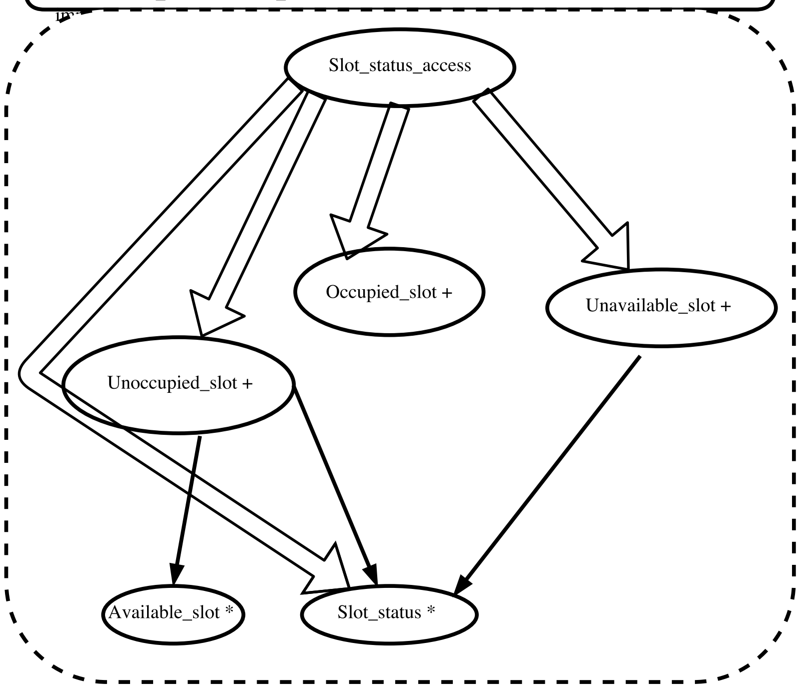
Board

```
feature -- Constructor
make_default
  ensure board_set: Current ~ bta.templates.default_board
make_easy
  ensure board_set: Current ~ bta.templates.easy_board
make_cross
  ensure board_set: Current ~ bta.templates.cross_board
make_plus
  ensure board_set: Current ~ bta.templates.plus_board
make_pyramid
  ensure board_set: Current ~ bta.templates.pyramid_board
make_arrow
  ensure board_set: Current ~ bta.templates.arrow_board
make_diamond
  ensure board_set: Current ~ bta.templates.diamond_board
make_skull
  ensure board_set: Current ~ bta.templates.skull_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(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: r1 ≤ r2
    valid_column_range: c1 ≤ c2
  ensure
    slots_in_range_set: ∀ r1 ≤ row ≤ r2 : ∀ c1 ≤ column ≤ c2 : (row.item ≥ r1 and
row.item ≤  r2 and column.item ≥ c2 and column.item ≤  c2 ⇒ status_of(row.item,
column.item) ~ status)
    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) : BOOLEAN
  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 ~ ∀ 1 ≤  m ≤ 7 : ∀ 1 ≤  n ≤ 7 : (m.item < r1 and m.item > r2) or
    (n.item < c1 and n.item > c2) ⇒ other.status_of (m.item, n.item).is_equal (current.status_of
        (m.item, n.item))
unavailable_slot: UNAVAILABLE_SLOT
  ensure Result = ssa.unavailable_slot
    occupied_slot: OCCUPIED_SLOT
  ensure Result = ssa.occupied_slot
    unoccupied_slot: UNOCCUPIED_SLOT
  ensure Result = ssa.unoccupied_slot
feature -- Queries
number_of_rows: INTEGER
  ensure correct_result: Result = imp.height
number_of_columns: INTEGER
    result :=imp.width
  ensure correct_result: result = (imp.width)
is_valid_row(r: INTEGER): BOOLEAN ensure
    correct_result: result = (r > 0 and r ≤ number_of_rows)
is_valid_column(c: INTEGER): BOOLEAN ensure
    correct_result: result = (c > 0 and c ≤ number_of_columns)
status_of(r, c: INTEGER): SLOT_STATUS
  require
    valid_row: is_valid_row (r)
    valid_column: is_valid_column (c)
  ensure correct_result: Result = imp.item (r, c)
number_of_occupied_slots: INTEGER
feature -- Equality
is_equal(other: like Current): BOOLEAN
  ensure then correct_result: result = (current.out ~ other.out)
feature -- Output
out: STRING
  local s : STRING
feature {NONE} -- Implementation
   ssa:SLOT_STATUS_ACCESS
   bta: BOARD_TEMPLATES_ACCESS
```



Game

```
feature -- Constructors
    make_from_board (new_board: BOARD)
  ensure
    board_set: board.out ~ new_board.out
make_easy
  ensure
    board_set: board ~ bta.templates.easy_board
make_cross
  ensure
    board_set: board.out ~ bta.templates.cross_board.out
make_plus
  ensure
    board_set: board ~ bta.templates.plus_board
make_pyramid
  ensure
    board_set: board.out ~ bta.templates.pyramid_board.out
make_arrow
  ensure
    board_set: board.out ~ bta.templates.arrow_board.out
make_diamond
  ensure
    board_set: board.out ~ bta.templates.diamond_board.out
make_skull
  ensure
    board_set: board.out ~ bta.templates.skull_board.out
feature -- Commands
move_left (r, c: INTEGER)
  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).is_equal (board.occupied_slot)
    middle_slot_occupied: board.status_of (r, c-1).is_equal (board.occupied_slot)
    to_slot_unoccupied: board.status_of (r, c-2).is_equal (board.unoccupied_slot)
    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 )
move_right (r, c: INTEGER)
  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).is_equal (board.occupied_slot)
    middle_slot_occupied: board.status_of (r, c + 1).is_equal (board.occupied_slot)
    to_slot_unoccupied:  board.status_of (r, c + 2).is_equal (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)
  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).is_equal (board.occupied_slot)
    middle_slot_occupied: board.status_of (r - 1, c).is_equal (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-2, r, c, c)
move_down (r, c: INTEGER)
  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+1, c, c)
feature -- Status Queries
is_over: BOOLEAN
  ensure
    correct_result: Result = not ∃ 1 ≤  arrow ≤ board.number_of_rows : ∃ 1 ≤  columns ≤
board.number_of_columns                      moving_peg (rows.item,columns.item)
is_won: BOOLEAN
  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
feature -- Output
out: STRING
feature -- checking if the peg can move left/right/up/down
moving_peg (r,c: INTEGER): BOOLEAN
feature -- Auxiliary Routines
boolean_to_yes_no (b: BOOLEAN): STRING
feature -- Board
    bta: BOARD_TEMPLATES_ACCESS
    board: BOARD
```