Lab 2: Report

EECS3311,
Fall 2017.
Michael Mierzwa.
213550702
mierzwam

Section 1: Contract View

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

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'
                  'status'.
       require
             valid_row: is_valid_row (r)
             valid_column: is_valid_column (c)
       ensure
             slot_set: imp.item (r, c) ~ status
             slots_not_in_range_unchanged: matches_slots_except
                                        (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: r1 <= r2
             valid_column_range: c1 <= c2
```

```
        ensure
              slots_in_range_set: across
                          r1 |..| r2 as i
                    all
                          across
                                c1 |..| c2 as j
                          all
                                imp.item (i.item, j.item) ~ status
                          end
                    end
              slots_not_in_range_unchanged: matches_slots_except
                                            (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: 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 (c1) and is_valid_column
                                 (c2)
              valid_row_range: r1 <= r2
              valid_column_range: c1 <= c2
        ensure
              correct_result: Result 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 other.status_of
                                (i.item, j.item) ~ 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_output: Result = (Current.out ~ other.out)

feature -- Output

    out: STRING_8
            -- String representation of current board.

feature -- Queries

    is_valid_column (c: INTEGER_32): BOOLEAN
            -- Is 'x' 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: Result = (r >= 1 and r <= number_of_rows)

    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: "Michael Mierzwa"
    date: "$Date$"
    revision: "$Revision$"

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'.

    can_down (r, c: INTEGER_32): BOOLEAN

    can_left (r, c: INTEGER_32): BOOLEAN

    can_move (r, c: INTEGER_32): BOOLEAN

    can_right (r, c: INTEGER_32): BOOLEAN

    can_up (r, c: INTEGER_32): BOOLEAN


feature -- Board
```

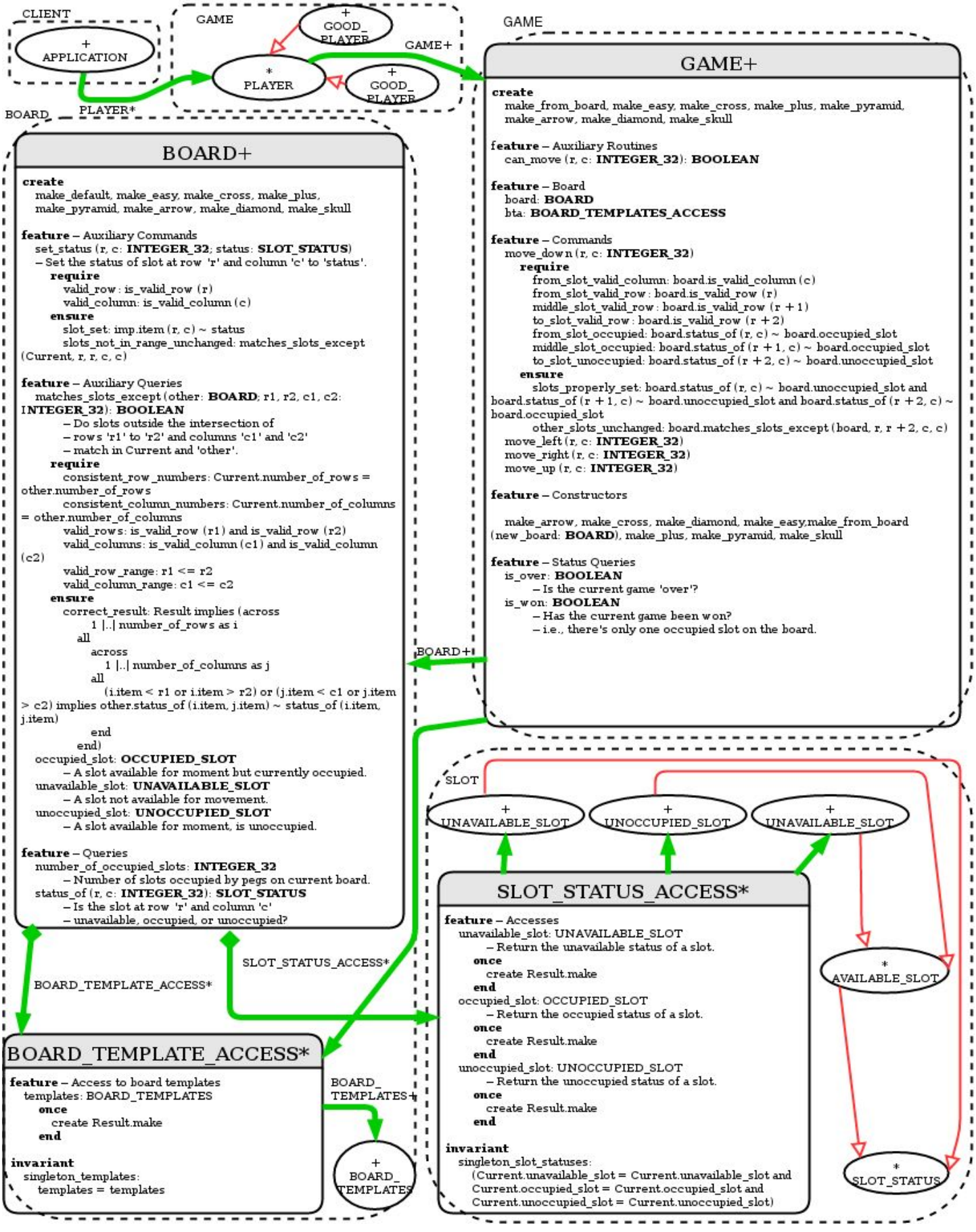
```
    board: BOARD

    bta: BOARD_TEMPLATES_ACCESS

feature -- Commands

    move_down (r, c: INTEGER_32)
       require
             from_slot_valid_column: board.is_valid_column (c)
             from_slot_valid_row: board.is_valid_row (r)
             middle_slot_valid_row: board.is_valid_row (r + 1)
             to_slot_valid_row: board.is_valid_row (r + 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 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 + 2, c, c)

    move_left (r, c: INTEGER_32)
       require
             from_slot_valid_row: board.is_valid_row (r)
             from_slot_valid_column: board.is_valid_column (c)
             middle_slot_valid_column: board.is_valid_column ((c - 1))
             to_slot_valid_column: board.is_valid_column ((c - 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 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)

    move_right (r, c: INTEGER_32)
```

```
        require
              from_slot_valid_row: board.is_valid_row (r)
              from_slot_valid_column: board.is_valid_column (c)
              middle_slot_valid_column: board.is_valid_column ((c + 1))
              to_slot_valid_column: board.is_valid_column ((c + 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 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))

    move_up (r, c: INTEGER_32)
        require
              from_slot_valid_column: board.is_valid_column (c)
              from_slot_valid_row: board.is_valid_row (r)
              middle_slot_valid_row: board.is_valid_row (r - 1)
              to_slot_valid_row: board.is_valid_row (r - 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 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 - 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.out ~ bta.Templates.easy_board.out

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

    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: Result = not (across
                        1 |..| board.number_of_rows as i
                  some
                        across
                              1 |..| board.number_of_columns as j
                        some
                              can_move (i.item, j.item)
                        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 =
                        (board.number_of_occupied_slots = 1)
            winning_a_game_means_game_over: Result implies is_over
end -- class GAME
```

Section 2: Architectural Diagram



Section 3: Tests

Normal:

```
add_boolean_case (agent matches_slots_except_good)
...
matches_slots_except_good: BOOLEAN
 local
       board, b2: BOARD
 do
       comment ("test: matches_slots_except")
       create board.make_default
       b2:= board
       board.set_status (1, 1, board.unoccupied_slot)
       Result := board.matches_slots_except (b2, 1, 1, 1, 1)
       check Result end
 end
```

This case tests whether or not board's 'board' and 'b2' match at all slots except for the single slot that I change in this test, which is (1,1).

Precondition:

```
add_violation_case_with_tag("valid_row_range", agent test_bad_match_pre)
...
test_bad_match_pre
 local
       board: BOARD
       test: BOOLEAN
 do
       comment ("test: matches_slots_except pre")
       create board.make_default
       test:=    board.matches_slots_except (board, 3, 1, 1, 2)
 end
```

This case calls a normal board and proceeds to try and retrieve a value for matches_slots_except, the problem is that r1(3) is greater than r2 (1), which should violate the precondition: "valid_row_range".

Postcondition:

```
add_violation_case_with_tag("correct_result",agent test_bad_match_post)
...
test_bad_match_post
 local
       board: BOARD
       b2: BAD_BOARD
       test: BOOLEAN
 do
       comment ("test: matches_slots_except post")
       create board.make_arrow
       create b2.make
       test:=    b2.matches_slots_except (board, 1, 1, 1, 1)
 end
...
class
    BAD_BOARD
inherit
    BOARD
       redefine
             matches_slots_except
       end
create
    make
Feature -- Constructor
    make
    do
       create imp.make_filled (ssa.unavailable_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
```

This case calls a new class called BAD_BOARD, which is a child of BOARD. This bad board redefines matches_slots_except to reverse the result of board's matches_slots_except, but after it is reversed in the child, it calls the parent post condition and results in the inequality TRUE implies FALSE. This violates the post condition.

END