7/10/2023

Phan Quoc Buu

Phòng thực nghiệm IoT - FPT Telecom

**ARCHERY GAME**

Game Example

# MỤC LỤC

[CHƯƠNG 1: MỤC LỤC 1](#_Toc144131961)

[CHƯƠNG 2: Chương 1: ARCHERY GAME PLAY 3](#_Toc144131962)

[2.1 Cách chơi 3](#_Toc144131963)

[2.2 Phần cứng 5](#_Toc144131964)

[CHƯƠNG 3: Thiết kế Archery game 6](#_Toc144131965)

[3.1 6](#_Toc144131966)

[3.2 Sơ đồ trình tự (The sequence diagram) 6](#_Toc144131967)

[3.3 Quản lý tài nguyên đối tượng 8](#_Toc144131968)

[3.3.1 Quản lý Struct 8](#_Toc144131969)

[3.3.2 Quản lý task 9](#_Toc144131970)

[3.3.3 Quản lý signal 10](#_Toc144131971)

[3.3.4 Quản lý hiển thị đối tượng 11](#_Toc144131972)

[CHƯƠNG 4: Logic code 13](#_Toc144131973)

[4.1 Lập trình hướng đối tượng. 13](#_Toc144131974)

[4.2 Cung tên (Archery) 13](#_Toc144131975)

[4.3 Mũi tên (Arrow) 15](#_Toc144131976)

[4.4 Vụ nổ (Bang) 17](#_Toc144131977)

[4.5 Ranh giới an toàn (Border) 19](#_Toc144131978)

[4.6 Thiên thạch (Meteoroid) 21](#_Toc144131979)

[4.7 Màn hình (Screen) 23](#_Toc144131980)

[CHƯƠNG 5: Màn hình hiển thị 26](#_Toc144131981)

[5.1 Nguyên lý 26](#_Toc144131982)

[5.2 Bitmap và animation. 27](#_Toc144131983)

# PHỤ LỤC HÌNH ẢNH

[Hình 1.1‑1: Menu game 3](#_Toc143611334)

[Hình 1.1‑2: Các thành phần trong game 3](#_Toc143611335)

[Hình 1.1‑3: Game Over 4](#_Toc143611336)

[Hình 1.2‑1 AK Embedded Base Kit - STM32L151 5](#_Toc143611337)

[Hình 2.1‑1: The sequence diagram 6](#_Toc143611338)

[Hình 3.1‑1: Archery – sequence 14](#_Toc143611339)

[Hình 3.2‑1: Arrow - sequence 16](#_Toc143611340)

[Hình 3.3‑1: Bang - sequence 18](#_Toc143611341)

[Hình 3.4‑1: Border - sequence 20](#_Toc143611342)

[Hình 3.5‑1: Meteoroid - sequence 22](#_Toc143611343)

[Hình 3.6‑1: Screen - sequence 24](#_Toc143611344)

[Hình 4.1‑1: Xử lý hiển thị và cập nhật màn hình 26](#_Toc143611345)

# PHỤ LỤC BẢNG

[Bảng 1: Quản lý Struct 9](#_Toc143611378)

[Bảng 2: Quản lý task 10](#_Toc143611379)

[Bảng 3: Quản lý Signal 11](#_Toc143611380)

[Bảng 4: Quản lý Bitmap 12](#_Toc143611381)

# ARCHERY GAME PLAY

## Cách chơi

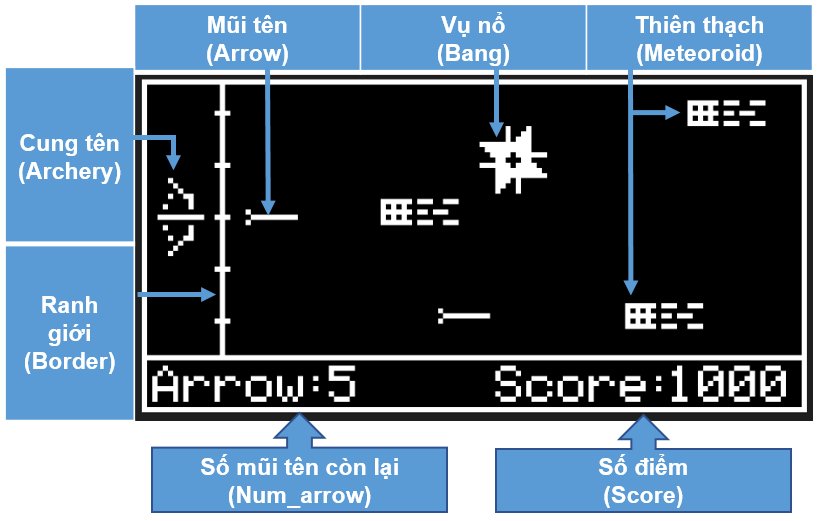
Phần mô tả sau đây về “Archery game” phục vụ mục đích là giải thích cách chơi và các đặc tính kỹ thuật nhằm mục đích thiết kế và triển khai phần mềm và phát triển các dự án game ở phần sau chương trình này.



Hình 1.1‑1: Menu game

Trò chơi bắt đầu với màn hình Menu game với nhiều chọn lựa:

* Archery Game: chọn vào để 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.
* Exit: vào màn hình nghỉ.



Hình 1.1‑2: Các thành phần trong game

**Cách chơi game:**

Trong game này bạn sẽ điều khiển 1 cung tên (Archery) di chuyển lên xuống chọn vị trí và bắn ra mũi tên (Arrow) tiêu diệt các đối tượng là các thiên thạch (Meteoroid) đang bay đến. Với số lượng mũi tên giới hạn, mỗi lần bắn tên số mũi tên sẽ giảm xuống và không bắn được nếu số mũi tên bằng “0” được hiển thị ở bên dưới góc trái màn hình hiển thị.

Với mỗi lần mũi tên (Arrow) bắn trúng thiên thạch (Meteoroid) thì thiên thạch sẽ tạo ra vụ nổ (Bang) và được +10 điểm vào “score” ở bên dưới góc phải màn hình. Và số lượng mũi tên cũng được khôi phục + 1 mũi tên. Ngoài ra khi mũi tên đi hết màn hình thì mũi tên cũng được xóa và khôi phục +1 mũi tên.

Với mỗi 200 điểm kiếm được thì tốc độ của thiên thạch (Meteoroid) sẽ tăng dần lên tạo độ khó cho game.

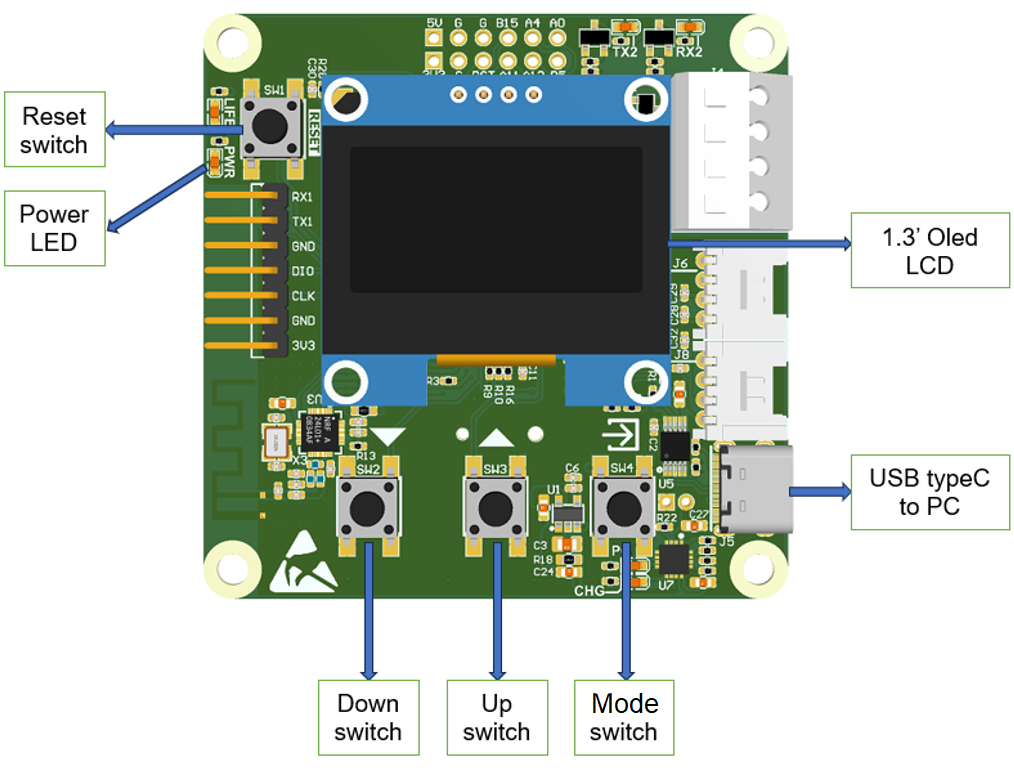


Hình 1.1‑3: Game Over

Game sẽ kết thúc khi thiên thạch (Meteoroid) chạm phải ranh giới (Border). Số điểm sẽ được lưu và bạn sẽ vào màn hình “Game Over” bạn có 3 lựa chọn là:

* Restart: chơi lại.
* Charts: vào xem bảng xếp hạng.
* Menu: về lại menu game.

## Phần cứng



Hình 1.2‑1 AK Embedded Base Kit - STM32L151

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

KIT tích hợp LCD Oled 1.3", 3 nút nhấn, và 1 loa Buzzer phát nhạc, với các trang bị này thì đã đủ để học hệ thống event-driven thông qua thực hành thiết kế máy chơi game..

KIT cũng tích hợp RS485, NRF24L01+, và Flash lên đến 32MB, thích hợp cho prototype các ứng dụng thực tế trong hệ thống nhúng hay sử dụng như: truyền thông có dây, không dây wireless, các ứng dụng lưu trữ data logger,...

**Thao tác trong game:**

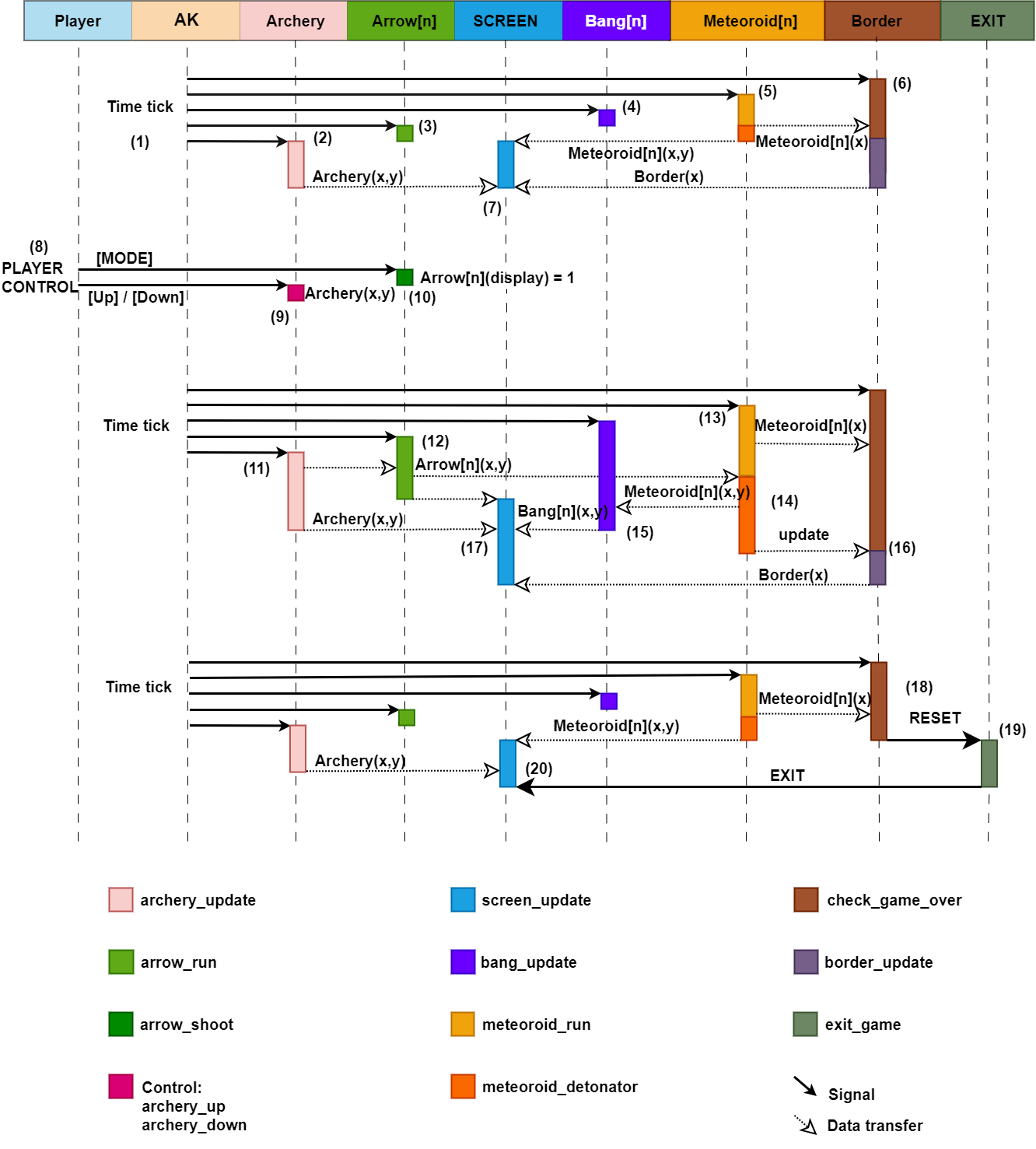
* Trong game người chơi có thể sử dụng 2 nút [Up] và [Down] để điều hướng chọn mục hay điều khiển Cung tên (Archery) lên xuống.
* Nút [Mode] dùng để chọn vào các mục trong Menu game hay bắn ra các mũi tên lúc chơi game.
* Nút [Reset] để khởi động lại kit.

# Thiết kế Archery game

## 

## Sơ đồ trình tự (The sequence diagram)

Sơ đồ trình tự được sử dụng để mô tả và hiển thị trình tự của các thông điệp và tương tác giữa các đối tượng trong một hệ thống.



Hình 2.2‑1: The sequence diagram

Trong sơ đồ trình tự trên:

* Time tick – Common: mô phỏng trạng thái của game lúc hoạt động bình thường.
* Time tick – Control: mô phỏng trạng thái của game lúc có các tác động điều khiển qua nút nhấn.
* Time tick – Game over: mô phỏng trạng thái xử lý lúc game over.

Các thành phần:

* Player Control: người chơi sẽ chơi game thông qua 3 nút là [Up], [Down] và [Mode].
* AK – Timer: là nơi tạo ra Time tick gửi tính hiệu cập nhật thường xuyên cho các đối tượng.
* Archery: mô tả đối tượng Archery game trong lúc xử lý.
* Arrow[n]: Đối tượng số n đại diện cho tất cả các đối tượng mũi tên khác.
* Bang[n]: Đối tượng số n đại diện cho tất cả các đối tượng vụ nổ khác.
* Border: Đối tượng vùng an toàn.
* Meteoroid[n]: Đối tượng số n đại diện cho tất cả các đối tượng thiên thạch khác.
* Screen: Đối tượng đại diện cho màn hình hiển thị.
* Exit: Đại diện cho xử lý Exit.

**Chú thích:**

(1) Time tick: là một Signal được tạo ra bởi 1 timer thường xuyên. Time tick kiểm soát thời gian tính toán logic và cập nhật khung hình của game.

(2) Archery – time tick: tại time tick sẽ cập nhật trạng thái của Archery và hiển thị trên màn hình (SCREEN)

(3) Arrow[n] – time tick: tại time tick vì arrow[n].visible = 0 nên bỏ qua cũng không hiển thị ra màn hình

(4) Bang [n] – time tick: tại time tick vì bang[n].visible = 0 nên bỏ qua cũng không hiển thị ra màn hình

(5) Meteoroid[n] – time tick: tại time tick đối tượng Thiên thạch (Meteoroid) chạy 2 signal để thay đổi vị trí của thiên thạch và kiểm tra thiên thạch có bị mũi tên bắn trúng. Cập nhật vị trí hiện tại để hiển thị trên màn hình.

(6) Border[n] – time tick: tại time tick đối tượng Border kiểm tra update vị trí hiện tại và so sánh vị trí với Meteoroid để kiểm tra điều kiện “game over”.

(7) Screen: là màn hình hiện thị của game, là nơi tiếp nhận các thông số của đối tượng để hiển thị ra màn hình.

(8) Player control: gồm 3 thao tác là [Mode], [Up], [Down]. [Mode] điều khiển Arrow[n] bắn ra.

[Up] / [Down] điều khiển vị trí của Archery.

(9) Archery – control: vị trí mới của cung tên sau khi được điều khiển.

(10) Arrow – control: điều khiển mũi tên bắn ra Arrow[n].visible = 1.

(11) Archery – time tick: Cập nhật vị trí Archery sang cho Arrow[n] và Screen.

(12) Arrow[n] – time tick: Vì lần này mũi tên đã được bắn ra nên time tick sẽ chuyển thành Arrow\_run.

(13) Meteoroid[n] – time tick: Arrow bắn trúng Meteoroid nên sẽ chuyển sang cập nhật trạng thái.

(14) Meteoroid\_detonator: chuyển Bang[n].visible = 1 và số điểm +10

(15) Bang[n] – time tick: Bang nhận thông tin vị trí của Meteoroid rồi cập nhật thông tin hiển thị ra màn hình.

(16) Border – time tick: Border update tùy thuộc vào số điểm hiện tại của game nếu số điểm tăng lên 200 thì sẽ dịch border đi một đoạn rồi cập nhật ra màn hình.

(17) Screen: màn hình cập nhật các giá trị mới và hiển thị các đối tượng ra màn hình.

(18) Trường hợp: Meteoroid trùng vị trí với Border đạt điều kiện “game over”

(19) Exit: Border gửi tín hiệu Reset để reset các đối tượng.

(20) Screen: Exit gửi đến screen tín hiệu exit để hiển thị ra màn hình “Game Over”

## Quản lý tài nguyên đối tượng

Sau khi xác định được các đối tượng trong game mà chúng ta cần, tiếp theo chúng ta phải liệt kê ra các thuộc tính, các task, các signal và bitmap mà trong game sẽ sử dụng tới.

Việc liệt kê càng chi tiết thì việc làm game diễn ra càng nhanh và tạo tình rõ ràng minh bạch cho phần tài nguyên giúp phần code game diễn ra xuông sẽ hơn.

### Quản lý Struct

Việc liệt kê các thuộc tính của đối tượng trong game có các tác dụng quan trọng sau:

* Định rõ thông tin: Liệt kê các thuộc tính giúp xác định rõ thông tin về đối tượng trong game.
* Thiết kế cấu trúc dữ liệu: Liệt kê thuộc tính giúp xác định cấu trúc dữ liệu phù hợp để lưu trữ thông tin của đối tượng.
* Giảm rủi ro và lỗi: Khi bạn xác định trước các thuộc tính cần thiết, bạn giảm thiểu khả năng bỏ sót hoặc nhầm lẫn trong việc xử lý và sử dụng các thuộc tính.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STRUCT** | | | | |
| Đối tượng | Kiểu dữ liệu | Thuộc tính | Ghi chú | Tác dụng |
| Archery | uint32\_t | x | archery.x | Tạo độ x |
| uint32\_t | y | archery.y | Tạo độ y |
| bool | visible | archery.visible | Hiển thị |
| uint8\_t | action\_image | archery.action\_image | (\*) |
| Arrow | uint32\_t | x | arrow[n].x | Tạo độ x |
| uint32\_t | y | arrow[n].y | Tạo độ y |
| bool | visible | arrow[n].visible | Hiển thị |
| uint8\_t | action\_image | arrow[n].action\_image | (\*) |
| Meteoroid | uint32\_t | x | meteoroid[n].x | Tạo độ x |
| uint32\_t | y | meteoroid[n].y | Tạo độ y |
| bool | visible | meteoroid[n].visible | Hiển thị |
| uint8\_t | action\_image | meteoroid[n].action\_image | (\*) |
| Bang | uint32\_t | x | bang[n].x | Tạo độ x |
| uint32\_t | y | bang[n].y | Tạo độ y |
| Bool | visible | bang[n].visible | Hiển thị |
| uint8\_t | action\_image | bang[n].action\_image | (\*) |
| Border | uint32\_t | x | border.x | Tạo độ x |
| uint32\_t | y | border.y | Tạo độ y |
| uint32\_t | h | border.h | Chiều cao h |
| Bool | visible | border.visible | Hiển thị |
| Biến | | | | |
| Tên biến | Kiểu dữ liệu | Biến | Giá trị đầu | Tác dụng |
| Score | uint32\_t | ar\_game\_score | 10 | Điểm |
| Số mũi tên | uint8\_t | num\_arrow | (\*\*)eeprom | Số mũi tên hiện tại |
| Tốc độ của mũi tên | uint8\_t | arrow\_speed | 5 | Tốc độ của mũi tên |
| Tốc độ của thiên thạch | Uint8\_t | meteoroid\_speed | (\*\*)eeprom | Tốc độ của thiên thạch |

Bảng 1: Quản lý Struct

(\*) action\_image là 1 biến chuyên dụng để chuyển animation, có thể cần thiết với các game muốn sinh động còn không có thể bỏ qua

(\*\*) giá trị ban đầu được lưu trữ trong eeprom để không bị mất lúc reset kit

### Quản lý task

Trong lập trình sự kiện (event-driven programming), một task là một đơn vị công việc độc lập được thực thi khi xảy ra một sự kiện cụ thể. Tác dụng của task là xử lý và đáp ứng cho các sự kiện trong hệ thống. Một số tác dụng quan trọng của task code:

* Xử lý sự kiện: Task được sử dụng để xử lý các sự kiện khi chúng xảy ra. Mỗi task có thể được liên kết với một sự kiện cụ thể và thực thi một loạt các hành động khi sự kiện đó xảy ra.
* Đồng bộ hóa: Task cung cấp cơ chế đồng bộ hóa cho việc xử lý các sự kiện. Khi một sự kiện xảy ra, task tương ứng được kích hoạt và thực thi. Các task khác có thể đợi cho đến khi task hiện tại hoàn thành trước khi được kích hoạt. Điều này giúp đảm bảo rằng các hành động xử lý sự kiện được thực hiện theo một thứ tự nhất định và tránh xung đột.
* Quản lý luồng điều khiển: Task cho phép quản lý luồng điều khiển trong ứng dụng event-driven. Bằng cách sử dụng task, bạn có thể xác định thứ tự thực thi của các hành động khi xảy ra các sự kiện khác nhau.
* Tách biệt logic: Sử dụng task giúp tách biệt logic xử lý sự kiện ra khỏi logic chính của ứng dụng. Điều này giúp tăng tính sạch sẽ, dễ đọc và dễ bảo trì của mã nguồn.
* Phân cấp nhiệm vụ: Task level cho phép việc xắp xếp trình tự ưu tiên xử lý của task.

|  |  |  |
| --- | --- | --- |
| Tasks ID | Task level | App tasks |
| AR\_GAME\_ARCHERY\_ID | TASK\_PRI\_LEVEL\_4 | ar\_game\_archery\_handle |
| AR\_GAME\_ARROW\_ID | TASK\_PRI\_LEVEL\_4 | ar\_game\_arrow\_handle |
| AR\_GAME\_BANG\_ID | TASK\_PRI\_LEVEL\_4 | ar\_game\_bang\_handle |
| AR\_GAME\_BORDER\_ID | TASK\_PRI\_LEVEL\_4 | ar\_game\_border\_handle |
| AR\_GAME\_METEOROID\_ID | TASK\_PRI\_LEVEL\_4 | ar\_game\_meteoroid\_handle |
| AR\_GAME\_SCREEN\_ID | TASK\_PRI\_LEVEL\_4 | scr\_archer\_game\_handle |

Bảng 2: Quản lý task

### Quản lý signal

Signal là một công cụ để thông báo về sự kiện và gọi hàm xử lý tương ứng trong hệ thống. Nó được sử dụng để gửi thông điệp và điều khiển luồng xử lý trong các ứng dụng và hệ thống phức tạp.

Trong game này Signal có các tác dụng như:

* Xử lý sự kiện: Signal được sử dụng để xử lý các sự kiện trong game.
* Giao tiếp giữa các thành phần: Signal cho phép giao tiếp giữa các thành phần trong game.
* Bất đồng bộ và xử lý đa luồng: Signal cung cấp khả năng xử lý bất đồng bộ và đa luồng trong game. Khi một tín hiệu được phát ra, các hàm xử lý có thể được gọi đồng thời hoặc song song trong các luồng khác nhau.

|  |  |  |
| --- | --- | --- |
| Đối tượng | Task ID | Signal |
| Archery | AR\_GAME\_ARCHERY\_ID | AR\_GAME\_ARCHERY\_SETUP |
| AR\_GAME\_ARCHERY\_UPDATE |
| AR\_GAME\_ARCHERY\_UP |
| AR\_GAME\_ARCHERY\_DOWN |
| AR\_GAME\_ARCHERY\_RESET |
| Arrow | AR\_GAME\_ARROW\_ID | AR\_GAME\_ARROW\_SETUP |
| AR\_GAME\_ARROW\_RUN |
| AR\_GAME\_ARROW\_SHOOT |
| AR\_GAME\_ARROW\_RESET |
| Bang | AR\_GAME\_BANG\_ID | AR\_GAME\_BANG\_SETUP |
| AR\_GAME\_BANG\_UPDATE |
| AR\_GAME\_BANG\_RESET |
| Border | AR\_GAME\_BORDER\_ID | AR\_GAME\_BORDER\_SETUP |
| AR\_GAME\_BORDER\_UPDATE |
| AR\_GAME\_BORDER\_RESET |
| AR\_GAME\_CHECK\_GAME\_OVER |
| Meteoroid | AR\_GAME\_METEOROID\_ID | AR\_GAME\_METEOROID\_SETUP |
| AR\_GAME\_METEOROID\_RUN |
| AR\_GAME\_METEOROID\_DETONATOR |
| AR\_GAME\_METEOROID\_RESET |
| Screen | AR\_GAME\_SCREEN\_ID | AR\_GAME\_INITIAL\_SETUP |
| AR\_GAME\_TIME\_TICK |
| AR\_GAME\_RESET |
| AR\_GAME\_EXIT\_GAME |

Bảng 3: Quản lý Signal

* Setup: Signal khởi tạo thông số ban đầu cho các đối tượng
* Update/Run: Signal cập nhật theo thời gian Time tick
* Reset: khởi tạo lại các giá trị hiện tại trước khi end game
* Exit: thoát game

### Quản lý hiển thị đối tượng

#### Bitmap

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

Animation là ứng dụng việc nối ảnh của của nhiều ảnh liên tiếp tạo thành hoạt ảnh cho đổi tượng muốn miêu tả. Trong game này tôi dùng biến “action\_image” trong đối tượng để thay đổi hoạt ảnh thành animation.

|  |  |  |  |
| --- | --- | --- | --- |
| Tên đối tượng | | Hình ảnh | Mô tả |
| Archery  (15x15px) | Archery\_I |  | Cung lúc có tên |
| Archery\_II |  | Cung lúc hết tên |
| Arrow (10 x 5 px) | |  | Mũi tên |
| Bang  (15x15px)  (10x10px) | Bang\_I |  | Vụ nổ animation 1 |
| Bang\_II |  | Vụ nổ animation 2 |
| Bang\_III |  | Vụ nổ animation 3 |
| Meteoroid  (20x10px) | Meteoroid\_I |  | Thiên thạch animation 1 |
| Meteoroid\_II |  | Thiên thạch animation 2 |
| Meteoroid\_III |  | Thiên thạch animation 3 |

Bảng 4: Quản lý Bitmap

# Logic code

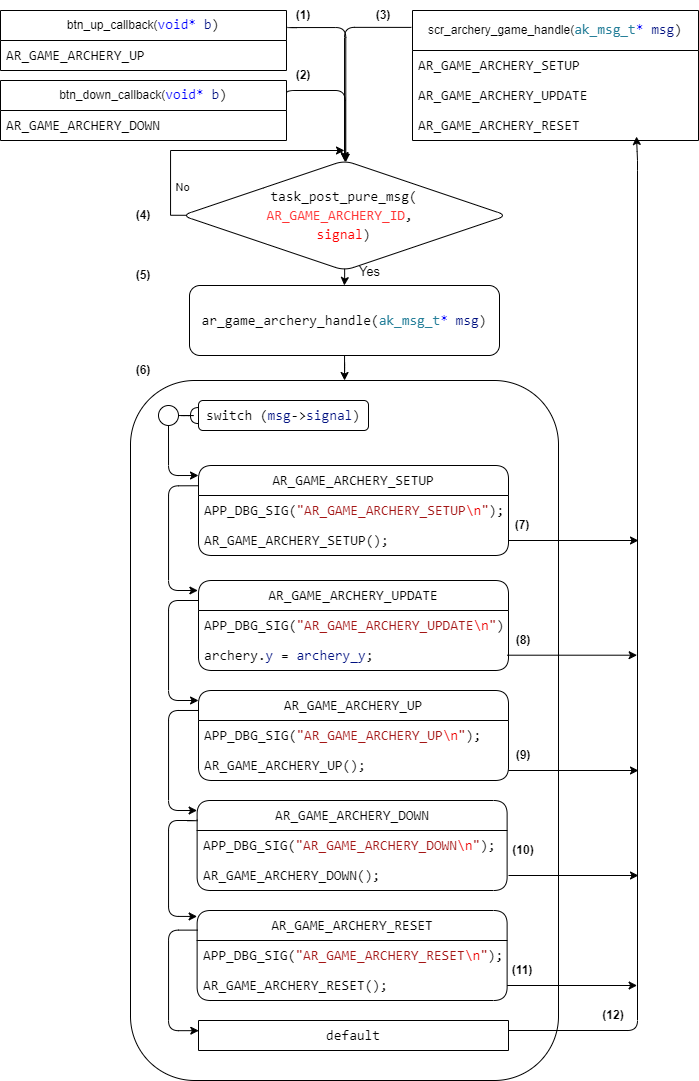
## Lập trình hướng đối tượng.

Lập trình hướng đối tượng (Object-Oriented Programming - OOP) là một phương pháp lập trình được sử dụng rộng rãi trong việc phát triển phần mềm. Nó tập trung vào việc tổ chức code thành các đối tượng riêng biệt, có tính chất, thuộc tính và hành vi riêng. Các đối tượng này tương tác với nhau thông qua việc gửi thông điệp và thực hiện phương thức của mình.

|  |  |
| --- | --- |
|  | Giải thích về sơ đồ nguyên lý:  Sơ đồ nguyên lý sẽ là 1 vòng lặp hoàn chỉnh:  Phần 1: Sẽ là nơi cấp phát “Signal” hay nơi gửi lệnh cho đối tượng.  Phần 2: task\_post\_pure\_msg() là hàm gửi message đến đối tượng dựa vào Tast\_ID đã tạo trước và kèm với 1 signal để gửi lệnh.  Phần 3: hàm \_handle() là hàm được đăng ký đi kèm với Task\_ID là nơi chứa các hàm xử lý của từng Signal trong các đối tượng.  Phần 4: hàm swich() dựa vào signal gửi để xác định và thực hiện xử lý cần thiết.  Sau đó trở về lại nơi cấp phát đăng ký từ trước.  (Giải thích: việc trở về scr\_archery\_game\_handle() vì hệ thống có con trỏ hệ thống đến vị trí màn hình hiện tại nên nếu không có bước nào tiếp theo thì sẽ trở vệ ví trí con trỏ đang trỏ đến.) |

## Cung tên (Archery)

Đối tượng Cung tên là đối tượng quan trọng nhất game. Trạng thái bình thường của Cung tên sẽ được duy trì bởi 2 biến là archery.x và archery.y xác đinh tọa độ xy của đối tượng trên màn hình. Khi có bất cứ tín hiệu thay đổi nào từ 2 nút nhấn [Up] và [Down] sẽ thay đổi giá trị của biến “archery\_y” và sau “Time tick” sẽ cập nhật vào thuộc tính “archery.y” trước khi cập nhật vẽ lại màn hình theo thuộc tính mới. Hai thuộc tính x và y cũng là neo điểm để đối tượng Mũi tên (Arrow) kế thừa vị trí để thực hiện hành động bắn ra.



Hình 3.2‑1: Archery – sequence

**Ghi chú:**

(1) Hàm btn\_up\_callback(void\* b) là hàm kiểm tra button [up] có thể gửi tới signal:

* AR\_GAME\_ARCHERY\_UP

(2) Hàm btn\_down\_callback(void\* b) là hàm kiểm tra button [down] có thể gửi tới signal:

* AR\_GAME\_ARCHERY\_DOWN

(3) Hàm scr\_archery\_game\_handle(ak\_msg\_t\* msg) là hàm xử lý màn hình (Screen) có thể gửi tới 3 signal:

* AR\_GAME\_ARCHERY\_SETUP
* AR\_GAME\_ARCHERY\_UPDATE
* AR\_GAME\_ARCHERY\_RESET

(4) Hàm gửi pure message thông qua “task\_id” và “signal” của đối tượng Cung tên (Archery).

(5) Hàm ar\_game\_archery\_handle(ak\_msg\_t\* msg) tiếp nhận message chứa “signal” và xử lý.

(6) AR\_GAME\_ARCHERY\_SETUP là signal được sử dụng để setup các giá trị bạn đầu cho đối tượng.

(7) AR\_GAME\_ARCHERY\_UPDATE là signal được sử dụng để cập nhật vị trí hình ảnh mới cho đối tượng.

(8) AR\_GAME\_ARCHERY\_UP là signal được sử dụng cập nhật giá trị vị trí tạo độ “archery.y” khi nhấn nút [up].

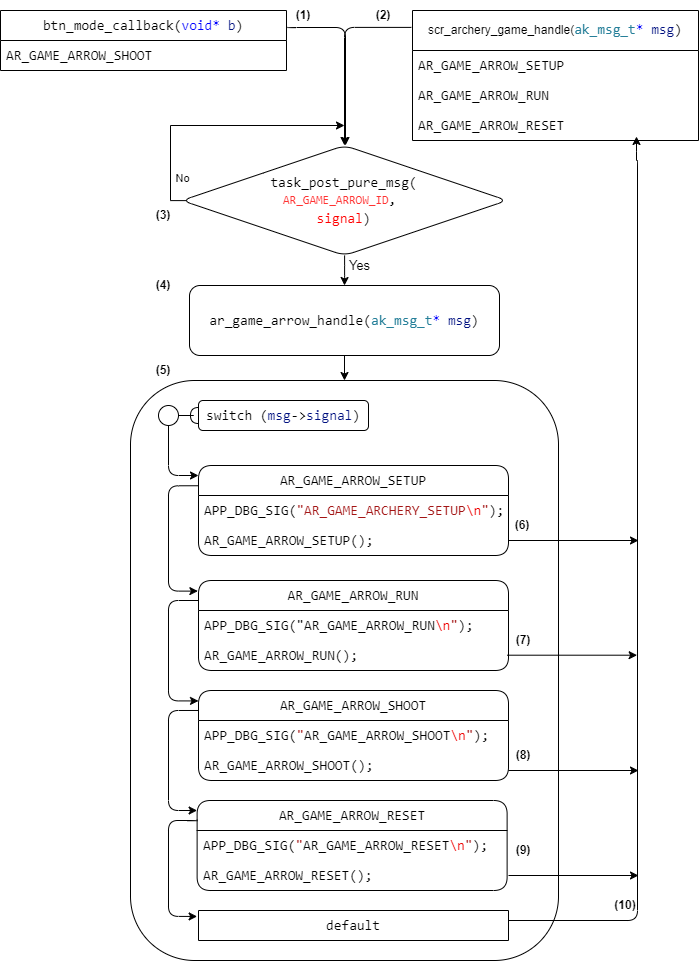
(9) AR\_GAME\_ARCHERY\_DOWN là signal được sử dụng cập nhật giá trị vị trí tạo độ archery.y khi nhấn nút [down].

(10) AR\_GAME\_ARCHERY\_RESET là signal được sử dụng để reset các giá trị ban đầu cho đối tượng trước khi thoát game.

(11) Sau khi xử lý xong yếu cầu theo signal thì sẽ có lệnh “break” để kết thúc xử lý và quay lại scr\_archery\_game\_handle() để đợi xử lý tast\_post tiếp theo.

## Mũi tên (Arrow)

Đối tượng Mũi tên là đối tượng đa thành phần được tạo thành từ 1 mảng các đối tượng arrow[n] mỗi đối tượng trong thành phần phải được tính toán riêng biệt. Nên để game hoạt động trôi chảy thì phải giới hạn số lượng mũi tên sẽ xuất hiện cùng lúc trên màn hình. Mỗi lần nhấn nút [Mode] sẽ có 1 mũi tên bắn ra.



Hình 3.3‑1: Arrow - sequence

**Ghi chú:**

(1) Hàm btn\_mode\_callback(void\* b) là hàm kiểm tra button [mode] có thể gửi tới signal:

* AR\_GAME\_ARROW\_SHOOT

(2) Hàm scr\_archery\_game\_handle(ak\_msg\_t\* msg) là hàm xử lý màn hình (Screen) có thể gửi tới 3 signal:

* AR\_GAME\_ARROW\_SETUP
* AR\_GAME\_ARROW\_RUN
* AR\_GAME\_ARROW\_RESET

(3) Hàm gửi pure message thông qua “task\_id” và “signal” của đối tượng Mũi tên (Arrow).

(4) Hàm ar\_game\_arrow\_handle (ak\_msg\_t\* msg) tiếp nhận message chứa “signal” và xử lý.

(5) Hàm “switch” nhận tín hiệu “signal” rồi thực thiện lệnh ứng với “signal” đó.

(6) AR\_GAME\_ARROW\_SETUP là signal được sử dụng để setup các giá trị bạn đầu cho đối tượng.

(7) AR\_GAME\_ARROW\_RUN là signal được sử dụng để cập nhật vị trí di chuyển theo thời gian và cập nhật hình ảnh mới cho đối tượng.

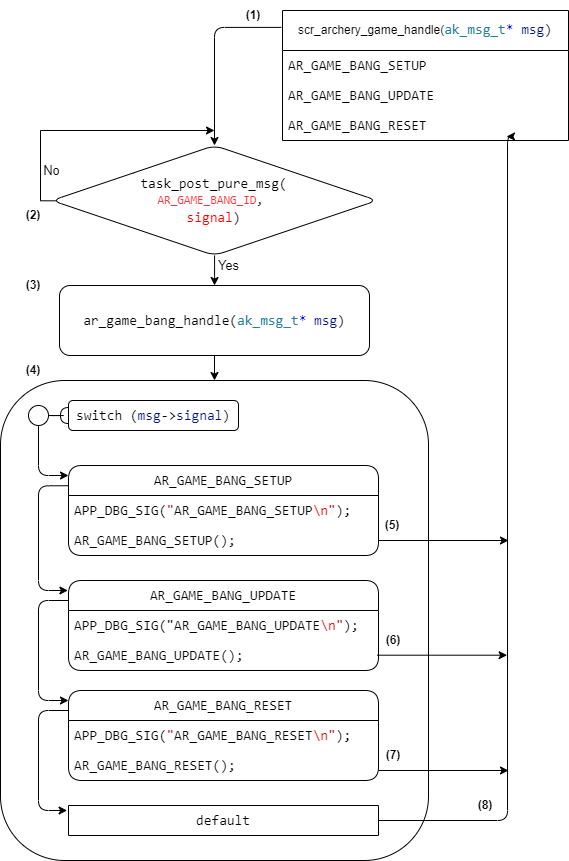
(8) AR\_GAME\_ARROW\_SHOOT là signal được sử dụng để tiếp nhận tín hiệu từ nút bấm [Mode] để nhận lệnh bắn tên.

(9) AR\_GAME\_ARROW\_RESET là signal được sử dụng để reset các giá trị ban đầu cho đối tượng trước khi thoát game.

(10) Sau khi xử lý xong yếu cầu theo signal thì sẽ có lệnh “break” để kết thúc xử lý và quay lại scr\_archery\_game\_handle() để đợi xử lý tast\_post tiếp theo.

## Vụ nổ (Bang)

Vụ nổ (Bang) là đối tượng xuất hiện có điều kiện thông qua sự va chạm của 2 đối tượng Mũi tên (Arrow) và Thiên thạch (Meteoroid). Vụ nổ xuất hiện sau vụ va chạm, kế thừa tạo độ của Thiên thạch tạo ra hoạt ảnh nổ tại tạo độ đó kèm theo âm thanh.



Hình 3.4‑1: Bang - sequence

**Ghi chú:**

(1) Hàm scr\_archery\_game\_handle(ak\_msg\_t\* msg) là hàm xử lý màn hình (Screen) có thể gửi tới 3 signal:

* AR\_GAME\_BANG\_SETUP
* AR\_GAME\_BANG\_UPDATE
* AR\_GAME\_BANG\_RESET

(2) Hàm gửi pure message thông qua “task\_id” và “signal” của đối tượng Vụ nổ (Bang).

(3) Hàm ar\_game\_bang\_handle (ak\_msg\_t\* msg) tiếp nhận message chứa “signal” và xử lý.

(4) Hàm “switch” nhận tín hiệu “signal” rồi thực thiện lệnh ứng với “signal” đó.

(5) AR\_GAME\_BANG\_SETUP là signal được sử dụng để setup các giá trị bạn đầu cho đối tượng.

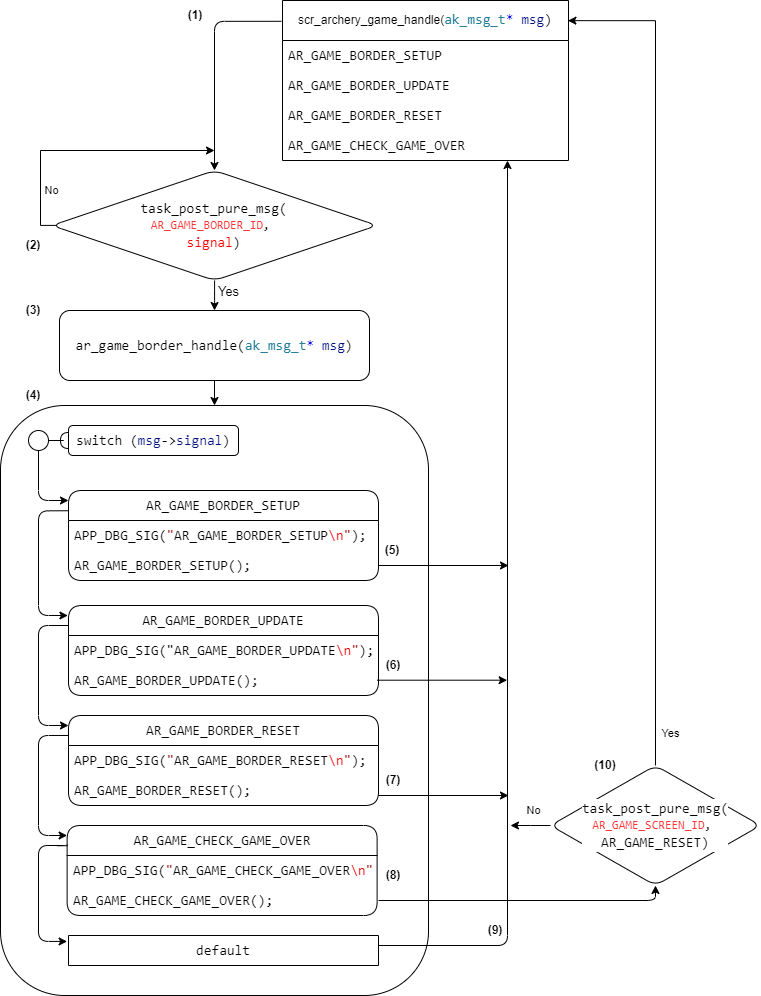
(6) AR\_GAME\_BANG\_UPDATE là signal được sử dụng để cập nhật hình ảnh theo thời gian cho đối tượng.

(7) AR\_GAME\_BANG\_RESET là signal được sử dụng để reset các giá trị ban đầu cho đối tượng trước khi thoát game.

(8) Sau khi xử lý xong yếu cầu theo signal thì sẽ có lệnh “break” để kết thúc xử lý và quay lại scr\_archery\_game\_handle() để đợi xử lý tast\_post tiếp theo.

## Ranh giới an toàn (Border)

Ranh giới an toàn (Border) là đối tượng bất động trên màn hình cũng chính là điều khiện thua của game. Border vạch ra vùng giới hạn mục tiêu của chúng ta là ngăn không cho Thiên thạch (Meteoroid) chạm vào Border. Nên ngoài ra, đối tượng còn đảm nhiệm nhiệm vụ kiểm tra điều kiện thua.



Hình 3.5‑1: Border - sequence

**Ghi chú:**

(1) Hàm scr\_archery\_game\_handle(ak\_msg\_t\* msg) là hàm xử lý màn hình (Screen) có thể gửi tới 3 signal:

* AR\_GAME\_BORDER\_SETUP
* AR\_GAME\_BORDER\_UPDATE
* AR\_GAME\_BORDER\_RESET
* AR\_GAME\_CHECK\_GAME\_OVER

(2) Hàm gửi pure message thông qua “task\_id” và “signal” của đối tượng Ranh giới an toàn (Border).

(3) Hàm ar\_game\_border\_handle (ak\_msg\_t\* msg) tiếp nhận message chứa “signal” và xử lý.

(4) Hàm “switch” nhận tín hiệu “signal” rồi thực thiện lệnh ứng với “signal” đó.

(5) AR\_GAME\_BORDER\_SETUP là signal được sử dụng để setup các giá trị bạn đầu cho đối tượng.

(6) AR\_GAME\_BORDER\_UPDATE là signal được sử dụng để cập nhật hình ảnh theo thời gian cho đối tượng.

(7) AR\_GAME\_BORDER\_RESET là signal được sử dụng để reset các giá trị ban đầu cho đối tượng trước khi thoát game.

(8) AR\_GAME\_CHECK\_GAME\_OVER là signal được sử dụng để kiểm tra điều kiện thua của game.

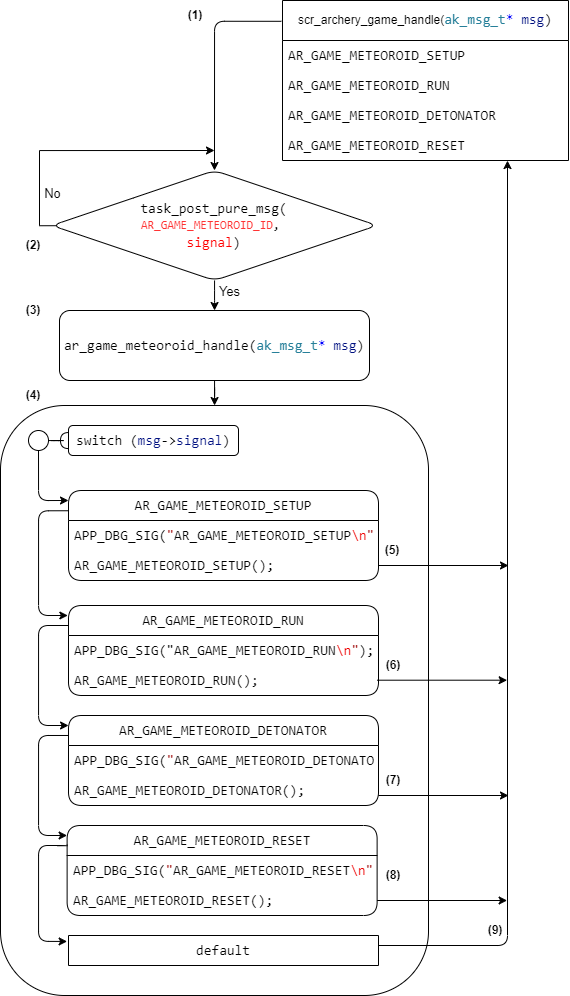
(9) Sau khi xử lý xong yếu cầu theo signal thì sẽ có lệnh “break” để kết thúc xử lý và quay lại scr\_archery\_game\_handle() để đợi xử lý tast\_post tiếp theo.

(10) Sau khi kiểm tra ở bước (8) nếu sai (No) thì về theo bước (9). Nếu kiểm tra đúng (Yes) thì sẽ gửi pure message về scr\_archery\_game\_handle(ak\_msg\_t\* msg) với signal:

* AR\_GAME\_RESET

## Thiên thạch (Meteoroid)

Thiên thạch (Meteoroid) là đối tượng phản diện trong game. Số lượng Thiên thạch suất hiện đồng thời trên màn hình là 5. Mỗi khi Thiên thạch bị phá hủy thì sẽ cộng 10 điểm vào score rồi Thiên thạch sẽ được tái tạo ở 1 vị trí ngẫu nghiên trong 1 khoãng xác định. Với mỗi 200 điểm kiếm được thì Thiên thạch tăng tốc độ lên 1 bật. Nếu Thiên thạch chạm vào vùng an toàn thì game sẽ kết thúc.



Hình 3.6‑1: Meteoroid - sequence

**Ghi chú:**

(1) Hàm scr\_archery\_game\_handle(ak\_msg\_t\* msg) là hàm xử lý màn hình (Screen) có thể gửi tới 3 signal:

* AR\_GAME\_METEOROID\_SETUP
* AR\_GAME\_METEOROID\_RUN
* AR\_GAME\_METEOROID\_DETONATOR
* AR\_GAME\_METEOROID\_RESET

(2) Hàm gửi pure message thông qua “task\_id” và “signal” của đối tượng Thiên thạch (Meteoroid).

(3) Hàm ar\_game\_meteoroid\_handle (ak\_msg\_t\* msg) tiếp nhận message chứa “signal” và xử lý.

(4) Hàm “switch” nhận tín hiệu “signal” rồi thực thiện lệnh ứng với “signal” đó.

(5) AR\_GAME\_METEOROID\_SETUP là signal được sử dụng để setup các giá trị bạn đầu cho đối tượng.

(6) AR\_GAME\_METEOROID\_RUN là signal được sử dụng để cập nhật hình ảnh theo thời gian cho đối tượng.

(7) AR\_GAME\_METEOROID\_DETONATOR là signal được sử dụng để kiểm tra điều kiện va chạm của Thiên thạch (Meteoroid) và Mũi tên (Arrow).

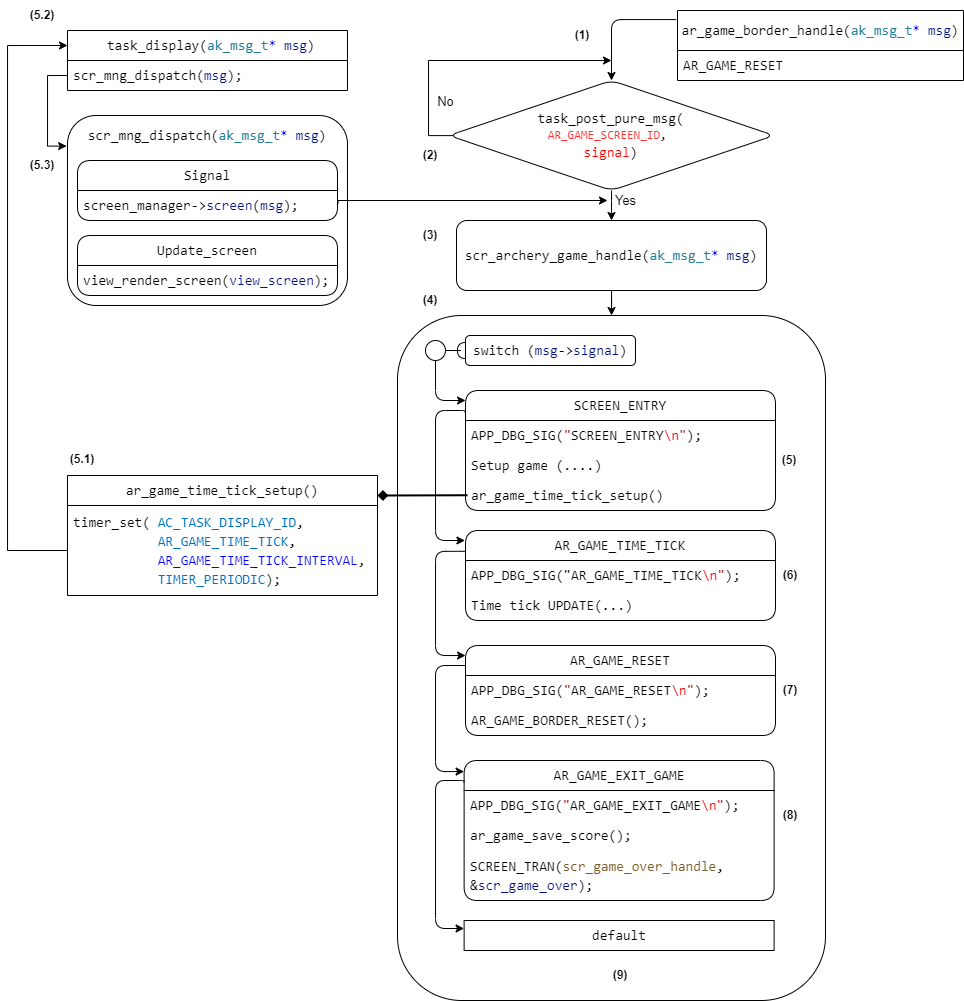
(8) AR\_GAME\_METEOROID\_RESET là signal được sử dụng để reset các giá trị ban đầu cho đối tượng trước khi thoát game.

(9) Sau khi xử lý xong yếu cầu theo signal thì sẽ có lệnh “break” để kết thúc xử lý và quay lại scr\_archery\_game\_handle() để đợi xử lý tast\_post tiếp theo.

## Màn hình (Screen)

Màn hình game là task chính của game. Task bao gồm 4 tác vụ chính là

* Screen Entry dùng để setup các đối tượng game và tạo timer.
* Timer: thực hiện tính toán và cập nhập màn hình theo chu kỳ.
* Reset: thực hiện reset các đối tượng.
* Exit: thực hiện lưu điểm và chuyển màn hình.



Hình 3.7‑1: Screen - sequence

**Ghi chú:**

(1) Hàm ar\_game\_border\_handle(ak\_msg\_t\* msg) là hàm xử lý Ranh giới an toàn (Border) có thể gửi tới signal🡪 AR\_GAME\_RESET thông báo thua game.

(2) Hàm gửi pure message thông qua “task\_id” và “signal” của đối tượng Màn hình (Screen).

(3) Hàm scr\_archery\_game (ak\_msg\_t\* msg) tiếp nhận message chứa “signal” và xử lý.

(4) Hàm “switch” nhận tín hiệu “signal” rồi thực thiện lệnh ứng với “signal” đó.

(5) SCREEN\_ENTRY là signal được sử dụng để setup các giá trị bạn đầu cho đối tượng và gửi các task\_post “setup” cho các đối tượng khác đồng thời khởi tạo Timer🡪 Time Tick.

(5.1) ar\_game\_time\_tick\_setup() khởi tạo Timer dạng periodic, dùng “task\_display”, signal “AR\_GAME\_TIME\_TICK” vởi thời gian timer là AR\_GAME\_TIME\_TICK\_INTERVAL.

(5.2) Từ task\_display(ak\_msg\_t\* msg) sang hàm scr\_mng\_dispatch(msg)

(5.3) Hàm scr\_mng\_dispatch(ak\_msg\_t\* msg) thì signal gửi sang Màn hình đang hiển thị hiện tại và đồng thời vẽ lại màn hình.

(6) AR\_GAME\_TIME\_TICK là signal được sử dụng gửi các task\_post “update” trạng thái cho các đối tượng khác.

(7) AR\_GAME\_RESET là signal được sử dụng để gửi các task\_post “reset” cho các đối tượng khác.

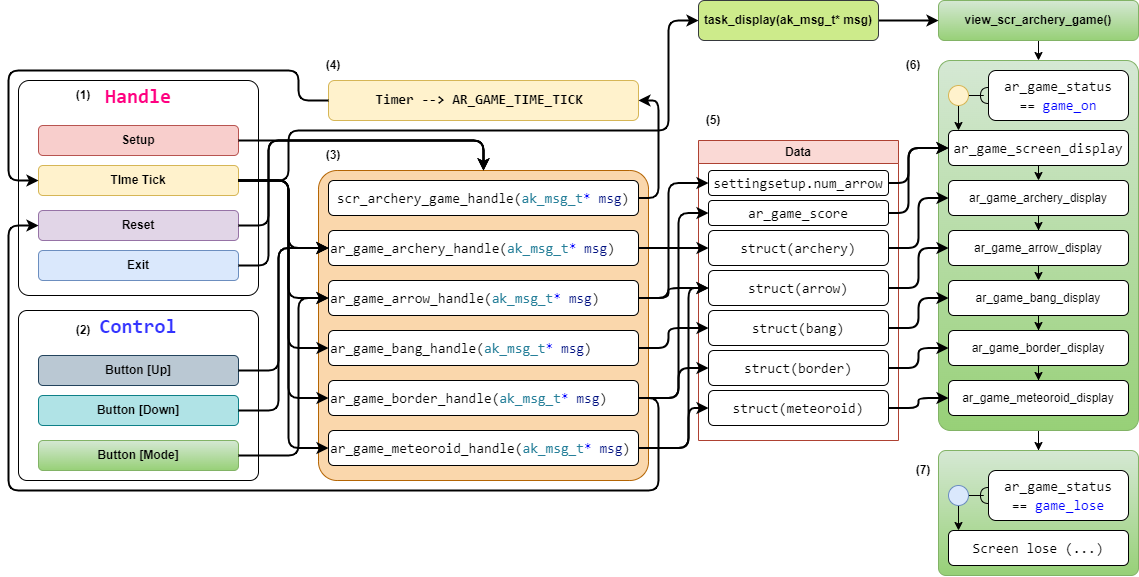
(8) AR\_GAME\_EXIT\_GAME là signal được sử dụng để lưu điểm và chuyển màn hình.

(9) Sau khi xử lý xong yếu cầu theo signal thì sẽ có lệnh “break” để kết thúc xử lý và đợi “signal” tiếp theo được gửi tới.

# Màn hình hiển thị

## Nguyên lý

Màn hình hiển thị là nơi biểu thị các data đã qua các bước tính toán lên màn hình. Màn hình được cập nhật thông qua 1 timer tần suất cập nhật càng nhanh thì game càng mượt.



Hình 4.1‑1: Xử lý hiển thị và cập nhật màn hình

**Ghi chú:**

(1) Handle: xử lý các tác vụ trong game.

* Setup: khởi tạo các giá trị đầu cho các đối tượng (tất cả đối tượng)
* Time Tick: cập nhật trạng thái đối tượng và cập nhật màn hình.
* Reset: khởi tạo lại giá trị các đối tượng trước khi thoát game.
* Exit: lưu và thoát game.

(2) Control: các tác vụ thao tác nút bấm của game.

* Button [Up]: gửi message đến đối tượng Cung tên (Archery).
* Button [Down]: gửi message đến đối tượng Cung tên (Archery).
* Button [Mode]: gửi message đến đối tượng Mũi tên (Arrow).

(3) Các hàm xử lý tiếp nhận signal của các đối tượng.

(4) Trong hàm xử lý scr\_archery\_game\_handle(ak\_msg\_t\* msg) có khởi tạo 1 timer để tạo Time Tick.

(5) Sau khi xử lý qua các đối tượng thì chúng ta có các Data của từng đối tượng.

(6) Các Data được xử lý và hiển thị ra màn hình ở hàm view\_scr\_archery\_game(), và hình ảnh này được cập nhật theo tầng số phụ thuộc vào Time Tick thông qua task\_display.

(7) Trong hàm view\_scr\_archery\_game() hiển thi còn được kiểm soát bằng trạng thái của game nhờ biến “ar\_game\_status”.

## Bitmap và animation.

Mỗi đối tượng trong game sẽ có Bitmap của riêng đối tượng đó kết hợp với Data được xử lý để hiện thị đối tượng ra màn hình.

Để game thêm độ hấp dẫn thì animation giúp đối tượng có thêm tính linh động. Và animation là sự kết hợp của các bitmap của cùng 1 đối tượng theo thời gian.