```systemverilog
// Handles user's config of the initial state before GoL begins to run

module userInput (clk, reset, KEY, SW0, start_game, row_select, col_select, set_initial, 
cell_state, GrnPixels);
    input logic clk, reset;
    input logic [3:0] KEY;
    input logic SW0, start_game;
    output logic [7:0] row_select, col_select;
    output logic set_initial, cell_state;
    output logic [15:0][15:0] GrnPixels;

    logic [7:0] row, col;
    logic [7:0] saved_row, saved_col; // Save cursor position
    logic [3:0] ps, ns; // present state, next state for each key
    logic blink_state;
    logic [31:0] blink_counter;
    logic key_press_detected;
    parameter BLINK_INTERVAL = 250000; // Adjust for desired blink speed

    // State encoding
    enum {off, press} state;

    // Next state logic for each key
    always_comb begin
        ns[3] = (KEY[3] ? press : off);
        ns[2] = (KEY[2] ? press : off);
        ns[1] = (KEY[1] ? press : off);
        ns[0] = (KEY[0] ? press : off);
    end

    // State registers and other logic
    always_ff @(posedge clk or posedge reset) begin
        if (reset) begin
            ps <= 4'b0000;
            row <= 8'd8;  // Starting in the middle
            col <= 8'd8;
            saved_row <= 8'd8;
            saved_col <= 8'd8;
            cell_state <= 1'b0;
            blink_counter <= 0;
            blink_state <= 0;
            key_press_detected <= 0;
            set_initial <= 0;
        end else begin
            ps <= ns;
            key_press_detected <= 0;

            // Detect key press
            if ((ps[3] == off && ns[3] == press) || (ps[2] == off && ns[2] == press) || (ps[
1] == off && ns[1] == press) || (ps[0] == off && ns[0] == press)) begin
                key_press_detected <= 1;
            end
            if (key_press_detected) begin
                set_initial <= 1;
            end else begin
                set_initial <= 0;
            end

            // Cursor Movement
            if (!start_game) begin
                if (ps[1] == off && ns[1] == press && row < 8'd15) row <= row + 8'd1;  // 
Move down
                if (ps[2] == off && ns[2] == press && row > 8'd0) row <= row - 8'd1;  // 
Move up
                if (ps[3] == off && ns[3] == press && col < 8'd15) col <= col + 8'd1; // 
Move right
                if (ps[0] == off && ns[0] == press && col > 8'd0) col <= col - 8'd1;  // 
Move left
                saved_row <= row;
                saved_col <= col;
            end else begin
                row <= saved_row;
                col <= saved_col;
                end

                // Cell State Toggle
                if (SW0 == 1) cell_state <= 1'b1; // Set cell on
                else cell_state <= 1'b0;          // Set cell off

                // Blinking Cursor Logic
                if (!start_game) begin
                    if (blink_counter == BLINK_INTERVAL) begin
                        blink_counter <= 0;
                        blink_state <= ~blink_state;
                    end else begin
                        blink_counter <= blink_counter + 1;
                    end
                end else begin
                    blink_state <= 0; // Turn off blinking when the game is running
                end
            end
        end

        // Set the current cursor position to blink green
        always_comb begin
            GrnPixels = '{default: 0};
            if (blink_state) begin
                GrnPixels[row][col] = 1'b1;
            end
        end

        assign row_select = row;
        assign col_select = col;
endmodule







module userInput_testbench();
   logic clk, reset;
   logic [3:0] KEY;
   logic SW0;
   logic [7:0] row_select;
   logic [7:0] col_select;
   logic set_initial;
   logic cell_state;
   logic [15:0][15:0] GrnPixels;

   userInput dut (clk, reset, KEY, SW0, row_select, col_select, set_initial, cell_state,
GrnPixels);

   // Set up a simulated clock.
   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk; // Forever toggle the clock
   end

   // Test the design.
   initial begin
      reset <= 1; @(posedge clk); // reset every time we start
      @(posedge clk);
      reset <= 0; @(posedge clk);
      @(posedge clk);

      // Move the cursor and set cells
      KEY <= 4'b1110;        // Move right
      SW0 <= 1;              // Set cell
      @(posedge clk);        @(posedge clk);
      KEY <= 4'b1101;        // Move down
      @(posedge clk);        @(posedge clk);
      SW0 <= 0;              // Clear cell
```

```
        @(posedge clk);       @(posedge clk);
        KEY <= 4'b1111;       // No movement
        @(posedge clk);       @(posedge clk);

        // Additional test cases
        KEY <= 4'b1110;       // Move right
        @(posedge clk);       @(posedge clk);
        KEY <= 4'b1111;       // No movement
        @(posedge clk);       @(posedge clk);
        KEY <= 4'b1101;       // Move down
        SW0 <= 1;             // Set cell
        @(posedge clk);       @(posedge clk);
        KEY <= 4'b1111;       // No movement
        @(posedge clk);       @(posedge clk);
        KEY <= 4'b1110;       // Move right
        @(posedge clk);       @(posedge clk);
        KEY <= 4'b1111;       // No movement
        SW0 <= 0;             // Clear cell
        @(posedge clk);       @(posedge clk);
        KEY <= 4'b1101;       // Move down
        @(posedge clk);       @(posedge clk);
        KEY <= 4'b1111;       // No movement
        @(posedge clk);       @(posedge clk);
        KEY <= 4'b1011;       // Move left
        @(posedge clk);       @(posedge clk);
        KEY <= 4'b1111;       // No movement
        SW0 <= 1;             // Set cell
        @(posedge clk);       @(posedge clk);
        KEY <= 4'b0111;       // Move up
        @(posedge clk);       @(posedge clk);
        KEY <= 4'b1111;       // No movement
        @(posedge clk);       @(posedge clk);

        $stop;
    end
endmodule

```