Benjamin Korobkin

bjk125

EECS3311

Lab 2 Report

Peg Solitaire

Fall 2017

Contract View

```
note
	description: "A board for the peg solitaire game."
	author: "Benjamin Korobkin"
	date: "October 17, 2017"
	revision: "N/A"

class interface
	BOARD

create
	make_default,
	make_easy,
	make_cross,
	make_plus,
	make_pyramid,
	make_arrow,
	make_diamond,
	make_skull

feature -- Auxiliary Commands

	set_status (r, c: INTEGER_32; status: SLOT_STATUS)
			-- Set the status of slot at row 'r' and column 'c' to 'status'.
		require
			valid_row: is_valid_row (r)
			valid_column: is_valid_column (c)
		ensure
			slot_set: imp.item (r, c).is_equal (status)
			slots_not_in_range_unchanged: matches_slots_except (old Current, r, r, c, c)

	set_statuses (r1, r2, c1, c2: INTEGER_32; status: SLOT_STATUS)
			-- Set the range of slots to 'status':
			-- intersection of rows 'r1' to 'r2' and
			-- columns 'c1' to 'c2'.
		require
			valid_rows: is_valid_row (r1) and is_valid_row (r2)
			valid_columns: is_valid_column (c1) and is_valid_column (c2)
			valid_row_range: (r2 - r1) >= 0
			valid_column_range: (c2 - c1) >= 0




		ensure
```

```
                slots_in_range_set: across
                        1 |..| number_of_rows as i
                    all
                        across
                            1 |..| number_of_columns as j
                        all
                        (i.item >= r1 and i.item <= r2 and j.item >= c1 and j.item <= c2) implies
(status_of (i.item, j.item) ~ status)
                        end
                    end
            slots_not_in_range_unchanged: matches_slots_except (old Current, r1, r2, c1,c2)

feature -- Auxiliary Queries

    matches_slots_except (other: BOARD; r1, r2, c1, c2: INTEGER_32): BOOLEAN
                -- Do slots outside the intersection of
                -- rows 'r1' to 'r2' and columns 'c1' and 'c2'
                -- match in Current and 'other'.
            require
                consistent_row_numbers: other.number_of_rows = number_of_rows
                consistent_column_numbers: other.number_of_columns = number_of_columns
                valid_rows:
 (r1 <= number_of_rows) and (r1 >= 1) and (r2 >= 1) and (r2 <= number_of_rows)
                valid_columns:
    (c1 <= number_of_columns) and (c1 >= 1) and (c2 >= 1) and (c2 <= number_of_columns)
                valid_row_range: r1 <= r2
                valid_column_range: c1 <= c2
            ensure
                correct_result: (Result = True) implies across
                        1 |..| number_of_rows as i
                    all
                        across
                            1 |..| number_of_columns as j
                        all
                            (i.item < r1 or i.item > r2 or j.item < c1 or j.item > c2)
implies status_of (i.item, j.item) ~ (other.status_of (i.item, j.item))
                        end
                    end

    occupied_slot: OCCUPIED_SLOT
                -- A slot available for moment but currently occupied.
            ensure
                    Result = ssa.occupied_slot

    unavailable_slot: UNAVAILABLE_SLOT
                -- A slot not available for movement.
            ensure
                    Result = ssa.Unavailable_slot
```

```
	unoccupied_slot: UNOCCUPIED_SLOT
			-- A slot available for moment and currently unoccupied.
		ensure
				Result = ssa.Unoccupied_slot

feature -- Constructor

	make_arrow
			-- Initialize a Arrow board.
		ensure
			board_set: Current ~ bta.Templates.arrow_board

	make_cross
			-- Initialize a Cross board.
		ensure
			board_set: Current ~ bta.Templates.cross_board

	make_default
			-- Initialize a default board with all slots unavailable.
		ensure
			board_set: Current ~ bta.Templates.default_board

	make_diamond
			-- Initialize a Diamond board.
		ensure
			board_set: Current ~ bta.Templates.diamond_board

	make_easy
			-- Initialize an easy board.
		ensure
			board_set: Current ~ bta.Templates.easy_board

	make_plus
			-- Initialize a Plus board.
		ensure
			board_set: Current ~ bta.Templates.plus_board

	make_pyramid
			-- Initialize a Pyramid board.
		ensure
			board_set: Current ~ bta.Templates.pyramid_board

	make_skull
			-- Initialize a Skull board.
		ensure
			board_set: Current ~ bta.Templates.skull_board
```

```
feature -- Equality

	is_equal (other: like Current): BOOLEAN
			-- Is current board equal to 'other'?
		ensure then
			correct_result: Result = out.is_equal (other.out)

feature -- Output

	out: STRING_8
			-- String representation of current board.

feature -- Queries

	is_valid_column (c: INTEGER_32): BOOLEAN
			-- Is 'c' a valid column number?
		ensure
			correct_result: Result = (c >= 1 and c <= number_of_columns)

	is_valid_row (r: INTEGER_32): BOOLEAN
			-- Is 'r' a valid row number?
		ensure
			correct_result: (r >= 1 and r <= number_of_rows) implies (Result = True)

	number_of_columns: INTEGER_32
			-- Number of columns in the board of game.
		ensure
			correct_result: Result = imp.width

	number_of_occupied_slots: INTEGER_32
			-- Number of slots occupied by pegs on current board.

	number_of_rows: INTEGER_32
			-- Number of rows in the board of game.
		ensure
			correct_result: Result = imp.height

	status_of (r, c: INTEGER_32): SLOT_STATUS
			-- Is the slot at row 'r' and column 'c'
			-- unavailable, occupied, or unoccupied?
		require
			valid_row: is_valid_row (r)
			valid_column: is_valid_column (c)
		ensure
			correct_result: Result = imp.item (r, c)

end -- class BOARD
```

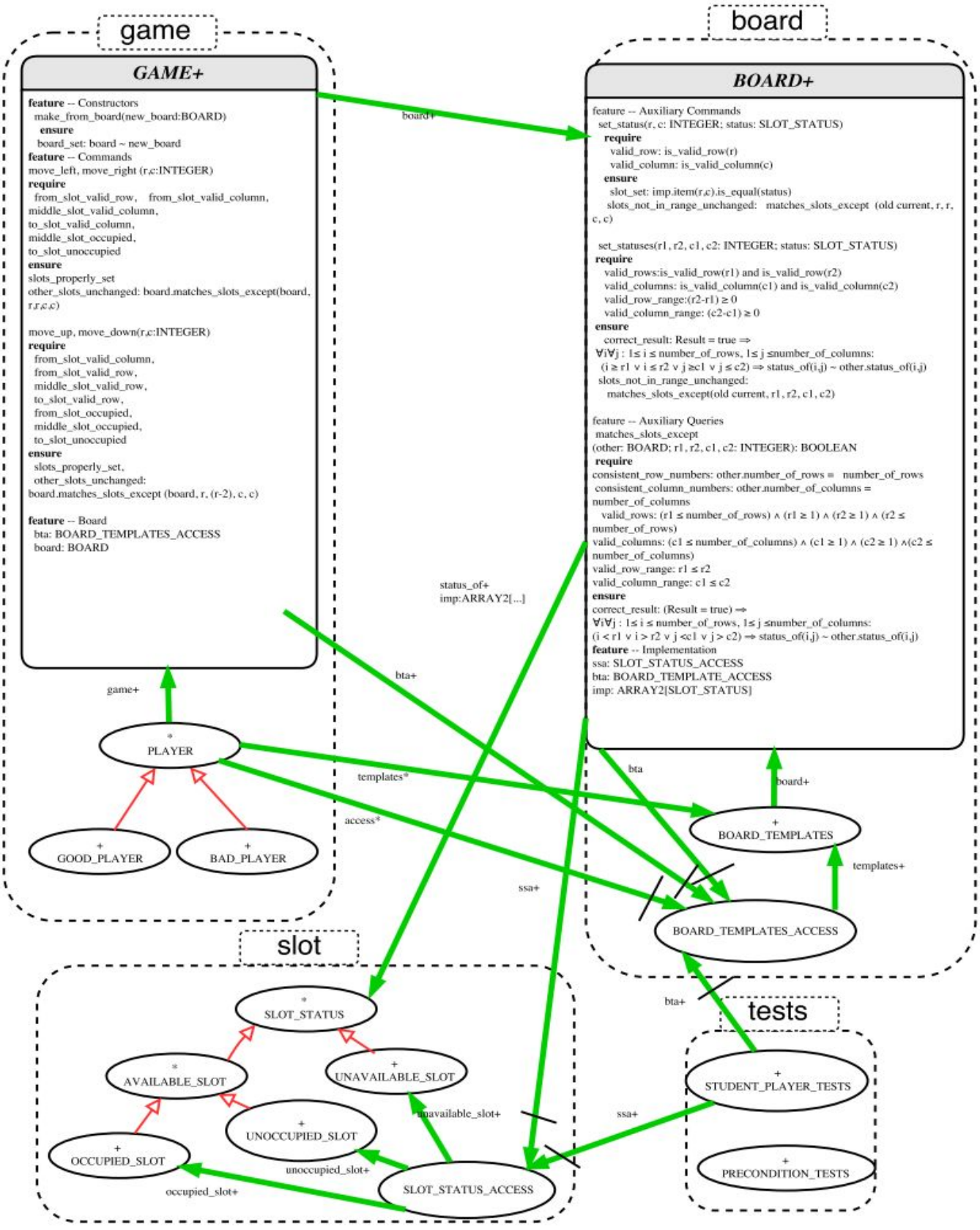
```
note
	description: "A game of peg solitaire."
	author: "Benjamin Korobkin"
	date: "October 17, 2017"
	revision: "N/A"

class interface
	GAME

create
	make_from_board,
	make_easy,
	make_cross,
	make_plus,
	make_pyramid,
	make_arrow,
	make_diamond,
	make_skull

feature -- Auxiliary Routines

	boolean_to_yes_no (b: BOOLEAN): STRING_8
			-- 'Yes' or 'No' corresponding to 'b'.

feature -- Board

	board: BOARD

	bta: BOARD_TEMPLATES_ACCESS

feature -- Commands

	move_down (r, c: INTEGER_32)
		require
			from_slot_valid_column: c >= 1 and c <= board.number_of_columns
			from_slot_valid_row: r >= 1 and r <= (board.number_of_rows - 2)
			middle_slot_valid_row: (r + 1) >= 2 and (r + 1) <= (board.number_of_rows - 1)
			to_slot_valid_row: (r + 2) >= 3 and (r + 2) <= (board.number_of_rows)
			from_slot_occupied: board.status_of (r, c) ~ board.occupied_slot
			middle_slot_occupied: board.status_of ((r + 1), c) ~ board.occupied_slot
			to_slot_unoccupied: board.status_of ((r + 2), c) ~ board.unoccupied_slot
		ensure
			slots_properly_set: board.status_of (r, c) ~ board.unoccupied_slot
				board.status_of ((r + 1), c) ~ board.unoccupied_slot
				board.status_of ((r + 2), c) ~ board.occupied_slot
			other_slots_unchanged: board.matches_slots_except (board, r, (r + 2), c, c)

	move_left (r, c: INTEGER_32)
```

```
            require
                from_slot_valid_row: r >= 1 and r <= board.number_of_rows
                from_slot_valid_column: c >= 3 and c <= board.number_of_columns
                middle_slot_valid_column: (c - 1) >= 2 and (c - 1) <=
(board.number_of_columns - 1)
                to_slot_valid_column: (c - 2) >= 1 and (c - 2) <= (board.number_of_columns - 2)
                from_slot_occupied: board.status_of (r, c) ~ board.occupied_slot
                middle_slot_occupied: board.status_of (r, (c - 1)) ~ board.occupied_slot
                to_slot_unoccupied: board.status_of (r, (c - 2)) ~ board.unoccupied_slot
            ensure
                slots_properly_set: board.status_of (r, c) ~ board.unoccupied_slot
                        board.status_of (r, c - 1) ~ board.unoccupied_slot
                        board.status_of (r, c - 2) ~ board.occupied_slot
                other_slots_unchanged: board.matches_slots_except (board, r, r, c - 2, c)

        move_right (r, c: INTEGER_32)
            require
                from_slot_valid_row: r >= 1 and r <= board.number_of_rows
                from_slot_valid_column: c >= 1 and c <= (board.number_of_columns - 2)
                middle_slot_valid_column: (c + 1) >= 2 and (c + 1) <=
(board.number_of_columns - 1)
                to_slot_valid_column: (c + 2) >= 3 and (c + 2) <= board.number_of_columns
                from_slot_occupied: board.status_of (r, c) ~ board.occupied_slot
                middle_slot_occupied: board.status_of (r, (c + 1)) ~ board.occupied_slot
                to_slot_unoccupied: board.status_of (r, (c + 2)) ~ board.unoccupied_slot
            ensure
                slots_properly_set: board.status_of (r, c) ~ board.unoccupied_slot
                        board.status_of (r, (c + 1)) ~ board.unoccupied_slot
                        board.status_of (r, (c + 2)) ~ board.occupied_slot
                other_slots_unchanged: board.matches_slots_except (board, r, r, c, (c + 2))

        move_up (r, c: INTEGER_32)
            require
                from_slot_valid_column: c >= 1 and c <= board.number_of_columns
                from_slot_valid_row: r >= 3 and r <= board.number_of_rows
                middle_slot_valid_row: (r - 1) >= 2 and (r - 1) <= (board.number_of_rows - 1)
                to_slot_valid_row: (r - 2) >= 1 and (r - 2) <= (board.number_of_rows - 2)
                from_slot_occupied: board.status_of (r, c) ~ board.occupied_slot
                middle_slot_occupied: board.status_of ((r - 1), c) ~ board.occupied_slot
                to_slot_unoccupied: board.status_of ((r - 2), c) ~ board.unoccupied_slot
            ensure
                slots_properly_set: board.status_of (r, c) ~ board.unoccupied_slot
                        board.status_of ((r - 1), c) ~ board.unoccupied_slot
                        board.status_of ((r - 2), c) ~ board.occupied_slot
                other_slots_unchanged: board.matches_slots_except (board, r, (r - 2), c, c)

feature -- Constructors
```

```
	make_arrow
			-- Initialize a game with Arrow board.
		ensure
			board_set: board ~ bta.Templates.arrow_board

	make_cross
			-- Initialize a game with Cross board.
		ensure
			board_set: board ~ bta.Templates.cross_board

	make_diamond
			-- Initialize a game with Diamond board.
		ensure
			board_set: board ~ bta.Templates.diamond_board

	make_easy
			-- Initialize a game with easy board.
		ensure
			board_set: board ~ bta.Templates.easy_board

	make_from_board (new_board: BOARD)
			-- Initialize a game with 'new_board'.
		ensure
			board_set: board ~ new_board

	make_plus
			-- Initialize a game with Plus board.
		ensure
			board_set: board ~ bta.Templates.plus_board

	make_pyramid
			-- Initialize a game with Pyramid board.
		ensure
			board_set: board ~ bta.Templates.pyramid_board

	make_skull
			-- Initialize a game with Skull board.
		ensure
			board_set: board ~ bta.Templates.skull_board

feature -- Output

	out: STRING_8
			-- String representation of current game.
			-- Do not modify this feature!

feature -- Status Queries
```

```
	is_over: BOOLEAN
			-- Is the current game 'over'?
			-- i.e., no further movements are possible.
		ensure
			correct_result: across
					1 |..| board.number_of_rows as i
				all
					across
						1 |..| board.number_of_columns as j
					all
(j.item >= 3 and board.status_of (i.item, j.item) ~ board.occupied_slot and board.status_of (i.item,
(j.item - 1)) ~ board.occupied_slot and board.status_of (i.item, (j.item - 2)) ~ board.unoccupied_slot) or
(j.item <= 5 and board.status_of (i.item, j.item) ~ board.occupied_slot and board.status_of (i.item,
(j.item + 1)) ~ board.occupied_slot and board.status_of (i.item, (j.item + 2)) ~ board.unoccupied_slot)
or (i.item >= 3 and board.status_of (i.item, j.item) ~ board.occupied_slot and board.status_of ((i.item -
1), j.item) ~ board.occupied_slot and board.status_of ((i.item - 2), j.item) ~ board.unoccupied_slot) or
(i.item <= 5 and board.status_of (i.item, j.item) ~ board.occupied_slot and board.status_of ((i.item + 1),
j.item) ~ board.occupied_slot and board.status_of ((i.item + 2), j.item) ~ board.unoccupied_slot)
implies (Result = False)
					end
				end

	is_won: BOOLEAN
			-- Has the current game been won?
			-- i.e., there's only one occupied slot on the board.
		ensure
			game_won_iff_one_occupied_slot_left: (Result = True) implies
					(board.number_of_occupied_slots = 1)
			winning_a_game_means_game_over: (Result = True) implies (is_over = True)
	end -- class GAME
```

BON Diagram:


game
GAME+
feature -- Constructors
make_from_board(new_board:BOARD)
ensure
board_set: board ~ new_board
feature -- Commands
move_left, move_right (r,c:INTEGER)
require
from_slot_valid_row, from_slot_valid_column,
middle_slot_valid_column,
to_slot_valid_column,
middle_slot_occupied,
to_slot_unoccupied
ensure
slots_properly_set
other_slots_unchanged: board.matches_slots_except(board, r,r,c,c)
move_up, move_down(r,c:INTEGER)
require
from_slot_valid_column,
from_slot_valid_row,
middle_slot_valid_row,
to_slot_valid_row,
from_slot_occupied,
middle_slot_occupied,
to_slot_unoccupied
ensure
slots_properly_set,
other_slots_unchanged:
board.matches_slots_except (board, r, (r-2), c, c)
feature -- Board
bta: BOARD_TEMPLATES_ACCESS
board: BOARD
board
BOARD+
feature -- Auxiliary Commands
set_status(r, c: INTEGER; status: SLOT_STATUS)
require
valid_row: is_valid_row(r)
valid_column: is_valid_column(c)
ensure
slot_set: imp.item(r,c).is_equal(status)
slots_not_in_range_unchanged: matches_slots_except (old current, r, r, c, c)
set_statuses(r1, r2, c1, c2: INTEGER; status: SLOT_STATUS)
require
valid_rows:is_valid_row(r1) and is_valid_row(r2)
valid_columns: is_valid_column(c1) and is_valid_column(c2)
valid_row_range:(r2-r1) ≥ 0
valid_column_range: (c2-c1) ≥ 0
ensure
correct_result: Result = true ⇒
∀i∀j : 1≤ i ≤ number_of_rows, 1≤ j ≤number_of_columns:
(i ≥ r1 ∨ i ≤ r2 ∨ j ≥c1 ∨ j ≤ c2) ⇒ status_of(i,j) ~ other.status_of(i,j)
slots_not_in_range_unchanged:
matches_slots_except(old current, r1, r2, c1, c2)
feature -- Auxiliary Queries
matches_slots_except
(other: BOARD; r1, r2, c1, c2: INTEGER): BOOLEAN
require
consistent_row_numbers: other.number_of_rows = number_of_rows
consistent_column_numbers: other.number_of_columns = number_of_columns
valid_rows: (r1 ≤ number_of_rows) ∧ (r1 ≥ 1) ∧ (r2 ≥ 1) ∧ (r2 ≤ number_of_rows)
valid_columns: (c1 ≤ number_of_columns) ∧ (c1 ≥ 1) ∧ (c2 ≥ 1) ∧(c2 ≤ number_of_columns)
valid_row_range: r1 ≤ r2
valid_column_range: c1 ≤ c2
ensure
correct_result: (Result = true) ⇒
∀i∀j : 1≤ i ≤ number_of_rows, 1≤ j ≤number_of_columns:
(i < r1 ∨ i > r2 ∨ j <c1 ∨ j > c2) ⇒ status_of(i,j) ~ other.status_of(i,j)
feature -- Implementation
ssa: SLOT_STATUS_ACCESS
bta: BOARD_TEMPLATE_ACCESS
imp: ARRAY2[SLOT_STATUS]
board+
status_of+
imp:ARRAY2[...]
bta+
game+
PLAYER
templates*
access*
GOOD_PLAYER
BAD_PLAYER
bta
board+
BOARD_TEMPLATES
templates+
ssa+
BOARD_TEMPLATES_ACCESS
slot
SLOT_STATUS
AVAILABLE_SLOT
UNAVAILABLE_SLOT
unavailable_slot+
UNOCCUPIED_SLOT
OCCUPIED_SLOT
unoccupied_slot+
occupied_slot+
SLOT_STATUS_ACCESS
bta+
tests
ssa+
STUDENT_PLAYER_TESTS
PRECONDITION_TESTS

Tests
matches_slots_except(other: BOARD; r1, r2, c1, c2: INTEGER) : BOOLEAN

```
test_matches_slots_except_pre1
	local
		g : GAME
		flag : BOOLEAN
	do
		comment("Invalid column range triggers precondition")
		create g.make_easy
		flag := g.board.matches_slots_except (g.board, 1, 2, 2, 1)
	end
```

The above test causes a **precondition** violation because the column range is invalid, as the value of c1 must be less than or equivalent to c2 (here 2 and 1 respectively). Since 2 is clearly greater than 1, the precondition is violated.

```
test_matches_slots_except_pre2
	local
		g:GAME
		flag:BOOLEAN
	do
		comment("Invalid row range triggers precondition")
		create g.make_easy
		flag := g.board.matches_slots_except (g.board, 5, 2, 6, 7)
	end
```

The above test causes a **precondition** violation because the row range is invalid, as the value of r1 must be less than or equal to r2 (here 5 and 2 respectively). Since 5 is clearly greater than 2, the precondition is violated.

```
test_matches_slots_except: BOOLEAN
		local
			g,f,h:GAME
			flag:BOOLEAN
		do
			comment ("test: matches_slots_except doesn't work with 2 different
boards. works with 2 of the same.")
			create g.make_arrow
			create f.make_cross
			flag := g.board.matches_slots_except (f.board, 1, 2, 1, 2)
			if flag = false then
				Result := true else Result := false
			end
			check Result end
			create h.make_arrow
			Result := h.board.matches_slots_except (g.board,2,2,3,3)
			check Result end
```

The above contains 2 Boolean test cases (**normal scenarios**). We first test to make sure that if two different boards are initialized, their slots do not match. We do this using a Boolean called flag. The Result is triggered if the boards do not match, causing the flag to initialize to false.
The second test Boolean test cases ensures that if two boards of the same type are initialized, their slots both match. Board g is an arrow board as is board h. Thus, they match and the Result is true.