots the user would like to eliminate. Note that the desired input format is u,v w,x y,z .... These are pairs of natural numbers with a comma between them and each pair is separated by a space. Store this value in memory and then determine what to output as follows:
    If the line read in is in the correct format then *output* : *out* := new BoardMoves() containing the pairs of integers
    Otherwise keep reading a line from the standard input until one of the above conditions are met

closeStream():

- transition : s:= close the input stream

printMsg(msg : $string$):

- transition : s:= Modify the output console to print out text contain in the string msg

Considerations

In java, closing the input stream corresponds to closing the System.in object

BoardController Module

Template Module

BoardController

Uses

TwoDotsBoard, BoardView, Color, PointT, BoardMoves

Syntax

Exported Constants

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
BoardController	TwoDotsBoard, BoardView	BoardController	
get	PointT	Color	
set	PointT, Color		
validateMoves	BoardMoves	$\mathbb{B}$	
updateBoard	BoardMoves		
updateView			
printMsg	$msg : string$		
modePrompt		Strategy	
closeViewStream			
getInput		BoardMoves	

Semantics

State variables

m : TwoDotsBoard v : BoardView

State invariant

None

Access Routine Semantics

BoardController(*model*,*view*)

- output: out := self
- transition: m := model, v := view
- exceptions: none

get(p)

- output : *out* := m.get(p)
- transition: none
- exceptions: none

set(p,c)

- transition: m.set(p,c)
- exception : none

validateMoves(b)

- output : *out* := m.validateMoves(b)

updateBoard(b)

- transition: m.updateBoard(b)

updateView()

- transition: v.printBoard(m)

printMsg(msg : $string$):

- transition view.printMsg(msg)

modePrompt():

- output: *out* := v.modePrompt()

closeViewStream():

- transition :v.closeStream()

getInput():

- output: if (m.validateMoves(v.getInput())) then *out* := else *out* := getInput()

StrategyGameMode Module

Template Module inherits Strategy

StrategyGameMode

Uses

Strategy, BoardView, BoardController, BoardMoves, TwoDotsBoard

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
play	TwoDotsBoard		
startUp	TwoDotsBoard		
checkWin		$\mathbb{B}$	
canContinue		$\mathbb{B}$	
updateData			
introMsg			
endMsg			

Semantics

State variables

c : BoardController
v : BoardView
$moves$: BoardMoves

State invariant

None

Access Routine Semantics

play(b)

- transition:
  - startUp(b)
  - introMsg()
  - if canContinue() then:
    * c.updateView()
    * c.updateBoard(c.getInput())
    * if checkWin() then
    * updateData()
    * if canContinue() then repeat these steps labeled with $*$

Consideration

In Java, this module would be implemented as an abstract class that implements the Strategy interface. Unimplemented methods are ones that are abstract methods and will be overridden by its children
This is the best that could be done to convey the idea of a "abstract class" given that MIS does not have the notion of an abstract class, following Dr Smith's advice to use Inheritance and leave a note for reader. Source: Here (will have to login to avenue)

MovesStrategy Module

Template Module inherits StrategyGameMode

MovesStrategy

Uses

StrategyGameMode, BoardView, BoardController, BoardMoves, TwoDotsBoard

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
play	TwoDotsBoard		
startUp	TwoDotsBoard		
checkWin		$\mathbb{B}$	
canContinue		$\mathbb{B}$	
updateData			
introMsg			
endMsg			

Semantics

Environment variables

State variables

$moveCount : \mathbb{N}$
$TARGET : \mathbb{N}$

State invariant

None

Access Routine Semantics

startUp(b)

- transition: $v, c, moveCount, TARGET :=$ new BoardView(), new BoardController(b, v), 15, 5

checkWin()

- transition: if $|moves| < TARGET$ then $moves := moves - 1$
- outupt: $out :=$ if $|moves| >= TARGET$ then $out := true$ else $out := false$

canContinue()

- output: $out := moveCount > 0$

updateData()

- transition: if $moveCount = 0$ then c.exit()
  else $c :=$ using c.printMsg(), modify the screen to print out how many moves are left, i.e print the value of $moveCount$

introMsg()

- transition: $c :=$ using c.printMsg(), modify the screen to print a message to state the rules. This includes instruction for how to enter input, how to win and how many total moves the user has

endMsg()
transition: $c :=$ using c.printMsg(), modify the screen by printing a message telling the user the game is over

CountDownTimer Module

Module

CountDownTimer

Uses

None

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
newTimer	$\mathbb{Z}$		IllegalArgumentException
isCancelled		$\mathbb{B}$	

Semantics

Environment variables

t : Represents the system clock

State variables

$cancelled : \mathbb{B}$
$multiplier : \mathbb{Z}$

State invariant

None

Access Routine Semantics

newTimer(time : $\mathbb{Z}$)

- transition: $cancelled := false$
  $multiplier := 1000$
  $t :=$ Use the system clock to start tracking the current time. once time*multiplier amount of time has passed then the transition cancelled := true happens

- exception: $exc :=$ time $< 1 \implies$ IllegalArgumentException

isCancelled()

- output: $out := \neg cancelled$

DEM Module

Template Module

DemT is Seq2D($\mathbb{Z}$)

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
total		$\mathbb{Z}$	
max		$\mathbb{Z}$	
ascendingRows		$\mathbb{B}$	

Semantics

Access Routine Semantics

total():

- output : out := $+(x, y : \mathbb{N} |$ validRow$(x) \wedge$ validCol(y) : $s[x][y])$
- exception: None

max():

- output: out := M such that $\forall(x : \text{Seq of } \mathbb{Z} \mid x \in s : \forall(y : \mathbb{Z} | y \in x : M \geq y)) \wedge (\exists i : \mathbb{N} | validRow(i) : M \in s[i])$
- exception: None

ascendingRows():

- output:out := $\forall(i : \mathbb{N} | i \in [0..|s| - 2] : sum(s[i]) < sum(s[i + 1]))$
- exception: None

Local Functions

validRow: $\mathbb{N} \rightarrow \mathbb{B}$
validRow$(n) \equiv (n \geq 0) \wedge (n <$ nRow$)$

validCol: $\mathbb{N} \to \mathbb{B}$
$\text{validCol}(c) \equiv (c \geq 0) \land (c < \text{nCol})$

sum: Seq of $\mathbb{Z} \to \mathbb{Z}$
$\text{sum}(s) \equiv (+x : \mathbb{Z} | x \in s : x)$