Assignment 5, Part 1, Specification

SFWR ENG 2AA4

April 13, 2008

The purpose of this software design exercise is to design, specify, implement and test a module for storing the state of an Othello game. The game board is represented as a two dimensional sequence, with the first dimension the row and the second dimension the column. The indexes are relative to the upper left hand corner of the board; that is, row 0 and column 0 are at the top left.

Othello Module

Module

Othello

Uses

N/A

Syntax

Exported Constants

SIZE = 8 *//size of the board in each direction*

Exported Types

cellT = { FREE, BLACK, WHITE }

Exported Access Programs

Routine name	In	Out	Exceptions
init			
move	integer, integer, cellT		OutOfBoundsException, InvalidMoveException, WrongPlayerException
switch_turn			ValidMoveExistsException
getb	integer, integer	cellT	OutOfBoundsException
get_turn		cellT	
count	cellT	integer	
is_valid_move	integer, integer, cellT	boolean	OutOfBoundsException
is_winning	cellT	boolean	
is_any_valid_move	cellT	boolean	
is_game_over		boolean	

Semantics

State Variables

b: boardT
blacksturn: boolean

State Invariant

count(BLACK) + count(WHITE) + count(FREE) = SIZE × SIZE

Assumptions

The init method is called for the abstract object before any other access routine is called for that object. The init method can be used to return the state of the game to the state of a new game.

Access Routine Semantics

init():

- transition:

$$blacksturn, b := true, < \begin{array}{c} < \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\ < \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\ < \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\ < \text{FREE}, \text{FREE}, \text{FREE}, \text{WHITE}, \text{BLACK}, \text{FREE}, \text{FREE}, \text{FREE} > \\ < \text{FREE}, \text{FREE}, \text{FREE}, \text{BLACK}, \text{WHITE}, \text{FREE}, \text{FREE}, \text{FREE} > \\ < \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\ < \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \\ < \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE}, \text{FREE} > \end{array} >$$

- exception none

move(i, j, c):

- transition: $blacksturn := \neg blacksturn$ and b such that
  UpdateNS$(i, j, c, b) \wedge$ UpdateWE$(i, j, c, b) \wedge$ UpdateNESW$(i, j, c, b) \wedge$ UpdateNWSE(i, j, c, b)
- exception $exc :=$ (InvalidPosition$(i, j) \Rightarrow$ OutOfBoundsException$|\neg$is_valid_move$(i, j, c) \Rightarrow$ InvalidMoveException$|\neg$is_correctPlayer$(blacksturn, c) \Rightarrow$ WrongPlayerException)

switch_turn():

- transition: $blacksturn := \neg blacksturn$
- exception $exc :=$ (is_any_valid_move() $\Rightarrow$ ValidMoveExistsException)

getb(i, j):

- output: $out := b[i, j]$
- exception $exc :=$ (InvalidPosition$(i, j) \Rightarrow$ OutOfBoundsException)

get_turn():

- output: $out :=$ ($blacksturn \Rightarrow$ BLACK$|\neg blacksturn \Rightarrow$ WHITE)
- exception: none

count(c):

- output: $+(i, j : \mathbb{N} | 0 \leq i < \text{SIZE} \wedge 0 \leq j < \text{SIZE} \wedge b[i, j] = c : 1)$
- exception: none

is_valid_move(i, j, c):

- output: $out := (b[i][j] = \text{FREE}) \wedge (\text{is\_validN}(i, j, c, b) \vee \text{is\_validS}(i, j, c, b) \vee \text{is\_validW}(i, j, c, b) \vee \text{is\_validE}(i, j, c, b) \vee \text{is\_validNW}(i, j, c, b) \vee \text{is\_validNE}(i, j, c, b) \vee \text{is\_validSW}(i, j, c, b) \vee \text{is\_validSE}(i, j, c, b))$
- exception $exc := (\text{InvalidPosition}(i, j) \Rightarrow \text{OutOfBoundsException})$

is_winning(c):

- output: $out := (c = \text{BLACK} \Rightarrow \text{count(BLACK)} > \text{count(WHITE)} | c = \text{WHITE} \Rightarrow \text{count(WHITE)} > \text{count(BLACK)} | c = \text{FREE} \Rightarrow \mathit{false})$
- exception: none

is_any_valid_move(): *//Returns true if a valid move exists for the current player*

- output:

  $out := \exists(i, j : \mathbb{N} | 0 \leq i < \text{SIZE} \wedge 0 \leq j < \text{SIZE} \wedge b[i][j] = \text{FREE} : (\mathit{blacksturn} \Rightarrow \text{is\_valid\_move}(i, j, \text{BLACK}) | \neg \mathit{blacksturn} \Rightarrow \text{is\_valid\_move}(i, j, \text{WHITE})))$

- exception: none

is_game_over(): *//Returns true if neither player has a valid move*

- output:

  $out := \neg\exists(i, j : \mathbb{N} | 0 \leq i < \text{SIZE} \wedge 0 \leq j < \text{SIZE} \wedge b[i][j] = \text{FREE} : \text{is\_valid\_move}(i, j, \text{BLACK})) \wedge \neg\exists(i, j : \mathbb{N} | 0 \leq i < \text{SIZE} \wedge 0 \leq j < \text{SIZE} \wedge b[i][j] = \text{FREE} : \text{is\_valid\_move}(i, j, \text{WHITE}))$

- exception: none

Local Types

boardT = sequence [SIZE, SIZE] of cellT

Local Functions

UpdateNS : integer $\times$ integer $\times$ cellT $\times$ boardT $\rightarrow$ boolean

$\text{UpdateNS}(i, j, c, b) \equiv \forall(k : \mathbb{N} | (i - \text{CountN}(i, j, c, b)) \leq k \leq (i + \text{CountS}(i, j, c, b)) : b[k, j] = c)$

UpdateWE : integer $\times$ integer $\times$ cellT $\times$ boardT $\rightarrow$ boolean

$\text{UpdateWE}(i, j, c, b) \equiv \forall(k : \mathbb{N} | (j - \text{CountW}(i, j, c, b)) \leq k \leq (j + \text{CountE}(i, j, c, b)) : b[i, k] = c)$

UpdateNESW : integer $\times$ integer $\times$ cellT $\times$ boardT $\rightarrow$ boolean

$\text{UpdateNESW}(i, j, c, b) \equiv \forall(k, l : \mathbb{N} | (i - \text{CountNE}(i, j, c, b)) \leq k \leq (i + \text{CountSW}(i, j, c, b)) \wedge ((j - \text{CountSW}(i, j, c, b)) \leq l \leq (j + \text{CountNE}(i, j, c, b)) : b[k, l] = c)$

UpdateNWSE : integer $\times$ integer $\times$ cellT $\times$ boardT $\rightarrow$ boolean

$\text{UpdateNWSE}(i, j, c, b) \equiv \forall(k, l : \mathbb{N} | (i - \text{CountNW}(i, j, c, b)) \leq k \leq (i + \text{CountSE}(i, j, c, b)) \wedge (j - \text{CountNW}(i, j, c, b)) \leq l \leq (j + \text{CountSE}(i, j, c, b)) : b[k, l] = c)$

CountN : integer $\times$ integer $\times$ cellT $\times$ boardT $\rightarrow$ integer

$$\begin{aligned}&\text{CountN}(i, j, c, b) \equiv \\ &\quad +(k : \mathbb{N} | \text{is\_validN}(i, j, c, b) \wedge \\ &\quad 0 < k < i \wedge \forall(l : \mathbb{N} | k \leq l < i : \text{hostile}(l, j, c, b)) : 1)\end{aligned}$$

CountS : integer $\times$ integer $\times$ cellT $\times$ boardT $\rightarrow$ integer

$$\begin{aligned}&\text{CountS}(i, j, c, b) \equiv \\ &\quad +(k : \mathbb{N} | \text{is\_validS}(i, j, c, b) \wedge \\ &\quad i < k < (\text{SIZE} - 1) \wedge \forall(l : \mathbb{N} | i < l \leq k : \text{hostile}(l, j, c, b)) : 1)\end{aligned}$$

etc.

is_validN : integer × integer × cellT × boardT → boolean

$$\text{is\_validN}(i, j, c, b) \equiv \exists(k : \mathbb{N} | 0 \leq k < i : \text{friendly}(k, j, c, b) \wedge \forall(l : \mathbb{N} | k < l < i : \text{hostile}(l, j, c, b)))$$

etc.

friendly: integer × integer × cellT × boardT → boolean

$$\text{friendly}(i, j, c, b) \equiv b[i, j] = c$$

hostile: integer × integer × cellT × boardT → boolean

$$\text{hostile}(i, j, c, b) \equiv (b[i, j] = \text{BLACK} \Rightarrow c = \text{WHITE} \,|b[i, j] = \text{WHITE} \Rightarrow c = \text{BLACK} \,|c = \text{FREE} \Rightarrow \mathit{false})$$

InvalidPosition: integer × integer → boolean

$$\text{InvalidPosition}(i, j) \equiv \neg((0 \leq i < \text{SIZE}) \wedge (0 \leq j < \text{SIZE}))$$

is_correctPlayer: boolean × cellT → boolean

$$\text{is\_correctPlayer}(bt, c) \equiv (bt \Rightarrow c = \text{BLACK} | \neg bt \Rightarrow c = \text{WHITE})$$