

epcbtech / archery-game Public

A game built on AK Embedded Base Kit to learn, understand and practice Event-Driven Programming, UML, Task, Signal, Timer, State-machine,...

MIT license

3 stars 3 forks Activity

Star

Notifications

<> Code

Issues

Pull requests

Actions

Projects

Security

Insights

main

Go to file

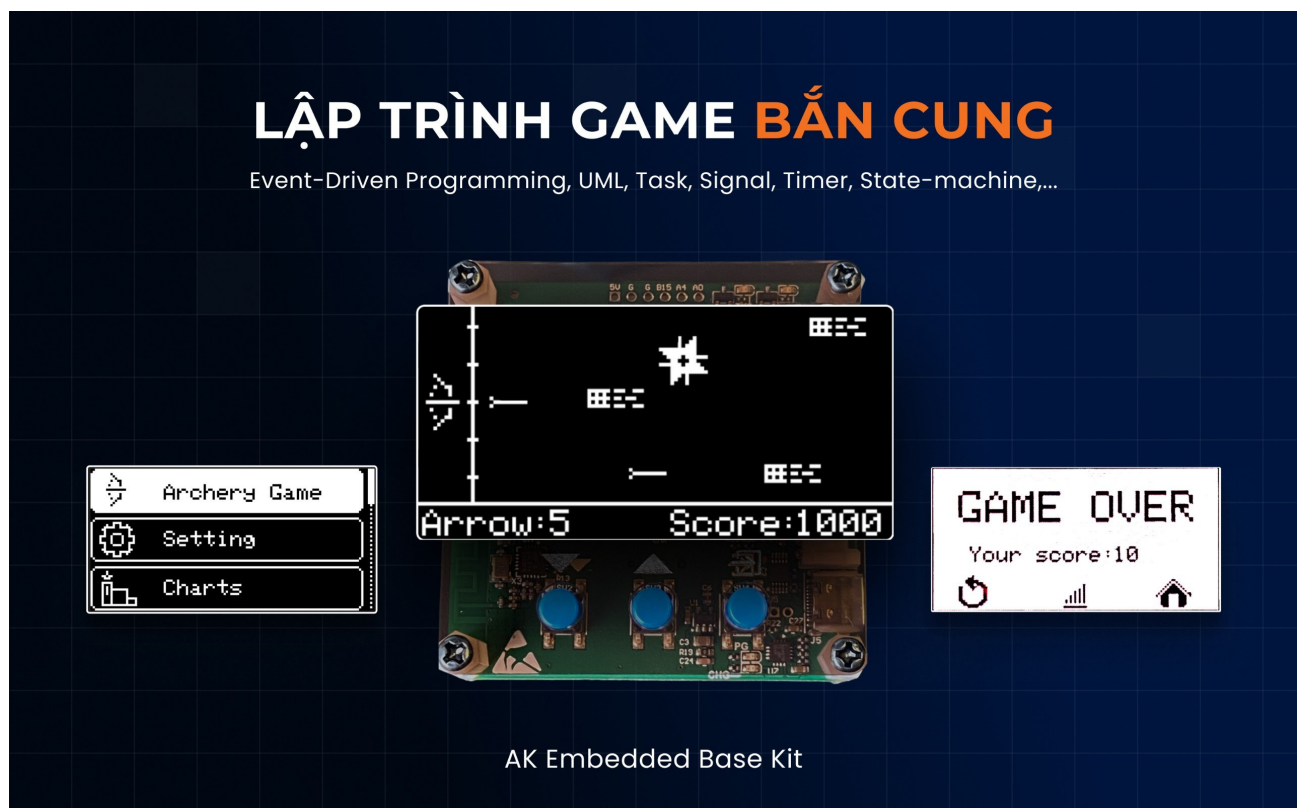
epcbtech [IMP] Update README.md (#1) ...

last week 6

[View code](#)

README.md

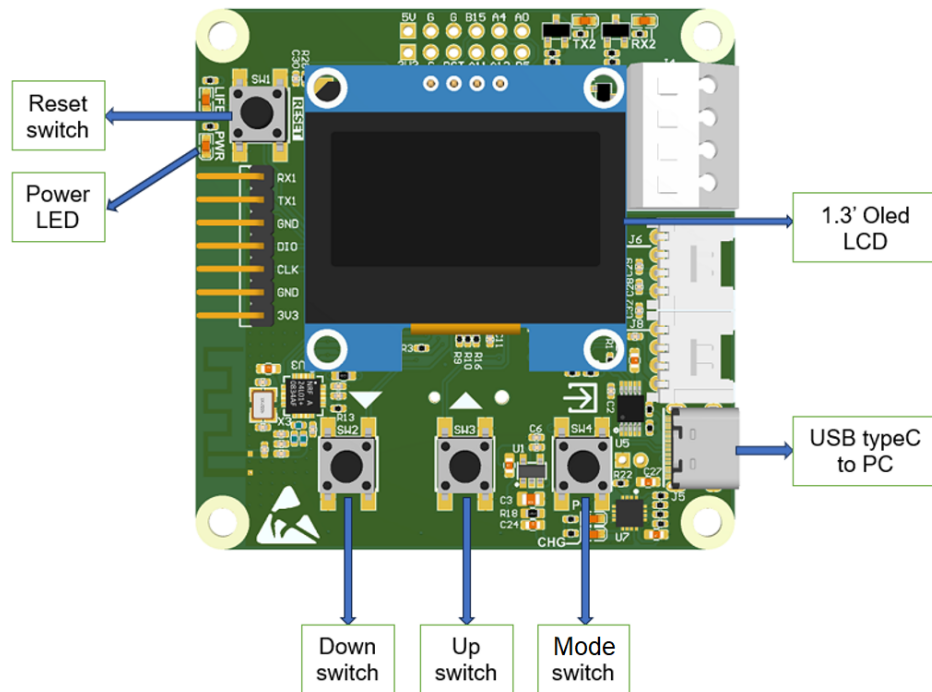
LẬP TRÌNH GAME BẮN CUNG





I. Giới thiệu [↗](#)

1.1 Phần cứng [↗](#)



Hình 1: AK Embedded Base Kit - STM32L151

AK Embedded Base Kit là một công cụ đánh giá dành cho các bạn học phần mềm nhúng 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,...

1.2 Mô tả trò chơi và đối tượng [↗](#)

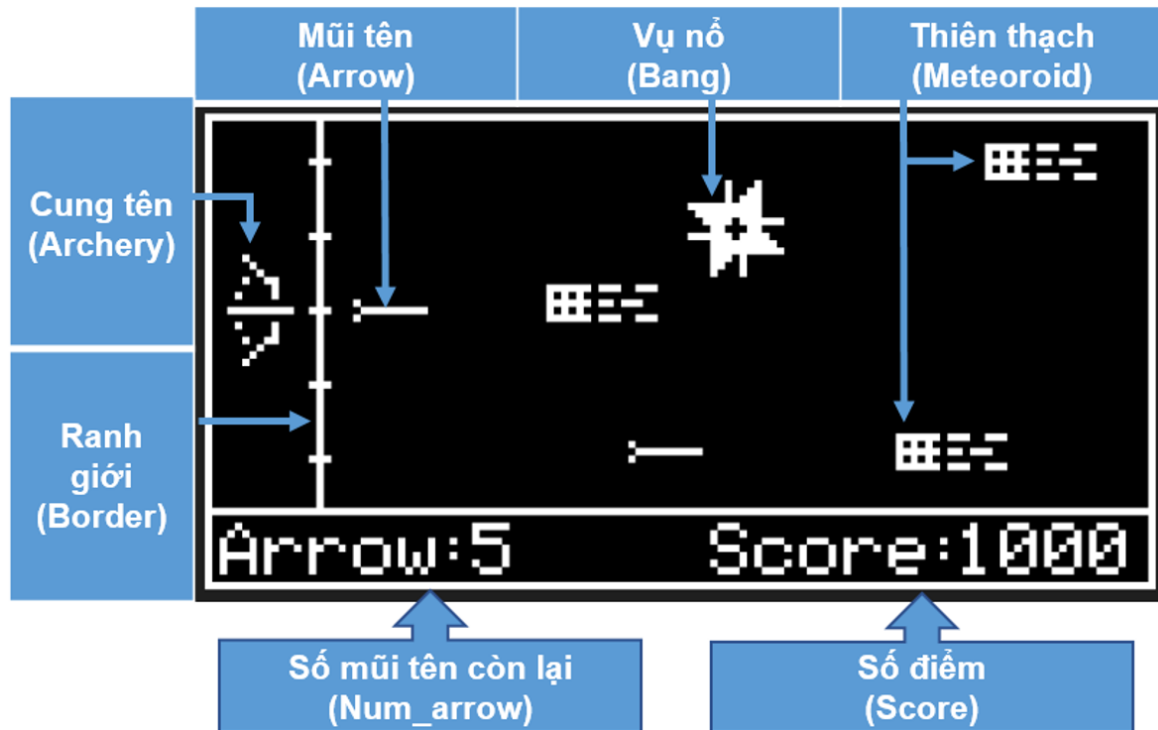
Phần mô tả sau đây về "Archery game" là giải thích cách chơi và cơ chế xử lý của trò chơi. Nhằm phục vụ cho việc thiết kế và phát triển trò chơi về sau.



Hình 2: 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 3: Màn hình game play và các đối tượng

Các đối tượng (Object) trong game: [🔗](#)

Đối tượng	Tên đối tượng	Mô tả
Cung tên	Archery	Di chuyển lên/xuống để chọn vị trí bắn ra mũi tên
Mũi tên	Arrow	Bắn ra từ cung tên, dùng để phá hủy thiên thạch
Vụ nổ	Bang	Hiệu ứng xuất hiện khi thiên thạch bị phá hủy
Ranh giới	Border	Vùng an toàn phải bảo vệ không cho thiên thạch rơi vào
Thiên thạch	Meteoroid	Vật thể bay về phía cung tên với tốc độ tăng dần, có khả năng phá hủy ranh giới

(*) Trong phần còn lại của tài liệu sẽ dùng tên của các đối tượng để đề cập đến đối tượng.

Cách chơi game: [↗](#)

- Trong trò chơi này bạn sẽ điều khiển Archery, di chuyển **lên/xuống** bằng hai nút **[Up]/[Down]**, để chọn vị trí **bắn ra Arrow**.
- Khi nhấn nút **[Mode]** Arrow sẽ được bắn ra, nhằm phá hủy các Meteoroid đang bay đến.
- Mục tiêu trò chơi là kiếm được càng nhiều điểm càng tốt, trò chơi sẽ kết thúc khi có Meteoroid chạm vào Border.

Cơ chế hoạt động: [↗](#)

- **Cách tính điểm:** Điểm được tính bằng số lượng Meteoroid bị phá hủy. Mỗi Meteoroid bị phá hủy tương ứng với 10 điểm. Số điểm tích lũy được sẽ hiển thị ở góc dưới bên phải màn hình.
- **Độ khó:** Mỗi khi tích lũy được 200 điểm, tốc độ bay của Meteoroid sẽ tăng lên một cấp độ. Độ khó ban đầu có thể cài đặt trong phần **setting**.
- **Giới hạn của Arrow:** Khi bắn thì số lượng Arrow hiện có sẽ giảm đi tương ứng số lượng Arrow đang bay, nếu Arrow hiện có giảm về "0" thì không thể bắn được và sẽ có âm thanh báo. Số lượng Arrow hiện có sẽ được hồi lại khi phá hủy được Meteoroid hoặc Arrow bay hết màn hình game. Số lượng Arrow được hiển thị ở góc dưới bên trái màn hình và có thể thay đổi trong phần **setting**.
- **Animation:** Để trò chơi thêm phần sinh động thì các đối tượng sẽ có thêm hoạt ảnh lúc di chuyển.
- **Kết thúc trò chơi:** Khi Meteoroid chạm vào Border, trò chơi sẽ kết thúc. Các đối tượng sẽ được reset và số điểm sẽ được lưu. Bạn sẽ vào màn hình "Game Over" với 3 lựa chọn là:
 - **Restart:** chơi lại.
 - **Charts:** vào xem bảng xếp hạng.
 - **Home:** về lại menu game.



Hình 4: Màn hình Game_over

II. Thiết kế - ARCHERY GAME

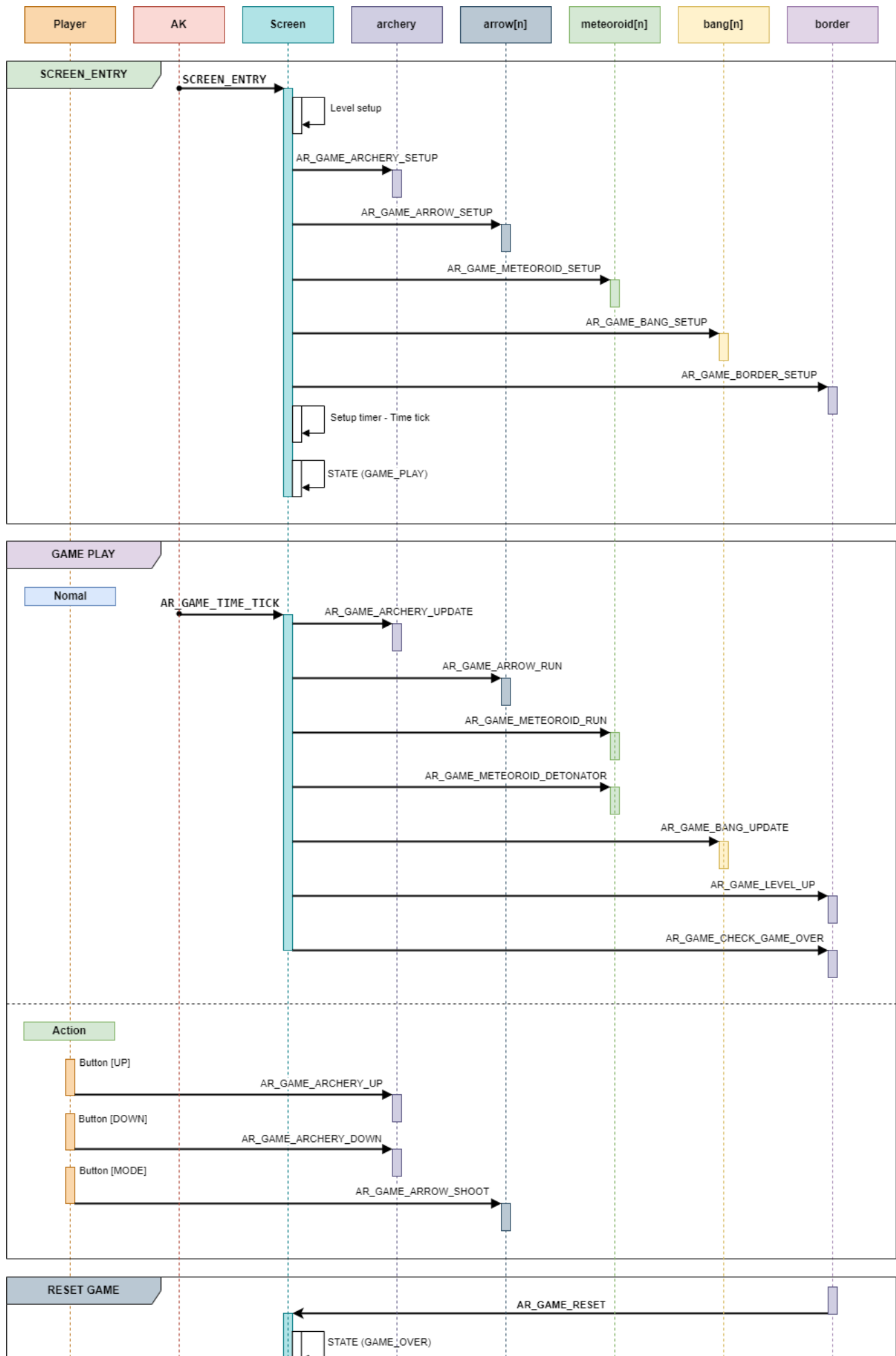
Các khái niệm trong event-driven:

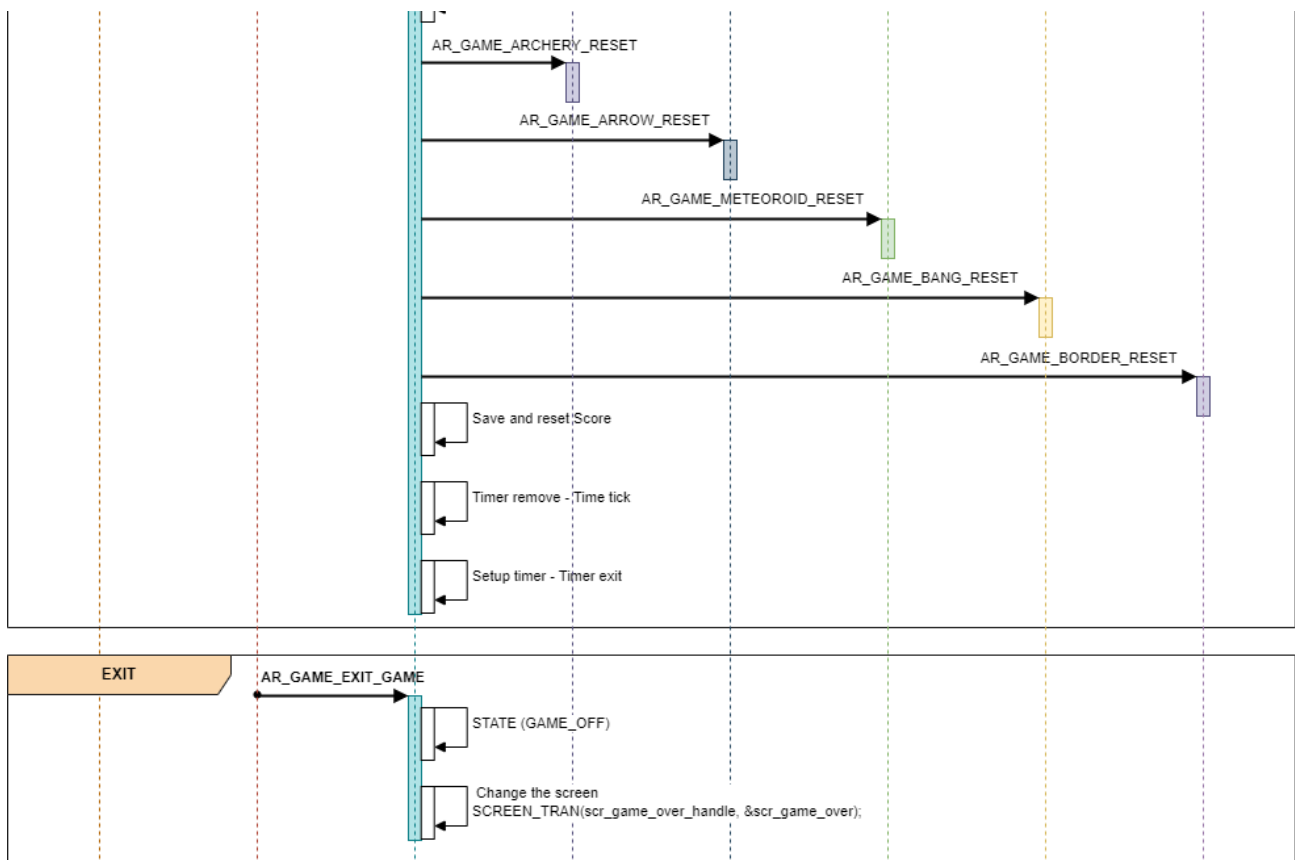
- **Event Driven:** Nôn na là một hệ thống gửi thư (gửi message) để thực thi các công việc. Trong đó, Task đóng vai trò là người nhận thư, Signal đại diện cho nội dung công việc. Task & Signal nền tảng của một hệ Event Driven.
- **Task:** Thông thường mỗi Task sẽ nhận một nhóm công việc nào đó, ví dụ: quản lý state-machine, quản lý hiển thị của màn hình, quản lý việc cập nhật phần mềm, quản lý hệ thống watchdog ...
- **Message:** Được chia làm 2 loại chính, Message chỉ chứa Signal, hoặc vừa chứa Signal và Data. Message tương đương với Signal.
- **Handler:** Chỗ thực thi một công việc nào đó thì gọi là Handler.

Chi tiết các khái niệm các bạn tham khảo tại bài viết: [AK Embedded Base Kit - STM32L151 - Event Driven: Task & Signal](#)

2.1 Sơ đồ trình tự

Sơ đồ trình tự được sử dụng để mô tả trình tự của các Message và luồng tương tác giữa các đối tượng trong một hệ thống.





Hình 5: The sequence diagram

Ghi chú: [↗](#)

SCREEN_ENTRY: Cài đặt các thiết lập ban đầu cho đối tượng trong game.

- **Level setup:** Thiết lập thông số cấp độ cho game.
- **AR_GAME_ARCHERY_SETUP:** Thiết lập thông số ban đầu cho đối tượng Archery
- **AR_GAME_ARROW_SETUP:** Thiết lập thông số ban đầu cho các đối tượng Arrow
- **AR_GAME_METEOROID_SETUP:** Thiết lập thông số ban đầu cho các đối tượng Meteoroid
- **AR_GAME_BANG_SETUP:** Thiết lập thông số ban đầu cho các đối tượng Bang
- **AR_GAME_BORDER_SETUP:** Thiết lập thông số ban đầu cho đối tượng Border
- **Setup timer - Time tick:** Khởi tạo Timer - Time tick cho game.
- **STATE (GAME_ON):** Cập nhật trạng thái game -> GAME_ON

GAME PLAY: Quá trình hoạt động của game.

GAME PLAY - Normal: Game hoạt động ở trạng thái bình thường.

- **AR_GAME_TIME_TICK:** Signal do Timer - Time tick gửi đến.
- **AR_GAME_ARCHERY_UPDATE:** Cập nhật trạng thái Archery.
- **AR_GAME_ARROW_RUN:** Cập nhật di chuyển của các Arrow theo thời gian.

- **AR_GAME_METEOROID_RUN:** Cập nhật di chuyển của các Meteoroid theo thời gian.
- **AR_GAME_METEOROID_DETONATOR:** Kiểm tra các Meteoroid có bị Arrow phá hủy.
- **AR_GAME_BANG_UPDATE:** Cập nhật hoạt ảnh vụ nổ theo thời gian
- **AR_GAME_BORDER_UPDATE:** Kiểm tra số điểm hiện tại để cập nhật tăng độ khó game.
- **AR_GAME_CHECK_GAME_OVER:** Kiểm tra Meteoroid chạm vào Border. Nếu chạm vào thì gửi Signal - **AR_GAME_RESET** đến **Screen**.

GAME PLAY - Action: Game hoạt động ở trạng thái có tác động của các nút nhấn.

- **AR_GAME_ARCHERY_UP:** Player nhấn nút **[Up]** điều khiển Archery di chuyển lên.
- **AR_GAME_ARCHERY_DOWN:** Player nhấn nút **[Down]** điều khiển Archery di chuyển xuống.
- **AR_GAME_ARROW_SHOOT:** Player nhấn nút **[Mode]** điều khiển Archery bắn Arrow ra.

RESET GAME: Quá trình cài đặt lại các thông số trước khi thoát game.

- **STATE (GAME_OVER):** Cập nhật trạng thái game -> **GAME_OVER**
- **AR_GAME_RESET:** Signal cài đặt lại game do Border gửi đến.
- **AR_GAME_ARCHERY_RESET:** Cài đặt lại đối tượng Archery trước khi thoát.
- **AR_GAME_ARROW_RESET:** Cài đặt lại đối tượng Arrow trước khi thoát.
- **AR_GAME_METEOROID_RESET:** Cài đặt lại đối tượng Meteoroid trước khi thoát.
- **AR_GAME_BANG_RESET:** Cài đặt lại đối tượng Bang trước khi thoát.
- **AR_GAME_BORDER_RESET:** Cài đặt lại đối tượng Border trước khi thoát.
- **Save and reset Score:** Lưu số điểm hiện tại và Cài đặt lại.
- **Timer remove - Timer tick:** Xóa Timer - Time tick
- **Setup timer - Timer exit:** Tạo 1 timer one shot để thoát game. Nhằm tạo ra một khoảng delay cho người chơi có thể nhận thức được là mình đã game over trước khi chuyển sang màn hình thông báo game over.

EXIT: Thoát khỏi game và chuyển sang màn hình Game Over.

- **AR_GAME_EXIT:** Signal do Timer exit gửi đến.
- **STATE (GAME_OFF):** Cập nhật trạng thái game -> **GAME_OFF**
- **Change the screen - SCREEN_TRAN(scr_game_over_handle, &scr_game_over):** Chuyển màn hình sang màn hình Game Over.

2.2 Chi tiết

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 suông sẽ hơn.

2.2.1 Thuộc tính đối tượng

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

- Giúp xác định rõ thông tin về đối tượng trong game.
- Giúp xác định cấu trúc dữ liệu phù hợp để lưu trữ thông tin của đối tượng.
- 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.

Trạng thái của một đối tượng được biểu diễn bởi các **thuộc tính**. Trong trò chơi này các đối tượng có các thuộc tính cụ thể là:

- **visible**: Quy định hiển thị của đối tượng.
- **x, y**: Quy định vị trí của đối tượng trên màn hình.
- **action_image**: Quy định hoạt ảnh tạo animation.

Ví dụ:

```
typedef struct {  
    bool visible;  
    uint32_t x, y;  
    uint8_t action_image;  
} ar_game_archery_t;  
  
extern ar_game_archery_t archery;
```



Áp dụng struct cho các đối tượng:

struct	Các biến
ar_game_archery_t	archery
ar_game_arrow_t	arrow[MAX_NUM_ARROW]
ar_game_bang_t	bang[NUM_BANG]
ar_game_border_t	border
ar_game_meteoroid_t	meteoroid[NUM_METEORIDS]

(*) Các đối tượng có số lượng nhiều thì sẽ được khai báo dạng mảng.

Các biến quan trọng:

- **ar_game_score:** Điểm của trò chơi.
- **ar_game_status:** Trạng thái quả trò chơi.
 - GAME_OFF: Tắt .
 - GAME_ON: Bật.
 - GAME_OVER: Đã thua.
- **ar_game_setting_t settingsetup :** Cấu hình cấp độ của trò chơi.
 - settingsetup.silent : Bật/tắt chế độ im lặng.
 - settingsetup.num_arrow : Cấu hình số lượng mũi tên.
 - settingsetup.arrow_speed : Cấu hình tốc độ mũi tên.
 - settingsetup.meteoroid_speed : Cấu hình tốc độ của thiên thạch.

2.2.2 Task

Trong lập trình event-driven, task là một đơn vị độc lập đảm nhiệm một nhóm công việc cụ thể. Khi hệ thống scheduler tìm thấy message liên quan đến task trong hàng đợi, hệ thống sẽ gọi hàm thực thi của task để xử lý message được gửi đến. Một số tác dụng quan trọng của task:

- **Xử lý sự kiện:** Task được sử dụng để xử lý các message được bắn đến khi có sự kiện 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 sẽ 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 sự kiện 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. Điều này giúp tăng tính sạch sẽ, dễ đọc.
- **Phân cấp nhiệm vụ:** Task level cho phép sắp xếp trình tự ưu tiên xử lý các message của task ở trong hàng đợi của hệ thống. Trong game các task level của game đều là 4 nên task nào được gọi trước sẽ xử lý trước.

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

Hình 6: Bảng Task của các đối tượng

2.2.3 Signal [↗](#)

Signal là một cơ chế truyền thông tin giữa các thành phần trong hệ thống event-driven. Khi một sự kiện xảy ra, nó có thể gửi một signal để thông báo cho các thành phần khác về việc xảy ra của sự kiện đó.

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

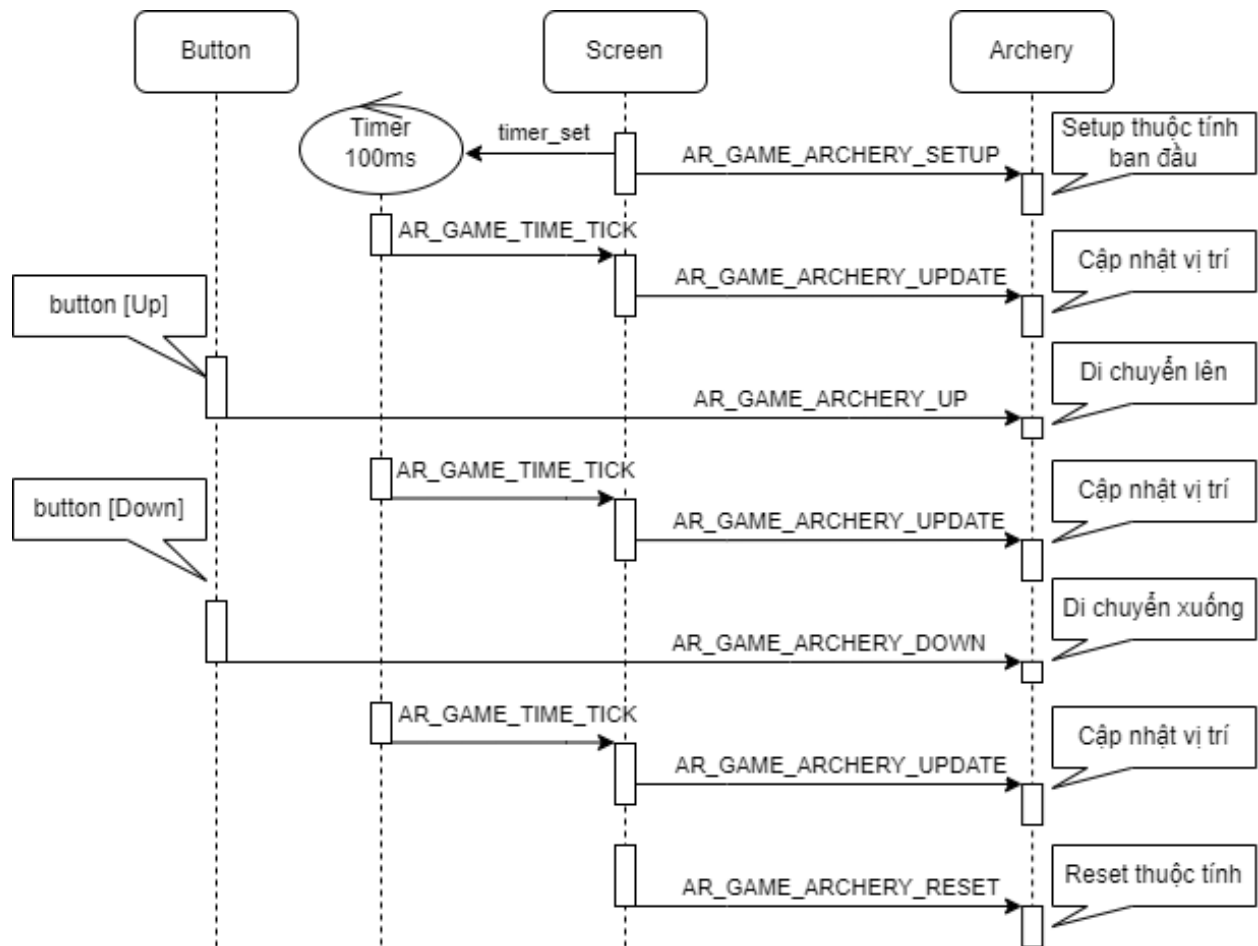
Hình 7: Bảng Signal của từng Task

(*) Tác dụng của các Signal trong game: xem tại Ghi chú - Hình 5

III. Hướng dẫn chi tiết code trong đối tượng [↗](#)

3.1 Archery. [↗](#)

Sequence diagram:



Hình 8: Archery sequence

Tóm tắt nguyên lý: Archery sẽ nhận Signal thông được gửi từ 2 nguồn là Screen và Button. Quá trình xử lý của đối tượng phần làm 3 giai đoạn:

- **Giai đoạn 1:** Bắt đầu game, cài đặt các thông số của Archery như vị trí và hình ảnh.
- **Giai đoạn 2:** Chơi game, trong giai đoạn này chia làm 2 hoạt động là:
 - Cập nhật: Screen gửi Signal cập nhật cho Archery mỗi 100ms để cập nhật trạng thái hiện tại của Archery.
 - Hành động: Button gửi Signal di chuyển lên/xuống cho Archery mỗi khi nhấn nút.
- **Giai đoạn 3:** Kết thúc game, thực hiện cài đặt lại trạng thái của Archery trước khi thoát game.

Code:

Trong code bạn có thể dùng macro để thay thế hàm void trong nhiều trường hợp.

```

#define TEN_DOAN_CODE()
do { \
    /*code*/ \
} while(0);
  
```

Khai báo: Thư viện, struct và biến.

```
#include "ar_game_archery.h"

ar_game_archery_t archery;
static uint32_t archery_y = AXIS_Y_ARCHERY;
```



AR_GAME_ARCHERY_SETUP() là một macro được dùng định nghĩa để cài đặt trạng thái ban đầu của trò chơi bắn cung. Nó đặt các giá trị của biến archery và sử dụng các hằng số được định nghĩa trước đó để thiết lập tọa độ, màu sắc và hình ảnh của cung.

```
#define AR_GAME_ARCHERY_SETUP() \
do { \
    archery.x = AXIS_X_ARCHERY; \
    archery.y = AXIS_Y_ARCHERY; \
    archery.visible = WHITE; \
    archery.action_image = 1; \
} while (0);
```



AR_GAME_ARCHERY_UP() là một macro được sử dụng để di chuyển cung lên trên. Nó giảm giá trị của archery_y bằng một giá trị STEP_ARCHERY_AXIS_Y và kiểm tra nếu giá trị mới bằng 0, nó được gán lại là 10.

```
#define AR_GAME_ARCHERY_UP() \
do { \
    archery_y -= STEP_ARCHERY_AXIS_Y; \
    if (archery_y == 0) {archery_y = 10;} \
} while(0);
```



AR_GAME_ARCHERY_DOWN() là một macro được sử dụng để di chuyển cung xuống dưới. Nó tăng giá trị của archery_y bằng một giá trị STEP_ARCHERY_AXIS_Y và kiểm tra nếu giá trị mới vượt quá 50, nó được gán lại là 50.

```
#define AR_GAME_ARCHERY_DOWN() \
do { \
    archery_y += STEP_ARCHERY_AXIS_Y; \
    if (archery_y > 50) {archery_y = 50;} \
} while(0);
```



AR_GAME_ARCHERY_RESET() là một macro được sử dụng để đặt lại trạng thái ban đầu của trò chơi bắn cung. Nó đặt lại giá trị của archery, archery_y và làm cho cung trở nên không hiển thị.

```
#define AR_GAME_ARCHERY_RESET() \
do { \
    archery.x = AXIS_X_ARCHERY; \
    archery.y = AXIS_Y_ARCHERY; \
    archery.visible = BLACK; \
    archery_y = AXIS_Y_ARCHERY; \
} while(0);
```



Hàm `ar_game_archery_handle()` là một hàm xử lý các thông điệp (messages) liên quan đến trò chơi cung bắn. Nó chứa một câu lệnh switch-case để xử lý các thông điệp khác nhau. Các thông điệp được gửi đến hàm này thông qua một tham số `msg` có kiểu dữ liệu `ak_msg_t`. Mỗi case trong switch-case xử lý một thông điệp cụ thể.

```
void ar_game_archery_handle(ak_msg_t* msg) {
    switch (msg->sig) {
        case AR_GAME_ARCHERY_SETUP: {
            APP_DBG_SIG("AR_GAME_ARCHERY_SETUP\n");
            AR_GAME_ARCHERY_SETUP();
        }
        break;

        case AR_GAME_ARCHERY_UPDATE: {
            APP_DBG_SIG("AR_GAME_ARCHERY_UPDATE\n");
            archery.y = archery_y;
        }
        break;

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

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

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

        default:
            break;
    }
}
```



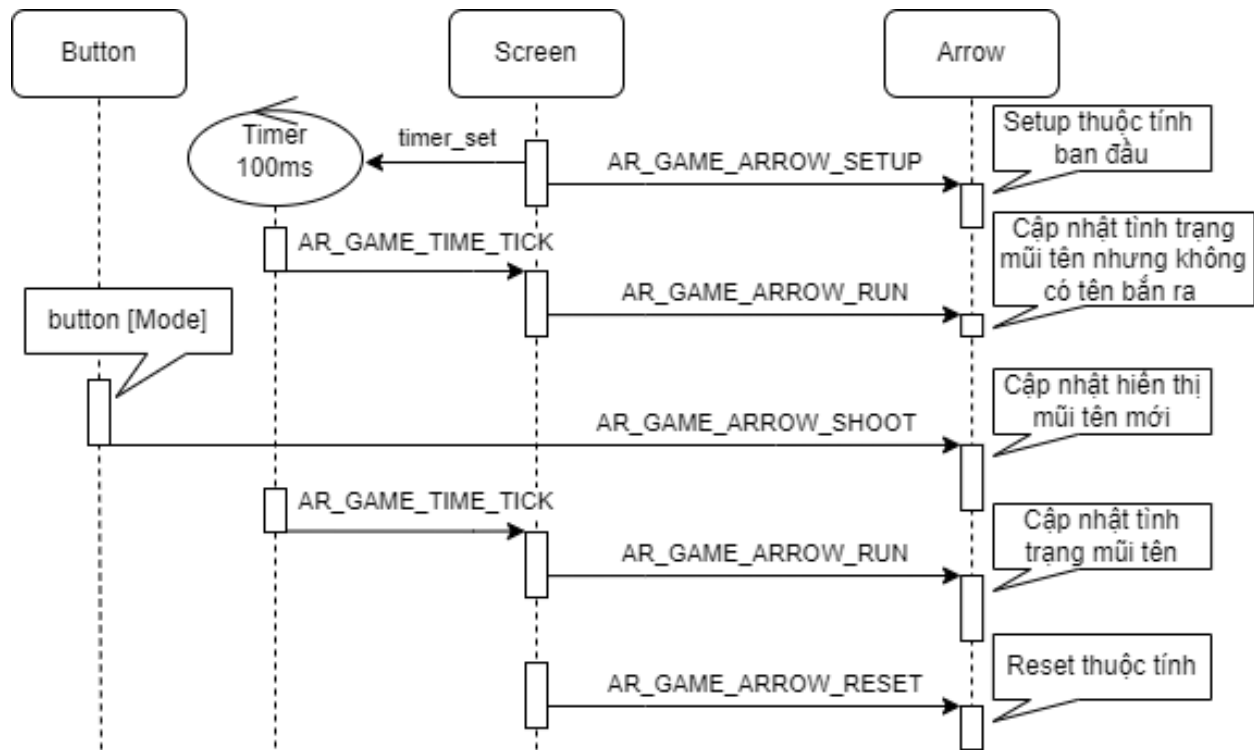

```

    }
}

```

3.2 Arrow [↗](#)

Sequence diagram:

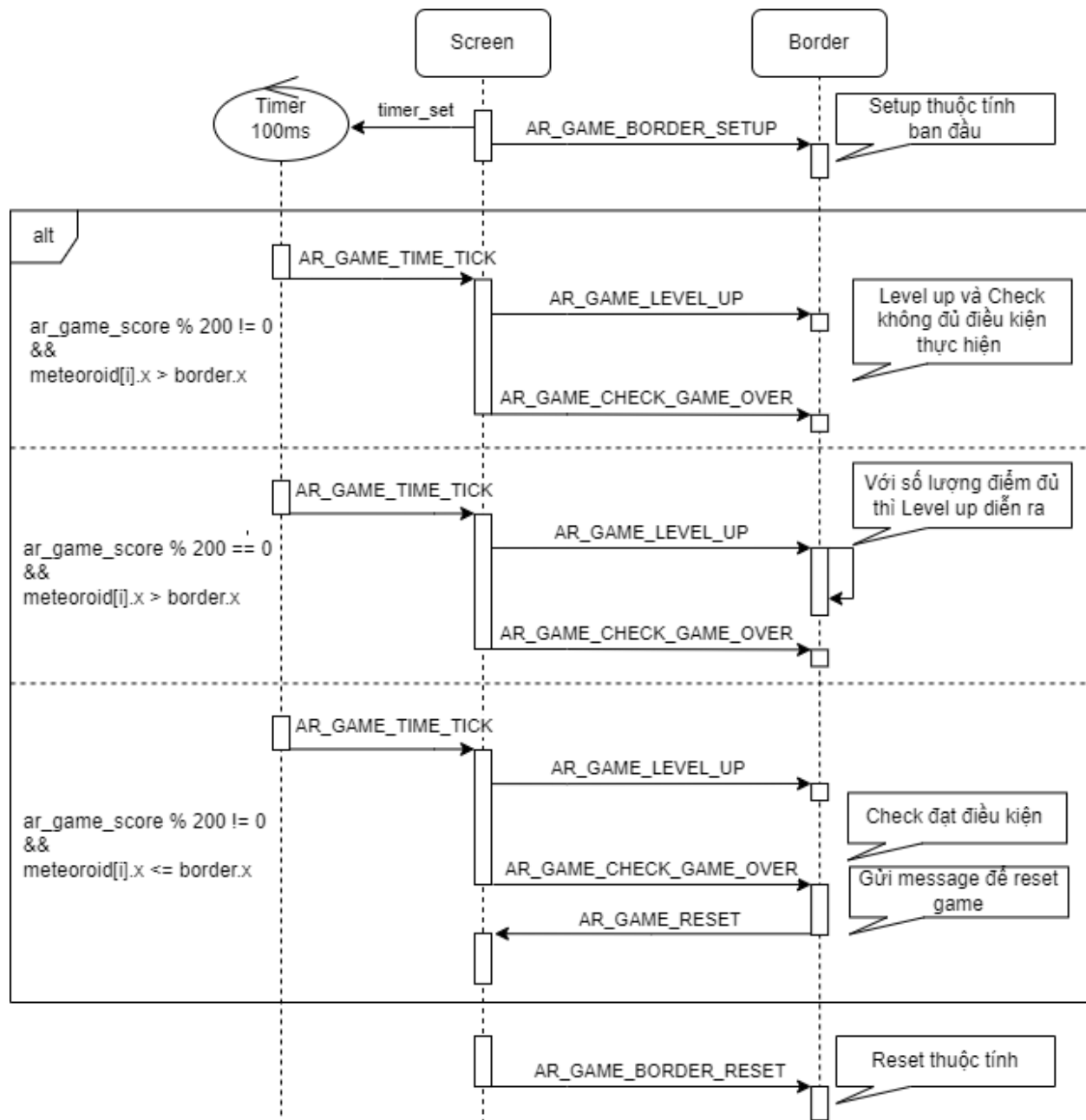


Hình 9: Arrow sequence

Tóm tắt nguyên lý: Arrow sẽ nhận Signal thông được gửi từ 2 nguồn là Screen và Button. Quá trình xử lý của đối tượng phần làm 3 giai đoạn:

- **Giai đoạn 1:** Bắt đầu game, cài đặt các thông số của Arrow. Tất cả Arrow vào trạng thái lặn, không hiển thị trên màn hình.
- **Giai đoạn 2:** Chơi game, trong giai đoạn này chia làm 2 hoạt động là:
 - Cập nhật: Screen gửi Signal di chuyển cho Arrow mỗi 100ms để tăng trạng thái của Arrow tạo sự di chuyển cho Arrow.
 - Hành động: Button gửi Signal bắn tên cho Arrow mỗi khi nhấn nút. Arrow sẽ kiểm tra số mũi tên và nếu còn thì sẽ cập nhật trạng thái để bắn mũi tên ra tại vị trí hiện tại của Archery
- **Giai đoạn 3:** Kết thúc game, thực hiện cài đặt lại trạng thái của Arrow trước khi thoát game.

Code: Tương tự Archery (link tham khảo [Archery_game](#))



Hình 11: Border sequence

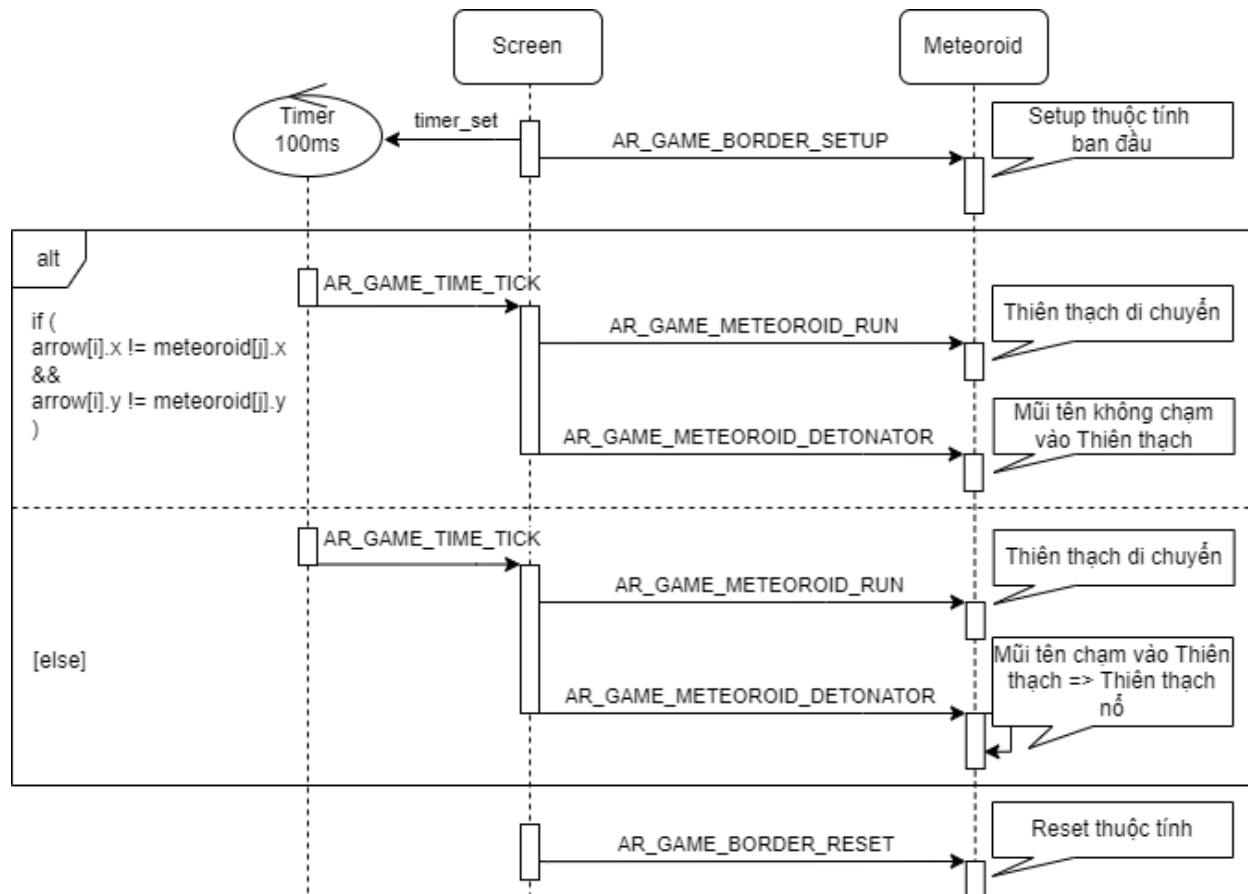
Tóm tắt nguyên lý: Border là 1 đối tượng bất động trong game. Có nhiệm vụ update level khi đến mốc điểm quy định và kiểm tra game over.

- **Giai đoạn 1:** Bắt đầu game, cài đặt thông số vị trí và hiển thị của Border.
- **Giai đoạn 2:** Chơi game, thực hiện các nhiệm vụ theo chu kỳ 100ms
 - Kiểm tra số điểm nếu số điểm thêm 200 thì tăng tốc độ của Meteoroid.
 - Kiểm tra vị trí của các Meteoroid nếu Meteoroid chạm vào Border thì gửi tín hiệu Reset đến Screen
- **Giai đoạn 3:** Kết thúc game, thực hiện cài đặt lại trạng thái của Border trước khi thoát game.

Code: Tương tự Archery (link tham khảo [Archery_game](#))

3.5 Meteoroid [↗](#)

Sequence diagram:



Hình 12: Meteoroid sequence

Tóm tắt nguyên lý: Meteoroid là đối tượng xuất hiện và di chuyển liên tục trong game nhận signal từ Screen. Chia làm 3 giai đoạn:

- **Giai đoạn 1:** Bắt đầu game, cài đặt thông số của Meteoroid. Cấp điểm xuất phát ngẫu nhiên cho Meteoroid, hiển thị lên màn hình.
- **Giai đoạn 2:** Chơi game, thực hiện các nhiệm vụ theo chu kỳ 100ms
 - Cập nhật vị trí và hoạt ảnh di chuyển cho Meteoroid
 - Kiểm tra vị trí của các Arrow nếu Arrow chạm vào Meteoroid thì thực hiện reset Arrow và Meteoroid rồi tạo Bang.
- **Giai đoạn 3:** Kết thúc game, thực hiện cài đặt lại trạng thái của Meteoroid trước khi thoát game.

Code: Tương tự Archery (link tham khảo [Archery_game](#))








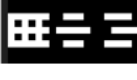

IV. Hiển thị và âm thanh trong trò chơi bắn cung

4.1 Đồ họa

Trong trò chơi, màn hình hiển thị là 1 màn hình **LCD Oled 1.3"** có kích thước là **128px*64px**. Nên các đối tượng được hiển thị trong game phải có kích thước hiển thị phù hợp với màn hình nên cần được thiết kế riêng.

Đồ họa được thiết kế từng phần theo từng đối tượng bằng phần mềm [Photopea](#)

Thiết kế đồ họa cho các đối tượng

Tên đối tượng		Hình ảnh	Mô tả
Archery (15x15 px)	bitmap_archery_I []		Cung lúc có tên
	bitmap_archery_II []		Cung lúc hết tên
Arrow (10x5px)	bitmap_arrow []		Mũi tên
Bang (15x15 px) (10x10 px)	bitmap_bang_I []		Vụ nổ animation 1
	bitmap_bang_II []		Vụ nổ animation 2
	bitmap_bang_III []		Vụ nổ animation 3
Meteoroid (20x10 px)	bitmap_meteoroid_I []		Thiên thạch animation 1
	bitmap_meteoroid_II []		Thiên thạch animation 2
	bitmap_meteoroid_III []		Thiên thạch animation 3

Hình 13: Bitmap của các đối tượng

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 đối tượng muốn miêu tả. Trong game, biến "action_image" trong đối tượng được sử dụng nối các ảnh theo thứ tự tạo thành animation.

Ghi chú: Trong thiết kế trên có nhiều ảnh khác nhau cho cùng 1 đối tượng để tạo animation cho đối tượng đó nhằm tăng tính chân thật lúc chơi game.


```

        SIZE_BITMAP_METEORIDS_X, \
        SIZE_BITMAP_METEORIDS_Y, \
        WHITE);
    }
    else if (meteoroid[i].action_image == 2) {
        view_render.drawBitmap( meteoroid[i].x, \
                                meteoroid[i].y, \
                                bitmap_meteoroid_II, \
                                SIZE_BITMAP_METEORIDS_X, \
                                SIZE_BITMAP_METEORIDS_Y, \
                                WHITE);
    }
    else if (meteoroid[i].action_image == 3) {
        view_render.drawBitmap( meteoroid[i].x, \
                                meteoroid[i].y, \
                                bitmap_meteoroid_III, \
                                SIZE_BITMAP_METEORIDS_X, \
                                SIZE_BITMAP_METEORIDS_Y, \
                                WHITE);
    }
}
}
}
}

```

Bang display:

```

void ar_game_bang_display() {
    for (uint8_t i = 0; i < NUM_BANG; i++) {
        if (bang[i].visible == WHITE) {
            if (bang[i].action_image == 1) {
                view_render.drawBitmap( bang[i].x, \
                                        bang[i].y, \
                                        bitmap_bang_I, \
                                        SIZE_BITMAP_BANG_I_X, \
                                        SIZE_BITMAP_BANG_I_Y, \
                                        WHITE);
            }
            else if (bang[i].action_image == 2) {
                view_render.drawBitmap( bang[i].x, \
                                        bang[i].y, \
                                        bitmap_bang_II, \
                                        SIZE_BITMAP_BANG_I_X, \
                                        SIZE_BITMAP_BANG_I_Y, \
                                        WHITE);
            }
            else if (bang[i].action_image == 3) {
                view_render.drawBitmap( bang[i].x + 2, \
                                        bang[i].y - 1, \
                                        bitmap_bang_III, \

```



```

        SIZE_BITMAP_BANG_II_X, \
        SIZE_BITMAP_BANG_II_Y, \
        WHITE);
    }
}
}

```

Border display:

```

void ar_game_border_display() {
    if (border.visible == WHITE) {
        view_render.drawFastVLine( border.x, \
                                   AXIS_Y_BORDER_ON, \
                                   AXIS_Y_BORDER_UNDER, \
                                   WHITE);

        for (uint8_t i = 0; i < NUM_METEORIDS; i++) {
            view_render.fillCircle( border.x, \
                                   meteoroid[i].y + 5, \
                                   1, \
                                   WHITE);
        }
    }
}

```



Game frame display:

```

void ar_game_frame_display() {
    view_render.setTextSize(1);
    view_render.setTextColor(WHITE);
    view_render.setCursor(2,55);
    view_render.print("Arrow:");
    view_render.print(settingsetup.num_arrow);
    view_render.setCursor(60,55);
    view_render.print(" Score:");
    view_render.print(ar_game_score);
    view_render.drawLine(0, LCD_HEIGHT, LCD_WIDTH, LCD_HEIGHT, WHITE);
    view_render.drawLine(0, LCD_HEIGHT-10, LCD_WIDTH, LCD_HEIGHT-10, WHITE);
    view_render.drawRect(0, 0, 128, 64, 1);
}

```



Screen display:

```

void view_scr_archery_game() {
    if (ar_game_status == GAME_ON) {
        ar_game_frame_display();
    }
}

```




```

    ar_game_archery_display();
    ar_game_arrow_display();
    ar_game_meteoroid_display();
    ar_game_bang_display();
    ar_game_border_display();
}
else if (ar_game_status == GAME_OVER) {
    view_render.clear();
    view_render.setTextSize(2);
    view_render.setTextColor(WHITE);
    view_render.setCursor(17, 24);
    view_render.print("YOU LOSE");
}
}
}

```

4.3 Âm thanh

Âm thanh được thiết kế qua website [Arduino Music](#)

Trong khi chơi, để trò chơi thêm phần sinh động và chân thật thì việc có âm thanh là điều cần thiết.

Các âm thanh cần thiết kể: nút nhấn, bắn tên, vụ nổ, nhạc game.

Code:

```

// Âm thanh Bắt đầu game
BUZZER_PlayTones(tones_SMB);

// Âm thanh Vụ nổ
BUZZER_PlayTones(tones BUM);

// Âm thanh nút nhấn
BUZZER_PlayTones(tones_cc);

// Âm thanh cảnh báo
BUZZER_PlayTones(tones_3beep);

// Merry Christmas
BUZZER_PlayTones(tones_merryChristmas);

/* _____BUZZER_____ */

void BUZZER_Sleep(bool sleep);
/* sleep = 0 : bật âm thanh
   sleep = 1 : tắt âm thanh */
static const Tone_TypeDef tones BUM[] = {
    {3000,3},

```



```
        {4500,6},
        { 0,0}
    };

    static const Tone_TypeDef tones_cc[] = {
        {2000,2},
        { 0,0},
    };

    static const Tone_TypeDef tones_startup[] = {
        {2000,3},
        { 0,3},
        {3000,3},
        { 0,3},
        {4000,3},
        { 0,3},
        {1200,4},
        { 0,6},
        {4500,6},
        { 0,0}    // <-- tones end
    };

    static const Tone_TypeDef tones_3beep[] = {
        {4000, 3},
        { 0,10},
        {1000, 6},
        { 0,10},
        {4000, 3},
        { 0, 0}
    };

    // "Super Mario bros." =
    static const Tone_TypeDef tones_SMB[] = {
        {2637,18}, // E7 x2
        { 0, 9}, // x3
        {2637, 9}, // E7
        { 0, 9}, // x3
        {2093, 9}, // C7
        {2637, 9}, // E7
        { 0, 9}, // x3
        {3136, 9}, // G7
        { 0,27}, // x3
        {1586, 9}, // G6
        { 0,27}, // x3

        {2093, 9}, // C7
        { 0,18}, // x2
        {1586, 9}, // G6
        { 0,18}, // x2
        {1319, 9}, // E6
        { 0,18}, // x2
```

```
{1760, 9}, // A6
{  0, 9}, // x1
{1976, 9}, // B6
{  0, 9}, // x1
{1865, 9}, // AS6
{1760, 9}, // A6
{  0, 9}, // x1
```

```
{1586,12}, // G6
{2637,12}, // E7
{3136,12}, // G7
{3520, 9}, // A7
{  0, 9}, // x1
{2794, 9}, // F7
{3136, 9}, // G7
{  0, 9}, // x1
{2637, 9}, // E7
{  0, 9}, // x1
{2093, 9}, // C7
{2349, 9}, // D7
{1976, 9}, // B6
{  0,18}, // x2
```

```
{2093, 9}, // C7
{  0,18}, // x2
{1586, 9}, // G6
{  0,18}, // x2
{1319, 9}, // E6
{  0,18}, // x2
{1760, 9}, // A6
{  0, 9}, // x1
{1976, 9}, // B6
{  0, 9}, // x1
{1865, 9}, // AS6
{1760, 9}, // A6
{  0, 9}, // x1
```

```
{1586,12}, // G6
{2637,12}, // E7
{3136,12}, // G7
{3520, 9}, // A7
{  0, 9}, // x1
{2794, 9}, // F7
{3136, 9}, // G7
{  0, 9}, // x1
{2637, 9}, // E7
{  0, 9}, // x1
{2093, 9}, // C7
{2349, 9}, // D7
{1976, 9}, // B6
```

```
{ 0, 0}
};

// Merry Christmas
static const Tone_TypeDef tones_merryChristmas[] = {
    {2637, 9}, // E7
    { 0, 9}, // x1
    {2637, 9}, // E7
    { 0, 9}, // x1
    {2637, 9}, // E7
    { 0,18}, // x2

    {2637, 9}, // E7
    { 0, 9}, // x1
    {2637, 9}, // E7
    { 0, 9}, // x1
    {2637, 9}, // E7
    { 0,18}, // x2

    {2637, 9}, // E7
    { 0, 9}, // x1
    {3136, 9}, // G7
    { 0, 9}, // x1
    {2093, 9}, // C7
    { 0, 9}, // x1
    {2349, 9}, // D7
    { 0, 9}, // x1
    {2637, 9}, // E7
    { 0,24}, // x2

    {2794, 9}, // F7
    { 0, 9}, // x1
    {2794, 9}, // F7
    { 0, 9}, // x1
    {2794, 9}, // F7
    { 0, 9}, // x1
    {2794, 9}, // F7
    { 0, 9}, // x1
    {2794, 9}, // F7
    { 0, 9}, // x1
    {2637, 9}, // E7
    { 0, 9}, // x1
    {2637, 9}, // E7
    { 0, 9}, // x1
    {2637, 9}, // E7
    { 0, 9}, // x1
    {2637, 9}, // E7
    { 0, 9}, // x1
    {2637, 9}, // E7
    { 0, 9}, // x1
    {2349, 9}, // D7
```

```
{ 0, 9}, // x1
{2349, 9}, // D7
{ 0, 9}, // x1
{2637, 9}, // E7
{ 0, 9}, // x1
{2349, 9}, // D7
{ 0, 9}, // x1
{3136, 9}, // G7
{ 0, 0} // <-- tones end
};
```

Ghi chú: Nếu không có thời gian hoặc không có khiếu âm nhạc thì tốt nhất nên dùng các thư viện trên github

Releases

No releases published

Packages

No packages published

Languages

● C 85.1% ● Assembly 6.4% ● C++ 4.9% ● HTML 3.1% ● Other 0.5%