Lab 03

**Báo cáo thực hành Lab 3 (17/05)**

Họ và tên: Phạm Đức Long

Mã số sinh viên: 20225737

[**I, Lưu ý: 1**](#_8e5h7aeyha3x)

[1. Cam kết 1](#_ocbu2prrpkmw)

[2. Hình ảnh, video 1](#_xs4m6fyvweo0)

[**II, Nội dung 2**](#_r76pd5xg6rvt)

[3.1. Làm quen với TouchGFX và FreeRTOS 2](#_b155g9d54ux6)

[3.1.1. Tạo project 2](#_nw3z2kh693yv)

[3.1.2. Lập trình ghép nối và xử lý sự kiện 2](#_aqchephj4xs1)

[3.1.3. Lập trình ghép nối nút bấm USER\_BUTTON 3](#_x0qvjki7gbqn)

[3.2. Xây dựng ứng dụng đồng hồ bấm giây 4](#_wtr5gg4x2f52)

[3.3. Lập trình ghép nối nút bấm USER\_BUTTON để bật/tắt bấm giờ 5](#_t5hx6etbhsqe)

[3.4. Bài tập tự làm 6](#_jkrjthyyyusz)

[3.4.1. Ghép nối phần cứng: 6](#_77h9iqdl4vce)

[3.4.2. Mô tả thiết kế và hoạt động của game 8](#_el3k1vdfr7j2)

[3.4.3. Chi tiết các hàm xử lý và cơ chế hoạt động: 10](#_4rho3pjkxczu)

# **I, Lưu ý:**

## **1. Cam kết**

- Nội dung và mã nguồn trong báo cáo thực hành này là do em và bạn Nguyễn Bá Hoàng (20225844) cùng tự làm. Bất cứ nội dung nào tham khảo từ bên ngoài thì sẽ được nêu rõ nguồn gốc và tác giả.

## **2. Hình ảnh, video**

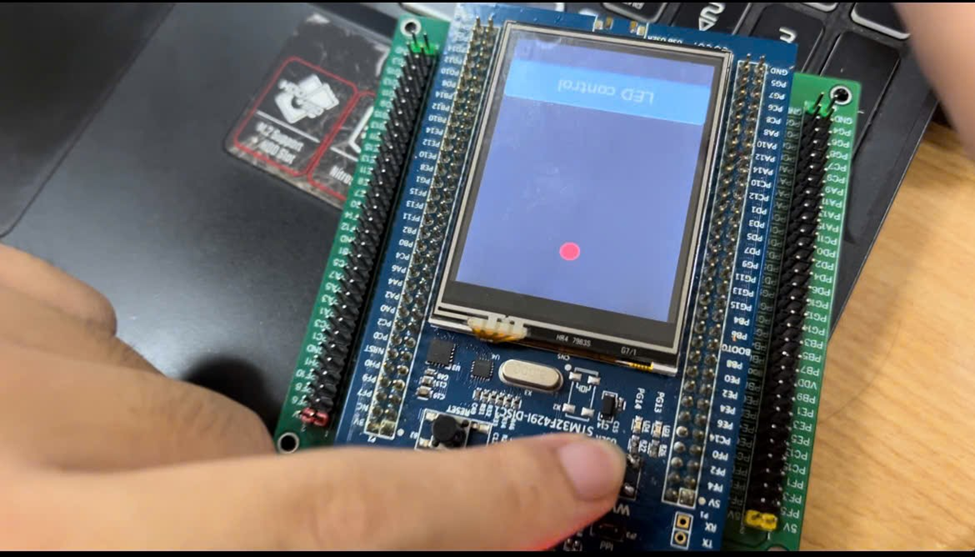
- Tất cả các hình ảnh, video về các bài tập trong buổi thực hành được nhóm em chụp và quay sẽ có ở [link](https://drive.google.com/drive/folders/1LrEbsx9BB5iof-fZEZEA9biSURXxq3v4?usp=sharing) để đảm bảo chất lượng hình ảnh, video và dung lượng file báo cáo ạ.

- Tất cả các bài lab từ bài 1 đến bài 3 em xin được tổng hợp ở [đây](https://drive.google.com/drive/folders/1vS1T02RXkDhAoV9ndzVOVtjsMHtRbMoc?usp=sharing) để thầy có thể nhìn thấy hết ạ.

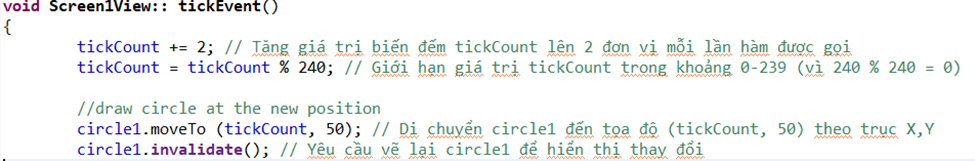
# **II, Nội dung**

## **3.1. Làm quen với TouchGFX và FreeRTOS**

### **3.1.1. Tạo project**



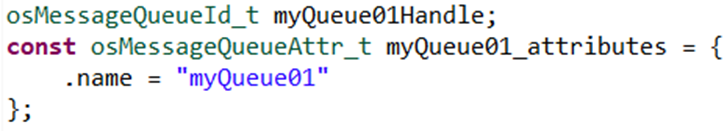
### **3.1.2. Lập trình ghép nối và xử lý sự kiện**

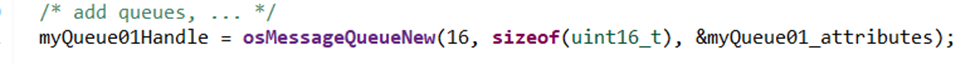


- Giải thích: Hàm tickEvent() điều khiển chuyển động lặp lại của hình tròn (circle1) theo chiều ngang. Mỗi khi hàm được thực thi, hình tròn dịch chuyển sang phải 2 pixel (thông qua tickCount += 2), nhưng luôn được giữ trong khoảng từ 0 đến 239 pixel nhờ phép chia dư (% 240). Khi hình tròn chạm đến biên phải (240px), nó sẽ ngay lập tức quay về vị trí gốc (0px), tạo ra hiệu ứng di chuyển tuần hoàn không dừng. Hàm moveTo() đảm nhiệm việc thay đổi vị trí, còn invalidate() đảm bảo thay đổi này được hiển thị kịp thời trên giao diện.

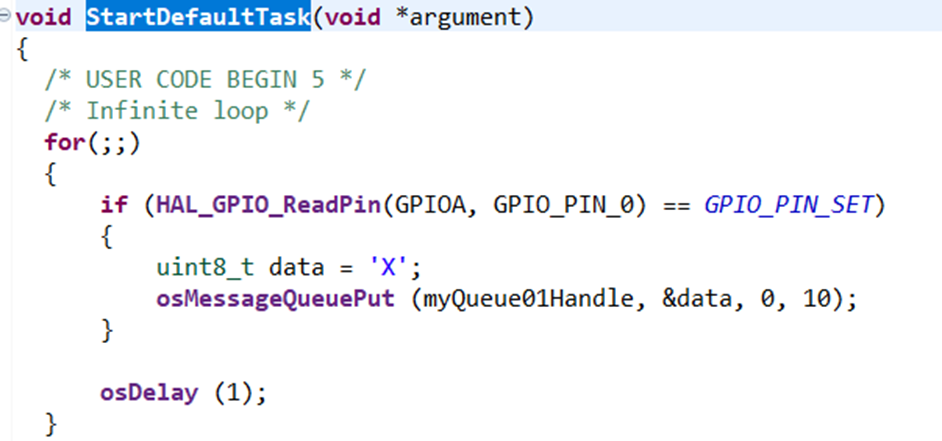
### **3.1.3. Lập trình ghép nối nút bấm USER\_BUTTON**

- Các khai báo:

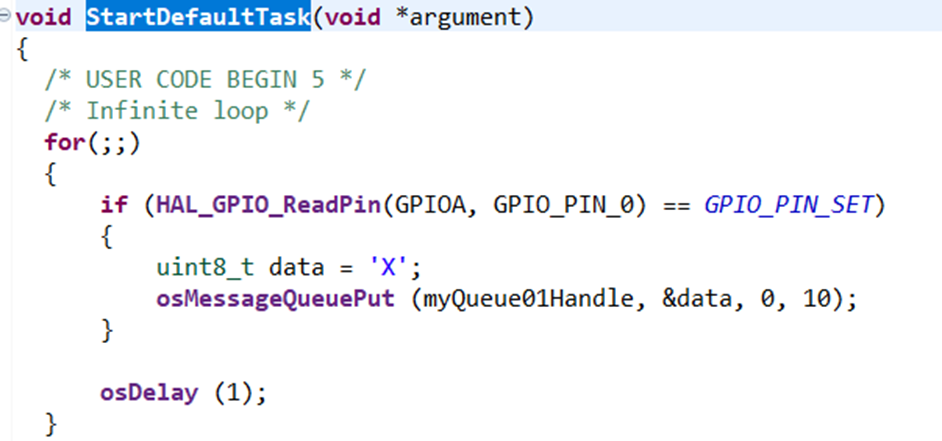




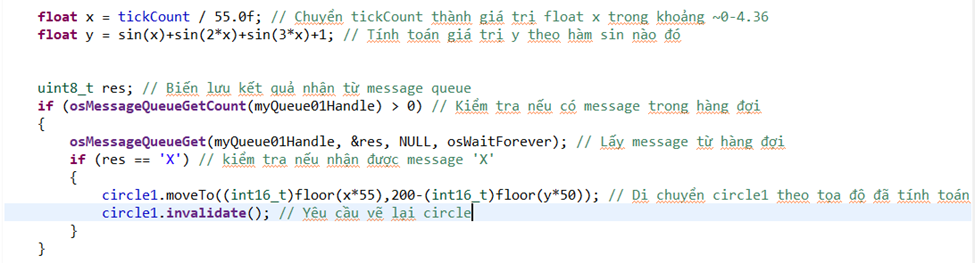
- Tạo queue trước khi tạo task trong file main.c để có thể sử dụng:



- Sau đó polling trạng thái nút bấm trong DefaultTask, gửi ‘X’ vào queue



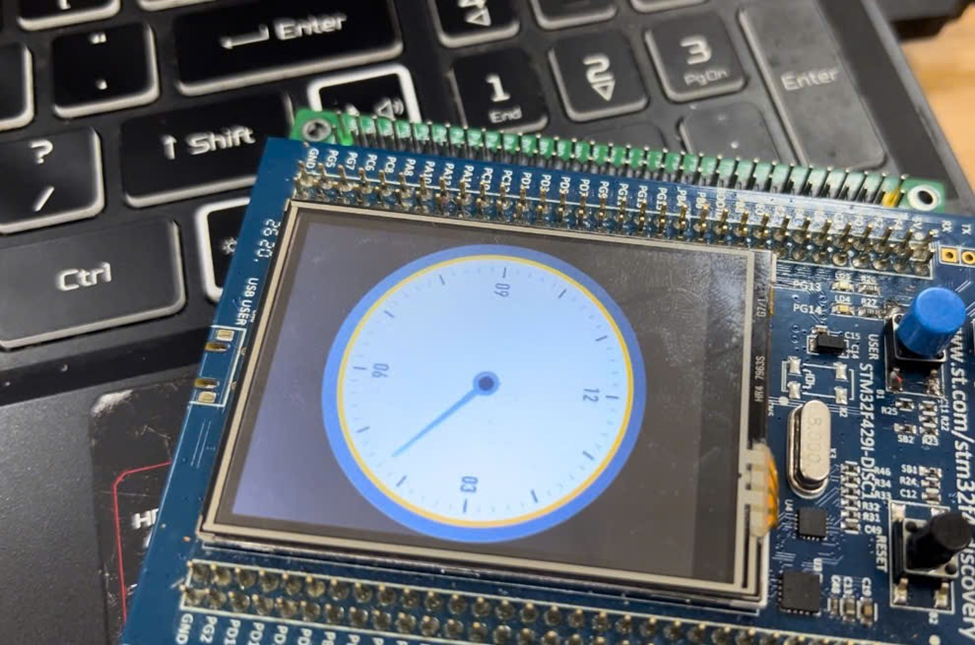
- Hiệu chỉnh tickEvent của Screen1View để nhận dữ liệu từ queue và cập nhật animation:



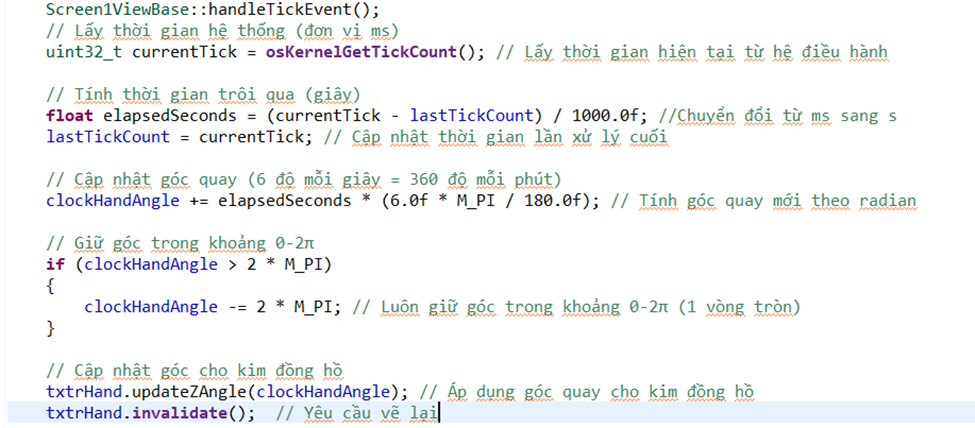
- Tổng quan: Hàm này tạo ra chuyển động phức tạp theo dạng hình sin cho một vòng tròn trên cả trục X và trục Y. Trên trục X, vòng tròn dịch chuyển đều với vận tốc 2 pixel mỗi lần hàm được gọi, còn trên trục Y thì dao động theo sự kết hợp của nhiều hàm sin. Chuyển động này chỉ xảy ra khi nhận được tín hiệu 'X' từ hàng đợi message (lưu trong biến res), nghĩa là có thể điều khiển việc di chuyển thông qua thao tác từ bên ngoài, ở đây là nút bấm.

## **3.2. Xây dựng ứng dụng đồng hồ bấm giây**

- Tạo project:



- Sử dụng hàm osKernelGetTickCount( void ) tính và cập nhật góc dịch chuyển của kim theo tham số giây:



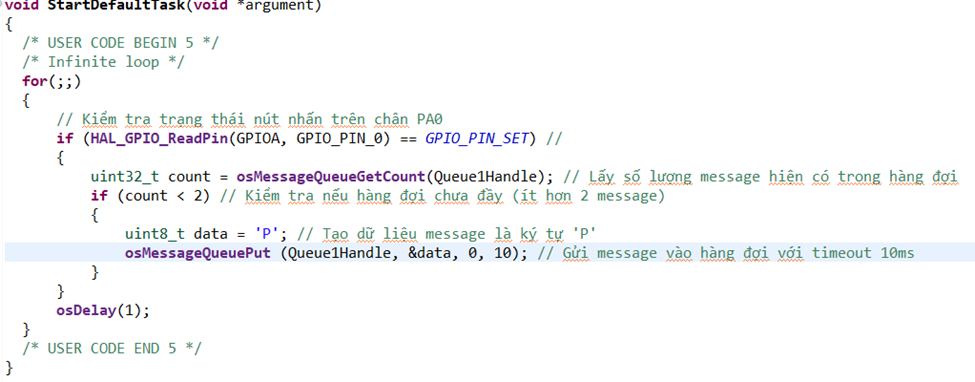
- Giải thích: Hàm này chịu trách nhiệm điều khiển chuyển động quay của kim đồng hồ với vận tốc 6 độ mỗi giây, tức là hoàn thành một vòng quay trong vòng 60 giây. Hệ thống sử dụng thời gian thực giữa các lần gọi hàm để tính toán góc quay phù hợp, giúp kim chuyển động mượt mà ngay cả khi khoảng cách thời gian giữa các lần gọi không đều nhau. Để tránh lỗi tràn số, góc quay được giới hạn trong phạm vi từ 0 đến 2π, và sau đó cập nhật hình ảnh hiển thị lên giao diện bằng cách gọi hàm invalidate.

## **3.3. Lập trình ghép nối nút bấm USER\_BUTTON để bật/tắt bấm giờ**

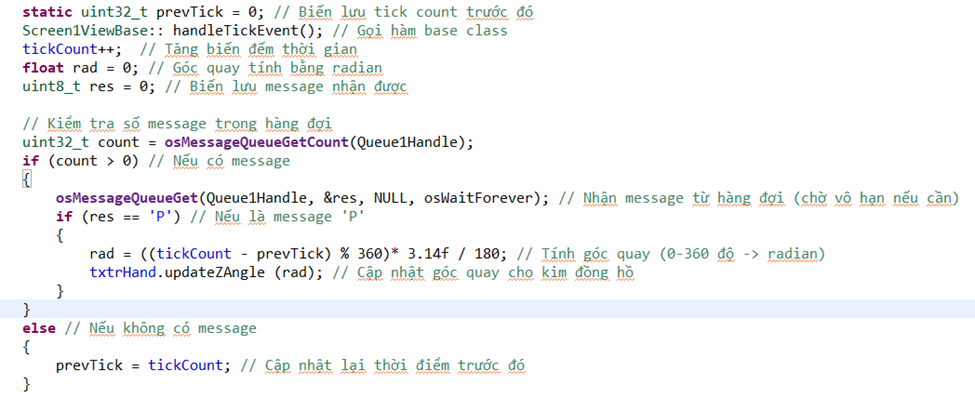
*(\*) Phần 3.3 này thực chất nằm trong 3.2 của handout*

- Giải thích hoạt động của chương trình:

+ Liên tục polling trạng thái nút bấm trong default task, gửi dữ liệu vào queue:



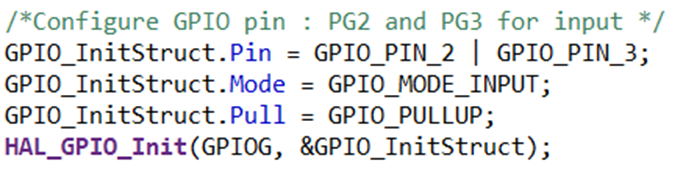
+ Hiệu chỉnh hàm handleTickEvent của Screen1View để nhận dữ liệu từ queue và cập nhật hoạt động của kim đồng hồ:

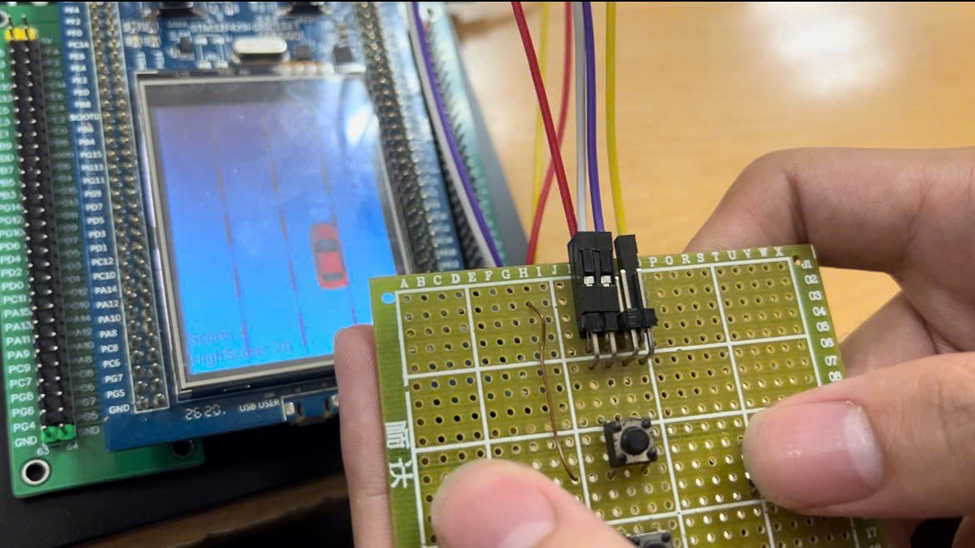


- Tổng quan: Hệ thống này sử dụng hàng đợi message để điều khiển hoạt động của kim đồng hồ. Trong đó, task StartDefaultTask đóng vai trò là producer, gửi message 'P' vào queue mỗi khi phát hiện nút được nhấn, đồng thời giới hạn tối đa 2 message nhằm tránh tình trạng queue bị đầy. Ngược lại, hàm handleTickEvent đóng vai trò consumer, nhận các message này và tính toán góc quay của kim đồng hồ dựa trên khoảng thời gian giữa các lần nhấn nút. Nếu không có message nào, hệ thống sẽ reset lại biến prevTick để đảm bảo mỗi lần nhấn nút mới sẽ khởi động một chuyển động quay bắt đầu từ vị trí hiện tại. Cơ chế này giúp đồng bộ hóa hiệu quả giữa thao tác nhấn nút và quá trình quay của kim đồng hồ.

## **3.4. Bài tập tự làm**

### **3.4.1. Ghép nối phần cứng:**

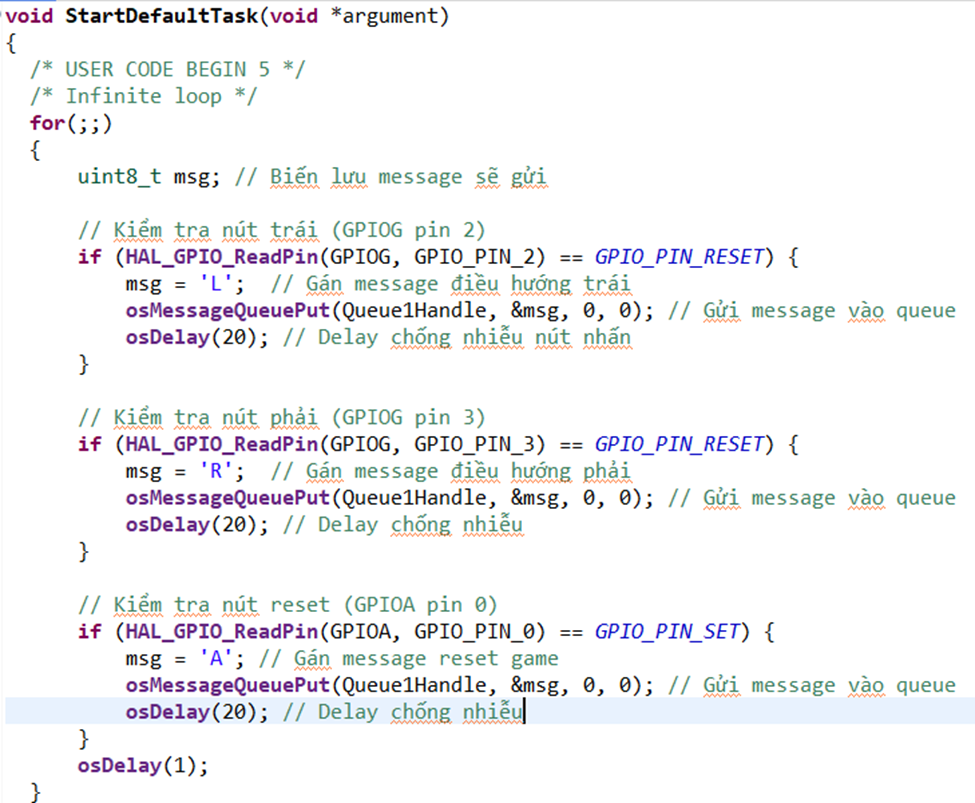


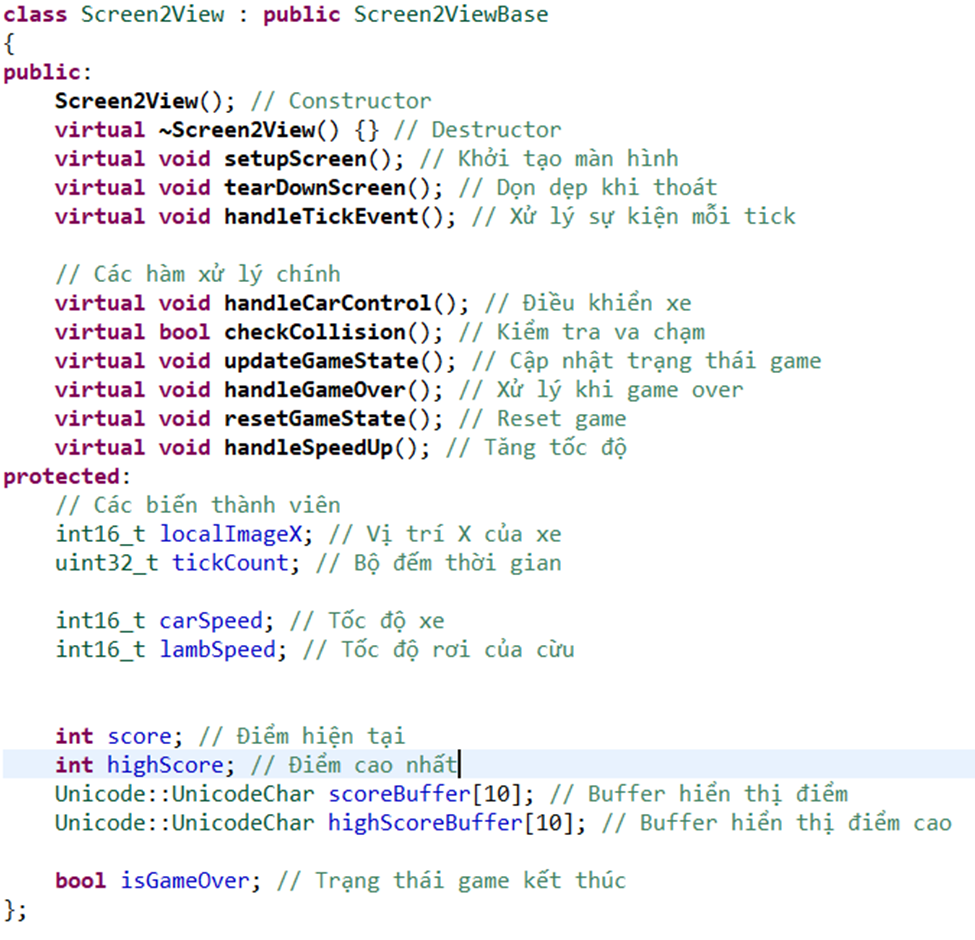


- Giải thích:

* Hệ thống cho phép điều khiển hướng di chuyển của xe sang trái hoặc phải thông qua các nút bấm. Trên mạch có tổng cộng 4 nút, trong đó các chân tín hiệu điều khiển (ví dụ như dây màu vàng trong hình) được kết nối vào các chân nằm ở phía dưới của mạch, còn các chân nối đất (như dây màu tím) được đấu vào các chân ở phía trên.
* Mỗi nút bấm đều có chân tín hiệu riêng biệt và được hàn ra 4 chân phía dưới để đảm bảo điều khiển độc lập. Trong khi đó, tất cả các chân nối đất của 4 nút được gom chung và hàn vào một điểm nối tại hàng chân phía trên. Tín hiệu từ các nút bấm này sau đó được kết nối tới các chân G2 và G3 của STM32, tương ứng với chức năng điều khiển xe rẽ trái hoặc phải trong trò chơi.

### **3.4.2. Mô tả thiết kế và hoạt động của game**





1. Thiết kế hệ thống điều khiển:

- Hệ thống áp dụng mô hình producer-consumer sử dụng hàng đợi message để xử lý tín hiệu đầu vào. Trong đó, hàm StartDefaultTask đảm nhận vai trò producer, liên tục giám sát trạng thái của ba nút điều khiển (trái, phải và reset), và gửi các ký hiệu tương ứng ('L', 'R', 'A') vào queue. Để loại bỏ nhiễu, mỗi lần nhấn nút sẽ kèm theo khoảng trễ 20ms, đảm bảo chỉ gửi đúng một message cho mỗi lần nhấn.

2. Cơ chế chuyển động:

- Trò chơi có hai dạng chuyển động chính: xe do người chơi điều khiển để né vật cản (nút trái/phải) và cừu rơi từ phía trên xuống. Hàm handleCarControl sẽ xử lý di chuyển xe dựa trên các message trong queue, còn updateGameState điều khiển chuyển động rơi của cừu với tốc độ do biến lambSpeed quyết định. Để tăng độ khó theo thời gian, hàm handleSpeedUp sẽ tự động nâng tốc độ rơi sau mỗi 500 tick. Hiệu ứng chuyển động của đường đua được mô phỏng bằng cách lần lượt hiển thị 5 khung hình (track 0 đến 4) trong hàm handleTickEvent.

3. Cơ chế phát hiện va chạm:

- Hàm checkCollision có nhiệm vụ kiểm tra xem xe có chạm vào vật thể hay không. Nếu phát hiện va chạm, handleGameOver sẽ xử lý kết thúc trò chơi bằng cách ẩn vật thể, hiển thị dòng chữ “Game Over”, đồng thời đặt biến isGameOver thành true. Việc kiểm tra va chạm diễn ra trong mỗi khung hình thông qua handleTickEvent.

4. Tính điểm

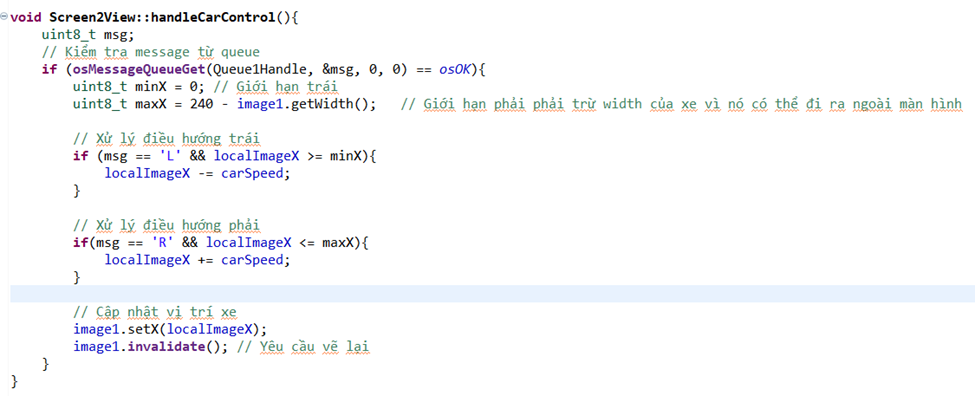
- Mỗi khi vật thể rơi xuống dưới màn hình (tọa độ y ≥ 320), người chơi sẽ được cộng thêm 1 điểm, xử lý bởi updateGameState. Nếu điểm hiện tại vượt qua điểm cao nhất cũ, biến highScore sẽ được cập nhật tương ứng. Vị trí xuất hiện mới của vật thể được xác định thông qua thuật toán sinh số giả ngẫu nhiên xorshift32, đồng thời tính toán dựa trên vị trí xe hiện tại để tạo ra độ khó hợp lý hơn.

5. Tổng quan:

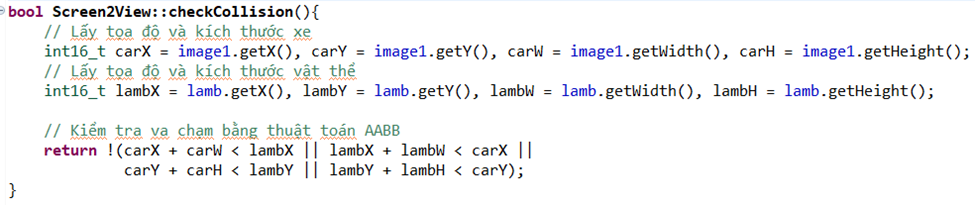
- Toàn bộ hoạt động game được tổ chức theo vòng lặp gồm ba giai đoạn: Nhận input -> Xử lý logic -> Hiển thị. Hàm StartDefaultTask thu thập các sự kiện điều khiển và đẩy vào hàng đợi, còn handleTickEvent xử lý game logic ở từng khung hình. Khi trò chơi kết thúc, hệ thống sẽ chờ người chơi gửi message ‘A’ để khởi động lại game qua resetGameState. Đồng thời, biến tickCount được dùng để đồng bộ thời gian trong game, giúp trò chơi ngày càng nhanh hơn nhờ cơ chế tăng tốc độ tự động từ hàm handleSpeedUp.

### **3.4.3. Chi tiết các hàm xử lý và cơ chế hoạt động:**

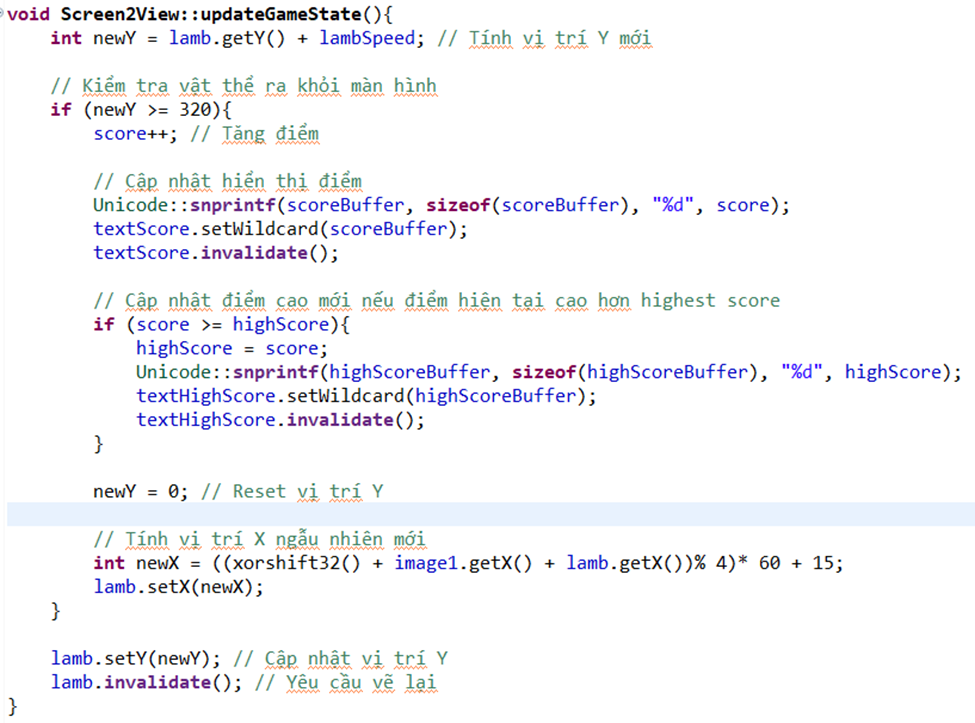
1. Điều khiển:



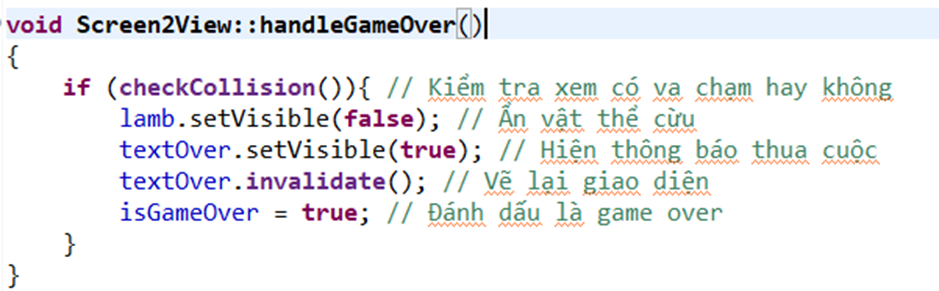
2. Xử lý va chạm:



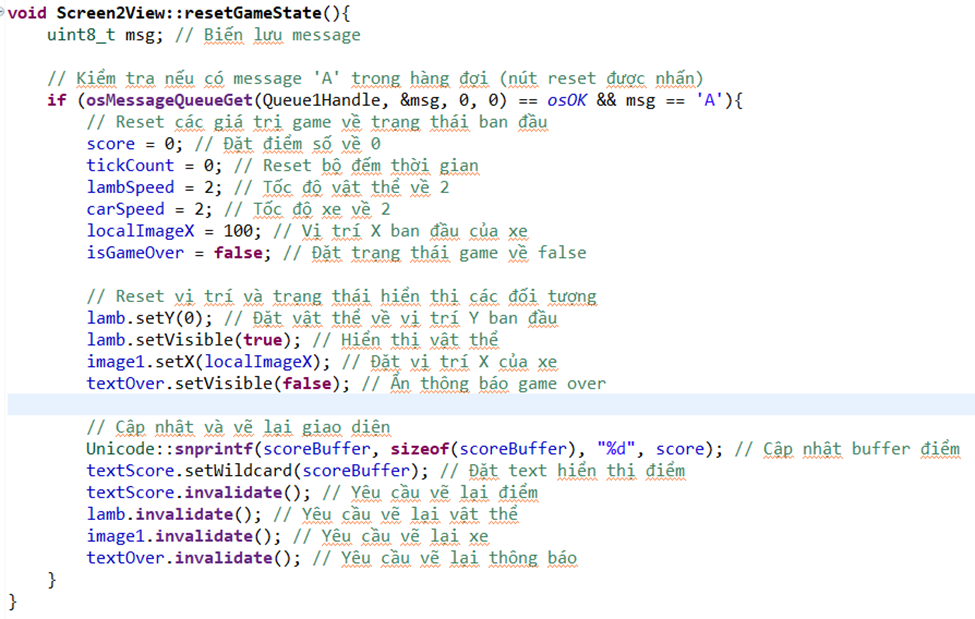
3. Cập nhật trạng thái:



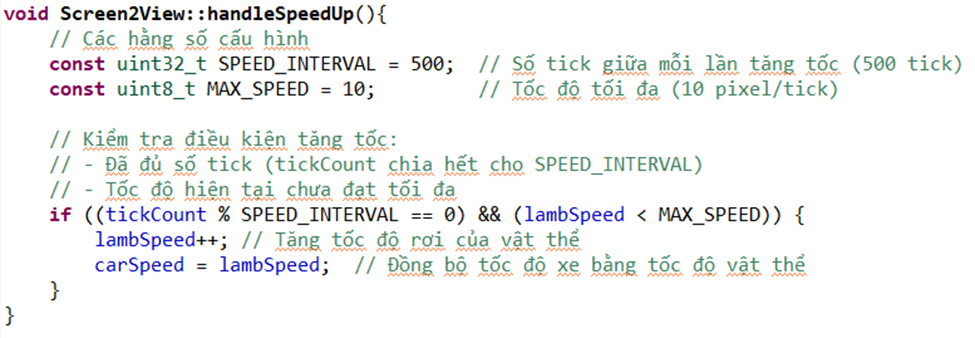
4. Xử lý kết thúc:



5. Reset game:



6. Cơ chế speed-up:



7. Xử lí tick:

