**Báo Cáo Đồ án Hệ thống nhúng Căn Bản**

**Nhóm:**

Đặng Đình Khang-15520336

Phạm Văn Duy -15520169

Võ Minh Kha -15520330

**1.Server**  
server.js: file chạy node application

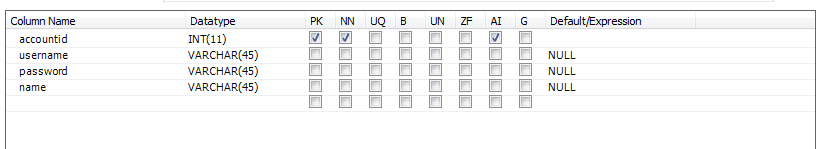
route: thư mục các file xử lý request

views: thư mục các file giao diện

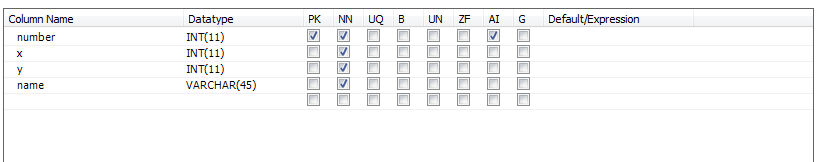
public: thư mục jquery.js, CSS

***1/ DATABASE***

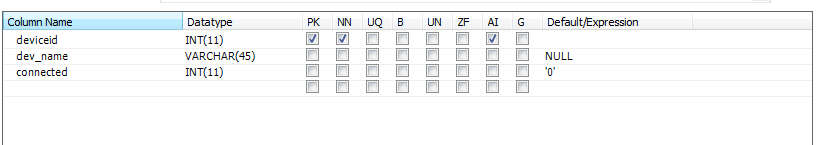
Tài khoản:



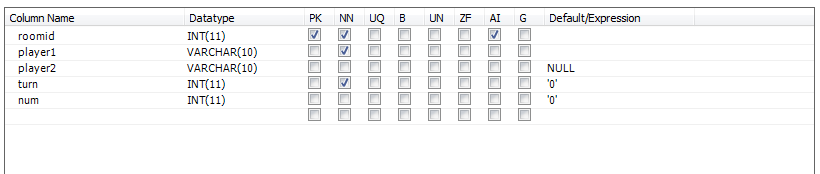
Map của user tạo:



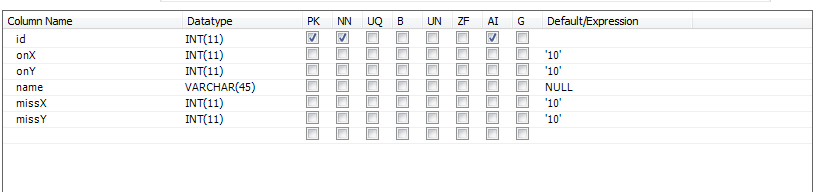
Device: các board kết nối tới server



Room: num là số player ở trong phòng, turn là lượt bắn trước



Fire: danh sách các tọa độ mà player đã bắn



***2/ Đăng nhập – Đăng ký:***

* 1. Đăng nhập:

main.js

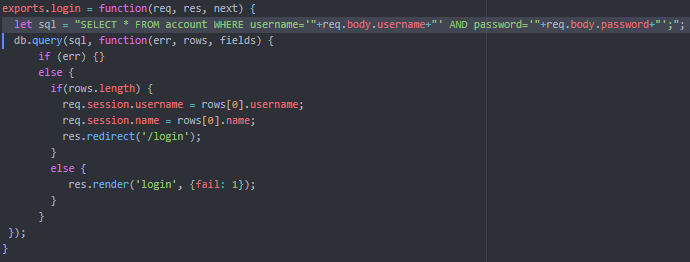


Link truy cập: /login. Tham số fail dùng để hiển thị dòng đăng nhập thất bại.

POST /login - xử lý Form đăng nhập:

* Truy xuất database, nếu có kết quả tức là tài khoản và password nhập vào đúng, sau đó tạo session cho user. Nếu không có kết quả, tải lại trang với dòng thông báo lỗi.

user.js



* 1. Đăng ký:

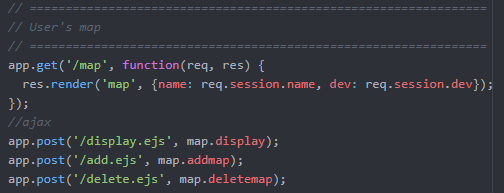
SELECT trong database và kiểm tra username, password nhập từ form có tồn tại hay không. Nếu có tải lại trang với thông báo lỗi cùng với form. Nếu không tải lại trang với dòng thông báo đăng ký thành công.

user.js

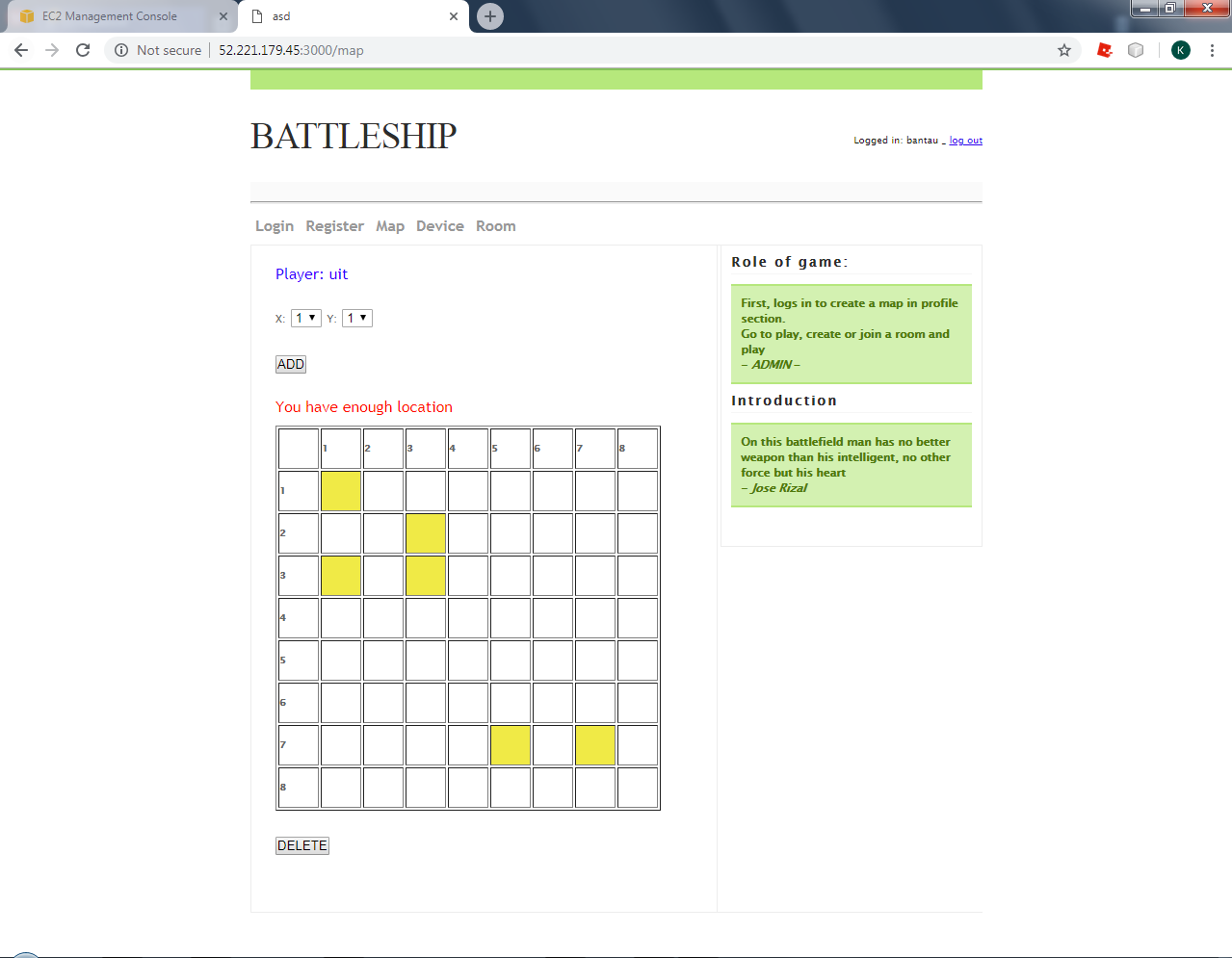


***3/ Tạo map cho user:***

main.js



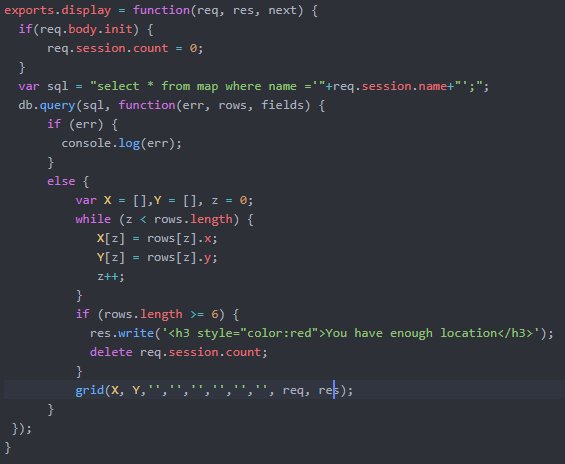
Giao diện:



Khi trang được tải, có một AJAX request /display.ejs với tham số init khởi tạo session count (số tọa độ user đã có) cho user.

* SELECT database trong table map, đưa các tọa độ chọn được vào hai mảng X[], Y[].
* Nếu user có đủ 6 tọa độ thì hủy session count.
* Hàm grid dùng để vẽ map 9\*9 như hình trên, 2 tham số đầu là mảng tọa độ của user,2 tham số tiếp theo là vị trí xy của con trỏ, 2 tham số tiếp theo là mảng tọa độ bắn trượt, 2 tham số tiếp theo là mảng tọa độ bắn trúng, id trong html, 2 object request và response.

map.js



Khi có sự kiện nhấn nút ADD, POST AJAX /add.ejs được request:

* INSERT tọa độ được chọn vào database và tăng biến session count.
* Sau đó request AJAX /display.ejs lại để cập nhật lại map và kiểm tra user có đủ 6 tọa độ hay chưa.

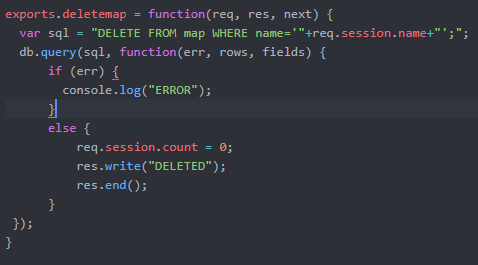
map.js



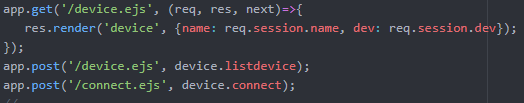
Khi có sự kiện nút nhấn DELETE, POST AJAX /delete.ejs được request:

* Xóa tất cả tọa độ trong database, cập nhật lại session, request AJAX /display.ejs để cập nhật lại map

map.js



***4/ Kết nối board:***

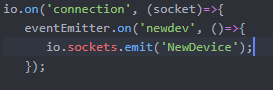
main.js  


Require thư viện ‘net’ trong server.js, listen port 6969 và tạo event listener.

Kết nối giao diện client với server bằng socket.io.

Khi nhấn nút kết nối trên tay cầm, board gửi mã số của board cho server và thêm vào database.

Khi nhận dữ liệu từ board, server kiểm tra có dấu : trong dữ liệu không, nếu có thì đó là lệnh chơi game, nếu không thì đó là mã của board và thêm vào database. Sau đó phát sự kiện có board vừa kết nối tới listener trong socket.io rồi emit lên những client đang ở trang device để request POST AJAX /device.ejs để cập nhật danh sách các board.



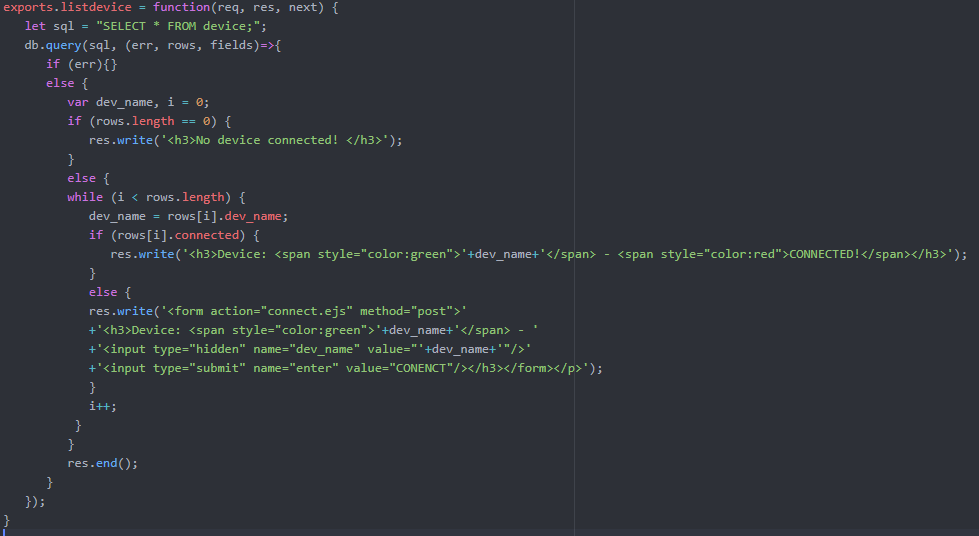
server.js

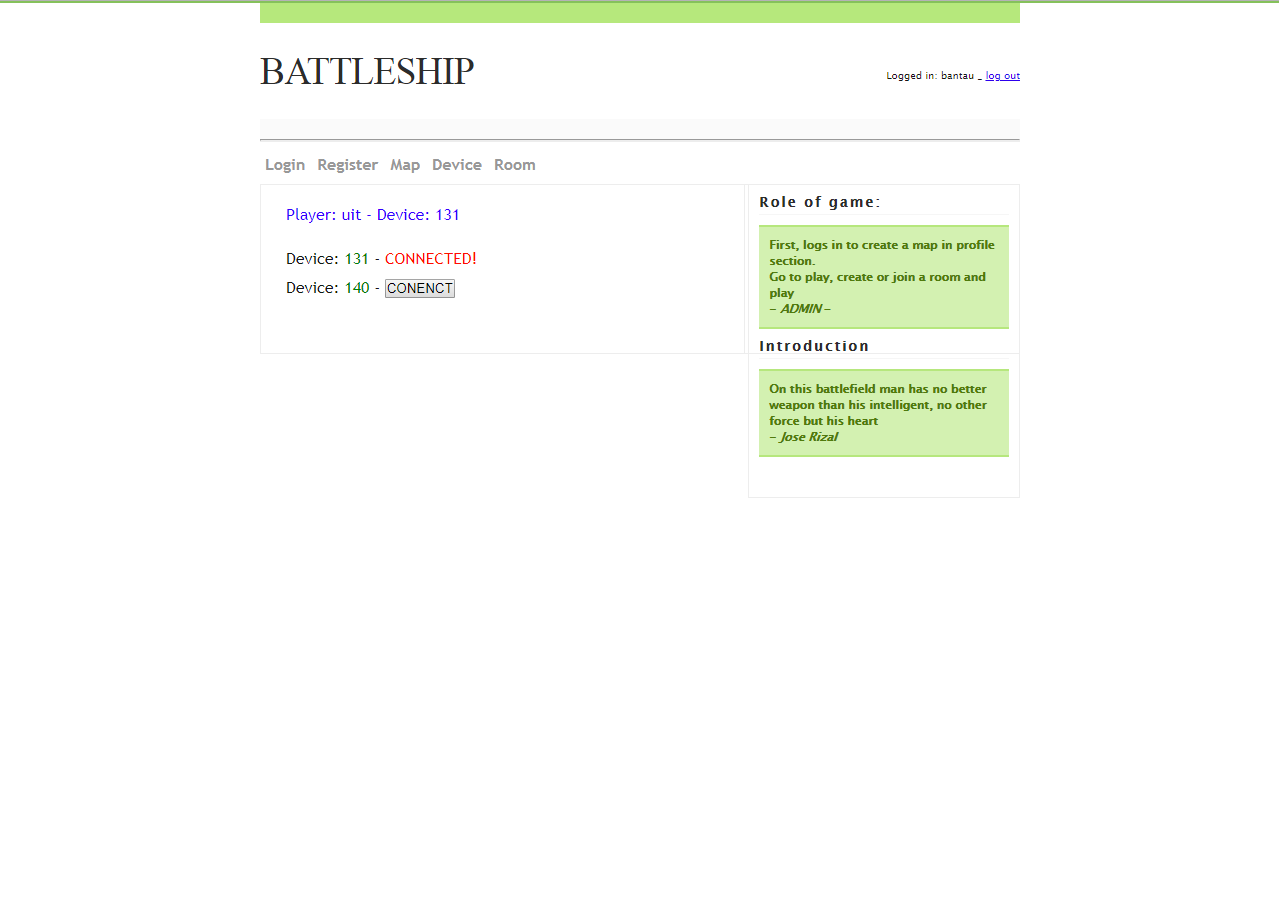


Khi user nhấn CONNECT, update lại trạng thái của board đó trong database, tạo session tên board vừa kết nối ‘dev’ và tải lại trang với trạng thái board được cập nhật.

Cách hiển thị danh sách board: SELECT trong database table device, nếu board đã connected thì hiển thị dòng connected. Nếu chưa thì tạo form gồm nút connected, hidden input mang tên board để Update trạng thái trong database.

device.js





***5/ Tạo room:***

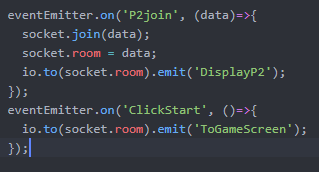
main.js

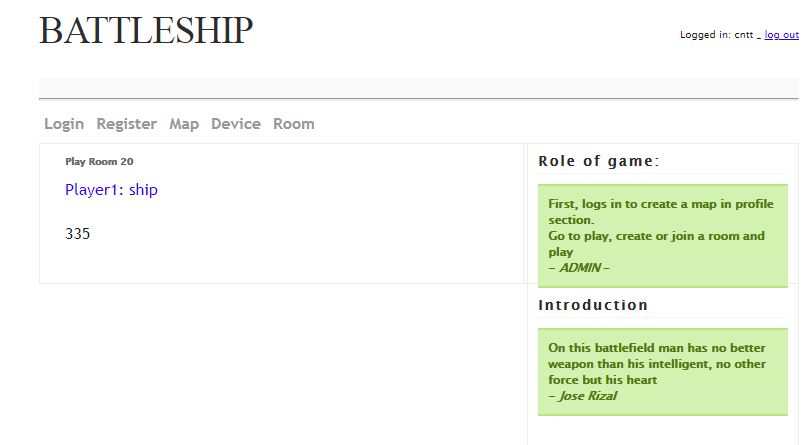


/room/index: trang hiển thị danh sách các room được tạo. POST AJAX được request mỗi 5s để cập nhật lại danh sách room.

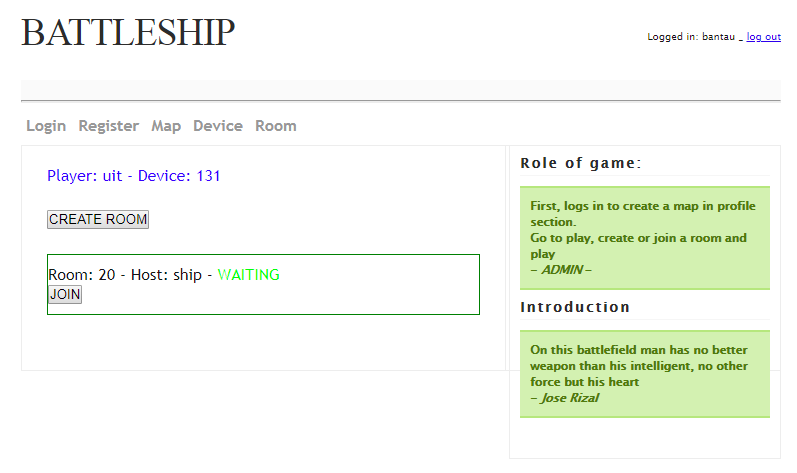
Khi user nhấn nút tạo room, INSERT vào database room mới, player1 là người nhấn tạo room, số player trong room ‘num’ = 1. Sau đó SELECT lại room mới tạo để gán session roomid cho user.

Giao diện phòng chờ được kết nối với server bằng socket.io, khi giao diện được tải lên, có một AJAX request /room/player1 để hiển thị tên player1 và join room socket.io với session roomid, sau đó socket.io emit lên client để cập nhật player2 nhưng lúc này player2 chưa join vào.



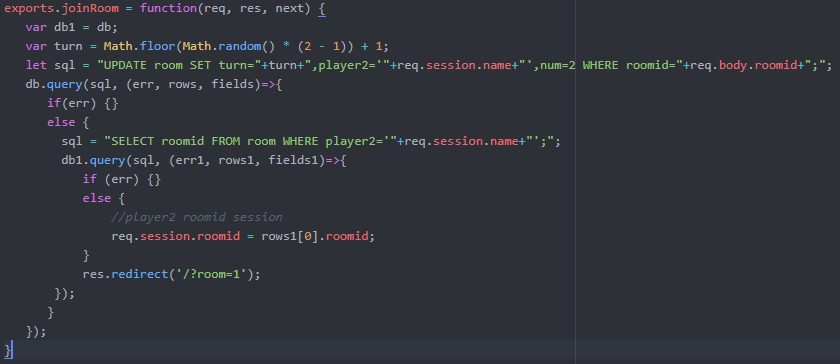
Giao diện của phòng chờ:

Các user khác cập nhật danh sách room:



Khi user nhấn join, link /room/join.ejs được request

* Thực hiện hàm random lượt chơi trong khoảng 1,2
* Cập nhật player2, turn, num = 2 trong database table room
* Chuyển đến giao diện phòng chờ, thực hiện request AJAX /room/player1 để join room socket.io như ở trên. Socket.io emit tới những client đã join vào room tên ‘roomid’ mà user đã tạo hoặc join, để cập nhật player2 vừa mới join vào room.
* Chỉ player1 mới thấy nút START GAME

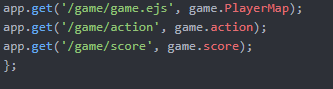


Player1 Nút START GAME dẫn đến link /room/start, player1 đươc chuyển đến màn hình game, và emit sự kiện click tới socket.io. Socket.io emit ngược lại cho player2 để chuyển đến màn hình game bằng window.location trên code client. Io.to(socket.room).emit chỉ emit sự kiện tới những client đã join vào room (roomid đã được tạo).

Trong quá trình thực thi hàm callback trong link AJAX hiển thị player2 /room/player2\_join.ejs, mỗi player được cấp một session turn bằng với cột turn trong database được insert khi player2 join vào.

***6/ Game:***

main.js



Khởi tạo màn hình game:

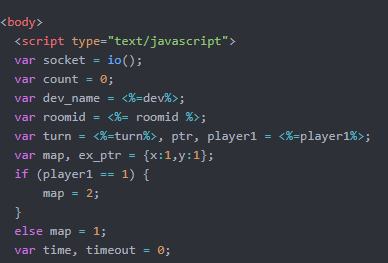
* SELECT các cột trong table room
* Tạo session score
* Chuyển chuỗi string của mã board thành giá trị integer.
* Nếu là player1 và turn của room = 1 thì tải trang với các giá trị turn = 1; player1 = 1 để gán vào các biến trong code javascript trên client.

game.js



Khởi tạo các biến trong javascript trên client:

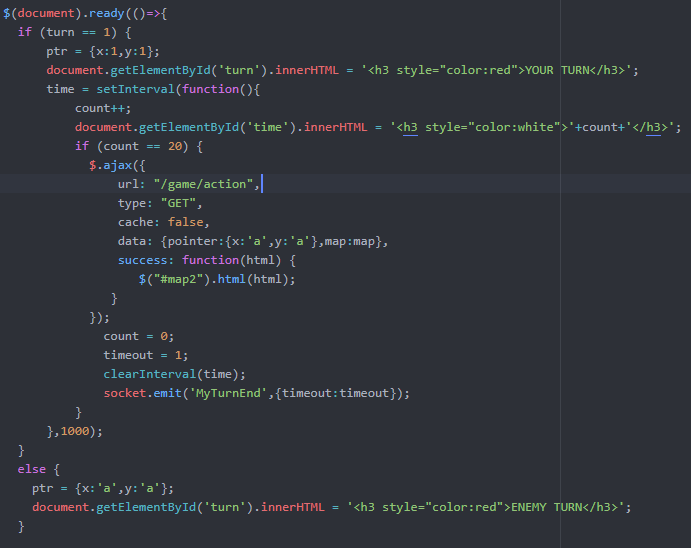
game.ejs



Biến map để chọn select map của địch khi request AJAX.

Khởi tạo timeout 20s, nếu hết 20s thì xóa bật cờ timeout và emit sự kiện kết thúc turn. Nếu không phải lượt của mình thì chỉ hiển thị ENEMY TURN trên màn hình.

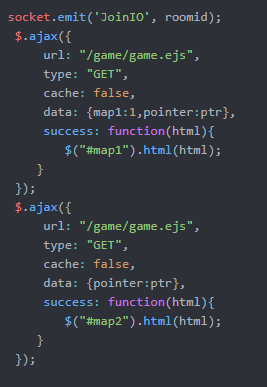
game.ejs



Sau khi khởi tại timer, join 2 client vào socket.io room như phần trên.

Sau đó khởi tạo 2 map của mình và địch. Cũng thực hiện SELECT từ table map và thực hiện hàm grid để vẽ. Với map địch thì truyền vào vị trí con trỏ ptr đã khởi tạo ở trên, nếu tới lượt turn = 1 thì ptr là {1:1} được hiển thị trên bản đồ, nếu không có lượt ptr = {a;a} và không được vẽ lên map.

game.ejs



Lệnh chơi game của board gồm có mã board : lệnh. Tách 2 lệnh thành 2 string. String đầu tiên để so sánh với device name mà user đã kết nối, string thứ hai để thực hiện lệnh. Có 5 lệnh chơi game 131:A, 131:S, 131:W, 131:D, 131:F. Nếu string 2 là F thì emit sự kiện fire tới event listener trong socket.io và socket.io emit lên client. Còn không thì emit Move để di chuyển con trỏ.

server.js

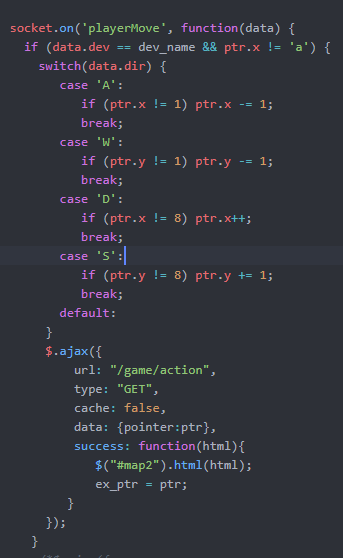


Ở client, khi nhận được sự kiện thì tiến hành so sánh mã board từ lệnh với mã board user connect và tọa độ con trỏ để đảm bảo chỉ tới lượt thì mới được thực thi. Ptr ={a;a} tức là không phải lượt của mình.

Với lệnh move, tiến hành so sánh và tăng giảm tọa độ con trỏ theo hướng đi, sau đó request AJAX thực hiện hàm action để cập nhật lại bản đồ với con trỏ mới.

Với lệnh fire, tiến hành request AJAX để cập nhật kết quả map sau khi bắn, hiển thị điểm và thông báo kết thúc lượt. Đồng thời xóa cờ timeout để thông báo là lượt kết thúc do player bắn.

game.ejs

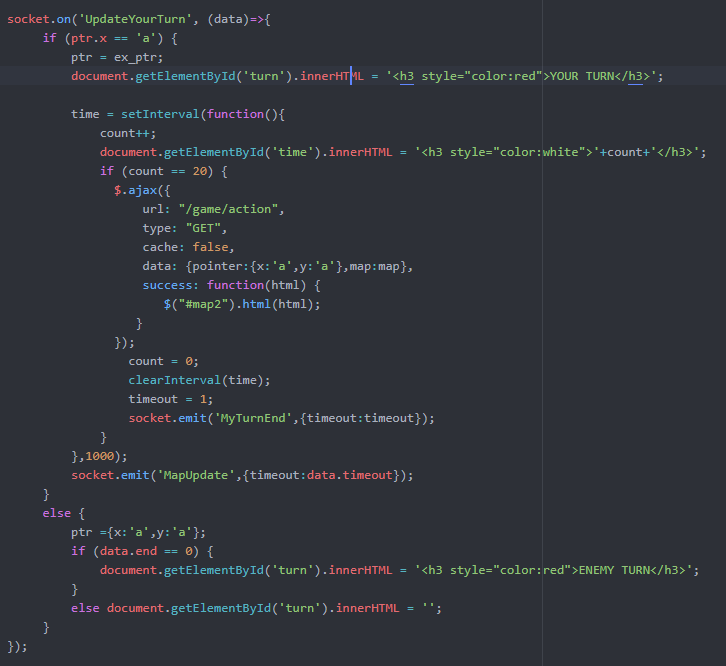


Socket.io nhận được sự kiện kết thúc từ player vừa bắn và emit lại cho cả 2 player trong room để cập nhật giao diện, chuyển đổi turn, pointer

Player đang có con trỏ = {a;a} sẽ được cấp con trỏ = con trỏ mà player đã bắn ở lượt của mình trước đó (ex\_ptr) và khởi tạo timeout và emit socket.io để hiện con trỏ.

Player sau khi bắn thì lưu con trỏ vào ex\_ptr và con trỏ trả về = {a;a}.

game.ejs

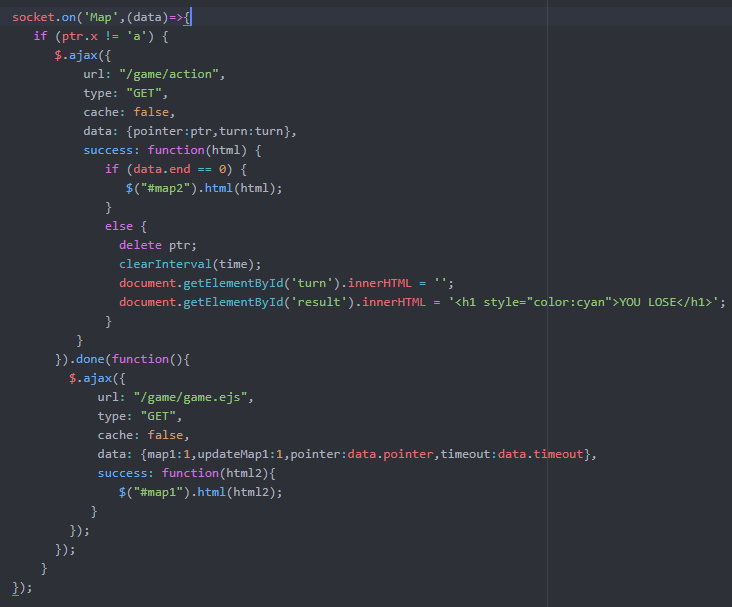


Cập nhật lại map cho player khi tới lượt của mình.

AJAX thứ nhất cập nhật map mới xuất hiện con trỏ trên màn hình.

AJAX thứ hai cập nhật vị trí mà player trước đã bắn vào mình.

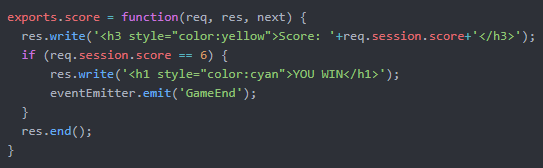
game.ejs



Mỗi lần player bắn trúng tàu địch, session score sẽ tăng 1, nếu bắn trùng vị trí cũ thì score giữ nguyên.

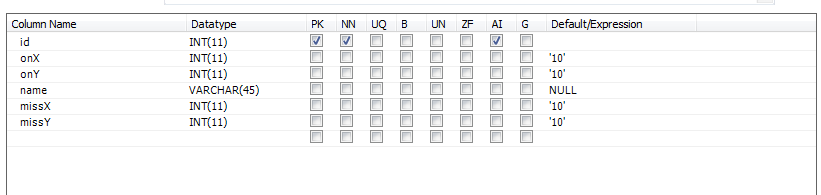
Nếu player bắn hết 6 tàu địch thì hiển thị dòng You win và emit sự kiện kết thúc game tới socket.io. Socket.io sẽ truyền tham số end = 1 khi cho player cập nhật map khi tới lượt của mình. Và hiển thị You lose.

game.js



Xét tọa độ trúng hoặc không trúng:

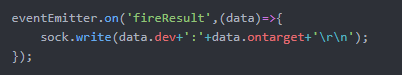
* AJAX request trong sự kiện ‘playerFire’ gửi lên server map, con trỏ và cờ fire. Map dùng để SELECT tọa độ của địch.
* Đầu tiên, SELECT những tọa độ mà mình đã bắn trước đó trong table fire. Nếu ở turn đầu tiên thì sẽ không có kết quả nào.



* Gán các tọa độ trúng và trượt vào các mảng.
* So sánh con trỏ với các tọa độ trên để xác định tọa độ bắn trúng có bị trùng hay không.
* SELECT tọa độ của địch trong table map bằng cách join bảng room và map với nhau.
* Nếu cờ báo trùng bật lên thì bỏ qua và hiển thị map với các tọa độ mình đã bắn.
* Nếu không có trùng thì xét cờ ontarget xem có trúng hay không
  + Nếu trúng thì thêm con trỏ vào mảng onX,Y, tăng score, và emit sự ‘fireresult’ tới event listener trong TCP/IP và trả về server mã board của mình : 1 ứng với bắn trúng.
  + Nếu trượt thì thêm vào mảng missX,Y
  + Cập nhật lại map.
* Sau cùng, truyền giá trị con trỏ đến socket.io. Socket.io truyền lên con trỏ cho client để thực hiện AJAX request ở phần cập nhật vị trí địch đã bắn.

Trong phần cập nhật vị trí địch đã bắn, nếu vị trí con trỏ = với các tọa độ thì emit sự kiện ‘fireresult’ với chuỗi trả về board là mã board : T ứng với bị bắn trúng.

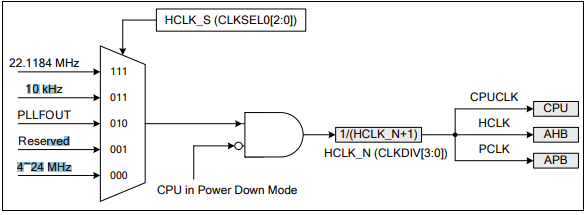
server.js



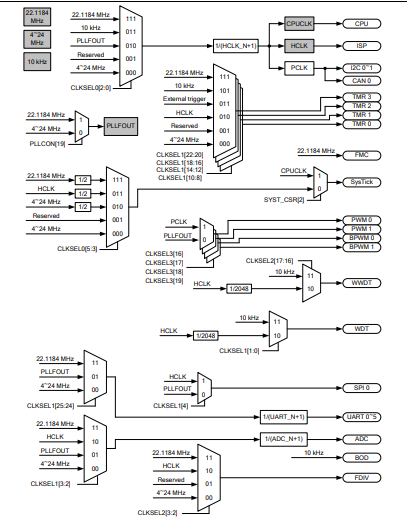
Board sau khi nhận dữ liệu trả về thì tách chuỗi. Nếu không phải mã của mình thì bỏ qua.

**2.Phần cứng:(Tay Cầm sử dụng KIT NUC131 và ESP8266)**

Để NUC 131 hoạt động ổn định và truyền nhận UART trước tiên phải set giá trị Clock NGUỒN và clock ,Baurate,khung truyền của UART



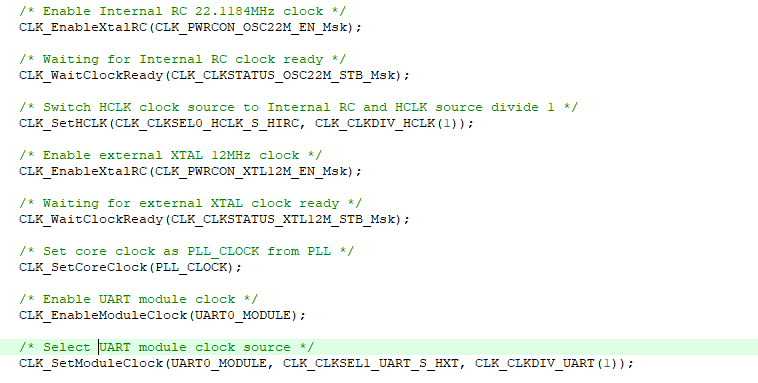
Hình 1\_System Clock Block Diagram NUC 131



Hình 2\_Clock Generator Global View Diagram

Như bạn thấy ở trong Hình 2 tất cả các phần trong NUC muốn hoạt động thì đề có thể set Clock riêng với bộ chia xung riêng .

Vì thế muốn hoạt động NUC 131 thì bộ thư viện của NUC131 cũng cung cấp các hàm set Clock và bộ chia



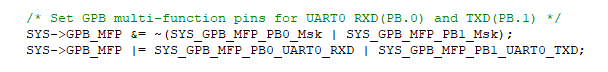
Set baurate cho UART :



Cấu hình khung truyền



Set 2 chân TX,RX của UART(PB.0 là RX,PB.1 là TX)



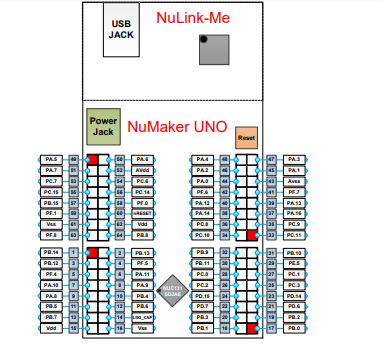
Xử lí với nút bấm:

Có 4 chế độ cơ bản nhất của IO là

* Input
* Open Drain
* Push-Pull Ouput
* Quasi-Bidirection .

Trong project trên sử dụng 6 chân GPIO:

**PA10,PA11,PA12, PA13,PA14,PA15**

sử dụng cho nút bấm. 

**Thiết lập GPIO để sử dụng ngắt khi nhấn nút nhấn xuống.**

GPIO\_SetMode(PA, BIT10, GPIO\_PMD\_QUASI);

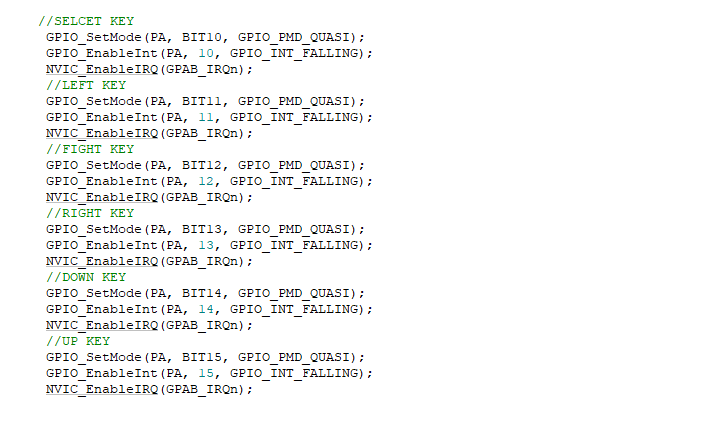
//Cho phép ngắt

GPIO\_EnableInt(PA, 10, GPIO\_INT\_FALLING);

Ở đây sử dụng ngắt GPIO\_INT\_FALLING(khi nhấn nút điện áp được kéo xuống 0 và cho phép ngắt xảy ra).

Cho phép vào ngắt ngắt của PA nằm trong hàm phục vụ ngắt GPAB\_IRQn

NVIC\_EnableIRQ(GPAB\_IRQn);



**Hàm phục vụ ngắt GPAB\_IRQHandler:**

void GPAB\_IRQHandler(void)

{

/\* To check if PA.10 interrupt occurred \*/

if(GPIO\_GET\_INT\_FLAG(PA, BIT10))

{

GPIO\_CLR\_INT\_FLAG(PA, BIT10);

btnselect=1;

PA->ISRC = PA->ISRC;

}

/\* To check if PA.11 interrupt occurred \*/

else if(GPIO\_GET\_INT\_FLAG(PA, BIT11))

{

GPIO\_CLR\_INT\_FLAG(PA, BIT11);

btnleft=1;

PA->ISRC = PA->ISRC;

}

/\* To check if PA.15 interrupt occurred \*/

else if(GPIO\_GET\_INT\_FLAG(PA, BIT15))

{

GPIO\_CLR\_INT\_FLAG(PA, BIT15);

btnup=1;

PA->ISRC = PA->ISRC;

}

/\* To check if PA.14 interrupt occurred \*/

else if(GPIO\_GET\_INT\_FLAG(PA, BIT14))

{

GPIO\_CLR\_INT\_FLAG(PA, BIT14);

btndown=1;

PA->ISRC = PA->ISRC;

}

/\* To check if PA.13 interrupt occurred \*/

else if(GPIO\_GET\_INT\_FLAG(PA, BIT13))

{

GPIO\_CLR\_INT\_FLAG(PA, BIT13);

btnright=1;

PA->ISRC = PA->ISRC;

}

/\* To check if PA.12 interrupt occurred \*/

else if(GPIO\_GET\_INT\_FLAG(PA, BIT12))

{

GPIO\_CLR\_INT\_FLAG(PA, BIT12);

btnfire=1;

PA->ISRC = PA->ISRC;

}

else

{

/\* Un-expected interrupt. Just clear all PA, PB interrupts \*/

PA->ISRC = PA->ISRC;

PB->ISRC = PB->ISRC;

}

}

Khi ngắt GPIO xảy ra do chúng ta nhấn nút bấm thì trong hàm ngắt sẽ so sánh kiểm tra xem ngắt xảy ra ở chân GPIO nào (PA111,PA12,PA13,PA14,PA14,PA15) .Khi biết các nút bấm nằm trong GPIO nào thì set các biến hoặc btnselect=1 hoặc btnup=1hoặc btndown=1 hoặc btnleft=1 hoặc btnright=1 hoặc btnfire=1;

Các biến này quyết định việc thực hiện trong hàm while(1) trong hàm main().

Nếu **btnselect=1** những lệnh gửi qua UART từ NUC 131 tới ESP:

**-“AT+RST\r\n”**

**-"AT+CWJAP=\"noname\",\"12345677\"\r\n" //kết nối wifi**

**-"AT+CIPSTART=\"TCP\",\"54.169.158.78\",6969\r\n" //kết nối địa chỉ IP qua TCP port 6969**

**-"AT+CIPSEND=3\r\n" //gửi 3 kí tự**

**-"131" //** **Tên board để user connect vào**

Các lệnh gửi xong thì **reset btnselect=0** để không gửi lại các lệnh đã gửi

Nếu **btnleft==1 thì các lệnh sẽ gửi:**

**"AT+CIPSEND=5\r\n"**//gửi 5 kí tự

**"131:A"** // server sẽ hiểu là muốn điều khiển con trỏ sang trái

Gửi **OK** 2 lệnh trên thì reset **btnleft=0 để thoát ra không gửi lại lệnh**

Nếu **btnright==1 thì các lệnh sẽ gửi:**

**"AT+CIPSEND=5\r\n"**//gửi 5 kí tự

**"131:D"** // server sẽ hiểu là muốn điều khiển con trỏ sang trái

Gửi **OK** 2 lệnh trên thì reset **btnright=0 để thoát ra không gửi lại lệnh**

Nếu **btnup==1 thì các lệnh sẽ gửi:**

**"AT+CIPSEND=5\r\n"**//gửi 5 kí tự

**"131:W"** // server sẽ hiểu là muốn điều khiển con trỏ lên trên

Gửi **OK** 2 lệnh trên thì reset **btnup=0 để thoát ra không gửi lại lệnh**

Nếu **btndown==1 thì các lệnh sẽ gửi:**

**"AT+CIPSEND=5\r\n"**//gửi 5 kí tự

**"131:S"** // server sẽ hiểu là muốn điều khiển con trỏ xuống dưới

Gửi **OK** 2 lệnh trên thì reset **btndown=0 để thoát ra không gửi lại lệnh**

Nếu **btnfire==1 thì các lệnh sẽ gửi:**

**"AT+CIPSEND=5\r\n"**//gửi 5 kí tự

**"131:S"** // server sẽ hiểu là muốn điều khiển con trỏ xuống dưới

Gửi **OK** 2 lệnh trên thì reset **btnfire=0 để thoát ra không gửi lại lệnh.**

**Còn trường hợp khác thì cho tất cả các btnselect=0,btnup=0,btndown=0,btnleft=0,btnright=0,btnfire=0.**

**//SELECT KEY**

if(btnselect==1){

sendComand("AT+RST\r\n");

Delay(10000000);

sendComand("AT+CWJAP=\"noname\",\"12345677\"\r\n");

Delay(10000000);

sendComand("AT+CIPSTART=\"TCP\",\"54.169.158.78\",6969\r\n");

Delay(1000000);

sendComand("AT+CIPSEND=3\r\n");

Delay(5000);

sendComand("131");

Delay(10000);

btnselect=0;

}

**//LEFT KEY**

else if(btnleft==1)

{

sendComand("AT+CIPSEND=5\r\n");

Delay(1000);

sendComand("131:A");

Delay(1000);

btnleft=0;

}

**//UP KEY**

else if(btnup==1)

{

sendComand("AT+CIPSEND=5\r\n");

Delay(1000);

sendComand("131:W");

Delay(1000);

btnup=0;

}

**//RIGHT KEY**

else if(btnright==1)

{

sendComand("AT+CIPSEND=5\r\n");

Delay(1000);

sendComand("131:D");

Delay(1000);

btnright=0;

}

**//down key**

else if(btndown==1)

{

sendComand("AT+CIPSEND=5\r\n");

Delay(1000);

sendComand("131:S");

Delay(1000);

btndown=0;

}

**//fire key**

else if(btnfire==1)

{

sendComand("AT+CIPSEND=5\r\n");

Delay(1000);

sendComand("131:F");

Delay(1000);

btnfire=0;

}

else

{

btnselect=0;

btnleft=0;

btnright=0;

btnup=0;

btndown=0;

btnfire=0;

}

**Hàm gửi lệnh UART**

**Để gửi lênh thì sử dụng hàm gửi lệnh do NUC 131 cung cấp**

**cmd là chuỗi ki tự lệnh chúng ta muốn gửi.**

**Biến i=0 sẽ chạy hết chiều dài chuỗi và lần lượt gửi từng kí tự với hàm**

UART\_WRITE(UART0,cmd[i]);

**Trong khi kí tự chưa gửi xong thì sẽ chờ đến khi kí tự gửi kết thúc mới quay lại gửi kí tự tiếp thoe qua hàm**

**while(!UART\_IS\_TX\_EMPTY(UART0)){};**

**Code truyền UART:**

while((i<strlen(cmd))){

UART\_WRITE(UART0,cmd[i]);

i++;

while(!UART\_IS\_TX\_EMPTY(UART0)){};

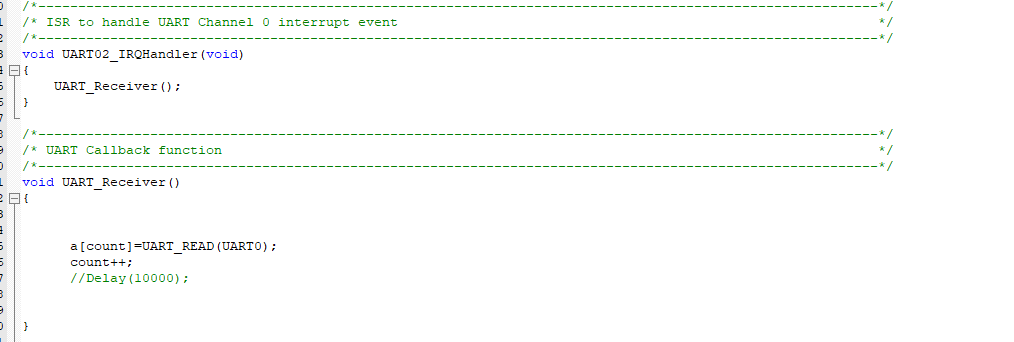
}

**Trong hàm sendComand(cmd) chúng ta sử dụng**

UART\_EnableInt(UART0, UART\_IER\_RDA\_IEN\_Msk);

**để cho phép ngắt nhận UART.**

**Hàm ngắt nhận UART:**



**Khi xảy ra ngắt vào hàm xử lí ngắt để nhận các giá trị trả về .Từng kí tự nhận về để đọc ra thì sử dụng hàm do NUC131 cung cấp UART\_READ(UART0);**

a[count]=UART\_READ(UART0);

count++;

**các giá trị được đọc ra từ UART sẽ được lưu vào mảng a[50] ,biến count++ sẽ tang vị trí trong mảng để lưu từng giá trị trả về**

**Để hàm ngắt nhận xảy ra hoàn tất thì sử dụng hàm Delay(int number)**

void Delay(int num){

int j=0;

for(j=0;j<num;j++){}

}

**Ở đây mình Delay(12000000); để việc nhận giá trị trả về hoàn tất.**

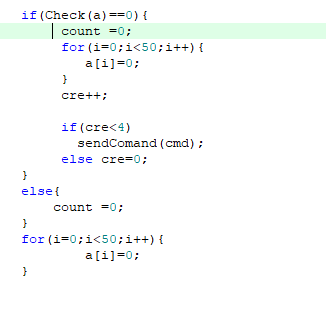
**Sau khi nhận kết thúc sẽ disable ngắt nhận UART đi**

*UART\_DisableInt(UART0,UART\_IER\_RDA\_IEN\_Msk);*

**Mảg a[] sẽ lưu toàn bộ kí tự trả về .**



**Hàm Check() được sử dụng để kiểm tra trong mảng trả về có chuỗi “OK” không.Sử dụng biến result để lưu giá trị khi có chuỗi “OK” thì result sẽ tăng lên.**



Nếu giá trị trả về không có **“OK”** thì reset mảng a[] sẽ gửi lại lệnh sendComand() .Nếu gửi 4 lần mà giá trị trả về vẫn không chứa chuỗi **“OK”** thì quá trình gửi sẽ dừng lại. Nếu chuỗi trả về có **“OK”** thì reset mảng a[] và gửi các lệnh tiếp theo.

**Code hàm sendComand(char \*cmd):**

void sendComand(char \*cmd){

uint8\_t i=0;

UART\_EnableInt(UART0, UART\_IER\_RDA\_IEN\_Msk);

while((i<strlen(cmd))){

UART\_WRITE(UART0,cmd[i]);

i++;

while(!UART\_IS\_TX\_EMPTY(UART0)){};

}

Delay(12000000);

UART\_DisableInt(UART0,UART\_IER\_RDA\_IEN\_Msk);

if(Check(a)==0){

count =0;

for(i=0;i<50;i++){

a[i]=0;

}

cre++;

if(cre<4)

sendComand(cmd);

else cre=0;

}

else{

count =0;

}

for(i=0;i<50;i++){

a[i]=0;

}

}