Lab 2

COURSE: EECS 3311

FALL 2017

NAME: Saad Saeed

CSE login: saeeds28

Contract view

Game:

```
note
	description: "A game of peg solitaire."
	author: ""
	date: "$Date$"
	revision: "$Revision$"

class
	GAME

inherit
	ANY
		redefine
			out
		end

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

feature -- Constructors

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

```
        end

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

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

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

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

make_arrow
                -- Initialize a game with Arrow board.
```

```
		do
			create board.make_arrow
		ensure
			board_set: board.out ~ bta.Templates.arrow_board.out
		end

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

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

feature -- Commands

	move_left (r, c: INTEGER_32)
		require
			from_slot_valid_row: board.is_valid_row (r)
			from_slot_valid_column: c > 2 and c <= 7
			middle_slot_valid_column: c - 1 > 1 and c - 1 < 7
			to_slot_valid_column: c - 2 >= 1 and c - 2 < 6
			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
		do
			board.set_status (r, c, board.unoccupied_slot)
			board.set_status (r, c - 1, board.unoccupied_slot)
```

```
            board.set_status (r, c - 2, board.occupied_slot)
        ensure
            slots_properly_set: board.status_of (r, c) ~ board.unoccupied_slot and board.status_of (r, c -
1) ~ board.unoccupied_slot and board.status_of (r, c - 2) ~ board.occupied_slot
            other_slots_unchanged: board.matches_slots_except (board, r, r, c, c - 2)
        end

move_right (r, c: INTEGER_32)
        require
            from_slot_valid_row: board.is_valid_row (r)
            from_slot_valid_column: c >= 1 and c < 6
            middle_slot_valid_column: c + 1 >= 2 and c + 1 < 7
            to_slot_valid_column: c + 2 > 2 and c + 2 <= 7
            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
        do
            board.set_status (r, c, board.unoccupied_slot)
            board.set_status (r, c + 1, board.unoccupied_slot)
            board.set_status (r, c + 2, board.occupied_slot)
        ensure
            slots_properly_set: board.status_of (r, c) ~ board.unoccupied_slot and board.status_of (r, c +
1) ~ board.unoccupied_slot and board.status_of (r, c + 2) ~ board.occupied_slot
            other_slots_unchanged: board.matches_slots_except (board, r, r, c, c + 2)
        end

move_up (r, c: INTEGER_32)
        require
            from_slot_valid_column: board.is_valid_column (c)
            from_slot_valid_row: r > 2 and r <= 7
            middle_slot_valid_row: r - 1 >= 2 and r - 1 < 7
            to_slot_valid_row: r - 2 >= 1 and r - 2 < 6
            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
        do
```

```
                board.set_status (r - 2, c, board.occupied_slot)
                board.set_statuses (r - 1, r, c, c, board.unoccupied_slot)
            ensure
                slots_properly_set: board.status_of (r, c) ~ board.unoccupied_slot and board.status_of (r - 1,
        c) ~ board.unoccupied_slot and board.status_of (r - 2, c) ~ board.occupied_slot
                other_slots_unchanged: board.matches_slots_except (board, r - 2, r, c, c)
            end

        move_down (r, c: INTEGER_32)
            require
                from_slot_valid_column: board.is_valid_column (c)
                from_slot_valid_row: r >= 1 and r < 6
                middle_slot_valid_row: r + 1 >= 2 and r + 1 < 7
                to_slot_valid_row: r + 2 > 2 and r + 2 <= 7
                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
            do
                board.set_status (r + 2, c, board.occupied_slot)
                board.set_statuses (r, r + 1, c, c, board.unoccupied_slot)
            ensure
                slots_properly_set: board.status_of (r, c) ~ board.unoccupied_slot and board.status_of (r + 1,
        c) ~ board.unoccupied_slot and board.status_of (r + 2, c) ~ board.occupied_slot
                other_slots_unchanged: board.matches_slots_except (board, r, r + 1, c, c)
            end

    feature {NONE} -- Moves Query

        moves_possible (r, c: INTEGER_32): BOOLEAN
                -- Checks to see if a move is possible in all four directions
            local
                right_move: BOOLEAN
                left_move: BOOLEAN
                up_move: BOOLEAN
                down_move: BOOLEAN
                bounded_right: BOOLEAN
```

```
                bounded_left: BOOLEAN
                bounded_up: BOOLEAN
                bounded_down: BOOLEAN
        do
                bounded_right := (c + 2) <= 7
                bounded_left := (c - 2) >= 1
                bounded_up := (r - 2) >= 1
                bounded_down := (r + 2) <= 7
                if bounded_right = True then
                        right_move := board.status_of (r, c) = board.occupied_slot and board.status_of (r, c
    + 1) = board.occupied_slot and board.status_of (r, c + 2) = board.unoccupied_slot
                end
                if bounded_left = True then
                        left_move := board.status_of (r, c) = board.occupied_slot and board.status_of (r, c -
    1) = board.occupied_slot and board.status_of (r, c - 2) = board.unoccupied_slot
                end
                if bounded_up = True then
                        up_move := board.status_of (r, c) = board.occupied_slot and board.status_of (r - 1,
    c) = board.occupied_slot and board.status_of (r - 2, c) = board.unoccupied_slot
                end
                if bounded_down = True then
                        down_move := board.status_of (r, c) = board.occupied_slot and board.status_of (r +
    1, c) = board.occupied_slot and board.status_of (r + 2, c) = board.unoccupied_slot
                end
                Result := right_move or left_move or up_move or down_move
        end

feature -- Status Queries

        is_over: BOOLEAN
                -- Is the current game 'over'?
                -- i.e., no further movements are possible.
        do
                Result := not across
                        1 |..| board.number_of_rows as m
                some
```

```
                        across
                                1 |..| board.number_of_columns as n
                        some
                                moves_possible (m.item, n.item)
                        end
                end
        ensure
                correct_result: Result = not across
                                1 |..| board.number_of_rows as m
                        some
                                across
                                        1 |..| board.number_of_columns as n
                                some
                                        moves_possible (m.item, n.item)
                                end
                        end
        end

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

feature -- Output

    out: STRING_8
                -- String representation of current game.
                -- Do not modify this feature!
        do
                create Result.make_empty
                Result.append ("Game is over: " + boolean_to_yes_no (is_over) + "%N")
```

```
                Result.append ("Game is won : " + boolean_to_yes_no (is_won) + "%N")
                Result.append ("Board Status:%N")
                Result.append (board.out)
        end

feature -- Auxiliary Routines

        boolean_to_yes_no (b: BOOLEAN): STRING_8
                -- 'Yes' or 'No' corresponding to 'b'.
        do
                if b then
                        Result := "Yes"
                else
                        Result := "No"
                end
        end

feature -- Board

        bta: BOARD_TEMPLATES_ACCESS

        board: BOARD

end -- class GAME
```

Board

```
note
        description: "A board for the peg solitaire game."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class
        BOARD
```

```
inherit
	ANY
		redefine
			out, is_equal
		end

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


feature -- Constructor
	make_default
			-- Initialize a default board with all slots unavailable.
		do
			create imp.make_filled (unavailable_slot, 7, 7)
		ensure
			board_set:
				Current ~bta.templates.default_board
		end

	make_easy
			-- Initialize an easy board.
		do
			make_default
			set_status (1, 4, unoccupied_slot)
			set_status (4, 4, unoccupied_slot)
			set_status (6, 4, unoccupied_slot)
			set_statuses (2, 3, 4, 4, occupied_slot)
			set_status (5, 4, occupied_slot)
```

```
        ensure
                board_set: Current ~ bta.templates.easy_board
        end

make_cross
                -- Initialize a Cross board.
        do
                make_default
                set_statuses (1, 7, 1, 7, unoccupied_slot)
                set_statuses (1, 1, 1, 2, unavailable_slot)
                set_statuses (1, 1,6,7,unavailable_slot)
                set_statuses (2, 2, 1, 2, unavailable_slot)
                set_statuses (2, 2, 6, 7, unavailable_slot)
                set_statuses (6, 6, 1, 2, unavailable_slot)
                set_statuses (6, 6, 6, 7, unavailable_slot)
                set_statuses (7, 7, 1, 2, unavailable_slot)
                set_statuses (7, 7, 6, 7, unavailable_slot)
                set_status (2, 4, occupied_slot)
                set_status (4, 4, occupied_slot)
                set_status (5, 4, occupied_slot)
                set_statuses (3, 3, 3, 5, occupied_slot)

        ensure
                board_set:
                        current ~ bta.templates.cross_board
        end

make_plus
                -- Initialize a Plus board.
        do
                make_default
                set_statuses (1, 7, 1, 7, unoccupied_slot)
                set_statuses (1, 1, 1, 2, unavailable_slot)
                set_statuses (1, 1,6,7,unavailable_slot)
                set_statuses (2, 2, 1, 2, unavailable_slot)
                set_statuses (2, 2, 6, 7, unavailable_slot)
```

```
		set_statuses (6, 6, 1, 2, unavailable_slot)
		set_statuses (6, 6, 6, 7, unavailable_slot)
		set_statuses (7, 7, 1, 2, unavailable_slot)
		set_statuses (7, 7, 6, 7, unavailable_slot)
		set_status (2, 4, occupied_slot)
		set_status (3, 4, occupied_slot)
		set_statuses (4, 4, 2, 6, occupied_slot)
		set_status (5, 4, occupied_slot)
		set_status (6, 4, occupied_slot)
	ensure
		board_set:
			current~bta.templates.plus_board
	end

make_pyramid
		-- Initialize a Pyramid board.
	do
		make_default
		set_statuses (1, 7, 1, 7, unoccupied_slot)
		set_statuses (1, 1, 1, 2, unavailable_slot)
		set_statuses (1, 1,6,7,unavailable_slot)
		set_statuses (2, 2, 1, 2, unavailable_slot)
		set_statuses (2, 2, 6, 7, unavailable_slot)
		set_statuses (6, 6, 1, 2, unavailable_slot)
		set_statuses (6, 6, 6, 7, unavailable_slot)
		set_statuses (7, 7, 1, 2, unavailable_slot)
		set_statuses (7, 7, 6, 7, unavailable_slot)
		set_status (2, 4, occupied_slot)
		set_statuses (3, 3, 3,5, occupied_slot)
		set_statuses (4, 4, 2, 6, occupied_slot)
		set_statuses (5, 5, 1, 7, occupied_slot)
	ensure
		board_set:
			current~bta.templates.pyramid_board
	end
```

```
make_arrow
            -- Initialize a Arrow board.
        do
            make_default
            set_statuses (1, 7, 1, 7, unoccupied_slot)
            set_statuses (1, 1, 1, 2, unavailable_slot)
            set_statuses (1, 1,6,7,unavailable_slot)
            set_statuses (2, 2, 1, 2, unavailable_slot)
            set_statuses (2, 2, 6, 7, unavailable_slot)
            set_statuses (6, 6, 1, 2, unavailable_slot)
            set_statuses (6, 6, 6, 7, unavailable_slot)
            set_statuses (7, 7, 1, 2, unavailable_slot)
            set_statuses (7, 7, 6, 7, unavailable_slot)
            set_status (1,4, occupied_slot)
            set_statuses (2, 2, 3, 5, occupied_slot)
            set_statuses (3, 3, 2, 6, occupied_slot)
            set_status (4, 4, occupied_slot)
            set_status (5, 4, occupied_slot)
            set_statuses (6, 6, 3, 5, occupied_slot)
            set_statuses (7, 7, 3, 5, occupied_slot)
            ensure
            board_set:
                    current ~ bta.templates.arrow_board
        end

make_diamond
            -- Initialize a Diamond board.
        do
            make_default
            set_statuses (1, 7, 1, 7, unoccupied_slot)
            set_statuses (1, 1, 1, 2, unavailable_slot)
            set_statuses (1, 1, 6, 7,unavailable_slot)
            set_statuses (2, 2, 1, 2, unavailable_slot)
            set_statuses (2, 2, 6, 7, unavailable_slot)
            set_statuses (6, 6, 1, 2, unavailable_slot)
            set_statuses (6, 6, 6, 7, unavailable_slot)
```

```
                set_statuses (7, 7, 1, 2, unavailable_slot)
                set_statuses (7, 7, 6, 7, unavailable_slot)
                set_status (1,4, occupied_slot)
                set_statuses (2, 2, 3, 5, occupied_slot)
                set_statuses (3, 3, 2, 6, occupied_slot)
                set_statuses (4, 4, 1, 3, occupied_slot)
                set_statuses (4, 4, 5, 7, occupied_slot)
                set_statuses (5, 5, 2, 6, occupied_slot)
                set_statuses (6, 6, 3, 5, occupied_slot)
                set_status (7, 4, occupied_slot)
        ensure
                board_set:
                        current~bta.templates.diamond_board
        end

make_skull
                -- Initialize a Skull board.
        do
                make_default
                set_statuses (1, 7, 1, 7, unoccupied_slot)
                set_statuses (1, 1, 1, 2, unavailable_slot)
                set_statuses (1, 1, 6, 7,unavailable_slot)
                set_statuses (2, 2, 1, 2, unavailable_slot)
                set_statuses (2, 2, 6, 7, unavailable_slot)
                set_statuses (6, 6, 1, 2, unavailable_slot)
                set_statuses (6, 6, 6, 7, unavailable_slot)
                set_statuses (7, 7, 1, 2, unavailable_slot)
                set_statuses (7, 7, 6, 7, unavailable_slot)
                set_statuses (1, 1, 3, 5, occupied_slot)
                set_statuses (2, 2, 3, 5, occupied_slot)
                set_statuses (3, 3, 2, 6, occupied_slot)
                set_status (4, 2, occupied_slot)
                set_status (4, 4, occupied_slot)
                set_status (4, 6, occupied_slot)
                set_statuses (5, 5, 2, 6, occupied_slot)
                set_statuses (6, 6, 3, 5, occupied_slot)
```

```
                set_statuses (7, 7, 3, 5, occupied_slot)
        ensure
                board_set:
                        current~bta.templates.skull_board
        end

feature -- Auxiliary Commands
    set_status (r, c: INTEGER; status: SLOT_STATUS)
                -- Set the status of slot at row 'r' and column 'c' to 'status'.
        require
                valid_row:
                        is_valid_column(r)
                valid_column:
                        is_valid_column(c)
        do
                imp.put (status, r, c)
        ensure
                slot_set:
                        imp.item (r, c) ~status
                slots_not_in_range_unchanged:
                        matches_slots_except(current,r,r,c,c)
        end

    set_statuses (r1, r2, c1, c2: INTEGER; 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:
                        r1<=r2
                valid_column_range:
                        c1<=c2
```

```
		do
			across
				r1 |..| r2 as n
			loop
				across
					c1 |..| c2 as m
				loop
					imp.put(status, n.item, m.item)
				end
			end
		ensure
			slots_in_range_set:
				across
					r1 |..| r2 as n
				all
					across
						c1 |..| c2 as m
					all
						imp.item (n.item, m.item)~status
					end
				end
			slots_not_in_range_unchanged:
				matches_slots_except (current, r1,r2,c1,c2)
		end

feature -- Auxiliary Queries
	matches_slots_except (
			other: BOARD; r1, r2, c1, c2: INTEGER)
	: 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:
					current.number_of_rows=other.number_of_rows
				consistent_column_numbers:
```

```
                current.number_of_columns=other.number_of_columns
        valid_rows:
                is_valid_row(r1) and is_valid_row(r2)
        valid_columns:
                is_valid_column(r1) and is_valid_column(r2)
        valid_row_range:
                r1<=r2
        valid_column_range:
                c1<=c2
    do
        Result:= true
        across
                1 |..| number_of_rows as m
        loop
                across
                        1 |..| number_of_columns as n
                loop
                        if      (n.item < c1 or n.item > c2) or (m.item < r1 or m.item > r2)
                        then
                                Result:= Current.status_of (n.item, m.item) ~ other.status_of
(n.item, m.item)
                end
        end
    ensure
        correct_result:
                across
                        1 |..| number_of_rows as i
                all
                        across
                                1 |..| number_of_columns as j
                        all
                                Current.status_of (i.item,j.item) = other.status_of (i.item,j.item) or
(i.item <= r2 and i.item >= r1 and j.item >= c1 and j.item <= c2 )
                end
        end
```

```
		end

	unavailable_slot: UNAVAILABLE_SLOT
			-- A slot not available for movement.
		do
			Result := ssa.unavailable_slot
		ensure
			Result = ssa.unavailable_slot
		end

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

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

feature -- Queries
	number_of_rows: INTEGER
			-- Number of rows in the board of game.
		do
			Result := imp.height
		ensure
			correct_result:
				Result = imp.height
		end
```

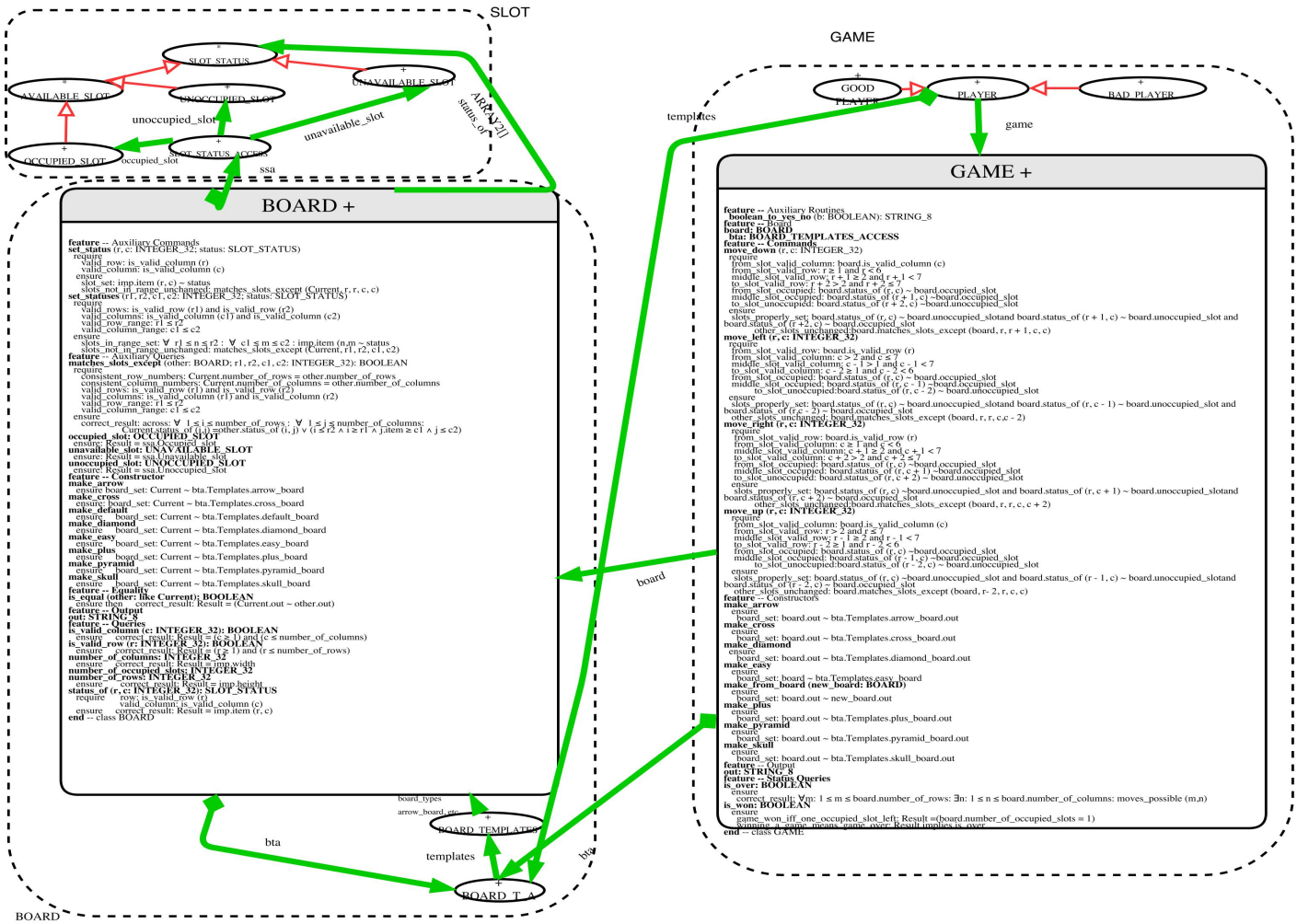
```
number_of_columns: INTEGER
                -- Number of columns in the board of game.
        do
                Result := imp.width
        ensure
                correct_result:
                        Result = imp.width
        end

is_valid_row (r: INTEGER): BOOLEAN
                -- Is 'r' a valid row number?
        do
                Result := r>=1 and r<= number_of_rows
        ensure
                correct_result:
                        Result = (r>=1) and (r<= number_of_rows)
        end

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

status_of (r, c: INTEGER): 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)
        do
```

```
                Result := imp.item (r, c)
        ensure
                correct_result:
                        Result = imp.item (r, c)
        end

    number_of_occupied_slots: INTEGER
                -- Number of slots occupied by pegs on current board.
        do
                across
                        1 |..| number_of_rows as n
                loop
                        across
                                1 |..| number_of_columns as m
                        loop
                                if status_of(n.item,m.item) = occupied_slot then
                                        result:=result+1
                                end
                        end
                end
        end

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

feature -- Output
    out: STRING
                -- String representation of current board.
        do
```

```
                create Result.make_empty

                --Result := Current.out
                across
                        1|..| number_of_rows as m
                loop
                        across
                                1 |..| number_of_columns as n
                        loop
                                if imp.item (m.item, n.item) ~ ssa.unavailable_slot
                                then
                                        Result.append ("*")
                                end

                                if imp.item (m.item, n.item) ~ ssa.occupied_slot
                                then
                                        Result.append ("O")
                                end

                                if imp.item (m.item, n.item) ~ ssa.unoccupied_slot
                                then
                                        Result.append (".")
                                end
                        end
                        Result.append ("%N")
                end

                Result.remove_tail (1)

        end

feature {NONE} -- Implementation

        ssa:SLOT_STATUS_ACCESS
        bta: BOARD_TEMPLATES_ACCESS
```

```
        -- Note: ARRAY2 takes row (y) and then column (x)
        imp: ARRAY2[SLOT_STATUS]
end
```

Architectural Diagram



TESTS

Precondition violation

test_correct_matches_slot_except_precondition

```
local
	board:BOARD
	test: BOOLEAN
do
	comment ("test: matches_slots_except_precondition for valid_row_range")
	create board.make_diamond
	test:= board.matches_slots_except (board, 6, 5, 2, 3)
end
```

This throws a valid_row_range exception because the r1 is supposed to be less than or equal to r2. In this case, this is not the case since r1 (6) is greater than r2 (5). Hence the correct precondition violation is thrown.

Postcondition violation

test_bad_matches_slots_except_postcondition

```
local
	board: BOARD
board2: BAD_MATCHES_SLOT_EXCEPT
test: BOOLEAN
do
comment ("test: matches_slots_except postcondition violation")
create board.make_diamond
create board2.make
test:=  board2.matches_slots_except (board, 1, 2, 3, 4)
end
```

Purposely tampered version of matches_slots_except

```
class
   BAD_MATCHES_SLOT_EXCEPT
inherit
   BOARD
      redefine
         matches_slots_except
      end
create
   make

feature
```

```
    make
    do
        create imp.make_filled (ssa.occupied_slot, 7, 7)
    end

    feature --bad matches
    matches_slots_except(board: BOARD; r1,r2,c1,c2:INTEGER):BOOLEAN
        do
            Result := not Precursor (board, r1,r2,c1,c2)
        end

end
```

A new class was defined where the matches_slots_except method is redefined in such a way that it negates the actual result of the correctly implemented method. The correct_result postcondition ensures that whatever the method returns is correct by looping through the board ranges. If we negate the whatever is returned, obviously the result does not match and the correct_result is thrown as expected.

Correct run

```
test_correct_matches_slots_except :BOOLEAN
        local
                board1,board2:BOARD
        do
                comment ("test: correct run for the matches_slots_except")
                create board1.make_arrow
                create board2.make_arrow

                board1.set_statuses (4, 4, 2, 3, board1.unavailable_slot)
                result:= board1.matches_slots_except (board2, 4, 4, 2, 3)
                check result end
        end
```

This is the correct implementation.