## COL215 – Digital Logic and System Design

Department of Computer Science & Engineering, IIT Delhi Semester I, 2025–26

# Lab Assignment 8 part III (Friday)

Developed By:
Bharmal Bhakar (2024CS10403)
Ritik Raj (2024CS10086)

## 1 Introduction

Part III requires generating a rival car at a random horizontal position at the top of the road and moving it vertically downward, as well as detecting collisions.

## 2. Design Decisions

The implementation involved three main components:

1. **Display_sprite_with_rival:** Handles background scrolling and pixel-level collision detection.
2. **lfsr8:** Generates pseudo-random numbers for the rival car's horizontal spawn position.
3. **rival_controller:** Manages the rival car's random spawning and vertical movement.
4. **Car_control_FSM:** Augments the FSM to transition to the COLLIDE state upon rival car collision.

### 2.1 Background Scrolling Logic

The appearance of vertical movement (scrolling) is achieved by cyclically shifting the starting address for reading data from the background ROM (bg_rom) every frame.

- Vertical Offset: The current scroll position is stored in bg_y_offset_reg.
- Update Condition: The offset is decremented by SCROLL_STEP (1 pixel) at the end of every full VGA frame (frame_end_clk), but only when the FSM is in the running state (running_fsm).
- Wrap-around: When the offset reaches the bottom boundary (bg_y_offset_reg <= BG_RESET_Y), it is reset to BG_RESET_Y + BG_HEIGHT - SCROLL_STEP to create a continuous loop of the road graphics.
- ROM Addressing: The final Y-address for the ROM read is calculated as the current screen Y-coordinate plus the offset, modulo the background height (BG_HEIGHT = 240).
  Final_Y = (Screen_Y_relative + bg_y_offset_reg) (mod BG_HEIGHT)

### 2.2. Pseudo-Random Number Generation (PRNG)

A Linear Feedback Shift Register (**LFSR**) was implemented to generate the random horizontal position for the rival car.

**LFSR Implementation (`lfsr8.v`) :**

**Structure:** An 8-bit LFSR (lfsr8) is used, with the output of the exclusive-OR (XOR) gates fed back into the shift register.

**Feedback Taps**: The new bit (newbit) is generated using taps at bits 7, 5, 4, and 3.

$newbit = q[7] \oplus q[5] \oplus q[4] \oplus q[3]$

**Reset:** On reset (btnC), the LFSR is synchronously initialized with the SEED parameter.

**SEED Selection :**

The SEED is determined by the bitwise XOR of the last 4 digits of the two Kerberos IDs

| Kerberos ID | Last 4 digit(Decimal) | Hexadecimal | Binary |
|---|---|---|---|
| 2024CS10403 | 0403 | 193 | 110010011 |
| 2024CS10086 | 0086 | 56 | 001010110 |

**SEED =** $11001001 \oplus 001010110$ **=** $111000101$ = 453(in decimal) = 1C5(in hexadecimal)

The calculated seed value exceeds 8bit, that's why to maintain it in under 8bit we took 8'hC5 as our seed value .

## 2.3. Rival Car Control (`rival_controller.v`)

The 8-bit random output (lfsr_q_out) is scaled to fit the valid road width relative to the background start (BG_OFFSET_X = 200).

- **Road Boundaries (Relative):** RIVAL_X_MIN_REL = 44 and RIVAL_X_MAX_REL = 104.
- Random Logic: The upper 7 bits of the frozen LFSR output (rand7) are used to calculate an offset within the range RIVAL_RANGE = 61 pixels.

  scaled = $(\frac{rand7}{128} \times \text{RIVAL\_RANGE}\ ) \sim \text{mult}[13:7]$ (where mult = rand7 × RIVAL_RANGE)

  rival_x = BG_OFFSET_X + RIVAL_X_MIN_REL + scaled

The rival car moves vertically every several frames to ensure a moderate speed.

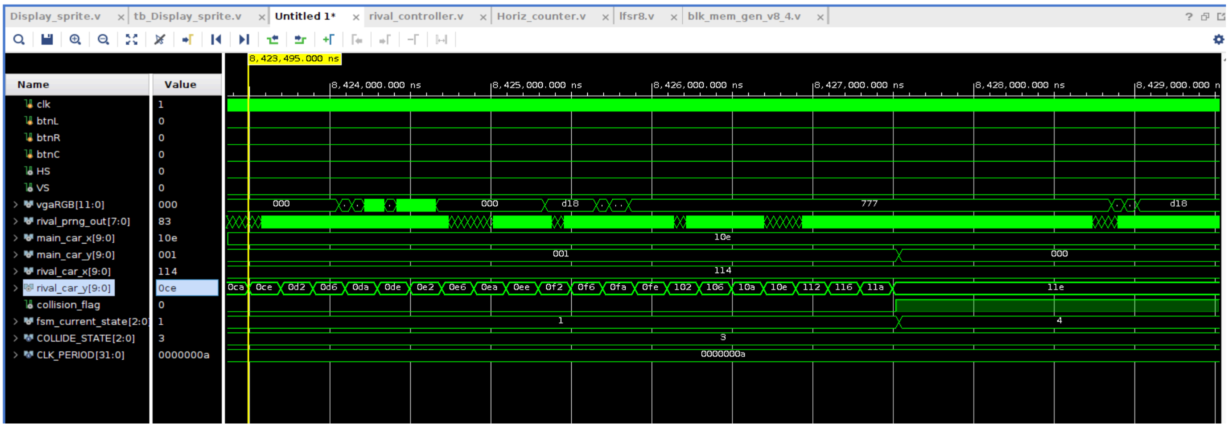
- **Frame Counter:** A frame_reg_count tracks the number of frames passed.
- **Movement Step:** When the counter reaches FRAME_THRESHOLD - 1 (e.g., 4 if FRAME_THRESHOLD=5), the counter is reset, and rival_y is incremented by DELTA_Y (e.g., 3 pixels).
- **Respawn:** When the car reaches the bottom threshold (rival_y > BOTTOM_Y_THRESH), the car's coordinates are updated to respawn it at the top (BG_OFFSET_Y) using a newly calculated random X position.

### 2.4. Car-to-Car Collision Detection

A dedicated **rect_collision** module is used to detect overlap between the two 14x16 pixel rectangular hit boxes (sprites).

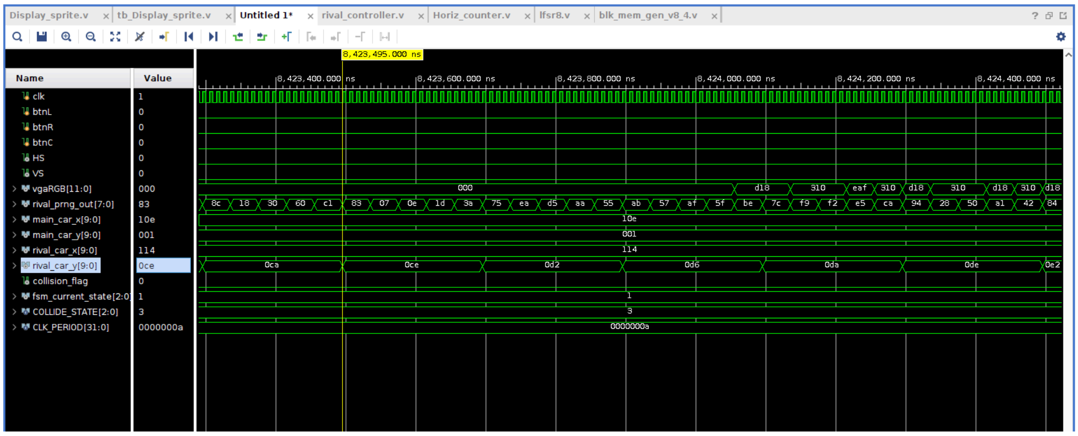
- **Logic:** The module outputs collide high if the two rectangles overlap on the screen.
- **FSM Integration:** The rival_collision_pixel output from the collision module is synchronized and fed to the main **Car_control_FSM** as rival_collision.
- **State Transition:** When rival_collision is high, the FSM forces an asynchronous transition to the **COLLIDE** state, regardless of the current state or button inputs. In the COLLIDE state, the running output is set to 0 to stop all movement and scrolling. The game remains halted until btnC is pressed.

## 3. Simulation Snapshots



| Name | Value |
|---|---|
| clk | 1 |
| btnL | 0 |
| btnR | 0 |
| btnC | 0 |
| HS | 0 |
| VS | 0 |
| vgaRGB[11:0] | 000 |
| rival_prng_out[7:0] | 83 |
| main_car_x[9:0] | 10e |
| main_car_y[9:0] | 001 |
| rival_car_x[9:0] | 114 |
| rival_car_y[9:0] | 0ce |
| collision_flag | 0 |
| fsm_current_state[2:0] | 1 |
| COLLIDE_STATE[2:0] | 3 |
| CLK_PERIOD[31:0] | 0000000a |

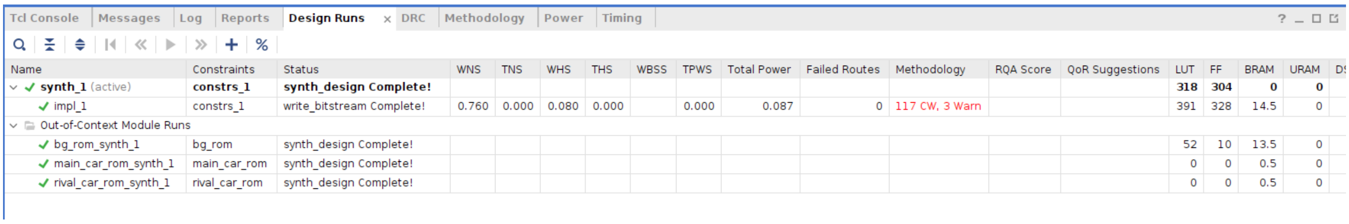
This simulation captures the rival car's movement (rival_car_y increasing vertically) and the game-over condition. When the rival car collides with the main car, the collision_flag asserts high, immediately forcing the FSM into the COLLIDE state (3'd4). In this state, the game halts, freezing the rival_car_y coordinate and disabling movement until btnC is pressed to restart



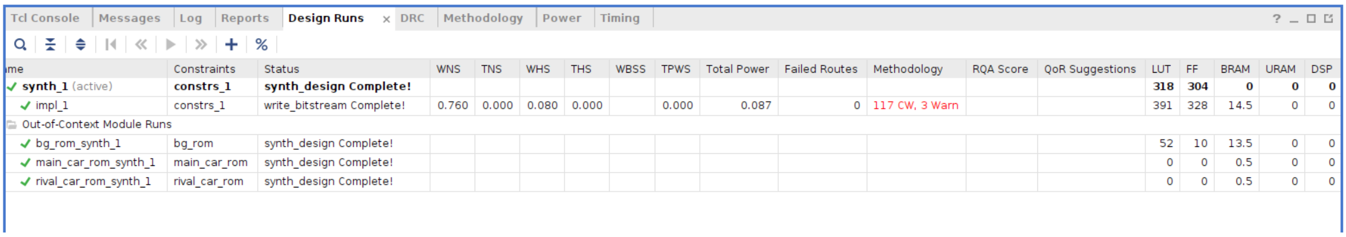
| Name | Value |
|---|---|
| clk | 1 |
| btnL | 0 |
| btnR | 0 |
| btnC | 0 |
| HS | 0 |
| VS | 0 |
| vgaRGB[11:0] | 000 |
| rival_prng_out[7:0] | 83 |
| main_car_x[9:0] | 10e |
| main_car_y[9:0] | 001 |
| rival_car_x[9:0] | 114 |
| rival_car_y[9:0] | 0ce |
| collision_flag | 0 |
| fsm_current_state[2:0] | 1 |
| COLLIDE_STATE[2:0] | 3 |
| CLK_PERIOD[31:0] | 0000000a |

This simulation verifies the **LFSR operation**, showing its output (rival_prng_out) continuously cycling through random values. This output is correctly scaled to set the rival car's fixed **random horizontal spawn position** (rival_car_x) upon startup, while its vertical position (rival_car_y) remains at the top boundary.

# 4. Synthesis Report



| Name | Constraints | Status | WNS | TNS | WHS | THS | WBSS | TPWS | Total Power | Failed Routes | Methodology | RQA Score | QoR Suggestions | LUT | FF | BRAM | URAM | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ✓ **synth_1** (active) | **constrs_1** | **synth_design Complete!** | | | | | | | | | | | | **318** | **304** | **0** | **0** | |
| ✓ impl_1 | constrs_1 | write_bitstream Complete! | 0.760 | 0.000 | 0.080 | 0.000 | | 0.000 | 0.087 | 0 | 117 CW, 3 Warn | | | 391 | 328 | 14.5 | 0 | |
| Out-of-Context Module Runs | | | | | | | | | | | | | | | | | | |
| ✓ bg_rom_synth_1 | bg_rom | synth_design Complete! | | | | | | | | | | | | 52 | 10 | 13.5 | 0 | |
| ✓ main_car_rom_synth_1 | main_car_rom | synth_design Complete! | | | | | | | | | | | | 0 | 0 | 0.5 | 0 | |
| ✓ rival_car_rom_synth_1 | rival_car_rom | synth_design Complete! | | | | | | | | | | | | 0 | 0 | 0.5 | 0 | |

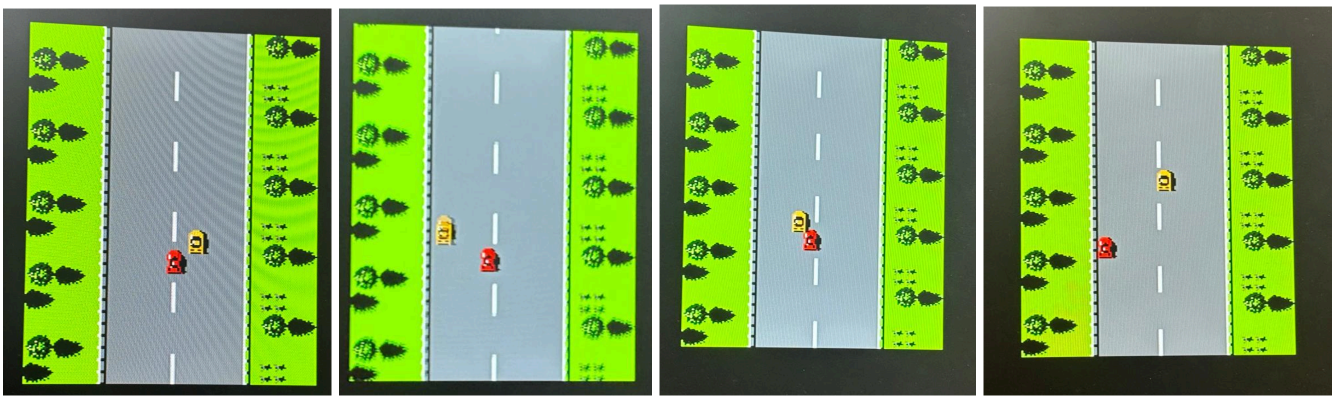


| me | Constraints | Status | WNS | TNS | WHS | THS | WBSS | TPWS | Total Power | Failed Routes | Methodology | RQA Score | QoR Suggestions | LUT | FF | BRAM | URAM | DSP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ✓ **synth_1** (active) | **constrs_1** | **synth_design Complete!** | | | | | | | | | | | | **318** | **304** | **0** | **0** | **0** |
| ✓ impl_1 | constrs_1 | write_bitstream Complete! | 0.760 | 0.000 | 0.080 | 0.000 | | 0.000 | 0.087 | 0 | 117 CW, 3 Warn | | | 391 | 328 | 14.5 | 0 | 0 |
| Out-of-Context Module Runs | | | | | | | | | | | | | | | | | | |
| ✓ bg_rom_synth_1 | bg_rom | synth_design Complete! | | | | | | | | | | | | 52 | 10 | 13.5 | 0 | 0 |
| ✓ main_car_rom_synth_1 | main_car_rom | synth_design Complete! | | | | | | | | | | | | 0 | 0 | 0.5 | 0 | 0 |
| ✓ rival_car_rom_synth_1 | rival_car_rom | synth_design Complete! | | | | | | | | | | | | 0 | 0 | 0.5 | 0 | 0 |

The final implementation (impl_1) consumes **391 LUTs** and **328 Flip-Flops (FFs)**. The design utilizes **14.5 BRAMs** (Block RAMs) in total, primarily for storing the background and both car sprites. The increase in logic compared to Part II is due to the addition of the LFSR, rival car controller, and scrolling logic

# 5. On-Board Implementation

This implementation successfully displays the main car, rival car and the road background on the VGA. The following snippets show the random generation of position of rival car, collisions between rival car and main car and collision with boundary of roads.

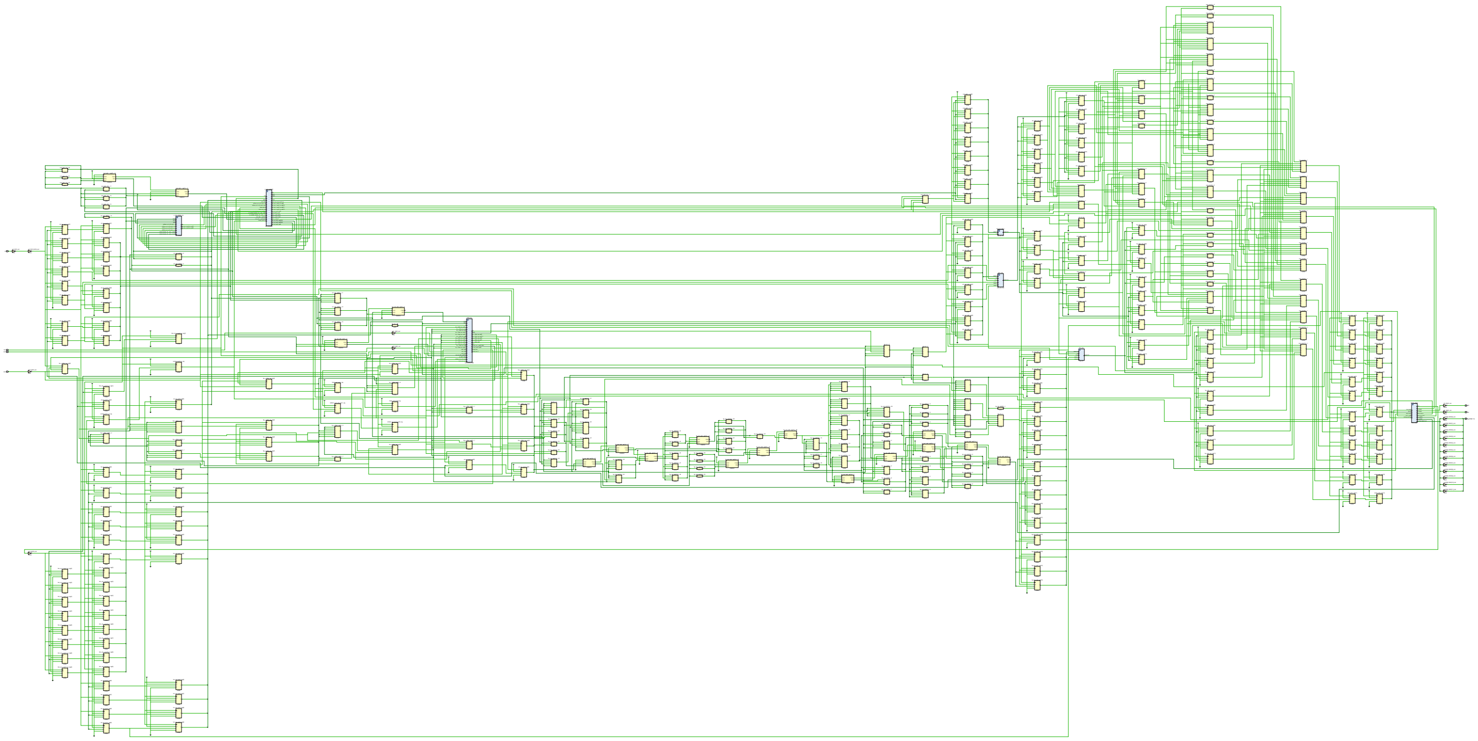


# 6. Conclusion

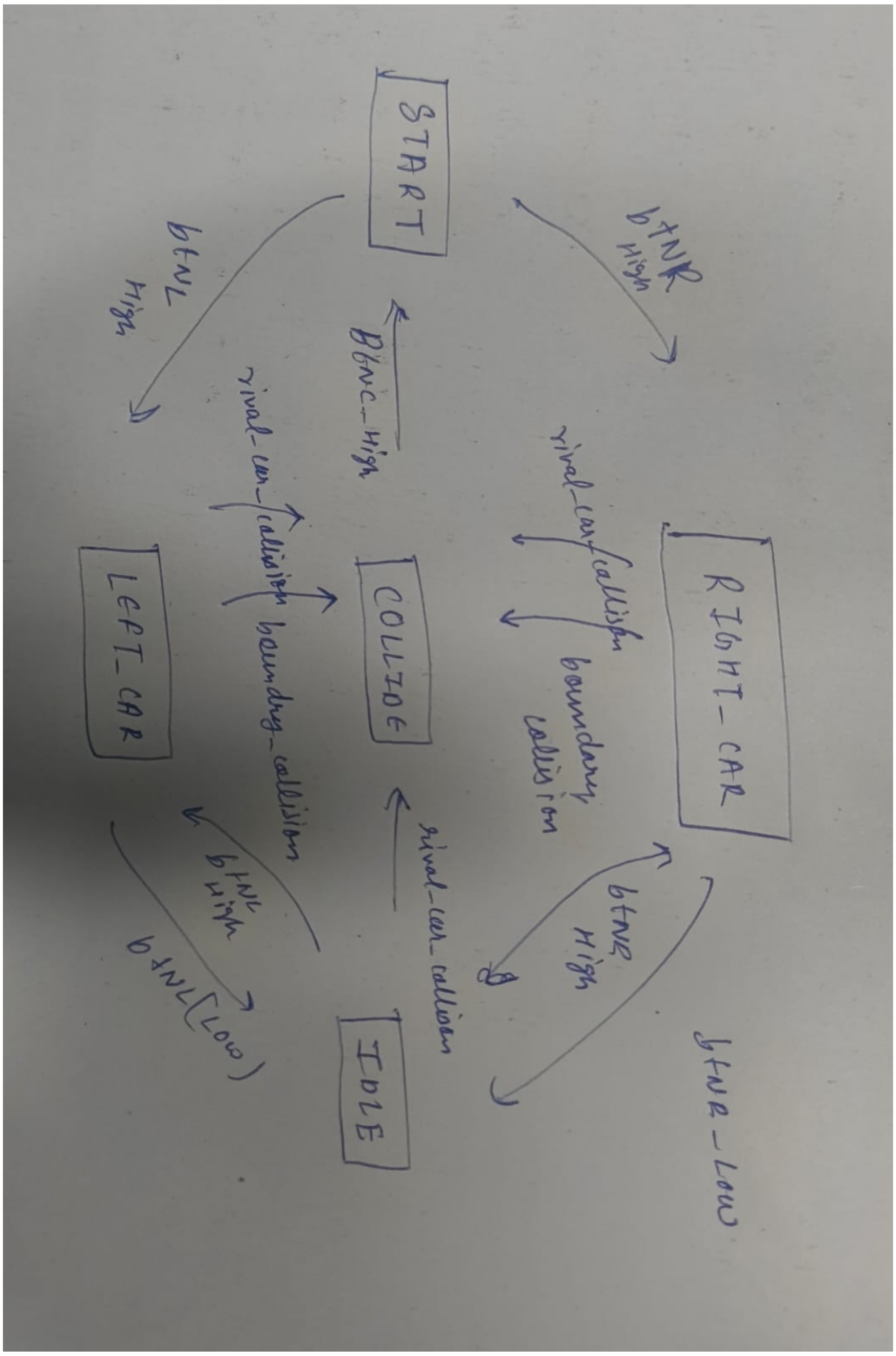
Part III successfully implemented dynamic gameplay via continuous background scrolling and a rival car spawned using an 8-bit LFSR. Crucial car-to-car collision detection was integrated, which forces the FSM into the COLLIDE state (3'd4), halting all movement until the game is reset via btnC.

# 7. Generated Schematic

The Generated Schematic for part3 of this project is attached below



## 8. FSM STATE DIAGRAMS


START
btnR
high
btnL
high
BtnC_High
rival_car_collision
boundary
collision
RIGHT_CAR
COLLIDE
LEFT_CAR
rival-car-collision
boundry_collision
btnR
High
btnL
high
rival-car collision
IDLE
btnL (Low)
btnR_Low