```
module update_cars (clk, reset, car2, car5, car7, car9, car11, car12);
   input logic clk, reset;
   output logic [15:0] car2, car5, car7, car9, car11, car12;

   /*

      need to figure out how many cars we want to use / how to use them
      thinking a 2D array

      DEFAULT CAR LAYOUT
      _________________________________
   F  |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|
   E  |
   D  |
   C  |X X X     X X X X X X X     X
   B  |X X       X X X X X X       X X
   A  |
   9  |X X X X X           X X X   X X
   8  |
   7  |  X X X X     X X X X   X     X
   6  |
   5  |X   X X X X     X X X X X
   4  |
   3  |
   2  |X X X         X x     X X X X X
   1  |
   0  |
       0 1 2 3 4 5 6 7 8 9 A B C D E F

       There are 6 lines with cars
       Plan is to have a 6 x 16 array of 1s ad 0s
       Rows 2, 7, 11 move LEFT
       Rows 5, 9, 12 move RIGHT

   */

   integer count;

   always_ff @(posedge clk) begin

      integer i;
      integer j;

      if (reset) begin
         count <= 0;
         car2  <= 16'b1110000110011111;
         car5  <= 16'b1011110011111000;
         car7  <= 16'b0111100111101001;
         car9  <= 16'b1111100000111011;
         car11 <= 16'b1100011111100011;
         car12 <= 16'b1110011111110000;
         end
      else begin
// need a counter in HERE, not draw_game
         if (count == 3000) begin

            count <= 0;

            // update the cars
            // rows moving LEFT: 2, 7, 11
            for(i = 15; i > 0; i--) begin
               car2[i]  <= car2[i - 1];
               car7[i]  <= car7[i - 1];
               car11[i] <= car11[i - 1];
            end

            car2[0]  <=  car2[15];
            car7[0]  <=  car7[15];
            car11[0] <=  car11[15];


            // rows moving RIGHT: 5, 9, 12
            for (j = 0; j < 15; j++) begin
               car5[j]  <= car5[j + 1];
               car9[j]  <= car9[j + 1];
               car12[j] <= car12[j + 1];
```

```
77                end
78
79                car5[15]  <= car5[0];
80                car9[15]  <= car9[0];
81                car12[15] <= car12[0];
82             end
83             else begin
84                count <= count + 1;
85                car2 <= car2;
86                car5 <= car5;
87                car7 <= car7;
88                car9 <= car9;
89                car11 <= car11;
90                car12 <= car12;
91             end
92          end
93       end
94    endmodule
95
96    module update_cars_testbench();
97       logic clk, reset;
98       logic [15:0] car2, car5, car7, car9, car11, car12;
99
100      update_cars dut(clk, reset, car2, car5, car7, car9, car11, car12);
101
102      // Set up the clock.
103      parameter CLOCK_PERIOD=100;
104      initial begin
105      clk <= 0;
106      forever #(CLOCK_PERIOD/2) clk <= ~clk;
107      end
108
109      // Set up the inputs to the design. Each line is a clock cycle.
110      initial begin
111                                       @(posedge clk);
112         reset <= 1;                   @(posedge clk);
113         reset <= 0;                   @(posedge clk);
114                                       @(posedge clk);
115                                       @(posedge clk);
116         // just let it run and generate car positions
117                                       @(posedge clk);
118                                       @(posedge clk);
119                                       @(posedge clk);
120                                       @(posedge clk);
121                                       @(posedge clk);
122                                       @(posedge clk);
123                                       @(posedge clk);
124                                       @(posedge clk);
125                                       @(posedge clk);
126                                       @(posedge clk);
127                                       @(posedge clk);
128                                       @(posedge clk);
129                                       @(posedge clk);
130                                       @(posedge clk);
131                                       @(posedge clk);
132                                       @(posedge clk);
133                                       @(posedge clk);
134                                       @(posedge clk);
135                                       @(posedge clk);
136                                       @(posedge clk);
137                                       @(posedge clk);
138
139         $stop; // End the simulation.
140      end
141
142   endmodule
143
```