Car Avoid Obstacle Game – Build on AK Embedded Base Kit

I. Giới thiệu

Car Avoid Obstacle Game là một trò chơi nhúng chạy trên AK Embedded Base Kit – STM32L151,sử dụng cấu trúc Event-Driven (Task–Signal–Message).Trò chơi được thiết kế nhằm giúp sinh viên rèn luyện kỹ năng lập trình hệ thống event-driven, quản lý đa đối tượng, và xây dựng hoạt ảnh hiển thị trong môi trường nhúng thực tế.

1.1 Phần cứng

Hình 1.1 AK Base Kit

AK Embedded Base Kit là một evaluation kit dành cho các bạn học software embedded nâng cao.

Các phần tử tích hợp trong kit

-Vi điều khiển chính STM32 ARM-based 32-bit MCU: STM32L151C8T6

-Truyền nhận không dây 2.4Ghz RF Transceiver ICs: NRF24L01P-R

-Giao tiếp có dây trong công nghiệp RS485 50Mbps: THVD1450DR

-Tích hợp loa dùng để làm game Buzzers Magnetic: MLT-8530

-Tích hợp NOR FLASH (32MB): W25Q256JVEIQTR

-Kết nối với các mạch ứng dụng chuẩn Seeedstudio Grove Header HY-4AW

-Tích hợp console qua USB type C sử dụng chip USB UART Converter: CH340E

1.2 Màn hình Menu

Vì giữ lại game Ạcherry nên trò chơi bắt đầu bằng màn hình Menu game với các lựa chọn sau:

Hình 1.2 Menu game

Archery Game: Chọn vào để bắt đầu chơi game.

Car Game: Chọn đề bắt đầu chơi game.

Setting: Chọn vào để cài đặt các thông số của game.

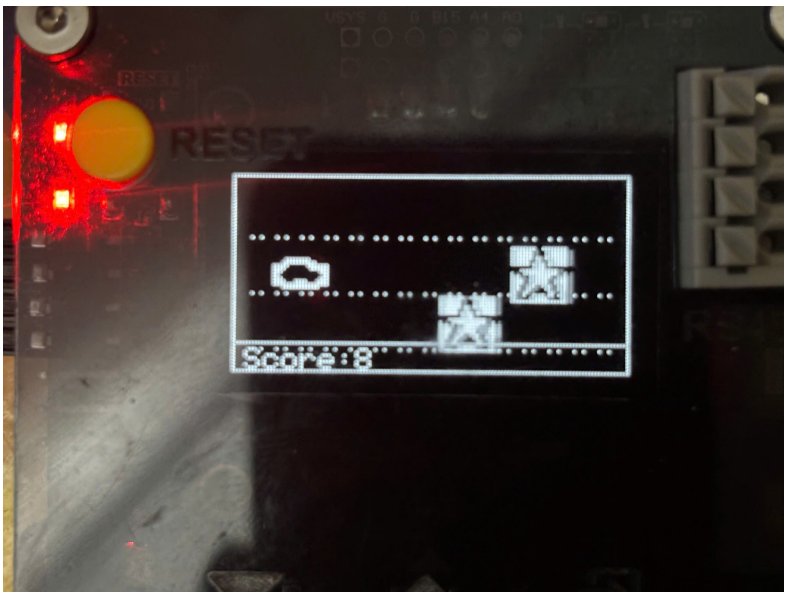
Charts: Chọn vào để xem top 3 điểm cao nhất đạt được.

Car Charts: Chọn để xem top 3 điểm cao nhất đạt được

Exit: Thoát menu vào màn hình chờ

1.3 Các đối tượng hoạt động trong game Car Avoid Obstacle

Đối tượng	Mô tả
Xe (Car)	Đối tượng chính, di chuyển giữa các lane.
Vật cản (Obstacle)	Xuất hiện ngẫu nhiên, đứng yên, xe phải tránh.



Hình 1.3 Giao diện game

1.4 Cách chơi game

-Xe di chuyển tự động theo trục X (từ trái sang phải).

-Người chơi dùng nút [UP] để đổi lên làn trên, nút [DOWN] để đổi xuống làn dưới.

-Có 3 làn đường (top, middle, bottom).

-Nhiệm vụ là tránh các vật cản (Obstacle) cố định hoặc thay đổi lane ngẫu nhiên.

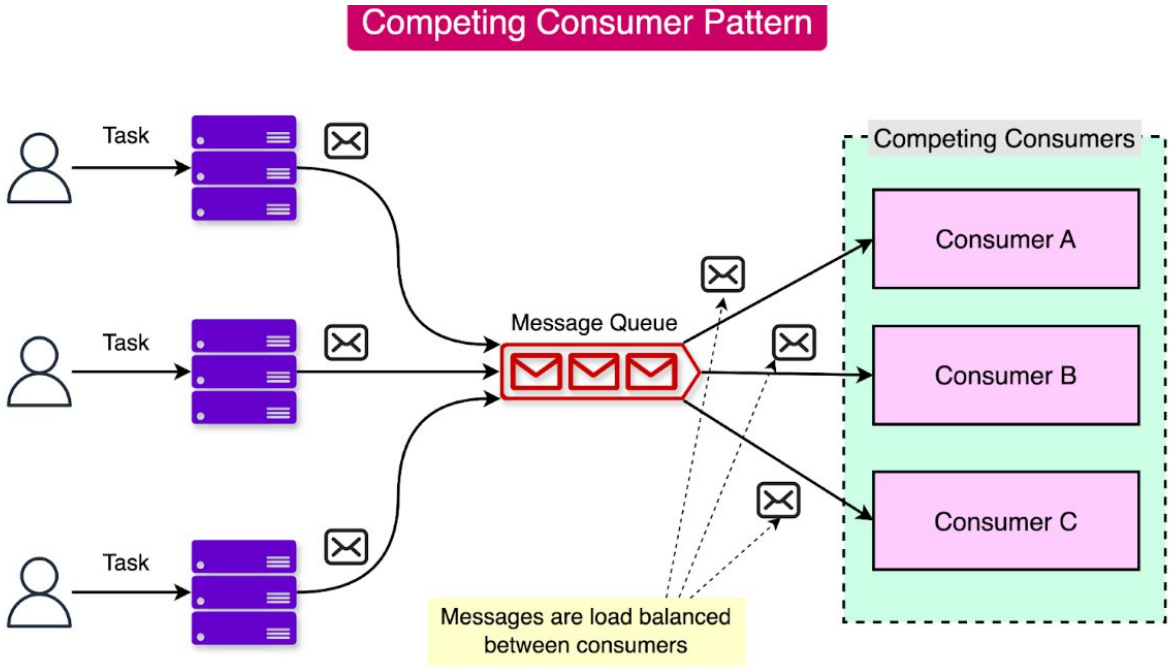
-Nếu va chạm, xe phát nổ và hiện màn hình Game Over.

Cách tính điểm:

Cứ vượt qua 1 vật cản sẽ cộng 2 điểm và sẽ tăng tốc độ nếu đạt đủ số điểm yêu cầu

II Thiết kế

2.1 Sơ lược về Event-Driven



Hình 2.1 Event-Driven

https://dev.to/syed_mohsinalinaqvi_266/understanding-application-architectures-beyond-mvc-3heh

-Event-Driven: Là mô hình hoạt động dựa trên cơ chế truyền và xử lý thông điệp (message) để thực thi các công việc trong hệ thống. Khi một sự kiện (event) xảy ra, nó sẽ phát đi một Signal/Message và được một Task khác tiếp nhận để xử lý thông qua Handler.

-Task: Là đơn vị thực thi đảm nhiệm một nhóm chức năng cụ thể như quản lý state-machine, hiển thị giao diện, điều khiển timer hoặc watchdog.

-Message (Signal): Là gói thông tin biểu thị một sự kiện hoặc yêu cầu cần xử lý; có thể chỉ chứa tín hiệu hoặc kèm dữ liệu.

-Handler: Là hàm hoặc khối mã đảm nhiệm việc xử lý khi Task nhận được một Message tương ứng.

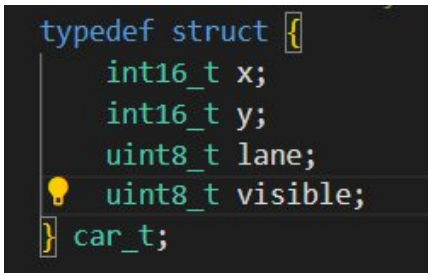
2.2 Thuộc tính đối tượng

Trong trò chơi Car Avoid Obstacle, việc xác định rõ các thuộc tính của từng đối tượng giúp cho việc quản lý, hiển thị và xử lý logic game trở nên dễ dàng, rõ ràng và nhất quán.

Mỗi đối tượng trong game được mô tả bằng struct (cấu trúc dữ liệu) chứa các thông tin đặc trưng như vị trí, trạng thái và khả năng hiển thị trên màn hình.

Đối tượng xe – car_t

Đây là đối tượng chính mà người chơi điều khiển. Xe chỉ di chuyển theo chiều dọc (lên hoặc xuống) giữa ba làn đường cố định, trong khi toạ độ X là cố định.



Giải thích:

-x, y: xác định vị trí hiển thị của xe trên màn hình.

-lane: xác định xe đang ở làn nào để xử lý chuyển làn và va chạm.

-visible: quy định trạng thái hiển thị của xe (ẩn khi game over hoặc reset).

Đối tượng vật cản – obstacle-t

```
typedef struct {
    int16_t x;          // Vị trí X (tọa độ top-left)
    int16_t y;          // Vị trí Y (tọa độ top-left)
    uint8_t lane;       // 0: top, 1: mid, 2: bottom
    uint8_t active;     // 1 = hiển thị, 0 = ẩn
    uint8_t type;
    bool counted;        //   cộng điểm
} cg_game_obstacle_t;
```

-active: cho biết obstacle có đang xuất hiện trên màn hình hay không.

-counted: giúp đảm bảo mỗi obstacle chỉ được cộng điểm một lần khi xe vượt qua.

-lane: xác định vị trí dọc của obstacle (theo 3 lane cố định).

-type: cho phép một số obstacle có khả năng di chuyển giữa các lane.

-x, y: vị trí hiển thị của obstacle trên màn hình**.**

Các biến quan trọng

- car_game_score :Lưu điểm số hiện tại của người chơi. Mỗi lần xe tránh được vật cản, điểm tăng thêm.

- car_game_state

Trạng thái hiện tại của trò chơi:

• GAME_OFF – chưa bắt đầu.

• GAME_PLAY – đang chơi.

• GAME_OVER – đã thua.

-car_x :Tọa độ X cố định của xe, được dùng để kiểm tra va chạm với obstacle.

-car_lane : Làn đường hiện tại của xe (0, 1, 2).

-obstacle_speed: Tốc độ di chuyển của vật cản, tăng dần theo điểm số để nâng độ khó.

2.3 Task và Signal trong Car Avoid Obstacle

2.3.1 Task

TASK ID	PRIORITY	HANDLER
CAR_GAME_SCREEN_ID	TASK_PRI_LEVEL_4	cg_game_car_handle
CAR_GAME_CAR_ID	TASK_PRI_LEVEL_4	scr_car_game_handle

Vai trò và tác dụng của Task

-Xử lý sự kiện: Mỗi Task được gắn với một nhóm sự kiện cụ thể. Khi sự kiện xảy ra, hệ thống gửi message đến Task tương ứng và Task sẽ thực thi các hành động cần thiết để phản hồi lại sự kiện đó.

-Đồng bộ hóa hoạt động: Task giúp đảm bảo việc xử lý sự kiện diễn ra theo trình tự rõ ràng, tránh xung đột dữ liệu. Hệ thống chỉ kích hoạt Task khi Task hiện tại đã hoàn tất quá trình xử lý của mình.

-Quản lý luồng điều khiển: Task cho phép tổ chức và kiểm soát luồng sự kiện trong ứng dụng. Thông qua các Task, ta có thể xác định rõ thứ tự và cách thức phản ứng của hệ thống trước các sự kiện khác nhau.

-Tách biệt logic xử lý: Việc chia chương trình thành các Task riêng biệt giúp tách biệt logic, khiến source code dễ đọc, dễ mở rộng và bảo trì.

-Phân cấp và ưu tiên (Task Level): Mỗi Task được gán một mức ưu tiên (priority level). Scheduler sẽ xử lý các Task có level cao hơn hoặc được gọi trước trước các Task khác.

Trong game Car Avoid Obstacle, tất cả các Task đều có mức ưu tiên level 4, nên Task nào được gửi message trước sẽ xử lý trước.

2.3.2 Signal

Message được chia làm 2 loại chính, Message chỉ chứa Signal và Message vừa chứa Signal và Data. Message tương đương với Signal

Task ID	Signal
CAR_GAME_SCREEN_ID	SCREEN_ENTRY CAR_GAME_TIME_TICK CAR_GAME_OBSTACLE_SETUP_SIG CAR_GAME_OBSTACLE_UPDATE_SIG CAR_GAME_OBSTACLE_DRAW_SIG CAR_GAME_CAR_COLLISION_SIG AC_DISPLAY_BUTTON_UP_RELEASED AC_DISPLAY_BUTTON_DOWN_RELEASED AC_DISPLAY_BUTTON_MODE_RELEASED CAR_GAME_RESET CAR_GAME_EXIT_GAME
CAR_GAME_CAR_ID	CAR_GAME_CAR_SETUP

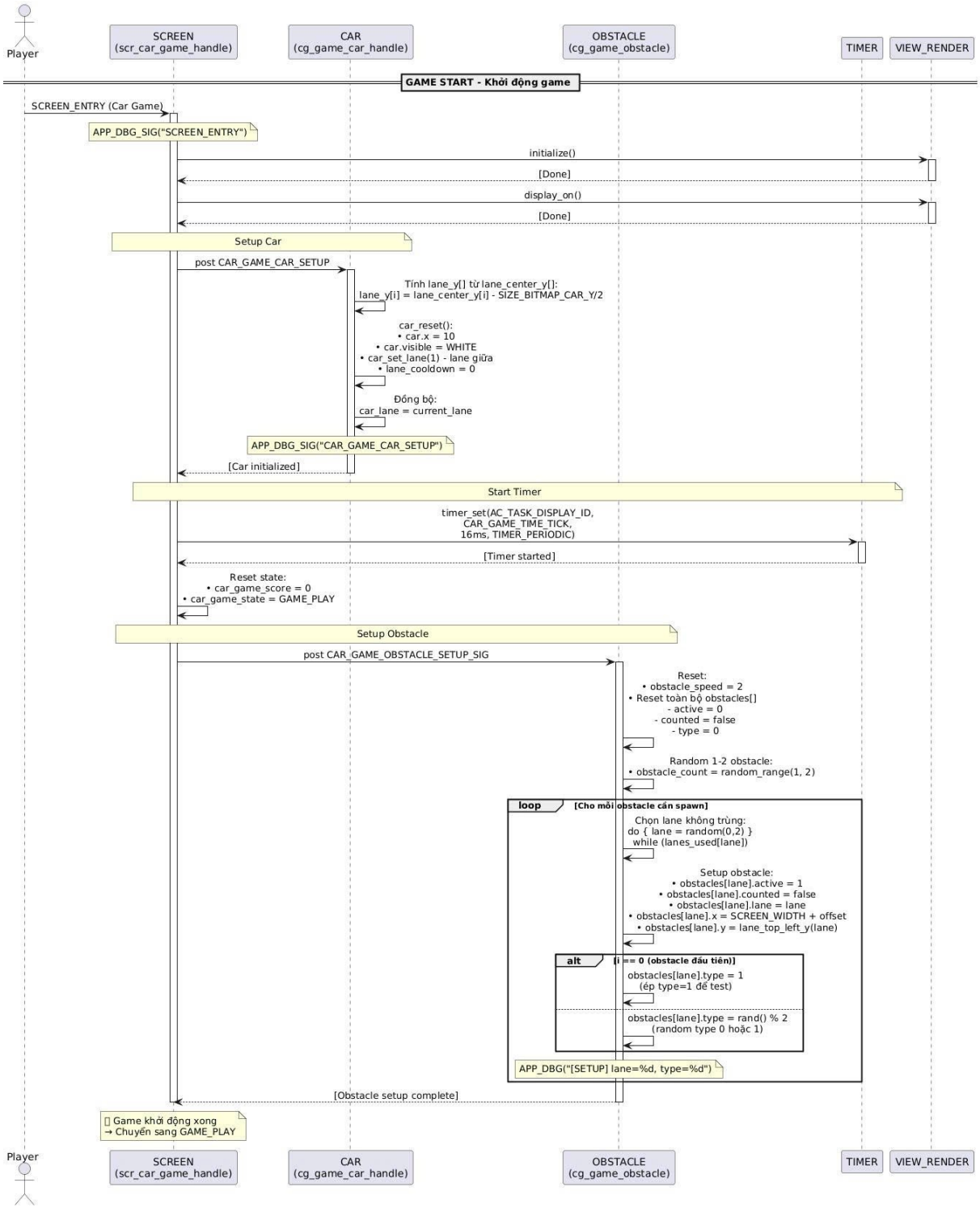
	CAR_GAME_CAR_UPDATE CAR_GAME_CAR_MOVE_LEFT CAR_GAME_CAR_MOVE_RIGHT CAR_GAME_CAR_RESET

-CAR_GAME_SCREEN_ID: điều phối toàn bộ logic và hiển thị.

-CAR_GAME_CAR_ID: xử lý riêng cho đối tượng xe.

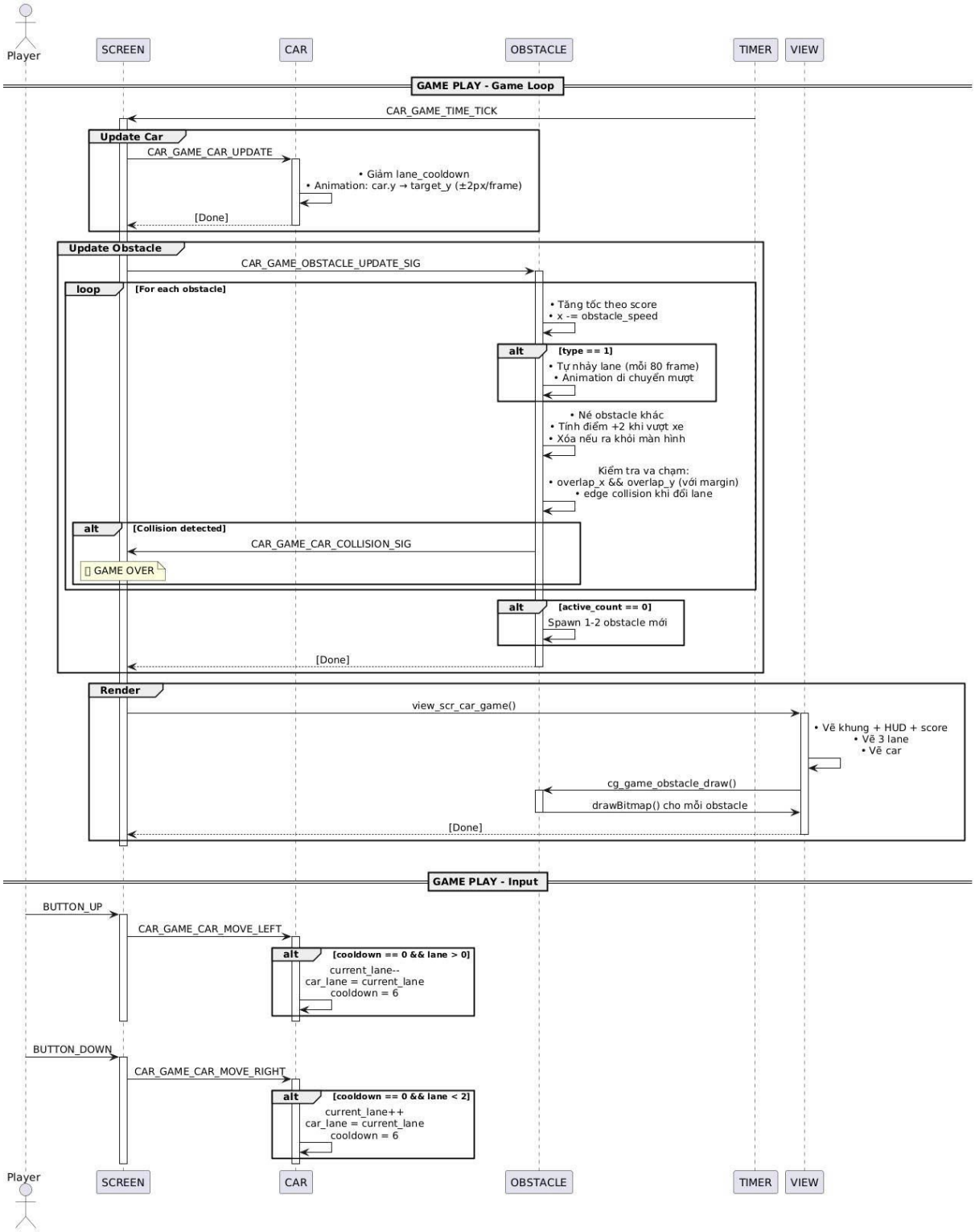
2.4 Sơ đồ quá trình của game

GAME ON



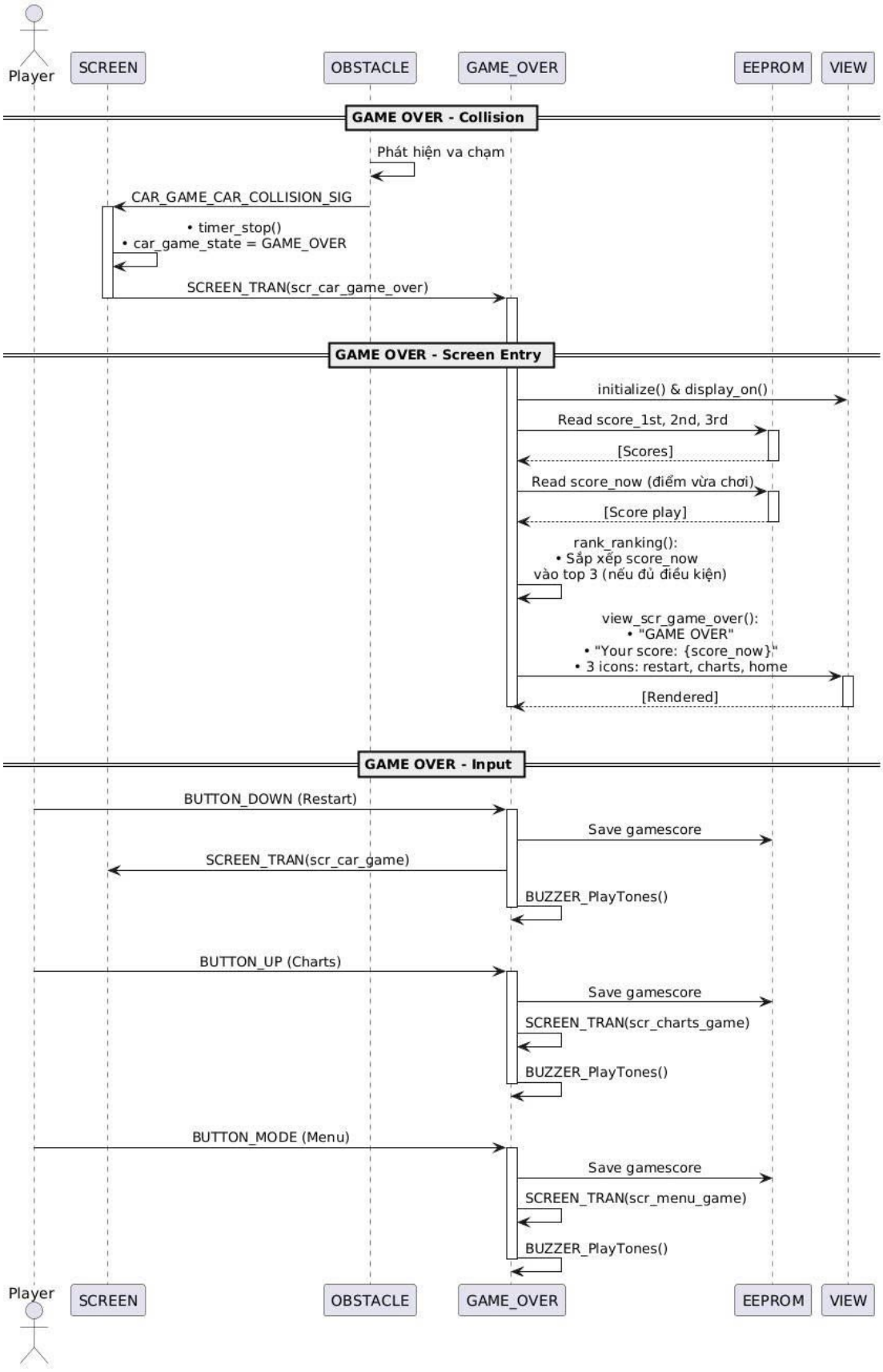
Hình 2.2 Sơ đồ tuần tự game on

GAME RUN



Hình 2.3 Sơ đồ tuần tự khi game play

GAME OVER



Hình 2.4 Sơ đồ tuần tự khi game over

2.4.1 Game On – Khởi tạo trò chơi

-SCREEN_ENTRY: Khi người dùng chọn bắt đầu chơi game, hệ thống khởi tạo màn hình chơi mới.

-CAR_GAME_CAR_SETUP: Thiết lập thông số mặc định cho đối tượng xe: vị trí, lane, trạng thái hiển thị.

-CAR_GAME_OBSTACLE_SETUP_SIG: Cài đặt thông số ban đầu cho các vật cản (số lượng, lane, loại obstacle).

-CAR_GAME_TIME_TICK: Tín hiệu do timer hệ thống (AK-Kernel) gửi đến với chu kỳ 16 ms để bắt đầu vòng lặp cập nhật khung hình.

-car_game_state = GAME_PLAY: Biến nội bộ lưu trạng thái hiện tại của game là đang chơi.

-car_game_score = 0: Đặt lại điểm số ban đầu cho lượt chơi mới.

2.4.2. Game Run – Trò chơi đang hoạt động

CAR_GAME_TIME_TICK: Timer gửi mỗi 16 ms để cập nhật toàn bộ game.

CAR_GAME_CAR_UPDATE: Cập nhật trạng thái và vị trí của xe.

CAR_GAME_OBSTACLE_UPDATE_SIG :Di chuyển obstacle, tăng tốc và kiểm tra va chạm.

CAR_GAME_CAR_COLLISION_SIG: Gửi khi xe va chạm obstacle → dừng game, chuyển sang Game Over.

2.4.3 Game over

-CAR_GAME_CAR_COLLISION_SIG: Gửi từ OBSTACLE đến SCREEN khi xe va chạm. Khi nhận, SCREEN dừng timer, đặt trạng thái GAME_OVER, và chuyển sang màn hình Game Over.

-SCREEN_ENTRY: Được gọi khi vào màn hình Game Over. Dùng để khởi tạo hiển thị, đọc điểm từ EEPROM và xếp hạng điểm.

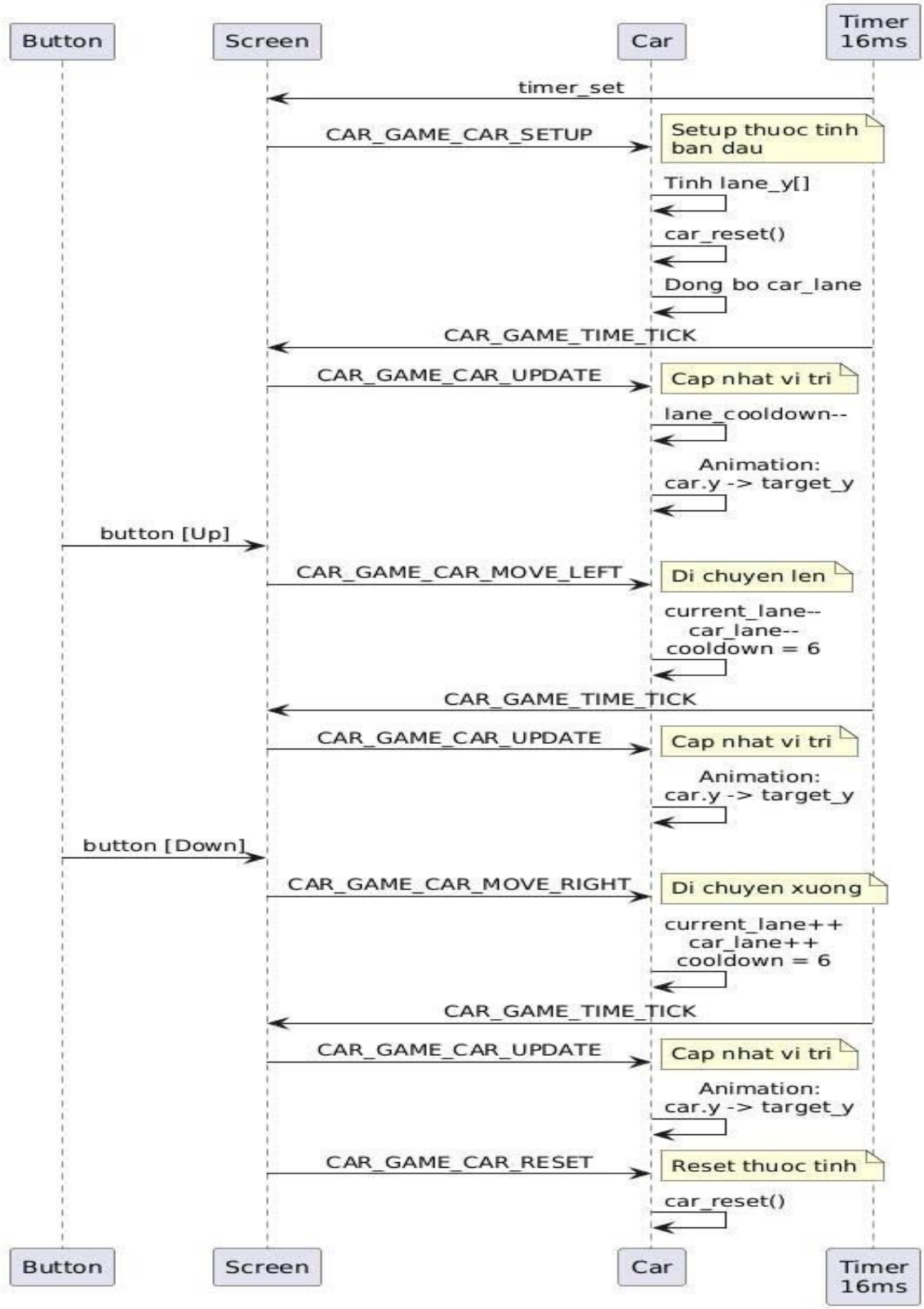
-AC_DISPLAY_BUTTON_MODE_RELEASED: Người chơi nhấn nút MODE → quay lại menu chính (scr_menu_game).

-AC_DISPLAY_BUTTON_UP_RELEASED: Người chơi nhấn UP → mở bảng xếp hạng (scr_car_charts).

-AC_DISPLAY_BUTTON_DOWN_RELEASED: Người chơi nhấn DOWN → khởi động lại game (scr_car_game).

III Hướng dẫn chi tiết code trong đối tượng

3.1 Car



Hình 3.1 Sơ đồ tuần tự đối tượng Car

Tóm tắt nguyên lý:

Đối tượng Car nhận Signal được gửi từ 2 nguồn chính là Screen và Button.

Quá trình xử lý của đối tượng được chia thành 3 giai đoạn:

Giai đoạn 1: Bắt đầu game

Xe được khởi tạo, cài đặt các thông số ban đầu như tọa độ, lane (làn đường) và trạng thái hiển thị.

Giai đoạn 2: Chơi game

Giai đoạn này chia làm 2 hoạt động:

Cập nhật:

Screen gửi Signal CAR_GAME_CAR_UPDATE liên tục (mỗi ~16 ms) để cập nhật trạng thái xe, hiệu ứng di chuyển, và thời gian cooldown giữa hai lần đổi lane.

Hành động:

Khi người chơi nhấn nút UP/DOWN, Button gửi Signal CAR_GAME_CAR_MOVE_LEFT hoặc CAR_GAME_CAR_MOVE_RIGHT giúp xe di chuyển lên hoặc xuống tương ứng giữa ba làn đường.

Giai đoạn 3: Kết thúc game

Khi trò chơi kết thúc hoặc người chơi khởi động lại, Screen gửi Signal CAR_GAME_CAR_RESET để xe trở về trạng thái ban đầu và sẵn sàng cho lượt chơi mới.

Code

-Khai báo: Thư viện, struct và biến

```
#include "cg_game_car.h"

car_t car = {
    .x = 30,
    .y = 0,
    .visible = WHITE,
};
```

-Hai biến toàn cục điều khiển hành vi của xe:

```
int16_t car_x = 30;     // Tọa độ X cố định của xe
uint8_t car_lane = 1;   // Lane hiện tại (0 = top, 1 = mid, 2 = bottom)
```

-Macro định nghĩa hành vi của Car

1. CAR_GAME_CAR_SETUP()

Là macro dùng để cài đặt trạng thái ban đầu của xe khi bắt đầu game.

Thiết lập toạ độ, hiển thị và vị trí làn ban đầu.

```
#define CAR_GAME_CAR_SETUP() \
do { \
    car.x = 10; \
    car.visible = WHITE; \
    current_lane = 1; \
    car_set_lane(current_lane); \
    car_lane = current_lane; \
    lane_cooldown = 0; \
} while(0);
```

2.CAR_GAME_CAR_MOVE_LEFT()

Là macro giúp xe di chuyển lên làn trên (khi người chơi nhấn nút UP).

Giảm giá trị lane hiện tại, có kiểm tra cooldown để tránh đổi lane quá nhanh.

```
#define CAR_GAME_CAR_MOVE_LEFT() \
do { \
    if (lane_cooldown == 0 && current_lane > 0) { \
        current_lane--; \
        car_lane = current_lane; \
        lane_cooldown = LANE_COOLDOWN_FRAMES; \
    } \
} while(0);
```

3.CAR_GAME_CAR_MOVE_RIGHT()

Là macro giúp xe di chuyển xuống làn dưới (khi người chơi nhấn nút DOWN).

```
#define CAR_GAME_CAR_MOVE_RIGHT() \
do { \
    if (lane_cooldown == 0 && current_lane < lane_count - 1) { \
        current_lane++; \
        car_lane = current_lane; \
        lane_cooldown = LANE_COOLDOWN_FRAMES; \
    } \
} while(0);
```

4.CAR_GAME_CAR_RESET()

Macro dùng để đặt lại trạng thái ban đầu của xe khi kết thúc game hoặc reset.

```
#define CAR_GAME_CAR_RESET() \
do { \
   car.x = 10; \
   car.visible = WHITE; \
   car_set_lane(1); \
   car_lane = current_lane; \
   lane_cooldown = 0; \
} while(0);
```

5.Hàm xử lý tín hiệu

Hàm cg_game_car_handle() là nơi xử lý toàn bộ các tín hiệu (Signal) được gửi đến đối tượng xe.

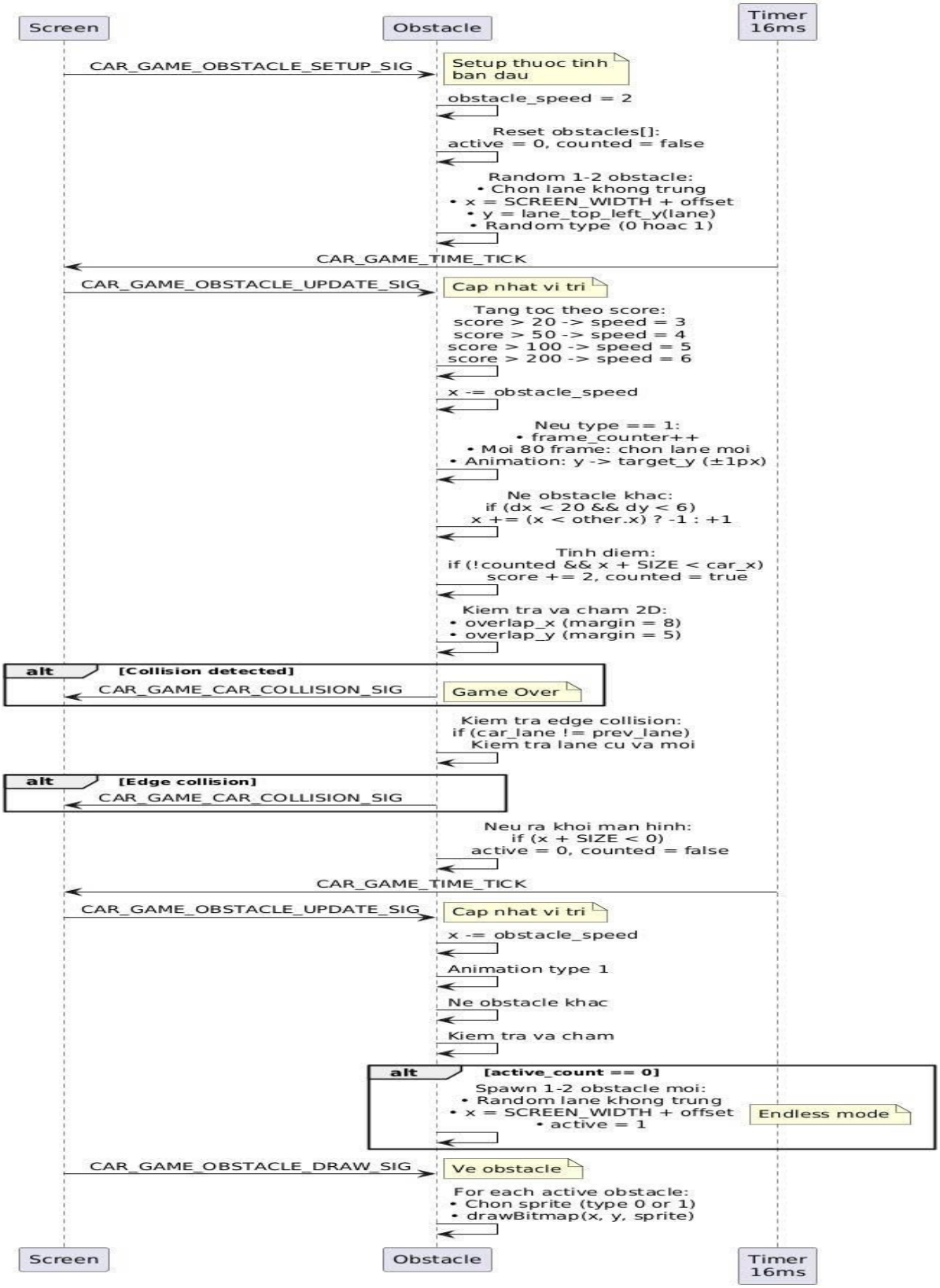
Hàm sử dụng switch-case để xác định và thực thi hành động tương ứng.

```
void cg_game_car_handle(ak_msg_t* msg) {
   switch (msg->sig) {

   case CAR_GAME_CAR_SETUP: {
      APP_DBG_SIG("CAR_GAME_CAR_SETUP\n");
      for (uint8_t i = 0; i < lane_count; i++) {
         lane_y[i] = lane_center_y[i] - SIZE_BITMAP_CAR_Y / 2;
      }
      car_reset();
      car_lane = current_lane;
   } break;

   case CAR_GAME_CAR_UPDATE: {
      if (lane_cooldown > 0) lane_cooldown--;
      static int16_t target_y = 0;
      static uint8_t moving = 0;

      if (!moving) {
         target_y = lane_y[current_lane];
         moving = 1;
      }

      if (car.y < target_y) car.y += 2;
      else if (car.y > target_y) car.y -= 2;
      if (car.y == target_y) moving = 0;
   } break;
```

```
    case CAR_GAME_CAR_MOVE_LEFT: {
        APP_DBG_SIG("CAR MOVE UP -> lane %d\n", current_lane);
        CAR_GAME_CAR_MOVE_LEFT();
    } break;

    case CAR_GAME_CAR_MOVE_RIGHT: {
        APP_DBG_SIG("CAR MOVE DOWN -> lane %d\n", current_lane);
        CAR_GAME_CAR_MOVE_RIGHT();
    } break;

    case CAR_GAME_CAR_RESET: {
        APP_DBG_SIG("CAR_GAME_CAR_RESET\n");
        CAR_GAME_CAR_RESET();
    } break;

    default:
        break;
    }
}
```

\

3.2 Obstacle



Hình 3.2 Sơ đồ tuần tự đối tượng Obstacle

Tóm tắt nguyên lý:

Đối tượng Obstacle nhận Signal được gửi từ Screen và Timer (16 ms).

Quá trình xử lý của đối tượng chia làm 3 giai đoạn chính:

Giai đoạn 1 – Bắt đầu game:

Cài đặt các thông số ban đầu cho các vật cản như vị trí, lane, loại obstacle (type 0 hoặc 1), tốc độ ban đầu.

Giai đoạn 2 – Chơi game:

Bao gồm hai hoạt động:

Cập nhật: Screen gửi signal cập nhật định kỳ để di chuyển obstacle sang trái, tăng tốc theo điểm, và kiểm tra va chạm.

Sinh obstacle mới: Khi không còn obstacle nào đang hoạt động, hệ thống tạo ngẫu nhiên 1–2 obstacle mới (Endless Mode).

Giai đoạn 3 – Kết thúc game:

Khi phát hiện va chạm với xe, gửi signal CAR_GAME_CAR_COLLISION_SIG đến Screen để kết thúc game.

Khai báo và biến

```
#include "cg_game_obstacle.h"
#include "car_game_signal.h"
#include "view_render.h"

#define SCREEN_WIDTH 128
#define SIZE_BITMAP_OBSTACLE_X 18
#define SIZE_BITMAP_OBSTACLE_Y 18
#define OBSTACLE_COUNT 3

static cg_game_obstacle_t obstacles[OBSTACLE_COUNT];
static int16_t obstacle_speed = 2;   // tốc độ ban đầu
```

Các biến liên kết với module Car:

```
extern int16_t car_x;
extern uint8_t car_lane;
extern uint32_t car_game_score;
```

Macro thiết lập và xử lý chính

1. Khởi tạo vị trí và tốc độ ban đầu

```
#define CG_GAME_OBSTACLE_INIT() \
do { \
   for (uint8_t i = 0; i < OBSTACLE_COUNT; i++) { \
      obstacles[i].active = 0; \
      obstacles[i].lane   = i; \
      obstacles[i].x      = SCREEN_WIDTH; \
      obstacles[i].y      = lane_top_left_y(i); \
      obstacles[i].counted = false; \
   } \
} while(0)
```

2. Setup obstacle mới khi bắt đầu game

```
#define CG_GAME_OBSTACLE_SETUP() \
do { \
   obstacle_speed = 2; \
   uint8_t count = random_range(1, 2); \
   for (uint8_t i = 0; i < count; i++) { \
      uint8_t lane = random_range(0, 2); \
      obstacles[lane].active = 1; \
      obstacles[lane].lane = lane; \
      obstacles[lane].x = SCREEN_WIDTH + random_range(32, 64); \
      obstacles[lane].y = lane_top_left_y(lane); \
      obstacles[lane].type = (i == 0) ? 1 : rand() % 2; \
   } \
} while(0)
```

3. Cập nhật vị trí và tăng tốc

```
#define CG_GAME_OBSTACLE_UPDATE(ob) \
do { \
   if (car_game_score > 20)  obstacle_speed = 3; \
   if (car_game_score > 50)  obstacle_speed = 4; \
   if (car_game_score > 100) obstacle_speed = 5; \
   if (car_game_score > 200) obstacle_speed = 6; \
   ob->x -= obstacle_speed; \
} while(0)
```

4. Vẽ obstacle trên màn hình

```
#define CG_GAME_OBSTACLE_DRAW(ob, sprite) \
do { \
   view_render.drawBitmap(ob->x, ob->y, sprite, \
      SIZE_BITMAP_OBSTACLE_X, SIZE_BITMAP_OBSTACLE_Y, WHITE); \
```

```
} while(0)
```

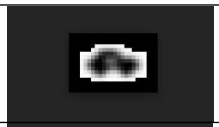
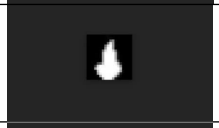
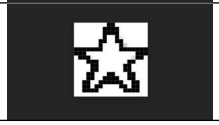
Hàm xử lý tín hiệu

```
void cg_game_obstacle_handle(ak_msg_t* msg) {
  switch (msg->sig) {

  // Thiết lập ban đầu khi bắt đầu game
  case CAR_GAME_OBSTACLE_SETUP_SIG:
    APP_DBG_SIG("CAR_GAME_OBSTACLE_SETUP_SIG\n");
    cg_game_obstacle_setup();
    break;

  // Cập nhật vị trí, điểm và kiểm tra va chạm
  case CAR_GAME_OBSTACLE_UPDATE_SIG:
    cg_game_obstacle_update();
    break;

  // Vẽ obstacle mỗi khung hình
  case CAR_GAME_OBSTACLE_DRAW_SIG:
    cg_game_obstacle_draw();
    break;

  default:
    break;
  }
}
```

IV Hiển thị trong game

4.1.1 Bitmap

Bitmap là một cấu trúc dữ liệu được sử dụng để lưu trữ và hiển thị hình ảnh trong game

Tên đối tượng	Hình ảnh	Mô tả
Car		**24x16px**
Obstacle 1		**17x17px**
Obstacle 0		**18x18px**

4.1.2 Code

Car display

```
void cg_game_car_display() {
  if (car.visible == WHITE) {
    view_render.drawBitmap( car.x,
                car.y,
                bitmap_car,
                SIZE_BITMAP_CAR_X,
                SIZE_BITMAP_CAR_Y,
                WHITE);
  }
}
```

Obstacle display

```
void cg_game_obstacle_draw(void) {
  for (uint8_t i = 0; i < OBSTACLE_COUNT; i++) {
    if (!obstacles[i].active) continue;

    const unsigned char* sprite = bitmap_obstacle;
    if (obstacles[i].type == 1) {
      sprite = bitmap_obstacle2;  // obstacle có animation nhảy lane
    }

    view_render.drawBitmap( obstacles[i].x,
                obstacles[i].y,
                sprite,
                SIZE_BITMAP_OBSTACLE_X,
```

```
                    SIZE_BITMAP_OBSTACLE_Y,
                    WHITE );
  }
}
```

Lane display

```
void car_game_lanes_display() {
  for (uint8_t x = 6; x < LCD_WIDTH-6; x += 8) {
    view_render.drawPixel(x,    12 + (SIZE_BITMAP_CAR_Y/2), WHITE);
    view_render.drawPixel(x+3,  12 + (SIZE_BITMAP_CAR_Y/2), WHITE);

    view_render.drawPixel(x,    30 + (SIZE_BITMAP_CAR_Y/2), WHITE);
    view_render.drawPixel(x+3,  30 + (SIZE_BITMAP_CAR_Y/2), WHITE);

    view_render.drawPixel(x,    48 + (SIZE_BITMAP_CAR_Y/2), WHITE);
    view_render.drawPixel(x+3,  48 + (SIZE_BITMAP_CAR_Y/2), WHITE);
  }
}
```

Frame display

```
void car_game_frame_display() {
  view_render.setTextSize(1);
  view_render.setTextColor(WHITE);
  view_render.drawRect(0, 0, LCD_WIDTH, LCD_HEIGHT, WHITE);
  view_render.drawLine(0, LCD_HEIGHT-10, LCD_WIDTH, LCD_HEIGHT-10, WHITE);

  view_render.setCursor(4, LCD_HEIGHT-8);
  view_render.print("Score:");
  view_render.print(car_game_score);
}
```

Screen display

```
void view_scr_car_game() {
  if (car_game_state == GAME_PLAY) {
    car_game_frame_display();
```

```
    car_game_lanes_display();
    cg_game_car_display();
    cg_game_obstacle_draw();
  }
  else if (car_game_state == GAME_OVER) {
    view_render.clear();
    view_render.setTextSize(2);
    view_render.setTextColor(WHITE);
    view_render.setCursor(16, 22);
    view_render.print("GAME OVER");
    view_render.setTextSize(1);
    view_render.setCursor(28, 44);
    view_render.print("Score: ");
    view_render.print(car_game_score);
  }
}
```